Date

JavaScript `Date` objects represent a single moment in time in a platform-independent format. `Date` objects encapsulate an integral number that represents milliseconds since the midnight at the beginning of January 1, 1970, UTC (the *epoch*).

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 epoch, timestamps, and invalid date

A JavaScript date is fundamentally specified as the time in milliseconds that has elapsed since the epoch, which is defined as the midnight at the beginning of January 1, 1970, UTC (equivalent to the UNIX epoch). This timestamp is *timezone-agnostic* and uniquely defines an instant in history.

Note: 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.

The maximum timestamp representable by a `Date` object is slightly smaller than the maximum safe integer (`Number.MAX_SAFE_INTEGER`, which is 9,007,199,254,740,991). A `Date` object can represent a maximum of ±8,640,000,000,000,000 milliseconds, or ±100,000,000 (one hundred million) days, relative to the epoch. This the range from April 20, 271821 BC to September 13, 275760 AD. Any attempt to represent a time outside this range results in the `Date` object holding a timestamp value of `NaN`, which is an "Invalid Date".

```
console.log(new Date(8.64e15).toString()); // "Sat Sep 13 275760 00:00:00 GMT+0000 (Coordinated Universal Time)"
console.log(new Date(8.64e15 + 1).toString()); // "Invalid Date"
```

There are various methods that allow you to interact with the timestamp stored in the date:

- You can interact with the timestamp value directly using the `getTime()` and `setTime()` methods.
- The `valueOf()` and `[@@toPrimitive]()` (when passed `"number"`) methods — which are automatically called in number coercion — return the timestamp, causing `Date` objects to behave like their timestamps when used in number contexts.
- All static methods (`Date.now()`, `Date.parse()`, and `Date.UTC()`) return timestamps instead of `Date` objects.
- The `Date()` constructor can be called with a timestamp as the only argument.

Date components and time zones

A date is represented internally as a single number, the *timestamp*. When interacting with it, the timestamp needs to be interpreted as a structured date-and-time representation. There are always two ways to interpret a timestamp: as a local time or as a Coordinated Universal Time (UTC), the global standard time defined by the World Time Standard. The local timezone is not stored in the date object, but is determined by the host environment (user's device).

Note: UTC should not be confused with the Greenwich Mean Time (GMT), because they are not always equal — this is explained in more detail in the linked Wikipedia page.

For example, the timestamp 0 represents a unique instant in history, but it can be interpreted in two ways:

- As a UTC time, it is midnight at the beginning of January 1, 1970, UTC,
- As a local time in New York (UTC-5), it is 19:00:00 on December 31, 1969.

The `getTimezoneOffset()` method returns the difference between UTC and the local time in minutes. Note that the timezone offset does not only depend on the current timezone, but also on the time represented by the `Date` object, because of daylight saving time and historical changes. In essence, the timezone offset is the offset from UTC time, at the time represented by the `Date` object and at the location of the host environment.

There are two groups of `Date` methods: one group gets and sets various date components by interpreting the timestamp as a local time, while the other uses UTC.

Component	**Local**		**UTC**	
	Get	**Set**	**Get**	**Set**
Year	`getFullYear()`	`setFullYear()`	`getUTCFullYear()`	`setUTCFullYear()`
Month	`getMonth()`	`setMonth()`	`getUTCMonth()`	`setUTCMonth()`
Date (of month)	`getDate()`	`setDate()`	`getUTCDate()`	`setUTCDate()`
Hours	`getHours()`	`setHours()`	`getUTCHours()`	`setUTCHours()`
Minutes	`getMinutes()`	`setMinutes()`	`getUTCMinutes()`	`setUTCMinutes()`
Seconds	`getSeconds()`	`setSeconds()`	`getUTCSeconds()`	`setUTCSeconds()`
Milliseconds	`getMilliseconds()`	`setMilliseconds()`	`getUTCMilliseconds()`	`setUTCMilliseconds()`
Day (of week)	`getDay()`	N/A	`getUTCDay()`	N/A

The `Date()` constructor can be called with two or more arguments, in which case they are interpreted as the year, month, day, hour, minute, second, and millisecond, respectively, in local time. `Date.UTC()` works similarly, but it interprets the components as UTC time and also accepts a single argument representing the year.

Note: Some methods, including the `Date()` constructor, `Date.UTC()`, and the deprecated `getYear()` / `setYear()` methods, interpret a two-digit year as a year in the 1900s. For example, `new Date(99, 5, 24)` is interpreted as June 24, 1999, not June 24, 99. See Interpretation of two-digit years for more information.

When a segment overflows or underflows its expected range, it usually "carries over to" or "borrows from" the higher segment. For example, if the month is set to 12 (months are zero-based, so December is 11), it become the January of the next year. If the day of month is set to 0, it becomes the last day of the previous month. This also applies to dates specified with the date time string format.

Date time string format

There are many ways to format a date as a string. The JavaScript specification only specifies one format to be universally supported: the *date time string format*, a simplification of the ISO 8601 calendar date extended format. The format is as follows:

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

- `YYYY` is the year, with four digits (`0000` to `9999`), or as an *expanded year* of `+` or `-` followed by six digits. The sign is required for expanded years. `-000000` is explicitly disallowed as a valid year.
- `MM` is the month, with two digits (`01` to `12`). Defaults to `01`.
- `DD` is the day of the month, with two digits (`01` to `31`). Defaults to `01`.
- `T` is a literal character, which indicates the beginning of the *time* part of the string. The `T` is required when specifying the time part.
- `HH` is the hour, with two digits (`00` to `23`). As a special case, `24:00:00` is allowed, and is interpreted as midnight at the beginning of the next day. Defaults to `00`.
- `mm` is the minute, with two digits (`00` to `59`). Defaults to `00`.
- `ss` is the second, with two digits (`00` to `59`). Defaults to `00`.
- `sss` is the millisecond, with three digits (`000` to `999`). Defaults to `000`.
- `Z` is the timezone offset, which can either be the literal character `Z` (indicating UTC), or `+` or `-` followed by `HH:mm`, the offset in hours and minutes from UTC.

Various components can be omitted, so the following are all valid:

- Date-only form: `YYYY`, `YYYY-MM`, `YYYY-MM-DD`
- Date-time form: one of the above date-only forms, followed by `T`, followed by `HH:mm`, `HH:mm:ss`, or `HH:mm:ss.sss`. Each combination can be followed by a time zone offset.

For example, `"2011-10-10"` (*date-only* form), `"2011-10-10T14:48:00"` (*date-time* form), or `"2011-10-10T14:48:00.000+09:00"` (*date-time* form with milliseconds and time zone) are all valid date time strings.

When the time zone offset is absent, **date-only forms are interpreted as a UTC time and date-time forms are interpreted as local time.** This is due to a historical spec error that was not consistent with ISO 8601 but could not be changed due to web compatibility. See Broken Parser – A Web Reality Issue.

`Date.parse()` and the `Date()` constructor both accept strings in the date time string format as input. Furthermore, implementations are allowed to support other date formats when the input fails to match this format.

Note: You are encouraged to make sure your input conforms to the date time string format above for maximum compatibility, because support for other formats is not guaranteed. However, there are some formats that are supported in all major implementations — like RFC 2822 format — in which case their usage can be acceptable. Always conduct cross-browser tests to ensure your code works in all target browsers. A library can help if many different formats are to be accommodated.

The `toISOString()` method returns a string representation of the date in the date time string format, with the time zone offset always set to `Z` (UTC).

Other ways to format a date

- `toISOString()` returns a string in the format `1970-01-01T00:00:00.000Z` (the date time string format introduced above, which is simplified ISO 8601). `toJSON()` calls `toISOString()` and returns the result.
- `toString()` returns a string in the format `Thu Jan 01 1970 00:00:00 GMT+0000 (Coordinated Universal Time)`, while `toDateString()` and `toTimeString()` return the date and time parts of the string, respectively. `[@@toPrimitive]()` (when passed `"string"` or `"default"`) calls `toString()` and returns the result.
- `toUTCString()` returns a string in the format `Thu, 01 Jan 1970 00:00:00 GMT` (generalized RFC 7231).

- `toLocaleDateString()`, `toLocaleTimeString()`, and `toLocaleString()` use locale-specific date and time formats, usually provided by the `Intl` API.

See the Formats of `toString` method return values section for examples.

Constructor

`Date()`

When called as a constructor, returns a new `Date` object. When called as a function, returns a string representation of the current date and time.

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.

`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 properties

These properties are defined on `Date.prototype` and shared by all `Date` instances.

`Date.prototype.constructor`

The constructor function that created the instance object. For `Date` instances, the initial value is the `Date` constructor.

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.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.

`Date.prototype[@@toPrimitive]()`

Converts this `Date` object to a primitive value.

Examples

Several ways to create a Date object

The following examples show several ways to create JavaScript dates:

> **Note:** When parsing date strings with the `Date` constructor (and `Date.parse`, they are equivalent), always make sure that the input conforms to the ISO 8601 format (`YYYY-MM-DDTHH:mm:ss.sssZ`) — the parsing behavior with other formats is implementation-defined and may not work across all browsers. A library can help if many different formats are to be accommodated.

```
const today = new Date();
const birthday = new Date("December 17, 1995 03:24:00"); // DISCOURAGED: may not work in all runtimes
const birthday2 = new Date("1995-12-17T03:24:00"); // This is ISO8601-compliant and will work reliably
const birthday3 = new Date(1995, 11, 17); // the month is 0-indexed
const birthday4 = new Date(1995, 11, 17, 3, 24, 0);
const birthday5 = new Date(628021800000); // passing epoch timestamp
```

Formats of toString method return values

```
const date = new Date("2020-05-12T23:50:21.817Z");
date.toString(); // Tue May 12 2020 18:50:21 GMT-0500 (Central Daylight Time)
date.toDateString(); // Tue May 12 2020
date.toTimeString(); // 18:50:21 GMT-0500 (Central Daylight Time)
date[Symbol.toPrimitive]("string"); // Tue May 12 2020 18:50:21 GMT-0500 (Central Daylight Time)

date.toISOString(); // 2020-05-12T23:50:21.817Z
date.toJSON(); // 2020-05-12T23:50:21.817Z

date.toUTCString(); // Tue, 12 May 2020 23:50:21 GMT

date.toLocaleString(); // 5/12/2020, 6:50:21 PM
date.toLocaleDateString(); // 5/12/2020
date.toLocaleTimeString(); // 6:50:21 PM
```

To get Date, Month and Year or Time

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

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 (GMT)
date = new Date(22, 1); // Wed Feb 01 1922 00:00:00 GMT+0000 (GMT)
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
const start = Date.now();

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

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

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

```
// To test a function and get back its return
function printElapsedTime(testFn) {
  const startTime = Date.now();
  const result = testFn();
  const endTime = Date.now();

  console.log(`Elapsed time: ${String(endTime - startTime)} milliseconds`);
  return result;
}

const 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

```
const 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 # sec-date-objects

Browser compatibility

Report problems with this compatibility data on GitHub

	■							
	Chrome ■	Edge ■	Firefox ■	Opera ■	Safari ■	Chrome Android ■	Firefox for Android ■	
`Date`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 3	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ C A
`@@toPrimitive`	✓ Chrome 47	✓ Edge 15	✓ Firefox 44	✓ Opera 34	✓ Safari 10	✓ Chrome 47 Android	✓ Firefox 44 for Android	✓ (A
`Date()` constructor	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 3	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ C A
`UTC`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 3	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ C A
`getDate`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 3	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ C A
`getDay`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 3	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ C A
`getFullYear`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 4	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ C A
`getHours`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 3	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ C A
`getMilliseconds`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 4	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ C A
`getMinutes`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 3	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ C A
`getMonth`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 3	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ C A

	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	
`getSeconds`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 3	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ C A
`getTime`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 3	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ C A
`getTimezoneOffset`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 3	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ C A
`getUTCDate`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 4	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ C A
`getUTCDay`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 4	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ C A
`getUTCFullYear`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 4	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ C A
`getUTCHours`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 4	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ C A
`getUTCMilliseconds`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 4	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ C A
`getUTCMinutes`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 4	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ C A
`getUTCMonth`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 4	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ C A
`getUTCSeconds`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 4	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ C A
`getYear` 🗑	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 3	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ C A
`now`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 10.5	✓ Safari 4	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ A
`parse`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 3	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ C A

	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	
ISO 8601 format	✓ Chrome 6	✓ Edge 12	✓ Firefox 4	✓ Opera 6	✓ Safari 5.1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ C A
`setDate`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 3	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ C A
`setFullYear`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 4	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ C A
`setHours`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 3	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ C A
`setMilliseconds`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 4	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ C A
`setMinutes`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 3	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ C A
`setMonth`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 3	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ C A
`setSeconds`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 3	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ C A
`setTime`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 3	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ C A
`setUTCDate`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 4	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ C A
`setUTCFullYear`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 4	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ C A
`setUTCHours`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 4	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ C A
`setUTCMilliseconds`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 4	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ C A
`setUTCMinutes`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 4	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ C A

	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	
`setUTCMonth`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 4	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ C A
`setUTCSeconds`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 4	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ C A
`setYear` 🗑	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 3	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ C A
`toDateString`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 5	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ C A
`toGMTString` 🗑	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 3	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ C A
`toISOString`	✓ Chrome 3	✓ Edge 12	✓ Firefox 1	✓ Opera 10.5	✓ Safari 4	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ A
`toJSON`	✓ Chrome 3	✓ Edge 12	✓ Firefox 1	✓ Opera 10.5	✓ Safari 4	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ A
`toLocaleDateString`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 5	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ C A
IANA time zone names in `timeZone` option	✓ Chrome 24	✓ Edge 14	✓ Firefox 52	✓ Opera 15	✓ Safari 7	✓ Chrome 25 Android	✓ Firefox 56 for Android	✓ A
`toLocaleDateString.locales`	✓ Chrome 24	✓ Edge 12	✓ Firefox 29	✓ Opera 15	✓ Safari 10	✓ Chrome 25 Android	✓ Firefox 56 for Android	✓ A
`toLocaleDateString.options`	✓ Chrome 24	✓ Edge 12	✓ Firefox 29	✓ Opera 15	✓ Safari 10	✓ Chrome 25 Android	✓ Firefox 56 for Android	✓ A
`toLocaleString`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 3	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ C A
IANA time zone names in `timeZone` option	✓ Chrome 24	✓ Edge 14	✓ Firefox 52	✓ Opera 15	✓ Safari 7	✓ Chrome 25 Android	✓ Firefox 56 for Android	✓ A
`toLocaleString.locales`	✓ Chrome 24	✓ Edge 12	✓ Firefox 29	✓ Opera 15	✓ Safari 10	✓ Chrome 25 Android	✓ Firefox 56 for Android	✓ A

	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	
`toLocaleString.options`	✓ Chrome 24	✓ Edge 12	✓ Firefox 29	✓ Opera 15	✓ Safari 10	✓ Chrome 25 Android	✓ Firefox 56 for Android	✓
`toLocaleTimeString`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 5	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓
IANA time zone names in `timeZone` option	✓ Chrome 24	✓ Edge 14	✓ Firefox 52	✓ Opera 15	✓ Safari 7	✓ Chrome 25 Android	✓ Firefox 56 for Android	✓
`toLocaleTimeString.locales`	✓ Chrome 24	✓ Edge 12	✓ Firefox 29	✓ Opera 15	✓ Safari 10	✓ Chrome 25 Android	✓ Firefox 56 for Android	✓
`toLocaleTimeString.options`	✓ Chrome 24	✓ Edge 12	✓ Firefox 29	✓ Opera 15	✓ Safari 10	✓ Chrome 25 Android	✓ Firefox 56 for Android	✓
`toString`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 3	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓
`toTimeString`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 5	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓
`toUTCString`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 4	✓ Safari 1	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓

mdn web docs_

							Android	

Tip: you can click/tap on a cell for more information.

✓ Full support · Partial support · Deprecated. Not for use in new websites. ··· Has more compatibility info.

See also

- `Date()` constructor

This page was last modified on Apr 22, 2023 by MDN contributors.