MDN Web Docs
moz://a

Date

JavaScript **`Date`** objects represent a single moment in time in a platform-independent format. `Date` objects contain a `Number` that represents milliseconds since 1 January 1970 UTC.

> **Note:** TC39 is working on Temporal, a new Date/Time API. Read more about it on the Igalia blog. It is not yet ready for production use!

Description

The ECMAScript epoch and timestamps

A JavaScript date is fundamentally specified as the number of milliseconds that have elapsed since midnight on January 1, 1970, UTC. This date and time are not the same as the **UNIX epoch** (the number of seconds that have elapsed since midnight on January 1, 1970, UTC), which is the predominant base value for computer-recorded date and time values.

Note: It's important to keep in mind that while the time value at the heart of a Date object is UTC, the basic methods to fetch the date and time or its components all work in the local (i.e. host system) time zone and offset.

It should be noted that the maximum `Date` is not of the same value as the maximum safe integer (`Number.MAX_SAFE_INTEGER` is 9,007,199,254,740,991). Instead, it is defined in ECMA-262 that a maximum of ±100,000,000 (one hundred million) days relative to January 1, 1970 UTC (that is, April 20, 271821 BCE ~ September 13, 275760 CE) can be represented by the standard `Date` object (equivalent to ±8,640,000,000,000,000 milliseconds).

Date format and time zone conversions

There are several methods available to obtain a date in various formats, as well as to perform time zone conversions. Particularly useful are the functions that output the date and time in Coordinated Universal Time (UTC), the global standard time defined by the World Time Standard.

(This time is historically known as *Greenwich Mean Time*, as UTC lies along the meridian that includes London—and nearby Greenwich—in the United Kingdom.) The user's device provides the local time.

In addition to methods to read and alter individual components of the local date and time (such as `getDay()` and `setHours()`), there are also versions of the same methods that read and manipulate the date and time using UTC (such as `getUTCDay()` and `setUTCHours()`).

Constructor

`Date()`

When called as a function, returns a string representation of the current date and time, exactly as `new Date().toString()` does.

`new Date()`

When called as a constructor, returns a new `Date` object.

Static methods

`Date.now()`

Returns the numeric value corresponding to the current time—the number of milliseconds elapsed since January 1, 1970 00:00:00 UTC, with leap seconds ignored.

`Date.parse()`

Parses a string representation of a date and returns the number of milliseconds since 1 January, 1970, 00:00:00 UTC, with leap seconds ignored.

> **Note:** Parsing of strings with `Date.parse` is strongly discouraged due to browser differences and inconsistencies.

`Date.UTC()`

Accepts the same parameters as the longest form of the constructor (i.e. 2 to 7) and returns the number of milliseconds since January 1, 1970, 00:00:00 UTC, with leap seconds ignored.

Instance methods

Instance methods

`Date.prototype.getDate()`

Returns the day of the month (`1` – `31`) for the specified date according to local time.

`Date.prototype.getDay()`

Returns the day of the week (`0` – `6`) for the specified date according to local time.

`Date.prototype.getFullYear()`

Returns the year (4 digits for 4-digit years) of the specified date according to local time.

`Date.prototype.getHours()`

Returns the hour (`0` – `23`) in the specified date according to local time.

`Date.prototype.getMilliseconds()`

Returns the milliseconds (`0` – `999`) in the specified date according to local time.

`Date.prototype.getMinutes()`

Returns the minutes (`0` – `59`) in the specified date according to local time.

`Date.prototype.getMonth()`

Returns the month (`0` – `11`) in the specified date according to local time.

`Date.prototype.getSeconds()`

Returns the seconds (`0` – `59`) in the specified date according to local time.

`Date.prototype.getTime()`

Returns the numeric value of the specified date as the number of milliseconds since January 1, 1970, 00:00:00 UTC. (Negative values are returned for prior times.)

`Date.prototype.getTimezoneOffset()`

Returns the time-zone offset in minutes for the current locale.

`Date.prototype.getUTCDate()`

Returns the day (date) of the month (`1` – `31`) in the specified date according to universal time.

`Date.prototype.getUTCDay()`

Returns the day of the week (`0` – `6`) in the specified date according to universal time.

`Date.prototype.getUTCFullYear()`

Returns the year (4 digits for 4-digit years) in the specified date according to universal time.

`Date.prototype.getUTCHours()`

Returns the hours (`0` – `23`) in the specified date according to universal time.

`Date.prototype.getUTCMilliseconds()`

Returns the milliseconds (`0` – `999`) in the specified date according to universal time.

`Date.prototype.getUTCMinutes()`

Returns the minutes (`0` – `59`) in the specified date according to universal time.

`Date.prototype.getUTCMonth()`

Returns the month (`0` – `11`) in the specified date according to universal time.

`Date.prototype.getUTCSeconds()`

Returns the seconds (`0` – `59`) in the specified date according to universal time.

`Date.prototype.getYear()`

Returns the year (usually 2–3 digits) in the specified date according to local time. Use `getFullYear()` instead.

`Date.prototype.setDate()`

Sets the day of the month for a specified date according to local time.

`Date.prototype.setFullYear()`

Sets the full year (e.g. 4 digits for 4-digit years) for a specified date according to local time.

`Date.prototype.setHours()`

Sets the hours for a specified date according to local time.

`Date.prototype.setMilliseconds()`

Sets the milliseconds for a specified date according to local time.

`Date.prototype.setMinutes()`
: Sets the minutes for a specified date according to local time.

`Date.prototype.setMonth()`
: Sets the month for a specified date according to local time.

`Date.prototype.setSeconds()`
: Sets the seconds for a specified date according to local time.

`Date.prototype.setTime()`
: Sets the `Date` object to the time represented by a number of milliseconds since January 1, 1970, 00:00:00 UTC. Use negative numbers for times prior.

`Date.prototype.setUTCDate()`
: Sets the day of the month for a specified date according to universal time.

`Date.prototype.setUTCFullYear()`
: Sets the full year (e.g. 4 digits for 4-digit years) for a specified date according to universal time.

`Date.prototype.setUTCHours()`
: Sets the hour for a specified date according to universal time.

`Date.prototype.setUTCMilliseconds()`
: Sets the milliseconds for a specified date according to universal time.

`Date.prototype.setUTCMinutes()`
: Sets the minutes for a specified date according to universal time.

`Date.prototype.setUTCMonth()`
: Sets the month for a specified date according to universal time.

`Date.prototype.setUTCSeconds()`
: Sets the seconds for a specified date according to universal time.

`Date.prototype.setYear()`

Sets the year (usually 2–3 digits) for a specified date according to local time. Use `setFullYear()` instead.

`Date.prototype.toDateString()`

Returns the "date" portion of the **`Date`** as a human-readable string like `'Thu Apr 12 2018'`.

`Date.prototype.toISOString()`

Converts a date to a string following the ISO 8601 Extended Format.

`Date.prototype.toJSON()`

Returns a string representing the **`Date`** using `toISOString()`. Intended for use by `JSON.stringify()`.

`Date.prototype.toGMTString()`

Returns a string representing the **`Date`** based on the GMT (UTC) time zone. Use `toUTCString()` instead.

`Date.prototype.toLocaleDateString()`

Returns a string with a locality sensitive representation of the date portion of this date based on system settings.

`Date.prototype.toLocaleString()`

Returns a string with a locality-sensitive representation of this date. Overrides the `Object.prototype.toLocaleString()` method.

`Date.prototype.toLocaleTimeString()`

Returns a string with a locality-sensitive representation of the time portion of this date, based on system settings.

`Date.prototype.toString()`

Returns a string representing the specified **`Date`** object. Overrides the `Object.prototype.toString()` method.

`Date.prototype.toTimeString()`

Returns the "time" portion of the `Date` as a human-readable string.

`Date.prototype.toUTCString()`

Converts a date to a string using the UTC timezone.

`Date.prototype.valueOf()`

Returns the primitive value of a `Date` object. Overrides the `Object.prototype.valueOf()` method.

Examples

Several ways to create a Date object

The following examples show several ways to create JavaScript dates:

> **Note:** Parsing of date strings with the `Date` constructor (and `Date.parse`, they are equivalent) is strongly discouraged due to browser differences and inconsistencies.

```
let today = new Date()
let birthday = new Date('December 17, 1995 03:24:00')
let birthday = new Date('1995-12-17T03:24:00')
let birthday = new Date(1995, 11, 17)            // the month is 0-indexed
let birthday = new Date(1995, 11, 17, 3, 24, 0)
let birthday = new Date(628021800000)            // passing epoch timestamp
```

To get Date, Month and Year or Time

```
const date = new Date();
const [month, day, year]       = [date.getMonth(), date.getDate(), date.ge
const [hour, minutes, seconds] = [date.getHours(), date.getMinutes(), date.get
```

Interpretation of two-digit years

`new Date()` exhibits legacy undesirable, inconsistent behavior with two-digit year values; specifically, when a `new Date()` call is given a two-digit year value, that year value does not get treated as a literal year and used as-is but instead gets interpreted as a relative offset — in some cases as an offset from the year `1900`, but in other cases, as an offset from the year `2000`.

```
let date = new Date(98, 1)          // Sun Feb 01 1998 00:00:00 GMT+0000 (G
let date = new Date(98, 1)          // Sun Feb 01 1998 00:00:00 GMT+0000 (
let date = new Date(22, 1)          // Wed Feb 01 1922 00:00:00 GMT+0000 (G
let date = new Date("2/1/22")       // Tue Feb 01 2022 00:00:00 GMT+0000 (GMT)

// Legacy method; always interprets two-digit year values as relative to 1900

date.setYear(98); date.toString()  // Sun Feb 01 1998 00:00:00 GMT+0000 (GMT)
date.setYear(22); date.toString()  // Wed Feb 01 1922 00:00:00 GMT+0000 (GMT)
```

So, to create and get dates between the years `0` and `99`, instead use the preferred `setFullYear()` and `getFullYear()` methods:.

```
// Preferred method; never interprets any value as being a relative offset
// but instead uses the year value as-is
date.setFullYear(98); date.getFullYear()  // 98 (not 1998)
date.setFullYear(22); date.getFullYear()  // 22 (not 1922, not 2022)
```

Calculating elapsed time

The following examples show how to determine the elapsed time between two JavaScript dates in milliseconds.

Due to the differing lengths of days (due to daylight saving changeover), months, and years, expressing elapsed time in units greater than hours, minutes, and seconds requires addressing a number of issues, and should be thoroughly researched before being attempted.

```
// Using Date objects
let start = Date.now()

// The event to time goes here:
doSomethingForALongTime()
let end = Date.now()
let elapsed = end - start // elapsed time in milliseconds
```

```
// Using built-in methods
let start = new Date()

// The event to time goes here:
doSomethingForALongTime()
let end = new Date()
let elapsed = end.getTime() - start.getTime() // elapsed time in milliseconds
```

```
// To test a function and get back its return
function printElapsedTime(fTest) {
  let nStartTime = Date.now(),
      vReturn = fTest(),
      nEndTime = Date.now()

  console.log(`Elapsed time: ${ String(nEndTime - nStartTime) } milliseconds`)
  return vReturn
}

let yourFunctionReturn = printElapsedTime(yourFunction)
```

Note: In browsers that support the Web Performance API's high-resolution time feature, `Performance.now()` can provide more reliable and precise measurements of elapsed time than `Date.now()`.

Get the number of seconds since the ECMAScript Epoch

```
let seconds = Math.floor(Date.now() / 1000)
```

In this case, it's important to return only an integer—so a simple division won't do. It's also important to only return actually elapsed seconds. (That's why this code uses `Math.floor()`, and *not* `Math.round()`.)

Specifications

Specification
ECMAScript Language Specification (ECMAScript) # sec-date-objects

Browser compatibility

Report problems with this compatibility data on GitHub

`Date`