Moment.js Documentation

Where to use it

Moment was designed to work both in the browser and in Node.JS.

Currently the following browsers are used for the ci system: IE8, IE9 on Windows 7, stable Chrome on Windows XP, Safari 10.8 on Mac and stable Firefox on Linux.

All code will work in both environments. All unit tests are run in both environments.

Node.js

edit

```
npm install moment
```

```
var moment = require('moment');
moment().format();
```

Note: In `2.4.0`, the globally exported moment object was **deprecated**. It will be removed in next major release.

Browser

edit

```
<script src="moment.js"></script>
<script>
    moment().format();
</script>
```

Moment.js is available on cdnjs.com. Remember though, cool kids concatenate their scripts to minimize http requests.

Bower

edit

bower

```
bower install --save moment
```

Notable files are `moment.js`, `locale/*.js` and `min/moment-with-locales.js`.

Require.js

edit

```
require.config({
    paths: {
        "moment": "path/to/moment",
    }
});
define(["moment"], function (moment) {
    moment().format();
});
```

Moment will still create a `moment` global, which is useful to plugins and other third-party code. If you wish to squash that global, use the `noGlobal` option on the module config.

```
require.config({
    config: {
        moment: {
            noGlobal: true
        }
    }
});
```

If you don't specify `noGlobal` then the globally exported moment will

print a deprecation warning. From next major release you'll have to export it yourself if you want that behavior.

For version `2.5.x`, in case you use other plugins that rely on Moment but are not AMD-compatible you may need to add `wrapShim: true` to your r.js config.

Note: To allow moment.js plugins to be loaded in requirejs environments, moment is created as a named module. Because of this, moment **must** be loaded exactly as as `"moment"`, using `paths` to determine the directory. Requiring moment with a path like `"vendor\moment"` will return `undefined`.

Note: From version **2.9.0** moment exports itself as an anonymous module, so if you're using only the core (no locales / plugins), then you don't need config if you put it on a non-standard location.

NuGet

edit

NuGet / Moment.js

```
Install-Package Moment.js
```

spm

edit

Static Package Manager / moment

```
spm install moment --save
```

meteor

edit

meteor / atmosphere / momentjs:moment

```
meteor add momentjs:moment
```

Other

edit

To use under **Java/Rhino**, see these instructions.

To use in **Demandware**, see these instructions.

Troubleshooting

edit

If you are having any troubles, try asking a question on Stack Overflow with the `momentjs` tag.

You can also use the GitHub issue tracker to find related issues or open a new issue.

Parse

Instead of modifying the native `Date.prototype`, Moment.js creates a wrapper for the `Date` object. To get this wrapper object, simply call `moment()` with one of the supported input types.

The `Moment` prototype is exposed through `moment.fn`. If you want to add your own functions, that is where you would put them.

For ease of reference, any method on the `Moment.prototype` will be referenced in the docs as `moment#method`. So `Moment.prototype.format` == `moment.fn.format` == `moment#format`.

Now 1.0.0+

edit

```
moment();                                    Method Signature
```

To get the current date and time, just call `moment()` with no parameters.

```
var now = moment();
```

This is essentially the same as calling `moment(new Date())`.

String 1.0.0+ edit

```
moment(String);                                    Method Signature
```

When creating a moment from a string, we first check if the string matches known ISO 8601 formats, then fall back to `new Date(string)` if a known format is not found.

```
var day = moment("1995-12-25");
```

Warning: Browser support for parsing strings is inconsistent. Because there is no specification on which formats should be supported, what works in some browsers will not work in other browsers.

For consistent results parsing anything other than ISO 8601 strings, you should use String + Format.

Supported ISO 8601 strings

An ISO 8601 string requires a date part.

```
2013-02-08  # A calendar date part
2013-W06-5  # A week date part
2013-039    # An ordinal date part
```

A time part can also be included, separated from the date part by a space or a uppercase T.

```
2013-02-08T09            # An hour time part separated by a
2013-02-08 09            # An hour time part separated by a
```

```
2013-02-08 09:30         # An hour and minute time part
2013-02-08 09:30:26      # An hour, minute, and second time
2013-02-08 09:30:26.123  # An hour, minute, second, and mil
2013-02-08 24:00:00.000  # hour 24, minute, second, millise
```

Any of the date parts can have a time part.

```
2013-02-08 09  # A calendar date part and hour time part
2013-W06-5 09  # A week date part and hour time part
2013-039 09    # An ordinal date part and hour time part
```

If a time part is included, an offset from UTC can also be included as `+-HH:mm`, `+-HHmm`, or `Z`.

```
2013-02-08 09+07:00            # +-HH:mm
2013-02-08 09-0100             # +-HHmm
2013-02-08 09Z                 # Z
2013-02-08 09:30:26.123+07:00  # +-HH:mm
```

Note: Automatic cross browser ISO-8601 support was added in version **1.5.0**. Support for the week and ordinal formats was added in version **2.3.0**.

If a string does not match any of the above formats and is not able to be parsed with `Date.parse`, `moment#isValid` will return false.

```
moment("not a real date").isValid(); // false
```

String + Format 1.0.0+ edit

```
moment(String, String);                    Method Signature
moment(String, String, String);
moment(String, String, Boolean);
moment(String, String, String, Boolean);
```

If you know the format of an input string, you can use that to parse a moment.

```
moment("12-25-1995", "MM-DD-YYYY");
```

The parser ignores non-alphanumeric characters, so both of the following will return the same thing.

```
moment("12-25-1995", "MM-DD-YYYY");
moment("12\25\1995", "MM-DD-YYYY");
```

The parsing tokens are similar to the formatting tokens used in `moment#format`.

Year, month, and day tokens

Input	Example	Description
`YYYY`	`2014`	4 digit year
`YY`	`14`	2 digit year
`Q`	`1..4`	Quarter of year. Sets month to first month in quarter.
`M MM`	`1..12`	Month number
`MMM MMMM`	`January..Dec`	Month name in locale set by `moment.locale()`
`D DD`	`1..31`	Day of month
`Do`	`1st..31st`	Day of month with ordinal
`DDD DDDD`	`1..365`	Day of year
`X`	`1410715640.579`	Unix timestamp
`x`	`1410715640579`	Unix ms timestamp

Week year, week, and weekday tokens

For these, the lowercase tokens use the locale aware week start days, and the uppercase tokens use the ISO week date start days.

Input	Example	Description
`gggg`	`2014`	Locale 4 digit week year
`gg`	`14`	Locale 2 digit week year

`w ww`	`1..53`	Locale week of year
`e`	`1..7`	Locale day of week
`GGGG`	`2014`	ISO 4 digit week year
`GG`	`14`	ISO 2 digit week year
`W WW`	`1..53`	ISO week of year
`E`	`1..7`	ISO day of week

Hour, minute, second, millisecond, and offset tokens

Input	Example	Description
`H HH`	`0..23`	24 hour time
`h hh`	`1..12`	12 hour time used with `a A`.
`a A`	`am pm`	Post or ante meridiem
`m mm`	`0..59`	Minutes
`s ss`	`0..59`	Seconds
`S`	`0..9`	Tenths of a second
`SS`	`0..99`	Hundreds of a second
`SSS`	`0..999`	Thousandths of a second
`Z ZZ`	`+12:00`	Offset from UTC as `+-HH:mm`, `+-HHmm`, or `Z`

Locale aware date and time formats are also available using `LT LTS L LL LLL LLLL`. They were added in version **2.2.1**, except `LTS` which was added **2.8.4**.

`Z ZZ` were added in version `1.2.0`.

`S SS SSS` were added in version `1.6.0`.

`X` was added in version `2.0.0`.

Unless you specify a timezone offset, parsing a string will create a date in the current timezone.

```
moment("2010-10-20 4:30",       "YYYY-MM-DD HH:mm");    // p
moment("2010-10-20 4:30 +0000", "YYYY-MM-DD HH:mm Z"); // p
```

If the moment that results from the parsed input does not exist, `moment#isValid` will return false.

```
moment("2010 13",           "YYYY MM").isValid();     // fa
moment("2010 11 31",        "YYYY MM DD").isValid();  // fa
moment("2010 2 29",         "YYYY MM DD").isValid();  // fa
moment("2010 notamonth 29", "YYYY MMM DD").isValid(); // fa
```

As of version `2.0.0`, a locale key can be passed as the third parameter to `moment()` and `moment.utc()`.

```
moment('2012 juillet', 'YYYY MMM', 'fr');
moment('2012 July',    'YYYY MMM', 'en');
```

Moment's parser is very forgiving, and this can lead to undesired behavior. As of version `2.3.0`, you may specify a boolean for the last argument to make Moment use strict parsing. Strict parsing requires that the format and input match exactly.

```
moment('It is 2012-05-25', 'YYYY-MM-DD').isValid();       /
moment('It is 2012-05-25', 'YYYY-MM-DD', true).isValid(); /
moment('2012-05-25',       'YYYY-MM-DD', true).isValid(); /
```

You can use both locale and strictness.

```
moment('2012-10-14', 'YYYY-MM-DD', 'fr', true);
```

Parsing two digit years

By default, two digit years above 68 are assumed to be in the 1900's and years below 68 are assumed to be in the 2000's. This can be changed by replacing the `moment.parseTwoDigitYear` method.

String + Formats 1.0.0+ edit

```
moment(String, String[], String, Boolean);  Method Signature
```

If you don't know the exact format of an input string, but know it could be one of many, you can use an array of formats.

This is the same as String + Format, only it will try to match the input to multiple formats.

```
moment("12-25-1995", ["MM-DD-YYYY", "YYYY-MM-DD"]);
```

Starting in version `2.3.0`, Moment uses some simple heuristics to determine which format to use. In order:

- Prefer formats resulting in valid dates over invalid ones.
- Prefer formats that parse more of the string than less and use more of the format than less, i.e. prefer stricter parsing.
- Prefer formats earlier in the array than later.

```
moment("29-06-1995", ["MM-DD-YYYY", "DD-MM", "DD-MM-YYYY"])
moment("05-06-1995", ["MM-DD-YYYY", "DD-MM-YYYY"]);
```

You may also specify a locale and strictness argument. They work the same was as they do in the single format case.

```
moment("29-06-1995", ["MM-DD-YYYY", "DD-MM-YYYY"], 'fr');
moment("29-06-1995", ["MM-DD-YYYY", "DD-MM-YYYY"], true);
moment("05-06-1995", ["MM-DD-YYYY", "DD-MM-YYYY"], 'fr', tr
```

Note: Parsing multiple formats is considerably slower than parsing a single format. If you can avoid it, it is much faster to parse a single format.

Special Formats 2.7.0+ edit

```
moment(String, moment.CUSTOM_FORMAT, [String], [Boolean]);
moment(String, [..., moment.ISO_8601, ...], [String], [Bool
```

ISO-8601 is a standard for time and duration display. Moment already supports parsing iso-8601 strings, but this can be specified explicitly

in the format/list of formats when constructing a moment.

To specify iso-8601 parsing use `moment.ISO_8601` constant. More formats will be added in the future.

Examples:

```
moment("2010-01-01T05:06:07", moment.ISO_8601);
moment("2010-01-01T05:06:07", ["YYYY", moment.ISO_8601]);
```

Object 2.2.1+ edit

```
moment({unit: value, ...});                    Method Signature
```

```
moment({ hour:15, minute:10 });
moment({ y    :2010, M     :3, d   :5, h    :15, m      :10
moment({ year :2010, month :3, day :5, hour :15, minute :10
moment({ years:2010, months:3, days:5, hours:15, minutes:10
moment({ years:2010, months:3, date:5, hours:15, minutes:10
```

You can create a moment by specifying some of the units in an object.

Omitted units default to 0 or the current date, month, and year.

`day` and `date` key both mean day-of-the-month.

`date` was added in **2.8.4**.

Unix Offset (milliseconds) 1.0.0+edit

```
moment(Number);                                Method Signature
```

Similar to `new Date(Number)`, you can create a moment by passing an integer value representing the number of *milliseconds* since the Unix Epoch (Jan 1 1970 12AM UTC).

```
var day = moment(1318781876406);
```

Unix Timestamp (seconds) 1.6.0+ edit

```
moment.unix(Number)                    Method Signature
```

To create a moment from a Unix timestamp (*seconds* since the Unix Epoch), use `moment.unix(Number)`.

```
var day = moment.unix(1318781876);
```

This is implemented as `moment(timestamp * 1000)`, so partial seconds in the input timestamp are included.

```
var day = moment.unix(1318781876.721);
```

Date 1.0.0+ edit

```
moment(Date);                          Method Signature
```

You can create a `Moment` with a pre-existing native Javascript `Date` object.

```
var day = new Date(2011, 9, 16);
var dayWrapper = moment(day);
```

This clones `Date` object; further changes to the `Date` won't affect the `Moment`, and vice-versa.

Array 1.0.0+ edit

```
moment(Number[]);                      Method Signature
```

You can create a moment with an array of numbers that mirror the parameters passed to new Date()

```
[year, month, day, hour, minute, second, millisecond]
```

```
moment([2010, 1, 14, 15, 25, 50, 125]); // February 14th, 3
```

Any value past the year is optional, and will default to the lowest possible number.

```
moment([2010]);         // January 1st
moment([2010, 6]);      // July 1st
moment([2010, 6, 10]); // July 10th
```

Construction with an array will create a date in the current timezone. To create a date from an array at UTC, use `moment.utc(Number[])`.

```
moment.utc([2010, 1, 14, 15, 25, 50, 125]);
```

Note: Because this mirrors the native `Date` parameters, months, hours, minutes, seconds, and milliseconds are all zero indexed. Years and days of the month are 1 indexed.

This is often the cause of frustration, especially with months, so take note!

If the date represented by the array does not exist, `moment#isValid` will return false.

```
moment([2010, 13]).isValid();     // false (not a real mont
moment([2010, 10, 31]).isValid(); // false (not a real day)
moment([2010, 1, 29]).isValid();  // false (not a leap year
```

ASP.NET JSON Date 1.3.0+ edit

```
moment(String);                          Method Signature
```

ASP.NET returns dates in JSON as `/Date(1198908717056)/` or `/Date(1198908717056-0700)/`

If a string that matches this format is passed in, it will be parsed correctly.

```
moment("/Date(1198908717056-0700)/"); // 2007-12-28T23:11:5
```

Moment Clone 1.2.0+ edit

```
moment(Moment);                                    Method Signature
```

All moments are mutable. If you want a clone of a moment, you can do so explicitly or implicitly.

Calling `moment()` on a moment will clone it.

```
var a = moment([2012]);
var b = moment(a);
a.year(2000);
b.year(); // 2012
```

Additionally, you can call `moment#clone` to clone a moment.

```
var a = moment([2012]);
var b = a.clone();
a.year(2000);
b.year(); // 2012
```

UTC 1.5.0+ edit

```
moment.utc();                                      Method Signature
moment.utc(Number);
moment.utc(Number[]);
moment.utc(String);
moment.utc(String, String);
moment.utc(String, String[]);
moment.utc(String, String, String);
moment.utc(Moment);
moment.utc(Date);
```

By default, moment parses and displays in local time.

If you want to parse or display a moment in UTC, you can use `moment.utc()` instead of `moment()`.

This brings us to an interesting feature of Moment.js. UTC mode.

While in UTC mode, all display methods will display in UTC time instead of local time.

```
moment().format();     // 2013-02-04T10:35:24-08:00
moment.utc().format(); // 2013-02-04T18:35:24+00:00
```

Additionally, while in UTC mode, all getters and setters will internally use the `Date#getUTC*` and `Date#setUTC*` methods instead of the `Date#get*` and `Date#set*` methods.

```
moment.utc().seconds(30) === new Date().setUTCSeconds(30);
moment.utc().seconds()   === new Date().getUTCSeconds();
```

It is important to note that though the displays differ above, they are both the same moment in time.

```
var a = moment();
var b = moment.utc();
a.format();  // 2013-02-04T10:35:24-08:00
b.format();  // 2013-02-04T18:35:24+00:00
a.valueOf(); // 1360002924000
b.valueOf(); // 1360002924000
```

Any moment created with `moment.utc()` will be in UTC mode, and any moment created with `moment()` will not.

To switch from UTC to local time, you can use moment#utc or moment#local.

```
var a = moment.utc([2011, 0, 1, 8]);
a.hours(); // 8 UTC
a.local();
a.hours(); // 0 PST
```

parseZone 2.3.0+ edit

```
moment.parseZone(String)                    Method Signature
```

Moment normally interprets input times as local times (or UTC times if `moment.utc()` is used). However, often the input string itself contains time zone information. `#parseZone` parses the time and then sets the zone according to the input string.

```
moment.parseZone("2013-01-01T00:00:00-13:00").zone(); // 78
```

`moment.parseZone` is equivalent to parsing the string and using `moment#zone` to parse the zone.

```
var s = "2013-01-01T00:00:00-13:00";
moment(s).zone(s);
```

Note: this method only works for a single string argument, not a string and format.

Validation 1.7.0+ edit

```
moment().isValid();                         Method Signature
```

Moment applies stricter initialization rules than the `Date` constructor.

```
new Date(2013, 25, 14).toString(); // "Sat Feb 14 2015 00:0
moment([2015, 25, 35]).format();   // 'Invalid date'
```

You can check whether the Moment considers the date invalid using `moment#isValid`. You can check the metrics used by `#isValid` using `moment#parsingFlags` which returns an object

The following parsing flags result in an invalid date:

- `overflow`: An overflow of a date field, such as a 13th month, a 32nd day of the month (or a 29th of February on non-leap years),

a 367th day of the year, etc. `overflow` contains the index of the invalid unit to match `#invalidAt` (see below); `-1` means no overflow.

- `invalidMonth`: An invalid month name, such as `moment('Marbruary', 'MMMM');`. Contains the invalid month string itself, or else null.
- `empty`: An input string that contains nothing parsable, such as `moment('this is nonsense');`. Boolean.
- `nullInput`: A `null` input, like `moment(null);`. Boolean.
- `invalidFormat`: An empty list of formats, such as `moment('2013-05-25', [])`. Boolean.
- `userInvalidated`: A date created explicitly as invalid, such as `moment.invalid()`. Boolean.

Additionally, if the Moment is parsed in strict mode, these flags must be empty for the Moment to be valid:

- `unusedTokens`: array of format substrings not found in the input string
- `unusedInput`: array of input substrings not matched to the format string

Note: Moment's concept of validity became more strict and consistent between 2.2 and 2.3.

Additionally, you can use `moment#invalidAt` to determine which date unit overflowed.

```
var m = moment("2011-10-10T10:20:90");
m.isValid(); // false
m.invalidAt(); // 5 for seconds
```

The return value has the following meaning:

0 years
1 months
2 days
3 hours
4 minutes
5 seconds
6 milliseconds

Note: In case of multiple wrong units the first one is returned (because days validity may depend on month, for example).

Defaults 2.2.1+ edit

```
moment("15", "hh")                                    Method Signature
```

You can create a moment object specifying only some of the units, and the rest will be defaulted to the current day, month or year, or 0 for hours, minutes, seconds and milliseconds.

Defaulting to now, when nothing is passed:

```
moment();  // current date and time
```

Defaulting to today, when only hours, minutes, seconds and milliseconds are passed:

```
moment(5, "HH");  // today, 5:00:00.000
moment({hour: 5});  // today, 5:00:00.000
moment({hour: 5, minute: 10});  // today, 5:10.00.000
moment({hour: 5, minute: 10, seconds: 20});  // today, 5:10
moment({hour: 5, minute: 10, seconds: 20, milliseconds: 300
```

Defaulting to this month and year, when only days and smaller units are passed:

```
moment(5, "DD");  // this month, 5th day-of-month
moment("4 05:06:07", "DD hh:mm:ss");  // this month, 4th da
```

Defaulting to this year, if year is not specified:

```
moment(3, "MM");  // this year, 3th month (April)
moment("Apr 4 05:06:07", "MMM DD hh:mm:ss");  // this year,
```

Get + Set

Moment.js uses overloaded getters and setters. You may be familiar with this pattern from it's use in jQuery.

Calling these methods without parameters acts as a getter, and calling them with a parameter acts as a setter.

These map to the corresponding function on the native `Date` object.

```
moment().seconds(30) === new Date().setSeconds(30);
moment().seconds()   === new Date().getSeconds();
```

If you are in UTC mode, they will map to the UTC equivalent.

```
moment.utc().seconds(30) === new Date().setUTCSeconds(30);
moment.utc().seconds()   === new Date().getUTCSeconds();
```

For convenience, both singular and plural method names exist as of version `2.0.0`.

Note: All of these methods mutate the original moment when used as setters.

Millisecond 1.3.0+ edit

```
moment().millisecond(Number);                Method Signature
moment().millisecond(); // Number
moment().milliseconds(Number);
moment().milliseconds(); // Number
```

Gets or sets the milliseconds.

Accepts numbers from 0 to 999. If the range is exceeded, it will bubble up to the seconds.

Second 1.0.0+ edit

```
moment().second(Number);                    Method Signature
moment().second(); // Number
moment().seconds(Number);
moment().seconds(); // Number
```

Gets or sets the seconds.

Accepts numbers from 0 to 59. If the range is exceeded, it will bubble up to the minutes.

Minute 1.0.0+ edit

```
moment().minute(Number);                    Method Signature
moment().minute(); // Number
moment().minutes(Number);
moment().minutes(); // Number
```

Gets or sets the minutes.

Accepts numbers from 0 to 59. If the range is exceeded, it will bubble up to the hours.

Hour 1.0.0+ edit

```
moment().hour(Number);                      Method Signature
moment().hour(); // Number
moment().hours(Number);
moment().hours(); // Number
```

Gets or sets the hour.

Accepts numbers from 0 to 23. If the range is exceeded, it will bubble up to the day.

Date of Month 1.0.0+ edit

```
moment().date(Number);                    Method Signature
moment().date(); // Number
moment().dates(Number);
moment().dates(); // Number
```

Gets or sets the day of the month.

Accepts numbers from 1 to 31. If the range is exceeded, it will bubble up to the months.

Note: `Moment#date` is for the date of the month, and `Moment#day` is for the day of the week.

Day of Week 1.3.0+ edit

```
moment().day(Number|String);              Method Signature
moment().day(); // Number
moment().days(Number|String);
moment().days(); // Number
```

Gets or sets the day of the week.

This method can be used to set the day of the week, with Sunday as 0 and Saturday as 6.

If the range is exceeded, it will bubble up to other weeks.

```
moment().day(-7); // last Sunday (0 - 7)
moment().day(7); // next Sunday (0 + 7)
moment().day(10); // next Wednesday (3 + 7)
moment().day(24); // 3 Wednesdays from now (3 + 7 + 7 + 7)
```

Note: `Moment#date` is for the date of the month, and `Moment#day` is for the day of the week.

As of **2.1.0**, a day name is also supported. This is parsed in the moment's current locale.

```
moment().day("Sunday");
moment().day("Monday");
```

Day of Week (Locale Aware) edit

2.1.0+

```
moment().weekday(Number);                    Method Signature
moment().weekday(); // Number
```

Gets or sets the day of the week according to the locale.

If the locale assigns Monday as the first day of the week, `moment().weekday(0)` will be Monday. If Sunday is the first day of the week, `moment().weekday(0)` will be Sunday.

As with `moment#day`, if the range is exceeded, it will bubble up to other weeks.

```
// when Monday is the first day of the week
moment().weekday(-7); // last Monday
moment().weekday(7); // next Monday
// when Sunday is the first day of the week
moment().weekday(-7); // last Sunday
moment().weekday(7); // next Sunday
```

ISO Day of Week 2.1.0+ edit

```
moment().isoWeekday(Number);                 Method Signature
moment().isoWeekday(); // Number
```

Gets or sets the ISO day of the week with `1` being Monday and `7` being Sunday.

```
moment().isoWeekday(1); // Monday
moment().isoWeekday(7); // Sunday
```

Day of Year 2.0.0+

edit

```
moment().dayOfYear(Number);                    Method Signature
moment().dayOfYear(); // Number
```

Gets or sets the day of the year.

Accepts numbers from 1 to 366. If the range is exceeded, it will bubble up to the years.

Week of Year 2.0.0+

edit

```
moment().week(Number);                         Method Signature
moment().week(); // Number
moment().weeks(Number);
moment().weeks(); // Number
```

Gets or sets the week of the year.

Because different locales define week of year numbering differently, Moment.js added `moment#week` to get/set the localized week of the year.

The week of the year varies depending on which day is the first day of the week (Sunday, Monday, etc), and which week is the first week of the year.

For example, in the United States, Sunday is the first day of the week. The week with January 1st in it is the first week of the year.

In France, Monday is the first day of the week, and the week with January 4th is the first week of the year.

The output of `moment#week` will depend on the locale for that moment.

When setting the week of the year, the day of the week is retained.

Week of Year (ISO) 2.0.0+

edit

```
moment().isoWeek(Number);                    Method Signature
moment().isoWeek(); // Number
moment().isoWeeks(Number);
moment().isoWeeks(); // Number
```

Gets or sets the ISO week of the year.

When setting the week of the year, the day of the week is retained.

Month 1.0.0+ edit

```
moment().month(Number|String);               Method Signature
moment().month(); // Number
moment().months(Number|String);
moment().months(); // Number
```

Gets or sets the month.

Accepts numbers from 0 to 11. If the range is exceeded, it will bubble up to the year.

Note: Months are zero indexed, so January is month 0.

As of **2.1.0**, a month name is also supported. This is parsed in the moment's current locale.

```
moment().month("January");
moment().month("Feb");
```

Before version **2.1.0**, if a moment changed months and the new month did not have enough days to keep the current day of month, it would overflow to the next month.

As of version **2.1.0**, this was changed to be clamped to the end of the target month.

```
// before 2.1.0
moment([2012, 0, 31]).month(1).format("YYYY-MM-DD"); // 201
// after 2.1.0
moment([2012, 0, 31]).month(1).format("YYYY-MM-DD"); // 201
```

Quarter 2.6.0+

edit

```
moment().quarter(); // Number                    Method Signature
moment().quarter(Number);
```

Gets the quarter (1 to 4).

```
moment('2013-01-01T00:00:00.000').quarter() // 1
moment('2013-04-01T00:00:00.000').subtract(1, 'ms').quarter
moment('2013-04-01T00:00:00.000').quarter() // 2
moment('2013-07-01T00:00:00.000').subtract(1, 'ms').quarter
moment('2013-07-01T00:00:00.000').quarter() // 3
moment('2013-10-01T00:00:00.000').subtract(1, 'ms').quarter
moment('2013-10-01T00:00:00.000').quarter() // 4
moment('2014-01-01T00:00:00.000').subtract(1, 'ms').quarter
```

Sets the quarter (1 to 4).

```
moment('2013-01-01T00:00:00.000').quarter(2) // '2013-01-01
moment('2013-02-05T05:06:07.000').quarter(2).format() // '2
```

Year 1.0.0+

edit

```
moment().year(Number);                          Method Signature
moment().year(); // Number
moment().years(Number);
moment().years(); // Number
```

Gets or sets the year.

Accepts numbers from -270,000 to 270,000.

Week Year 2.1.0+

edit

```
moment().weekYear(Number);          Method Signature
moment().weekYear(); // Number
```

Gets or sets the week-year according to the locale.

Because the first day of the first week does not always fall on the first day of the year, sometimes the week-year will differ from the month year.

For example, in the US, the week that contains Jan 1 is always the first week. In the US, weeks also start on Sunday. If Jan 1 was a Monday, Dec 31 would belong to the same week as Jan 1, and thus the same week-year as Jan 1. Dec 30 would have a different week-year than Dec 31.

Week Year (ISO) 2.1.0+ edit

```
moment().isoWeekYear(Number);          Method Signature
moment().isoWeekYear(); // Number
```

Gets or sets the ISO week-year.

Weeks In Year 2.6.0+ edit

```
moment().weeksInYear();          Method Signature
```

Gets the number of weeks according to locale in the current moment's year.

Weeks In Year (ISO) 2.6.0+ edit

```
moment().isoWeeksInYear();          Method Signature
```

Gets the number of weeks in the current moment's year, according to ISO weeks.

Get 2.2.1+ edit

```
moment().get('year');                    Method Signature
moment().get('month');  // 0 to 11
moment().get('date');
moment().get('hour');
moment().get('minute');
moment().get('second');
moment().get('millisecond');
```

String getter. In general

```
moment().get(unit) === moment()[unit]()
```

Units are case insensitive, and support plural and short forms: year (years, y), month (months, M), date (dates, D), hour (hours, h), minute (minutes, m), second (seconds, s), millisecond (milliseconds, ms).

Set 2.2.1+ edit

```
moment().set(String, Int);               Method Signature
moment().set(Object(String, Int));
```

Generic setter, accepting unit as first argument, and value as second:

```
moment().set('year', 2013);
moment().set('month', 3);  // April
moment().set('date', 1);
moment().set('hour', 13);
moment().set('minute', 20);
moment().set('second', 30);
moment().set('millisecond', 123);
```

```
moment().set({'year': 2013, 'month': 3});
```

Units are case insensitive, and support plural and short forms: year (years, y), month (months, M), date (dates, D), hour (hours, h), minute (minutes, m), second (seconds, s), millisecond (milliseconds, ms).

Maximum 2.7.0+

edit

```
moment.max(Moment[,Moment...]);                Method Signature
```

Returns the maximum (most distant future) of the given moment instances.

For example:

```
var a = moment().subtract(1, 'day');
var b = moment().add(1, 'day');
moment.max(a, b);  // b
```

With no arguments the function returns a moment instance with the current time.

Minimum 2.7.0+

edit

```
moment.min(Moment[,Moment...]);                Method Signature
```

Returns the minimum (most distant past) of the given moment instances.

For example:

```
var a = moment().subtract(1, 'day');
var b = moment().add(1, 'day');
moment.min(a, b);  // a
```

With no arguments the function returns a moment instance with the current time.

Manipulate

Once you have a `Moment`, you may want to manipulate it in some way. There are a number of methods to help with this.

Moment.js uses the fluent interface pattern, also known as method chaining. This allows you to do crazy things like the following.

```
moment().add(7, 'days').subtract(1, 'months').year(2009).ho
```

Note: It should be noted that moments are mutable. Calling any of the manipulation methods will change the original moment.

If you want to create a copy and manipulate it, you should use `moment#clone` before manipulating the moment. More info on cloning.

Add 1.0.0+ edit

```
moment().add(Number, String);          Method Signature
moment().add(Duration);
moment().add(Object);
```

Mutates the original moment by adding time.

This is a pretty robust function for adding time to an existing moment. To add time, pass the key of what time you want to add, and the amount you want to add.

```
moment().add(7, 'days');
```

There are some shorthand keys as well if you're into that whole brevity thing.

```
moment().add(7, 'd');
```

Key	Shorthand
years	y
quarters	Q
months	M
weeks	w
days	d
hours	h
minutes	m
seconds	s
milliseconds	ms

If you want to add multiple different keys at the same time, you can pass them in as an object literal.

```
moment().add(7, 'days').add(1, 'months'); // with chaining
moment().add({days:7,months:1}); // with object literal
```

There are no upper limits for the amounts, so you can overload any of the parameters.

```
moment().add(1000000, 'milliseconds'); // a million millise
moment().add(360, 'days'); // 360 days
```

Special considerations for months and years

If the day of the month on the original date is greater than the number of days in the final month, the day of the month will change to the last day in the final month.

```
moment([2010, 0, 31]);                  // January 31
moment([2010, 0, 31]).add(1, 'months'); // February 28
```

There are also special considerations to keep in mind when adding time that crosses over Daylight Savings Time. If you are adding years, months, weeks, or days, the original hour will always match the added

hour.

```
var m = moment(new Date(2011, 2, 12, 5, 0, 0)); // the day
m.hours(); // 5
m.add(1, 'days').hours(); // 5
```

If you are adding hours, minutes, seconds, or milliseconds, the assumption is that you want precision to the hour, and will result in a different hour.

```
var m = moment(new Date(2011, 2, 12, 5, 0, 0)); // the day
m.hours(); // 5
m.add(24, 'hours').hours(); // 6
```

Alternatively, you can use durations to add to moments.

```
var duration = moment.duration({'days' : 1});
moment([2012, 0, 31]).add(duration); // February 1
```

Before version `2.8.0`, the `moment#add(String, Number)` syntax was also supported. It has been deprecated in favor of `moment#add(Number, String)`.

```
moment().add('seconds', 1); // Deprecated in 2.8.0
moment().add(1, 'seconds');
```

Subtract 1.0.0+ edit

```
moment().subtract(Number, String);          Method Signature
moment().subtract(Duration);
moment().subtract(Object);
```

Mutates the original moment by subtracting time.

This is exactly the same as `moment#add`, only instead of adding time, it subtracts time.

```
moment().subtract(7, 'days');
```

Before version `2.8.0`, the `moment#subtract(String, Number)` syntax was also supported. It has been deprecated in favor of `moment#subtract(Number, String)`.

```
moment().subtract('seconds', 1); // Deprecated in 2.8.0
moment().subtract(1, 'seconds');
```

Start of Time 1.7.0+ edit

```
moment().startOf(String);                    Method Signature
```

Mutates the original moment by setting it to the start of a unit of time.

```
moment().startOf('year');    // set to January 1st, 12:00 a
moment().startOf('month');   // set to the first of this mo
moment().startOf('quarter');  // set to the beginning of th
moment().startOf('week');    // set to the first day of thi
moment().startOf('isoWeek'); // set to the first day of thi
moment().startOf('day');     // set to 12:00 am today
moment().startOf('hour');    // set to now, but with 0 mins
moment().startOf('minute');  // set to now, but with 0 seco
moment().startOf('second');  // same as moment().millisecon
```

These shortcuts are essentially the same as the following.

```
moment().startOf('year');
moment().month(0).date(1).hours(0).minutes(0).seconds(0).mi
```

```
moment().startOf('hour');
moment().minutes(0).seconds(0).milliseconds(0)
```

As of version **2.0.0**, `moment#startOf('day')` replaced `moment#sod`.

Note: `moment#startOf('week')` was added in version **2.0.0**.

As of version **2.1.0**, `moment#startOf('week')` uses the locale aware week start day.

Note: `moment#startOf('isoWeek')` was added in version **2.2.0**.

End of Time 1.7.0+ edit

```
moment().endOf(String);                    Method Signature
```

Mutates the original moment by setting it to the end of a unit of time.

This is the same as `moment#startOf`, only instead of setting to the start of a unit of time, it sets to the end of a unit of time.

```
moment().endOf("year"); // set the moment to 12-31 11:59:59
```

As of version **2.0.0**, `moment#endOf('day')` replaced `moment#eod`.

Note: `moment#endOf('week')` was added in version **2.0.0**.

As of version **2.1.0**, `moment#endOf('week')` uses the locale aware week start day.

Maximum From 2.1.0, Deprecated 2.7.0 edit

```
moment().max(Moment|String|Number|Date|Array);    Method Signature
```

NOTE: This function has been **deprecated** in **2.7.0**. Consider `moment.min` instead.

Limits the moment to a maximum of another moment value. So `a.max(b)` is the same as `a = moment.min(a, b)` (note that `max` is converted to `min`).

Sometimes, server clocks are not quite in sync with client clocks. This ends up displaying humanized strings such as "in a few seconds" rather than "a few seconds ago". You can prevent that with `moment#max()`:

This is the counterpart for `moment#min`.

```
var momentFromServer = moment(input);
var clampedMoment = momentFromServer.max();
```

You can pass anything to `moment#max` that you would pass to `moment()`.

```
moment().max(moment().add(1, 'd'));
moment().max("2013-04-20T20:00:00+0800");
moment().max("Jan 1 2001", "MMM D YYYY");
moment().max(new Date(2012, 1, 8));
```

Minimum From 2.1.0, Deprecated 2.7.0

edit

```
moment().min(Moment|String|Number|Date|Array);
```

Method Signature

NOTE: This function has been **deprecated** in **2.7.0**. Consider `moment.max` instead.

Limits the moment to a minimum of another moment value. So `a.min(b)` is the same as `a = moment.max(a, b)` (note that `min` is converted to `max`).

This is the counterpart for `moment#max`.

```
moment().min("2013-04-20T20:00:00+0800");
```

This can be used in conjunction with `moment#max` to clamp a moment to a range.

```
var start  = moment().startOf('week');
var end    = moment().endOf('week');
var actual = moment().min(start).max(end);
```

Local 1.5.0+

edit

```
moment().local();                                    Method Signature
```

Sets a flag on the original moment to internally use `Date#get*` and `Date#set*` instead of `Date#getUTC*` and `Date#setUTC*`.

```
var a = moment.utc([2011, 0, 1, 8]);
a.hours(); // 8 UTC
a.local();
a.hours(); // 0 PST
```

See moment.utc() for more information on UTC mode.

UTC 1.5.0+ edit

```
moment().utc();                                      Method Signature
```

Sets a flag on the original moment to internally use `Date#getUTC*` and `Date#setUTC*` instead of `Date#get*` and `Date#set*`.

```
var a = moment([2011, 0, 1, 8]);
a.hours(); // 8 PST
a.utc();
a.hours(); // 16 UTC
```

See moment.utc() for more information on UTC mode.

UTC Offset 2.9.0++ edit

```
moment().utcOffset();                                Method Signature
moment().utcOffset(Number|String);
```

Get the utc offset in minutes.

NOTE: Unlike `moment.fn.zone` this function returns the real offset from UTC, not the reverse offset (as returned by `Date.prototype.getTimezoneOffset`).

Getting the utcOffset of the current object:

```
moment().utcOffset(); // (-240, -120, -60, 0, 60, 120, 240,
```

Setting the utc offset by supplying minutes. Note that once you set an offset, its fixed and won't change on its own (i.e there are no DST rules). If you want an actual timezone -- time in a particular location, like `America/Los_Angeles`, consider moment-timezone.

```
moment().utcOffset(120);
```

If the input is less than `16` and greater than `-16`, it will interpret your input as hours instead.

```
// these are equivalent
moment().utcOffset(8);  // set hours offset
moment().utcOffset(480);  // set minutes offset (8 * 60)
```

It is also possible to set the utc offset from a string.

```
// these are equivalent
moment().utcOffset("08:00");
moment().utcOffset(8);
moment().utcOffset(480);
```

`moment#utcOffset` will search the string for the first match of `+00:00 +0000 -00:00 -0000`, so you can even pass an ISO8601 formatted string and the moment will be changed to that utc offset.

```
moment().utcOffset("2013-03-07T07:00:00+08:00");
```

Timezone Offset From 1.2.0, deprecated 2.9.0+ edit

```
moment().zone();                          Method Signature
moment().zone(Number|String);
```

NOTE: This function has been **deprecated** in **2.9.0**. Consider `moment.fn.utcOffset` instead.

Get the timezone offset in minutes.

```
moment().zone(); // (60, 120, 240, etc.)
```

As of version **2.1.0**, it is possible to set the offset by passing in the number of minutes offset from GMT.

```
moment().zone(120);
```

If the input is less than `16` and greater than `-16`, it will interpret your input as hours instead.

```
// these are equivalent
moment().zone(480);
moment().zone(8);
```

It is also possible to set the zone from a string.

```
moment().zone("-08:00");
```

`moment#zone` will search the string for the first match of `+00:00 +0000 -00:00 -0000`, so you can even pass an ISO8601 formatted string and the moment will be changed to that zone.

```
moment().zone("2013-03-07T07:00:00-08:00");
```

Display

Once parsing and manipulation are done, you need some way to display the moment.

Format 1.0.0+

edit

```
moment().format();                Method Signature
moment().format(String);
```

This is the most robust display option. It takes a string of tokens and replaces them with their corresponding values.

```
moment().format();                                  // "2014-
moment().format("dddd, MMMM Do YYYY, h:mm:ss a"); // "Sunda
moment().format("ddd, hA");                         // "Sun,
moment('gibberish').format('YYYY MM DD');           // "Inval
```

There are a couple conventions used with the naming of the

	Token	Output
Month	M	1 2 ... 11 12
	Mo	1st 2nd ... 11th 12th
	MM	01 02 ... 11 12
	MMM	Jan Feb ... Nov Dec
	MMMM	January February ... November December
Quarter	Q	1 2 3 4
Day of Month	D	1 2 ... 30 31
	Do	1st 2nd ... 30th 31st
	DD	01 02 ... 30 31
Day of Year	DDD	1 2 ... 364 365
	DDDo	1st 2nd ... 364th 365th
	DDDD	001 002 ... 364 365
Day of Week	d	0 1 ... 5 6
	do	0th 1st ... 5th 6th
	dd	Su Mo ... Fr Sa
	ddd	Sun Mon ... Fri Sat

	dddd	Sunday Monday ... Friday Saturday
Day of Week (Locale)	e	0 1 ... 5 6
Day of Week (ISO)	E	1 2 ... 6 7
Week of Year	w	1 2 ... 52 53
	wo	1st 2nd ... 52nd 53rd
	ww	01 02 ... 52 53
Week of Year (ISO)	W	1 2 ... 52 53
	Wo	1st 2nd ... 52nd 53rd
	WW	01 02 ... 52 53
Year	YY	70 71 ... 29 30
	YYYY	1970 1971 ... 2029 2030
Week Year	gg	70 71 ... 29 30
	gggg	1970 1971 ... 2029 2030
Week Year (ISO)	GG	70 71 ... 29 30
	GGGG	1970 1971 ... 2029 2030
AM/PM	A	AM PM
	a	am pm
Hour	H	0 1 ... 22 23
	HH	00 01 ... 22 23
	h	1 2 ... 11 12
	hh	01 02 ... 11 12
Minute	m	0 1 ... 58 59
	mm	00 01 ... 58 59
Second	s	0 1 ... 58 59
	ss	00 01 ... 58 59
Fractional Second	S	0 1 ... 8 9
	SS	0 1 ... 98 99

	SSS	0 1 ... 998 999
Timezone	z or zz	EST CST ... MST PST **Note:** as of **1.6.0**, the z/zz format tokens have been deprecated. Read more about it here.
	Z	-07:00 -06:00 ... +06:00 +07:00
	ZZ	-0700 -0600 ... +0600 +0700
Unix Timestamp	X	1360013296
Unix Millisecond Timestamp	x	1360013296123

`z ZZ` were added in **1.2.0**.

`S SS SSS` were added in **1.6.0**.

`X` was added in **2.0.0**.

`e E gg gggg GG GGGG` were added in **2.1.0**.

`x` was added in **2.8.4**.

Localized formats

Because preferred formatting differs based on locale, there are a few tokens that can be used to format a moment based on its locale.

There are upper and lower case variations on the same formats. The lowercase version is intended to be the shortened version of its uppercase counterpart.

Time	LT	8:30 PM
Time with seconds	LTS	8:30:25 PM
Month numeral, day of month, year	L	09/04/1986
	l	9/4/1986
Month name, day of month, year	LL	September

		4 1986
	ll	Sep 4 1986
Month name, day of month, year, time	LLL	September 4 1986 8:30 PM
	lll	Sep 4 1986 8:30 PM
Month name, day of month, day of week, year, time	LLLL	Thursday, September 4 1986 8:30 PM
	llll	Thu, Sep 4 1986 8:30 PM

`L LL LLL LLLL LT` are available in version **1.3.0**. `l ll lll llll` are available in **2.0.0**. `LTS` was added in **2.8.4**.

Escaping characters

To escape characters in format strings, you can wrap the characters in square brackets.

```
moment().format('[today] dddd'); // 'today Sunday'
```

Similarities and differences with LDML

Note: While these date formats are very similar to LDML date formats, there are a few minor differences regarding day of month, day of year, and day of week.

For a breakdown of a few different date formatting tokens across different locales, see [this chart of date formatting tokens.]

Formatting speed

To compare Moment.js formatting speed against other libraries, check out this comparison against other libraries.

Other tokens

If you are more comfortable working with strftime instead of LDML-like parsing tokens, you can use Ben Oakes' plugin. benjaminoakes/moment-strftime.

Default format

As of version **1.5.0**, calling `moment#format` without a format will default to `moment.defaultFormat`. Out of the box, `moment.defaultFormat` is the ISO8601 format `YYYY-MM-DDTHH:mm:ssZ`.

Time from now 1.0.0+ edit

```
moment().fromNow();                    Method Signature
moment().fromNow(Boolean);
```

A common way of displaying time is handled by `moment#fromNow`. This is sometimes called timeago or relative time.

```
moment([2007, 0, 29]).fromNow(); // 4 years ago
```

If you pass `true`, you can get the value without the suffix.

```
moment([2007, 0, 29]).fromNow();     // 4 years ago
moment([2007, 0, 29]).fromNow(true); // 4 years
```

The base strings are customized by the current locale.

The breakdown of which string is displayed for each length of time is outlined in the table below.

Range	Key	Sample Output
0 to 45 seconds	s	seconds ago
45 to 90 seconds	m	a minute ago
90 seconds to 45 minutes	mm	2 minutes ago ... 45 minutes ago
45 to 90 minutes	h	an hour ago
90 minutes to 22 hours	hh	2 hours ago ... 22 hours ago
22 to 36 hours	d	a day ago
36 hours to 25 days	dd	2 days ago ... 25 days ago
25 to 45 days	M	a month ago
45 to 345 days	MM	2 months ago ... 11 months ago
345 to 547 days (1.5 years)	y	a year ago
548 days+	yy	2 years ago ... 20 years ago

Time from X 1.0.0+ edit

```
moment().from(Moment|String|Number|Date|Array);          Method Signature
moment().from(Moment|String|Number|Date|Array, Boolean);
```

You may want to display a moment in relation to a time other than now. In that case, you can use `moment#from`.

```
var a = moment([2007, 0, 28]);
var b = moment([2007, 0, 29]);
a.from(b) // "a day ago"
```

The first parameter is anything you can pass to `moment()` or an actual `Moment`.

```
var a = moment([2007, 0, 28]);
var b = moment([2007, 0, 29]);
a.from(b);                      // "a day ago"
a.from([2007, 0, 29]);          // "a day ago"
```

```
a.from(new Date(2007, 0, 29)); // "a day ago"
a.from("2007-01-29");          // "a day ago"
```

Like `moment#fromNow`, passing `true` as the second parameter returns value without the suffix. This is useful wherever you need to have a human readable length of time.

```
var start = moment([2007, 0, 5]);
var end   = moment([2007, 0, 10]);
end.from(start);       // "in 5 days"
end.from(start, true); // "5 days"
```

Calendar Time 1.3.0+ edit

```
moment().calendar();                    Method Signature
moment().calendar(referenceTime);
```

Calendar time is displays time relative to given `referenceTime` (defaults to now), but slightly differently than `moment#fromNow`.

`moment#calendar` will format a date with different strings depending on how close to `referenceTime`'s date (today by default) the date is.

Last week	Last Monday 2:30 AM
The day before	Yesterday 2:30 AM
The same day	Today 2:30 AM
The next day	Tomorrow 2:30 AM
The next week	Sunday 2:30 AM
Everything else	7/10/2011

These strings are localized, and can be customized.

Difference 1.0.0+ edit

```
moment().diff(Moment|String|Number|Date|Array);  Method Signature
```

```
moment().diff(Moment|String|Number|Date|Array, String);
moment().diff(Moment|String|Number|Date|Array, String, Bool
```

To get the difference in milliseconds, use `moment#diff` like you would use `moment#from`.

```
var a = moment([2007, 0, 29]);
var b = moment([2007, 0, 28]);
a.diff(b) // 86400000
```

To get the difference in another unit of measurement, pass that measurement as the second argument.

```
var a = moment([2007, 0, 29]);
var b = moment([2007, 0, 28]);
a.diff(b, 'days') // 1
```

The supported measurements are years, months, weeks, days, hours, minutes, and seconds. For ease of development, the singular forms are supported as of **2.0.0**. Units of measurement other than milliseconds are available in version **1.1.1**.

By default, `moment#diff` will return number rounded down. If you want the floating point number, pass `true` as the third argument. Before **2.0.0**, `moment#diff` returned rounded number, not a rounded *down* number.

```
var a = moment([2008, 6]);
var b = moment([2007, 0]);
a.diff(b, 'years');       // 1
a.diff(b, 'years', true); // 1.5
```

If the moment is earlier than the moment you are passing to `moment.fn.diff`, the return value will be negative.

```
var a = moment();
var b = moment().add(1, 'seconds');
a.diff(b) // -1000
b.diff(a) // 1000
```

A easy way to think of this is by replacing `.diff(` with a minus operator.

```
          // a < b
a.diff(b) // a - b < 0
b.diff(a) // b - a > 0
```

Month and year diffs

`moment#diff` has some special handling for month and year diffs. It is optimized to ensure that two months with the same date are always a whole number apart.

So Jan 15 to Feb 15 should be exactly 1 month.

Feb 28 to Mar 28 should be exactly 1 month.

Feb 28 2011 to Feb 28 2012 should be exactly 1 year.

See more discussion on the month and year diffs here

This change to month and year diffs was made in **2.0.0**. As of version **2.9.0** diff also support quarter unit.

Unix Offset (milliseconds) 1.0.0+edit

```
moment().valueOf();                    Method Signature
+moment();
```

`moment#valueOf` simply outputs the number of milliseconds since the Unix Epoch, just like `Date#valueOf`.

```
moment(1318874398806).valueOf(); // 1318874398806
+moment(1318874398806); // 1318874398806
```

To get a Unix timestamp (the number of seconds since the epoch) from a `Moment`, use `moment#unix`.

Unix Timestamp (seconds) 1.6.0+ edit

```
moment().unix();                    Method Signature
```

`moment#unix` outputs a Unix timestamp (the number of seconds since the Unix Epoch).

```
moment(1318874398806).unix(); // 1318874398
```

This value is floored to the nearest second, and does not include a milliseconds component.

Days in Month 1.5.0+ edit

```
moment().daysInMonth();             Method Signature
```

Get the number of days in the current month.

```
moment("2012-02", "YYYY-MM").daysInMonth() // 29
moment("2012-01", "YYYY-MM").daysInMonth() // 31
```

As Javascript Date 1.0.0+ edit

```
moment().toDate();                  Method Signature
```

To get the native Date object that Moment.js wraps, use `moment#toDate`.

This will return the `Date` that the moment uses, so any changes to that `Date` will cause the moment to change. If you want a `Date` that is a copy, use `moment#clone` before you use `moment#toDate`.

`moment#native` has been replaced by `moment#toDate` and has been deprecated as of **1.6.0**.

As Array 1.7.0+ edit

```
moment().toArray();                    Method Signature
```

This returns an array that mirrors the parameters from `new Date()`.

```
moment().toArray(); // [2013, 1, 4, 14, 40, 16, 154];
```

As JSON 2.0.0+ edit

```
moment().toJSON();                     Method Signature
```

When serializing an object to JSON, if there is a `Moment` object, it will be represented as an ISO8601 string.

```
JSON.stringify({
    postDate : moment()
}); // '{"postDate":"2013-02-04T22:44:30.652Z"}'
```

As ISO 8601 String 2.1.0+ edit

```
moment().toISOString();                Method Signature
```

Formats a string to the ISO8601 standard.

```
moment().toISOString() // 2013-02-04T22:44:30.652Z
```

From version **2.8.4** the native `Date.prototype.toISOString` is used if available, for performance reasons.

Query

Is Before 2.0.0+ edit

```
moment().isBefore(Moment|String|Number|Date|Array); // signature
moment().isBefore(Moment|String|Number|Date|Array, String);
```

Check if a moment is before another moment.

```
moment('2010-10-20').isBefore('2010-10-21'); // true
```

If you want to limit the granularity to a unit other than milliseconds, pass the units as the second parameter.

```
moment('2010-10-20').isBefore('2010-12-31', 'year'); // fal
moment('2010-10-20').isBefore('2011-01-01', 'year'); // tru
```

Like `moment#isAfter` and `moment#isSame`, any of the units of time that are supported for `moment#startOf` are supported for `moment#isBefore`. Year, month, week, day, hour, minute, and second.

If nothing is passed to `moment#isBefore`, it will default to the current time.

NOTE: `moment().isBefore()` has undefined behavior and should not be used! If the code runs fast the initial created moment would be the same as the one created in isBefore to perform the check, so the result would be `false`. But if the code runs slower its possible that the moment created in isBefore is measurably after the one created in `moment()`, so the call would return `true`.

Is Same 2.0.0+

edit

```
moment().isSame(Moment|String|Number|Date|Array);  Method Signature
moment().isSame(Moment|String|Number|Date|Array, String);
```

Check if a moment is the same as another moment.

```
moment('2010-10-20').isSame('2010-10-20'); // true
```

If you want to limit the granularity to a unit other than milliseconds, pass the units as the second parameter.

```
moment('2010-10-20').isSame('2009-12-31', 'year'); // false
moment('2010-10-20').isSame('2010-01-01', 'year'); // true
moment('2010-10-20').isSame('2010-12-31', 'year'); // true
moment('2010-10-20').isSame('2011-01-01', 'year'); // false
```

Like `moment#isAfter` and `moment#isBefore`, any of the units of time that are supported for `moment#startOf` are supported for `moment#isSame`. Year, month, week, day, hour, minute, and second.

Is After 2.0.0+

edit

```
moment().isAfter(Moment|String|Number|Date|Array);  Method Signature
moment().isAfter(Moment|String|Number|Date|Array, String);
```

Check if a moment is after another moment.

```
moment('2010-10-20').isAfter('2010-10-19'); // true
```

If you want to limit the granularity to a unit other than milliseconds, pass the units as the second parameter.

```
moment('2010-10-20').isAfter('2010-01-01', 'year'); // fals
moment('2010-10-20').isAfter('2009-12-31', 'year'); // true
```

Like `moment#isSame` and `moment#isBefore`, any of the units of time that are supported for `moment#startOf` are supported for `moment#isAfter`. Year, month, week, day, hour, minute, and second.

If nothing is passed to `moment#isAfter`, it will default to the current time.

```
moment().isAfter(); // false
```

Is Between 2.9.0+ edit

```
moment().isBetween(moment-like, moment-like);  Method Signature
moment().isBetween(moment-like, moment-like, String);
// where moment-like is Moment|String|Number|Date|Array
```

Check if a moment is between two other moments, optionally looking at unit scale (minutes, hours, days, etc).

```
moment('2010-10-20').isBetween('2010-10-19', '2010-10-25');
```

If you want to limit the granularity to a unit other than milliseconds, pass the units as the second parameter.

```
moment('2010-10-20').isBetween('2010-01-01', '2012-01-01',
moment('2010-10-20').isBetween('2009-12-31', '2012-01-01',
```

Like `moment#isSame`, `moment#isBefore`, `moment#isAfter` any of the units of time that are supported for `moment#startOf` are supported for `moment#isAfter`. Year, month, week, day, hour, minute, and second.

Is Leap Year 1.0.0+ edit

```
moment().isLeapYear();  Method Signature
```

`moment#isLeapYear` returns `true` if that year is a leap year, and `false` if it is not.

```
moment([2000]).isLeapYear() // true
moment([2001]).isLeapYear() // false
moment([2100]).isLeapYear() // false
```

Is Daylight Saving Time 1.2.0+ edit

```
moment().isDST();                                    Method Signature
```

`moment#isDST` checks if the current moment is in daylight savings time.

```
moment([2011, 2, 12]).isDST(); // false, March 12 2011 is n
moment([2011, 2, 14]).isDST(); // true, March 14 2011 is DS
```

Is DST Shifted 2.3.0+ edit

```
moment('2013-03-10 2:30', 'YYYY-MM-DD HH:mm').isDSTShifted(
```

Another important piece of validation is to know if the date has been moved by a DST. For example, in most of the United States:

```
moment('2013-03-10 2:30', 'YYYY-MM-DD HH:mm').format(); //=
```

This is because daylight savings time shifts the time from 2:00 to 3:00, so 2:30 isn't a real time. The resulting time is browser-dependent, either adjusting the time forward or backwards. Use `moment#isDSTShifted` to test for this condition.

Note: before 2.3.0, Moment objects in this condition always returned `false` for `moment#isValid`; they now return `true`.

Is a Moment 1.5.0+

edit

```
moment.isMoment(obj);                    Method Signature
```

To check if a variable is a moment object, use `moment.isMoment()`.

```
moment.isMoment() // false
moment.isMoment(new Date()) // false
moment.isMoment(moment()) // true
```

Is a Date 2.9.0+

edit

```
moment.isDate(obj);                      Method Signature
```

To check if a variable is a native js Date object, use `moment.isDate()`.

```
moment.isDate(); // false
moment.isDate(new Date()); // true
moment.isDate(moment()); // false
```

i18n

Moment.js has robust support for internationalization.

You can load multiple locales and easily switch between them.

In addition to assigning a global locale, you can assign a locale to a specific moment.

Changing locale globally 1.0.0+ edit

```
// From 2.8.1 onward                                    Method Signature
moment.locale(String);
moment.locale(String[]);
moment.locale(String, Object);

// Deprecated in 2.8.1
moment.lang(String);
moment.lang(String[]);
moment.lang(String, Object);
```

By default, Moment.js comes with English locale strings. If you need other locales, you can load them into Moment.js for later use.

To load a locale, pass the key and the string values to `moment.locale`.

More details on each of the parts of the locale bundle can be found in the customization section.

```
moment.locale('fr', {
    months : "janvier_février_mars_avril_mai_juin_juillet_a
    monthsShort : "janv._févr._mars_avr._mai_juin_juil._aoû
    weekdays : "dimanche_lundi_mardi_mercredi_jeudi_vendred
    weekdaysShort : "dim._lun._mar._mer._jeu._ven._sam.".sp
    weekdaysMin : "Di_Lu_Ma_Me_Je_Ve_Sa".split("_"),
    longDateFormat : {
        LT : "HH:mm",
        LTS : "HH:mm:ss",
        L : "DD/MM/YYYY",
        LL : "D MMMM YYYY",
        LLL : "D MMMM YYYY LT",
        LLLL : "dddd D MMMM YYYY LT"
    },
    calendar : {
        sameDay: "[Aujourd'hui à] LT",
        nextDay: '[Demain à] LT',
        nextWeek: 'dddd [à] LT',
        lastDay: '[Hier à] LT',
        lastWeek: 'dddd [dernier à] LT',
```

```
        sameElse: 'L'
    },
    relativeTime : {
        future : "dans %s",
        past : "il y a %s",
        s : "quelques secondes",
        m : "une minute",
        mm : "%d minutes",
        h : "une heure",
        hh : "%d heures",
        d : "un jour",
        dd : "%d jours",
        M : "un mois",
        MM : "%d mois",
        y : "une année",
        yy : "%d années"
    },
    ordinalParse : /\d{1,2}(er|ème)/,
    ordinal : function (number) {
        return number + (number === 1 ? 'er' : 'ème');
    },
    meridiemParse: /PD|MD/,
    isPM: function (input) {
        return input.charAt(0) === 'M';
    },
    // in case the meridiem units are not separated around
    // this function (look at locale/id.js for an example)
    // meridiemHour : function (hour, meridiem) {
    //     return /* 0-23 hour, given meridiem token and ho
    // },
    meridiem : function (hours, minutes, isLower) {
        return hours < 12 ? 'PD' : 'MD';
    },
    week : {
        dow : 1, // Monday is the first day of the week.
        doy : 4  // The week that contains Jan 4th is the f
    }
});
```

Once you load a locale, it becomes the active locale. To change active

locales, simply call `moment.locale` with the key of a loaded locale.

```
moment.locale('fr');
moment(1316116057189).fromNow() // il y a une heure
moment.locale('en');
moment(1316116057189).fromNow() // an hour ago
```

`moment.locale` returns the locale used. This is useful because Moment won't change locales if it doesn't know the one you specify.

```
moment.locale('fr'); // 'fr'
moment.locale('tq'); // 'fr'
```

You may also specify a list of locales, and Moment will use the first one it has localizations for.

```
moment.locale(['tq', 'fr']); // 'fr'
```

Moment will also try locale specifier substrings from most-specific to least-specific until it finds a locale it knows. This is useful when supplying Moment with a locale string pulled from the user's environment, such as `window.navigator.language`.

```
moment.locale('en-NZ'); // 'en'
```

Finally, Moment will search intelligently through an array of locales and their substrings.

```
moment.locale('en-NZ', 'en-AU'); // 'en-au', not 'en'
```

Changing locales locally 1.7.0+ edit

```
// From version 2.8.1 onward                    Method Signature
moment().locale(String);

// Deprecated version 2.8.1
moment().lang(String);
```

A global locale configuration can be problematic when passing around

moments that may need to be formatted into different locale.

In **1.7.0** we added instance specific locale configurations.

```
moment.locale('en'); // default the locale to English
var globalLocale = moment();
var localLocale = moment();

localLocale.locale('fr'); // set this instance to use Frenc
localLocale.format('LLLL'); // dimanche 15 juillet 2012 11:
globalLocale.format('LLLL'); // Sunday, July 15 2012 11:01

moment.locale('es'); // change the global locale to Spanish
localLocale.format('LLLL'); // dimanche 15 juillet 2012 11:
globalLocale.format('LLLL'); // Domingo 15 Julio 2012 11:03

localLocale.locale(false); // reset the instance locale
localLocale.format('LLLL'); // Domingo 15 Julio 2012 11:03
globalLocale.format('LLLL'); // Domingo 15 Julio 2012 11:03
```

If you call `moment#locale` with no parameters, you get back the locale configuration that would be used for that moment.

```
var fr = moment().locale('fr');
fr.locale().months(moment([2012, 0])) // "janvier"
fr.locale('en');
fr.locale().months(moment([2012, 0])) // "January"
```

If you need to access the locale data for a moment, this is the preferred way to do so.

As of **2.3.0**, you can also specify an array of locale identifiers. It works the same was it does in the global locale configuration.

Loading locales in NodeJS 1.0.0 edit

```
moment.locale(String);                    Method Signature
```

Loading locales in NodeJS is super easy. If there is a locale file in

`moment-root/locale/` named after that key, the first call to `moment.locale` will load it.

```
var moment = require('moment');
moment.locale('fr');
moment(1316116057189).fromNow(); // il y a une heure
```

If you want your locale supported, create a pull request to the `develop` branch with the required locale and unit test files.

Loading locales in the browser edit

1.0.0+

```
// From 2.8.1 onward                                Method Signature
moment.locale(String, Object);

// Deprecated in 2.8.1
moment.lang(String, Object);
```

Loading locales in the browser just requires you to include the locale files.

```
<script src="moment.js"></script>
<script src="locale/fr.js"></script>
<script src="locale/pt.js"></script>
<script>
  moment.locale('fr');  // Set the default/global locale
  // ...
</script>
```

There are minified versions of all locales together:

```
<script src="moment.js"></script>
<script src="min/locales.js"></script>
```

Ideally, you would bundle all the files you need into one file to minimize http requests.

```
grunt embedLocales --embedLocales fr,it
```

```
<script src="min/moment-with-customlocales.js"></script>
```

Note: Locale files are defined in UMD style, so they should work seamlessly in all environments.

Adding your locale to Moment.js

edit

To add your locale to Moment.js, submit a pull request with both a locale file and a test file. You can find examples in `moment/locale/fr.js` and `moment/test/locale/fr.js`.

To run the tests in Node.js, do `npm install`, then `grunt`.

If all the tests pass, submit a pull request, and thank you for contributing!

Checking the current Moment.js locale 1.6.0+

edit

```
// From version 2.8.1 onward                    Method Signature
moment.locale();

// Deprecated in version 2.8.1
moment.lang();
```

If you are changing locales frequently, you may want to know what locale is currently being used. This is as simple as calling `moment.locale` without any parameters.

```
moment.locale('en'); // set to english
moment.locale(); // returns 'en'
moment.locale('fr'); // set to french
```

```
moment.locale(); // returns 'fr'
```

Listing the months and weekdays of the current Moment.js locale 2.3.0+

edit

```
moment.months()                    Method Signature
moment.monthsShort()
moment.weekdays()
moment.weekdaysShort()
moment.weekdaysMin()
```

It is sometimes useful to get the list of months or weekdays in a locale, for example when populating a dropdown menu.

```
moment.months();
```

Returns the list of months in the current locale.

```
[ 'January',
  'February',
  'March',
  'April',
  'May',
  'June',
  'July',
  'August',
  'September',
  'October',
  'November',
  'December' ]
```

Similarly, `moment.monthsShort` returns abbreviated month names, and `moment.weekdays`, `moment.weekdaysShort`, `moment.weekdaysMin` return lists of weekdays.

You can pass an integer into each of those functions to get a specific

month or weekday.

```
moment.weekdays(3); // 'Wednesday'
```

Note: Currently, weekdays always have Sunday as index 0, regardless of the local first day of the week.

Some locales make special considerations into account when formatting month names. For example, Dutch formats month abbreviations without a trailing period, but only if it's formatting the month between dashes. The `months` method supports passing a format in so that the months will be listed in the proper context.

```
moment.locale('nl');
moment.monthsShort(); // ['jan.', 'feb.', 'mrt.', ...]
moment.monthsShort('-MMM-'); // [ 'jan', 'feb', 'mrt', ...]
```

And finally, you can combine both the format option and the integer option.

```
moment.monthsShort('-MMM-', 3); // 'apr'
```

Accessing locale specific functionality 2.2.0+ edit

```
localeData = moment.localeData()            Method Signature
localeData.months()
localeData.monthsShort()
localeData.monthsParse()
localeData.weekdays()
localeData.weekdaysShort()
localeData.weekdaysMin()
localeData.weekdaysParse()
localeData.longDateFormat()
localeData.isPM()
localeData.meridiem()
localeData.calendar()
```

```
localeData.relativeTime()
localeData.pastFuture()
localeData.ordinal()
localeData.preparse()
localeData.postformat()
localeData.weeks()
localeData.invalidDate()
localeData.firstDayOfWeek()
localeData.firstDayOfYear()
```

You can access the properties of the currently loaded locale through the `moment.localeData(key)` function. It returns the current locale or a locale with the given key:

```
// get current locale
var currentLocaleData = moment.localeData();
var frLocaleData = moment.localeData('fr');
```

The returned object has the following methods:

```
localeData.months(aMoment);  // full month name of aMoment
localeData.monthsShort(aMoment);  // short month name of aM
localeData.monthsParse(longOrShortMonthString);  // returns
localeData.weekdays(aMoment);  // full weekday name of aMom
localeData.weekdaysShort(aMoment);  // short weekday name o
localeData.weekdaysMin(aMoment);  // min weekday name of aM
localeData.weekdaysParse(minShortOrLongWeekdayString);  //
localeData.longDateFormat(dateFormat);  // returns the full
localeData.isPM(amPmString);  // returns true iff amPmStrin
localeData.meridiem(hours, minutes, isLower);  // returns a
localeData.calendar(key, aMoment);  // returns a format tha
localeData.relativeTime(number, withoutSuffix, key, isFutur
localeData.pastFuture(diff, relTime);  // convert relTime s
localeData.ordinal(number);  // convert number to ordinal s
localeData.preparse(str);  // called before parsing on ever
localeData.postformat(str);  // called after formatting on
localeData.week(aMoment);  // returns week-of-year of aMome
localeData.invalidDate();  // returns a translation of 'Inv
localeData.firstDayOfWeek();  // 0-6 (Sunday to Saturday)
localeData.firstDayOfYear();  // 0-15 this and the first da
                              // to determine which is the
```

```
                    // year. dow == 1 and doy ==
                    // Monday and first week that
                    // first week of the year (bu
                    // Thursday).
```

Customize

Moment.js is very easy to customize. In general, you should create a locale setting with your customizations.

```
moment.locale('en-my-settings', {
    // customizations.
});
```

However, you can also overwrite an existing locale that has been loaded as well.

```
moment.locale('en', {
    // customizations
});
```

Any settings that are not defined are inherited from the default english settings.

Month Names 1.0.0+ edit

```
// From 2.8.1 onward                    Method Signature
moment.locale('en', {
    months : String[]
});
moment.locale('en', {
    months : Function
```

```
});

// Deprecated in 2.8.1
moment.lang('en', {
    months : String[]
});
moment.lang('en', {
    months : Function
});
```

`Locale#months` should be an array of the month names.

```
moment.locale('en', {
    months : [
        "January", "February", "March", "April", "May", "Ju
        "August", "September", "October", "November", "Dece
    ]
});
```

If you need more processing to calculate the name of the month, (for example, if there is different grammar for different formats), `Locale#months` can be a function with the following signature. It should always return a month name.

```
moment.locale('en', {
    months : function (momentToFormat, format) {
        // momentToFormat is the moment currently being for
        // format is the formatting string
        if (/^MMMM/.test(format)) { // if the format starts
            return nominative[momentToFormat.month()];
        } else {
            return subjective[momentToFormat.month()];
        }
    }
});
```

Month Abbreviations 1.0.0+ edit

```
// From 2.8.1 onward                                    Method Signature
moment.locale('en', {
    monthsShort : String[]
});
moment.locale('en', {
    monthsShort : Function
});

// Deprecated in 2.8.1
moment.lang('en', {
    monthsShort : String[]
});
moment.lang('en', {
    monthsShort : Function
});
```

`Locale#monthsShort` should be an array of the month abbreviations.

```
moment.locale('en', {
    monthsShort : [
        "Jan", "Feb", "Mar", "Apr", "May", "Jun",
        "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
    ]
});
```

Like `Locale#months`, `Locale#monthsShort` can be a callback function as well.

```
moment.locale('en', {
    monthsShort : function (momentToFormat, format) {
        if (/^MMMM/.test(format)) {
            return nominative[momentToFormat.month()];
        } else {
            return subjective[momentToFormat.month()];
        }
    }
});
```

Weekday Names 1.0.0+ edit

```
// From version 2.8.1 onward                    Method Signature
moment.locale('en', {
    weekdays : String[]
});
moment.locale('en', {
    weekdays : Function
});

// Deprecated version 2.8.1
moment.lang('en', {
    weekdays : String[]
});
moment.lang('en', {
    weekdays : Function
});
```

`Locale#weekdays` should be an array of the weekdays names.

```
moment.locale('en', {
    weekdays : [
        "Sunday", "Monday", "Tuesday", "Wednesday", "Thursd
    ]
});
```

`Locale#weekdays` can be a callback function as well.

```
moment.locale('en', {
    weekdays : function (momentToFormat, format) {
        return weekdays[momentToFormat.day()];
    }
});
```

Weekday Abbreviations 1.0.0+ edit

```
// From 2.8.1 onward                            Method Signature
```

```
moment.locale('en', {
    weekdaysShort : String[]
});
moment.locale('en', {
    weekdaysShort : Function
});

// Deprecated in 2.8.1
moment.lang('en', {
    weekdaysShort : String[]
});
moment.lang('en', {
    weekdaysShort : Function
});
```

`Locale#weekdaysShort` should be an array of the weekdays abbreviations.

```
moment.locale('en', {
    weekdaysShort : ["Sun", "Mon", "Tue", "Wed", "Thu", "Fr
});
```

`Locale#weekdaysShort` can be a callback function as well.

```
moment.locale('en', {
    weekdaysShort : function (momentToFormat, format) {
        return weekdaysShort[momentToFormat.day()];
    }
});
```

Minimal Weekday Abbreviations 1.7.0+

edit

```
// From 2.8.1 onward                    Method Signature
moment.locale('en', {
    weekdaysMin : String[]
});
```

```
moment.locale('en', {
    weekdaysMin : Function
});

// Deprecated in 2.8.1
moment.lang('en', {
    weekdaysMin : String[]
});
moment.lang('en', {
    weekdaysMin : Function
});
```

`Locale#weekdaysMin` should be an array of two letter weekday abbreviations. The purpose of these is for things like calendar pickers, thus they should be as small as possible.

```
moment.locale('en', {
    weekdaysMin : ["Su", "Mo", "Tu", "We", "Th", "Fr", "Sa"
});
```

`Locale#weekdaysMin` can be a callback function as well.

```
moment.locale('en', {
    weekdaysMin : function (momentToFormat, format) {
        return weekdaysMin[momentToFormat.day()];
    }
});
```

Long Date Formats 1.1.0+ edit

```
// From 2.8.1 onward                          Method Signature
moment.locale('en', {
    longDateFormat : Object
});

// Deprecated in 2.8.1
moment.lang('en', {
```

```
    longDateFormat : Object
});
```

`Locale#longDateFormat` should be an object containing a key/value pair for each long date format `L LL LLL LLLL LT LTS`. `LT` should be the time format, and is also used for `moment#calendar`.

```
moment.locale('en', {
    longDateFormat : {
        LT: "h:mm A",
        LTS: "h:mm:ss A",
        L: "MM/DD/YYYY",
        l: "M/D/YYYY",
        LL: "MMMM Do YYYY",
        ll: "MMM D YYYY",
        LLL: "MMMM Do YYYY LT",
        lll: "MMM D YYYY LT",
        LLLL: "dddd, MMMM Do YYYY LT",
        llll: "ddd, MMM D YYYY LT"
    }
});
```

You can eliminate the lowercase `l` tokens and they will be created automatically by replacing long tokens with the short token variants.

```
moment.locale('en', {
    longDateFormat : {
        LT: "h:mm A",
        LTS: "h:mm:ss A",
        L: "MM/DD/YYYY",
        LL: "MMMM Do YYYY",
        LLL: "MMMM Do YYYY LT",
        LLLL: "dddd, MMMM Do YYYY LT"
    }
});
```

Relative Time 1.0.0+

edit

```
// From 2.8.1 onward                                    Method Signature
moment.locale('en', {
    relativeTime : Object
});

// Deprecated in 2.8.1
moment.lang('en', {
    relativeTime : Object
});
```

`Locale#relativeTime` should be an object of the replacement strings for `moment#from`.

```
moment.locale('en', {
    relativeTime : {
        future: "in %s",
        past:   "%s ago",
        s:  "seconds",
        m:  "a minute",
        mm: "%d minutes",
        h:  "an hour",
        hh: "%d hours",
        d:  "a day",
        dd: "%d days",
        M:  "a month",
        MM: "%d months",
        y:  "a year",
        yy: "%d years"
    }
});
```

`Locale#relativeTime.future` refers to the prefix/suffix for future dates, and `Locale#relativeTime.past` refers to the prefix/suffix for past dates. For all others, a single character refers to the singular, and an double character refers to the plural.

If a locale requires additional processing for a token, It can set the token as a function with the following signature. The function should return a string.

```
function (number, withoutSuffix, key, isFuture) {
```

```
    return string;
}
```

The `key` argument refers to the replacement key in the `Locale#relativeTime` object. (eg. `s m mm h`, etc.)

The `number` argument refers to the number of units for that key. For `m`, the number is the number of minutes, etc.

The `withoutSuffix` argument will be true if the token will be displayed without a suffix, and false if it will be displayed with a suffix. (The reason for the inverted logic is because the default behavior is to display with the suffix.)

The `isFuture` argument will be true if it is going to use the future suffix/prefix and false if it is going to use the past prefix/suffix. The `isFuture` argument was added in version **1.6.0**.

AM/PM 1.6.0+ edit

```
// From 2.8.1 onward                                Method Signature
moment.locale('en', {
    meridiem : Function
});

// Deprecated in 2.8.1
moment.lang('en', {
    meridiem : Function
});
```

If your locale uses 'am/pm', `Locale#meridiem` can be ommitted, as those values are the defaults.

If your locale needs any different computation for am/pm, `Locale#meridiem` should be a callback function that returns the correct string based on hour, minute, and upper/lowercase.

```
moment.locale('zh-cn', {
    meridiem : function (hour, minute, isLowercase) {
        if (hour < 9) {
```

```
            return "早上";
        } else if (hour < 11 && minute < 30) {
            return "上午";
        } else if (hour < 13 && minute < 30) {
            return "中午";
        } else if (hour < 18) {
            return "下午";
        } else {
            return "晚上";
        }
    }
});
```

Before version **1.6.0**, `Locale#meridiem` was a map of upper and lowercase versions of am/pm.

```
moment.locale('en', {
    meridiem : {
        am : 'am',
        AM : 'AM',
        pm : 'pm',
        PM : 'PM'
    }
});
```

This has been deprecated. The **1.6.0** callback function syntax is now used instead.

AM/PM Parsing 2.1.0+ edit

```
// From 2.8.1 onward                                    Method Signature
moment.locale('en', {
    meridiemParse : RegExp
    isPM : Function
});

// Deprecated in 2.8.1
moment.lang('en', {
```

```
    meridiemParse : RegExp
    isPM : Function
});
```

`Locale#isPM` should return true if the input string is past 12 noon. This is used in parsing the `a A` tokens.

```
moment.locale('en', {
    isPM : function (input) {
        return ((input + '').toLowerCase()[0] === 'p');
    }
});
```

To configure what strings should be parsed as input, set the `meridiemParse` property.

```
moment.locale('en', {
    meridiemParse : /[ap]\.?m?\.?/i
});
```

Calendar 1.3.0+ edit

```
// From 2.8.1 onward                                    Method Signature
moment.locale('en', {
    calendar : Object
});

// Deprecated in 2.8.1
moment.lang('en', {
    calendar : Object
});
```

`Locale#calendar` should have the following formatting strings.

```
moment.locale('en', {
    calendar : {
        lastDay : '[Yesterday at] LT',
        sameDay : '[Today at] LT',
```

```
        nextDay : '[Tomorrow at] LT',
        lastWeek : '[last] dddd [at] LT',
        nextWeek : 'dddd [at] LT',
        sameElse : 'L'
    }
});
```

Each of the `Locale#calendar` keys can also be a callback function with the scope of the current moment. It should return a formatting string.

```
function () {
    return '[hoy a la' + ((this.hours() !== 1) ? 's' : '')
},
```

Ordinal 1.0.0+ edit

```
// From 2.8.1 onward                                    Method Signature
moment.locale('en', {
    ordinal : Function
});

// Deprecated in 2.8.1
moment.lang('en', {
    ordinal : Function
});
```

`Locale#ordinal` should be a function that returns the ordinal for a given number.

```
moment.locale('en', {
    ordinal : function (number, token) {
        var b = number % 10;
        var output = (~~ (number % 100 / 10) === 1) ? 'th'
            (b === 1) ? 'st' :
            (b === 2) ? 'nd' :
            (b === 3) ? 'rd' : 'th';
```

```
        return number + output;
    }
});
```

As of **2.0.0**, the ordinal function should return both the number and the ordinal. Previously, only the ordinal was returned.

As of **2.1.0**, the token parameter was added. It is a string of the token that is being ordinalized, for example: `M` or `d`.

For more information on ordinal numbers, see wikipedia

Relative Time Thresholds 2.7.0+ edit

```
moment.relativeTimeThreshold(unit);  // getter
moment.relativeTimeThreshold(unit, limit);  // setter
```
Method Signature

`duration.humanize` has thresholds which define when a unit is considered a minute, an hour and so on. For example, by default more than 45 seconds is considered a minute, more than 22 hours is considered a day and so on. To change those cutoffs use `moment.relativeTimeThreshold(unit, limit)` where limit is one of `s`, `m`, `h`, `d`, `M`.

unit	meaning	usage
s	seconds	least number of seconds to be considered a minute
m	minutes	least number of minutes to be considered an hour
h	hours	least number of hours to be considered a day
d	days	least number of days to be considered a month
M	months	least number of months to be considered a year

```
// Retrieve existing thresholds
moment.relativeTimeThreshold('s');  // 45
```

```
moment.relativeTimeThreshold('m');  // 45
moment.relativeTimeThreshold('h');  // 22
moment.relativeTimeThreshold('d');  // 26
moment.relativeTimeThreshold('M');  // 11

// Set new thresholds
moment.relativeTimeThreshold('s', 40);
moment.relativeTimeThreshold('m', 40);
moment.relativeTimeThreshold('h', 20);
moment.relativeTimeThreshold('d', 25);
moment.relativeTimeThreshold('M', 10);
```

NOTE: Retrieving thresholds was added in **2.8.1**.

Durations

Moment.js also has duration objects. Where a moment is defined as single points in time, durations are defined as a length of time.

Durations do not have a defined beginning and end date. They are contextless.

A duration is conceptually more similar to '2 hours' than to 'between 2 and 4 pm today'. As such, they are not a good solution to converting between units that depend on context.

For example, a year can be defined as 366 days, 365 days, 365.25 days, 12 months, or 52 weeks. Trying to convert years to days makes no sense without context. It is much better to use `moment#diff` for calculating days or years between two moments than to use `Durations`.

Creating 1.6.0+

edit

```
moment.duration(Number, String);          Method Signature
moment.duration(Number);
moment.duration(Object);
moment.duration(String);
```

To create a duration, call `moment.duration()` with the length of time in milliseconds.

```
moment.duration(100); // 100 milliseconds
```

If you want to create a moment with a unit of measurement other than milliseconds, you can pass the unit of measurement as well.

```
moment.duration(2, 'seconds');
moment.duration(2, 'minutes');
moment.duration(2, 'hours');
moment.duration(2, 'days');
moment.duration(2, 'weeks');
moment.duration(2, 'months');
moment.duration(2, 'years');
```

The same shorthand for `moment#add` and `moment#subtract` works here as well.

Key	Shorthand
years	y
months	M
weeks	w
days	d
hours	h
minutes	m
seconds	s
milliseconds	ms

Much like `moment#add`, you can pass an object of values if you need multiple different units of measurement.

```
moment.duration({
```

```
    seconds: 2,
    minutes: 2,
    hours: 2,
    days: 2,
    weeks: 2,
    months: 2,
    years: 2
});
```

As of **2.1.0**, moment supports parsing ASP.NET style time spans. The following formats are supported.

The format is an hour, minute, second string separated by colons like `23:59:59`. The number of days can be prefixed with a dot separator like so `7.23:59:59`. Partial seconds are supported as well `23:59:59.999`.

```
moment.duration('23:59:59');
moment.duration('23:59:59.999');
moment.duration('7.23:59:59.999');
moment.duration('23:59');          //added in 2.3.0
```

Humanize 1.6.0+ edit

```
moment.duration().humanize();                Method Signature
```

Sometimes, you want all the goodness of `moment#from` but you don't want to have to create two moments, you just want to display a length of time.

Enter `moment.duration().humanize()`.

```
moment.duration(1, "minutes").humanize(); // a minute
moment.duration(2, "minutes").humanize(); // 2 minutes
moment.duration(24, "hours").humanize();  // a day
```

By default, the return string is suffixless. If you want a suffix, pass in true as seen below.

```
moment.duration(1, "minutes").humanize(true); // in a minut
```

For suffixes before now, pass in a negative number.

```
moment.duration(-1, "minutes").humanize(true); // a minute
```

Milliseconds 1.6.0+ edit

```
moment.duration().milliseconds();          Method Signature
moment.duration().asMilliseconds();
```

To get the number of milliseconds in a duration, use `moment.duration().milliseconds()`.

It will return a number between 0 and 999.

```
moment.duration(500).milliseconds(); // 500
moment.duration(1500).milliseconds(); // 500
moment.duration(15000).milliseconds(); // 0
```

If you want the length of the duration in milliseconds, use `moment.duration().asMilliseconds()` instead.

```
moment.duration(500).asMilliseconds(); // 500
moment.duration(1500).asMilliseconds(); // 1500
moment.duration(15000).asMilliseconds(); // 15000
```

Seconds 1.6.0+ edit

```
moment.duration().seconds();          Method Signature
moment.duration().asSeconds();
```

To get the number of seconds in a duration, use `moment.duration().seconds()`.

It will return a number between 0 and 59.

```
moment.duration(500).seconds(); // 0
moment.duration(1500).seconds(); // 1
moment.duration(15000).seconds(); // 15
```

If you want the length of the duration in seconds, use `moment.duration().asSeconds()` instead.

```
moment.duration(500).asSeconds(); // 0.5
moment.duration(1500).asSeconds(); // 1.5
moment.duration(15000).asSeconds(); // 15
```

Minutes 1.6.0+ edit

```
moment.duration().minutes();                Method Signature
moment.duration().asMinutes();
```

As with the other getters for durations, `moment.duration().minutes()` gets the minutes (0 - 59).

`moment.duration().asMinutes()` gets the length of the duration in minutes.

Hours 1.6.0+ edit

```
moment.duration().hours();                  Method Signature
moment.duration().asHours();
```

As with the other getters for durations, `moment.duration().hours()` gets the hours (0 - 23).

`moment.duration().asHours()` gets the length of the duration in hours.

Days 1.6.0+ edit

```
moment.duration().days();                Method Signature
moment.duration().asDays();
```

As with the other getters for durations, `moment.duration().days()` gets the days (0 - 29).

`moment.duration().asDays()` gets the length of the duration in days.

Months 1.6.0+ edit

```
moment.duration().months();              Method Signature
moment.duration().asMonths();
```

As with the other getters for durations, `moment.duration().months()` gets the months (0 - 11).

`moment.duration().asMonths()` gets the length of the duration in months.

Note: The length of a duration in months is defined as 30 days.

Years 1.6.0+ edit

```
moment.duration().years();               Method Signature
moment.duration().asYears();
```

As with the other getters for durations, `moment.duration().years()` gets the years.

`moment.duration().asYears()` gets the length of the duration in years.

Note: The length of a duration in years is defined as 365 days.

Add Time 2.1.0+

edit

```
moment.duration().add(Number, String);      Method Signature
moment.duration().add(Number);
moment.duration().add(Duration);
moment.duration().add(Object);
```

Mutates the original duration by adding time.

```
var a = moment.duration(1, 'd');
var b = moment.duration(2, 'd');
a.add(b).days(); // 3
```

Subtract Time 2.1.0+

edit

```
moment.duration().subtract(Number, String); Method Signature
moment.duration().subtract(Number);
moment.duration().subtract(Duration);
moment.duration().subtract(Object);
```

Mutates the original duration by subtracting time.

```
var a = moment.duration(3, 'd');
var b = moment.duration(2, 'd');
a.subtract(b).days(); // 1
```

As Unit of Time 2.1.0+

edit

```
moment.duration().as(String);               Method Signature
```

As an alternate to `Duration#asX`, you can use `Duration#as('x')`.
All the shorthand keys from `moment#add` apply here as well.

```
duration.as('hours');
duration.as('minutes');
```

```
duration.as('seconds');
duration.as('milliseconds');
```

Get Unit of Time 2.1.0+ edit

```
moment.duration().get(String);          Method Signature
```

As an alternate to `Duration#x()` getters, you can use `Duration#get('x')`. All the shorthand keys from `moment#add` apply here as well.

```
duration.get('hours');
duration.get('minutes');
duration.get('seconds');
duration.get('milliseconds');
```

As JSON 2.9.0+ edit

```
moment.duration().toJSON();          Method Signature
```

When serializing a duration object to JSON, it will be be represented as an ISO8601 string.

```
JSON.stringify({
    postDuration : moment.duration(5, 'm')
}); // '{"postDuration":"PT5M"}'
```

Utilities

Moment exposes some methods which may be useful to people

extending the library or writing custom parsers.

Normalize Units 2.3.0+ edit

```
moment.normalizeUnits(String);                    Method Signature
```

Many of Moment's functions allow the caller to pass in aliases for unit enums. For example, all of the `get`s below are equivalent.

```
var m = moment();
m.get('y');
m.get('year');
m.get('years');
```

If you're extending the library, you may want access to Moment's facilities for that in order to better align your functionality with Moment's.

```
moment.normalizeUnits('y');      // 'year'
moment.normalizeUnits('Y');      // 'year'
moment.normalizeUnits('year');   // 'year'
moment.normalizeUnits('years');  // 'year'
moment.normalizeUnits('YeARS');  // 'year'
```

Invalid 2.3.0+ edit

```
moment.invalid(Object);                           Method Signature
```

You can create your own invalid Moment objects, which is useful in making your own parser.

```
var m = moment.invalid();
m.isValid();                     // false
m.format();                      // 'Invalid date'
m.parsingFlags().userInvalidated; // true
```

`invalid` also accepts an object which specifies which parsing flags to set. This will *not* set the `userInvalidated` parsing flag unless it's one of the properties specified.

```
var m = moment.invalid({invalidMonth: 'Actober'});
m.parsingFlags().invalidMonth; // 'Actober'
```

You need not specify parsing flags recognized by Moment; the Moment will be invalid nonetheless, and the parsing flags will be returned by `parsingFlags()`.

Plugins

Some other people have made plugins for Moment.js that may be useful to you.

Strftime

edit

```
npm install moment-strftime                    Method Signature
```

If you are more comfortable working with strftime instead of LDML-like parsing tokens, you can use Ben Oakes' plugin `moment-strftime`.

The repository is located at github.com/benjaminoakes/moment-strftime

ISO Calendar

edit

```
npm install moment-isocalendar                 Method Signature
```

If you are looking for a Python-like isocalendar method, you can use

Rocky Meza's plugin

`moment-isocalendar`

Calling the isocalendar method on a moment will return an array like the following:

`[year, week_of_year, day_of_week, minutes_since_midnight]`

```
moment().isocalendar(); // [2012, 8, 5, 870]
```

You can also reconstruct a moment from a isocalendar array.

```
moment.fromIsocalendar([2011, 51, 5, 870]).format('LLLL');
// "Friday, December 23 2011 2:30 PM"
```

The repository is located at github.com/fusionbox/moment-isocalendar

Date Ranges

edit

```
npm install moment-range                    Method Signature
```

If you need to work with date ranges, you can use Gianni Chiappetta's plugin `moment-range`.

Documentation can be found on the homepage github.com/gf3/moment-range.

And it is also available for the web at the repository below.

The repository is located at github.com/gf3/moment-range

Twix

edit

```
npm install twix                            Method Signature
```

Another range plugin is Isaac Cambron's library `Twix`. It has many range-related features and excels at formatting ranges readably. For

example,

```
var t = moment("1/25/1982 9:30 AM").twix("1/25/1982 1:30 PM
t.isCurrent(); // false
t.count('minutes'); // 241
t.format();  // 'Jan 25, 1982, 9:30 AM - 1:30 PM'
t.simpleFormat("h:m"); // '9:30 - 1:30'
```

Full documentation of all the options and features is here.

It's available on npm like so:

```
npm install twix
```

Or just grab the JS file from here.

Twitter

edit

If you're trying to format times for tweets like the way Twitter does, you can use the moment.twitter plugin by @hijonathan.

It's a simple way to display both short and long versions of human-readable timestamps.

```
moment().subtract(5, 'hours').twitter();
// 5 hours
```

Yes, it does smart pluralization.

```
moment().subtract(1, 'hour').twitter();
// 1 hour
```

Not short enough for you?

```
moment().subtract(6, 'days').twitterShort();
// 6d
```

Jalaali Calendar

edit

```
npm install moment-jalaali                    Method Signature
```

If you want to work with Jalaali calendar system (Jalali, Persian, Khorshidi or Shamsi), you can use Behrang Noruzi Niya's plugin `moment-jalaali`.

When installed, it will wrap `moment` and moment will be able to format and parse Jalaali years and months. Here is a short example:

```
var m = moment('1360/5/26', 'jYYYY/jM/jD'); // Parse a Jala
m.format('jYYYY/jM/jD [is] YYYY/M/D'); // 1360/5/26 is 1981
```

The repository is located at github.com/behrang/moment-jalaali.

MSDate

edit

If you are using OLE Automation dates in .NET check out Markit On Demand's `moment-msdate`. Using this plugin allows you to format OA dates into JavaScript dates and vice-versa.

Convert a `moment` to an OA date:

```
moment().toOADate(); // a floating point number
```

Or, convert an OA date to a `moment`:

```
moment.fromOADate(41493); // Wed Aug 07 2013 00:00:00 GMT-0
```

More information and detailed docs can be found on GitHub at http://markitondemand.github.io/moment-msdate/.

Fiscal Quarters

edit

If you ever have need for Fiscal, Calendar or Academic quarters, you can use the moment-fquarter plugin by @robgallen.

At it's simplest, just call the fquarter method on any moment object. It returns a formatted string with April being the first quarter.

```
moment("2013-01-01").fquarter();
// Q4 2012/13
```

You can pass in any month as the starting quarter, e.g. July

```
moment("2013-01-01").fquarter(7);
// Q3 2012/13
```

If you want calendar quarters, start in January

```
moment("2013-01-01").fquarter(1);
// Q1 2013
```

Precise Range

edit

The Precise Range plugin, written by Rob Dawson, can be used to display exact, human-readable representations of date/time ranges

```
moment("2014-01-01 12:00:00").preciseDiff("2015-03-04 16:05
 // 1 year 2 months 3 days 4 hours 5 minutes 6 seconds
```

```
moment.preciseDiff("2014-01-01 12:00:00", "2014-04-20 12:00
// 3 months 19 days
```

Recur

edit

```
npm install moment-recur                    Method Signature
```

If you need to work with recurring dates, you can use Casey Trimm's

plugin `moment-recur`.

This plugin will allow you to create length-based intervals (days, weeks, etc.) and calendar-based intervals (daysOfMonth, monthsOfYear, etc.).

It provides a `matches` function to test whether a date recurs according to the rules set, as well as generator functions to get the next and previous dates in a series.

The repository, documentation, and many more examples can be found at github.com/c-trimm/moment-recur

```
var interval = moment( "01/01/2014" ).recur().every(2).days
interval.matches( "01/03/2014" ); // true
interval.next( 2, "L" ); // ["01/03/2014", "01/05/2014"]
interval.forget( "days" ); // Remove a rule
interval.dayOfMonth( 10 ); // Calendar Interval
interval.matches( "05/10/2014" ); // true
interval.previous( 2, "L" ); // ["12/10/2013", "11/10/2013"
```

Parse Date Format

edit

```
npm install moment-parseformat
```

Method Signature

This plugin extracts the format of a date/time string.

```
var format = moment.parseFormat('Thursday, February 6th, 20
// dddd, MMMM Do, YYYY h:mma
moment().format(format); // format
```

That allows to create smart date inputs that let your users set a Date/Time and lets you extract the user's preferred format for future usage. Find an example usage of it at minutes.io.

The Plugin has been authored by @gr2m. Links: Demo | Source

Java DateFormat Parser

edit

```
npm install moment-jdateformatparser          Method Signature
```

If you want to work with the `java.text.DateFormat` you can use this plugin.

For example,

```
moment("2013-12-24 14:30").formatWithJavaDateFormat("dd.MM.
moment().toJavaDateFormatString("DD.MM.YYYY");  // returns
```

The repository is located at github.com/MadMG/moment-jdateformatparser

---