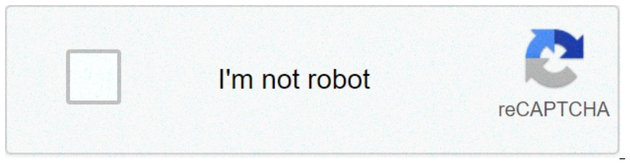

Continue

Js date now to iso string

The toISOString() method returns a string in simplified extended ISO format (ISO 8601), which is always 24 or 27 characters long (YYYY-MM-DDTHH:mm:ss.sssZ or ±YYYYYY-MM-DDTHH:mm:ss.sssZ, respectively). The timezone is always zero UTC offset, as denoted by the suffix "Z". A string representing the given date in the ISO 8601 format according to universal time.let today = new Date('05 October 2011 14:48 UTC') console.log(today.toISOString()) The above example uses parsing of a non–standard string value that may not be correctly parsed in non–Mozilla browsers.BCD tables only load in the browserSee also Reference > mongo Shell Methods > Object Constructors and MethodsDate()¶Returns a date either as a string or as a Date object.Date() returns the current date as a string in the mongo shell.new Date() returns the current date as a Date object. The mongo shell wraps the Date object with the ISODate helper. The ISODate is in UTC.You can specify a particular date by passing an ISO-8601 date string with a year within the inclusive range 0 through 9999 to the new Date() constructor or the ISODate() function. These functions accept the following formats:new Date("") returns the ISODate with the specified date.new Date("") specifies the datetime in the client's local timezone and returns the ISODate with the specified datetime in UTC.new Date("") specifies the datetime in UTC and returns the ISODate with the specified datetime in UTC.new Date() specifies the datetime as milliseconds since the UNIX epoch (Jan 1, 1970), and returns the resulting ISODate instance.Internally, Date objects are stored as a signed 64-bit integer representing the number of milliseconds since the Unix epoch (Jan 1, 1970).Not all database operations and drivers support the full 64-bit range. You may safely work with dates with years within the inclusive range 0 through 9999.If no document with _id equal to 1 exists in the products collection, the following operation inserts a document with the field dateAdded set to the current date:db.products.update({ _id: 1 }, { $set: { item: "apple" }, $setOnInsert: { dateAdded: new Date() } }, { upsert: true })To return the date as a string, use the Date() method, as in the following example:var myDateString = Date();The mongo shell wrap objects of Date type with the ISODate helper; however, the objects remain of type Date.The following example uses new Date() to return Date object with the specified UTC datetime.var myDate = new Date("2016-05-18T16:00:00Z");TipSee also: BSON Datemongo Shell DateGive Feedback← BulkWriteResult()ObjectId →On this page You can't perform that action at this time. You signed in with another tab or window. Reload to refresh your session. You signed out in another tab or window. Reload to refresh your session. Moment.js has been successfully used in millions of projects, and we are happy to have contributed to making date and time better on the web. As of September 2020, Moment gets over 12 million downloads per week! However, Moment was built for the previous era of the JavaScript ecosystem. The modern web looks much different these days. Moment has evolved somewhat over the years, but it has essentially the same design as it did when it was created in 2011. Given how many projects depend on it, we choose to prioritize stability over new features. As an example, consider that Moment objects are mutable. This is a common source of complaints about Moment. We address it in our usage guidance but it still comes as a surprise to most new users. Changing Moment to be immutable would be a breaking change for every one of the projects that use it. Creating a "Moment v3" that was immutable would be a tremendous undertaking and would make Moment a different library entirely. Since this has already been accomplished in other libraries, we feel that it is more important to retain the mutable API. Another common argument against using Moment in modern applications is its size. Moment doesn't work well with modern "tree shaking" algorithms, so it tends to increase the size of web application bundles. If one needs internationalization or time zone support, Moment can get quite large. Modern web browsers (and Node.js) expose internationalization and time zone support via the Intl object, codified as ECMA-402. Libraries like Luxon (and others) take advantage of this, reducing or removing the need to ship your own data files. Recently, Chrome Dev Tools started showing recommendations for replacing Moment for the size alone. We generally support this move. You may also want to read: The Moment team has discussed these issues at length. We recognize that many existing projects may continue to use Moment, but we would like to discourage Moment from being used in new projects going forward. Instead, we would like to recommend alternatives that are excellent choices for use in modern applications today. We would also like to promote the Temporal addition to the JavaScript language, which is looking for feedback and contributors. We now generally consider Moment to be a legacy project in maintenance mode. It is not dead, but it is indeed done. In practice, this means: We will not be adding new features or capabilities. We will not be changing Moment's API to be immutable. We will not be addressing tree shaking or bundle size issues. We will not be making any major changes (no version 3). We may choose to not fix bugs or behavioral quirks, especially if they are long-standing known issues. With specific regard to Moment's internationalization locale files: We may choose to not accept corrections to locale strings or localized date formats, especially if they have been argued successfully for their present form. You must make a new compelling argument for locale changes with significant, non-anecdotal evidence to support your position. If the string or format you are asking to change is reflected in the CLDR, then you must submit a change there first and have it accepted. However, since we understand that Moment is well established in millions of existing projects: We will address critical security concerns as they arise. We will release data updates for Moment-Timezone following IANA time zone database releases. Reasons to keep using Moment In most cases, you should not choose Moment for new projects. However there are some possible reasons you might want to keep using it. Browser support Moment works well on Internet Explorer 8 and higher. By contrast, Luxon only works on IE 10 and higher and requires a polyfill to do so. You can read more in Luxon's documentation. Other libraries have also had issues with Safari, especially on mobile devices. If you have a strong requirement to support older browsers, then you might want to stick with Moment for a bit longer. However, Day.js reports compatibility with IE8 and higher so you still may wish to consider that alternative. Dependency by other libraries Several other libraries, especially date pickers and graphing libraries, take Moment as a dependency. If you are using such a component and cannot find an alternative, then you are already including Moment in your project. Thus, it might make sense to continue using Moment throughout your project rather than including yet another date and time library. Familiarity If you are a long-time user of Moment, you may already understand its API and limitations well. If so, and the aforementioned issues are not a concern for you, then you certainly can continue to use it. edit There are several great options to consider using instead of Moment. When choosing, consider that: Some libraries are split into modules, plugins, or companion libraries. Some libraries use the ECMAScript Intl API for locales, time zones, or both. Some libraries still provide their own locale and time zone files like Moment and Moment-Timezone do. Here are the alternatives we recommend: Luxon Luxon can be thought of as the evolution of Moment. It is authored by Isaac Cambron, a long-time contributor to Moment. Please read Why does Luxon exist? and the For Moment users pages in the Luxon documentation. Locales: Intl provided Time Zones: Intl provided Day.js Day.js is designed to be a minimalist replacement for Moment.js, using a similar API. It is not a drop-in replacement, but if you are used to using Moment's API and want to get moving quickly, consider using Day.js. Locales: Custom data files that can be individually imported Time Zones: Intl provided, via a plugin date-fns Date-fns offers a series of functions for manipulating JavaScript Date objects. For more details, scroll to "Why date-fns?" on the date-fns home page. Locales: Custom data files that can be individually imported Time Zones: Intl provided, via a separate companion library js-Joda js-Joda is a JavaScript port of Java's Three-Ten Backport, which is the base for JSR-310 implementation of the Java SE 8 java.time package. If you are familiar with java.time, Joda-Time, or Noda Time, you will find js-Joda comparable. Locales: Custom data files via add-on module Time Zones: Custom data files via add-on module No Library JavaScript has always had a Date object, defined ECMAScript (ECMA-262) specification here. When using Date objects, be aware of the following: The Date object internally represents a Unix timestamp with millisecond precision. It offers functions that will convert to and from the system's local time zone, but it is always UTC internally. Unlike a Moment object, it can not be set to use another time zone; It has no concept of "mode". Using Date.parse, or new Date() has been problematic and implemented inconsistently in the past. The current specification defines parsing a variation of ISO 8601 strings, where date-only forms (like "2020-09-14") are parsed as UTC, instead of local time as they would be by ISO 8601. Even then, not all modern implementations have implemented this specification correctly (e.g., Safari). Other types of strings may work, but parsing them is implementation specific and can vary significantly - especially with older browsers. Depending on the implementation, and the components provided in the string, you may be surprised with the result. For these reasons, we agree with MDN's statement that parsing strings with the Date object is strongly discouraged. Modern JavaScript environments will also implement the by ECMA-402 specification, which provides the Intl object, and defines behavioral options of the Date object's toLocaleString, toLocaleDateString, and toLocaleTimeString functions. When using the Intl object, be aware of the following: Not every environment will implement the full specification. In particular, Node.js environments require internationalization support provided by ICU. See the Node.js documentation for further details. The ECMAScript Intl compatibility table (by kangax) can be useful in determining which features are supported and which are not. Most newer environments provide IANA time zone support via the timeZone option in the Intl.DateTimeFormat constructor (and in Date.toLocaleString, Date.toLocaleDateString, and Date.toLocaleTimeString). This option can be used to take the internal UTC-based timestamp of a Date object and get a string that has been converted to a named time zone. However, it can not be used to convert a Date object to a different time zone. If the Date and Intl objects meet your needs and you fully understand their limitations, then you might consider using them directly. edit One day soon, we hope there won't be a strong need for date and time libraries in JavaScript at all. Instead, we will be able to use capabilities of the JavaScript language itself. Though some capabilities are here today with Date and Intl, we know from experience and data that there is significant room for improvement. The effort to make better date and time APIs in the JavaScript language is being done via The ECMA TC39 Temporal Proposal. It is currently at Stage 3 of the TC39 process. Temporal will be a new global object that acts as a top-level namespace (like Math). It exposes many separate types of objects including Temporal.Instant, Temporal.ZonedDateTime, Temporal.PlainDateTime, Temporal.PlainDate, Temporal.PlainTime, Temporal.TimeZone and several others. The Temporal Cookbook shows many "recipes" with examples of how these objects can be used in different scenarios. You can try out Temporal today, via a non-production polyfill. Please give it a try, but don't use it in production (yet)! Please provide feedback, and consider contributing to this effort - especially if you have experience using Moment or other date and time libraries! Moment was designed to work both in the browser and in Node.js. All code should work in both of these environments, and all unit tests are run in both of these environments. Currently, the following browsers are used for the ci system: Chrome on Windows XP, IE 8, 9, and 10 on Windows 7, IE 11 on Windows 10, latest Firefox on Linux, and latest Safari on OSX 10.8 and 10.11. If you want to try the sample codes below, just open your browser's console and enter them. edit npm install moment var moment = require('moment'); // require moment().format(); Or in ES6 syntax: import moment from 'moment'; moment().format(); Note: if you want to work with a particular variation of moment timezone, for example using only data from 2012-2022, you will need to import it from the builds directory like so: import moment from 'moment-timezone/builds/moment-timezone-with-data-2012-2022'; Note: In 2.4.0, the globally exported moment object was deprecated. It will be removed in next major release. edit moment().format(); Moment.js is available on cdnjs.com and on jsDelivr. edit bower bower install --save moment Notable files are moment.js, locale/*.js and min/moment-with-locales.js. edit We strongly recommend reading this if you plan to use moment with Require.js. Also upgrade to 2.14.0 or above for best experience. As a start, you might have acquired moment through bower or node_modules or anything else that places moment.js together with a locales directory in a base folder. Then you should use a tool like adapt-pkg-main, or manually -- using packages config. requirejs.config({ packages: [{ name: 'moment', // This location is relative to baseUrl. Choose bower_components // or node_modules, depending on how moment was installed. location: '[bower_components|node_modules]/moment', main: 'moment' }] }); With the above setup, you can require the core with moment and de locale with moment/locale/de. // only needing core define(['moment'], function (moment) { console.log(moment().format('LLLL')); // 'Friday, June 24, 2016 1:42 AM' }); // core with single locale define(['moment', 'moment/locale/de'], function (moment) { moment.locale('de'); console.log(moment().format('LLLL')); // 'Freitag, 24. Juni 2016 01:42' }); // core with all locales define(['moment/min/moment-with-locales'], function (moment) { moment.locale('de'); console.log(moment().format('LLLL')); // 'Freitag, 24. Juni 2016 01:42' }); // async load locale define(['require', 'moment'], function(require, moment) { // Inside some module after the locale is detected. This is the // case where the locale is not known before module load time. require(['moment/locale/de'], function(localeModule) { // here the locale is loaded, but not yet in use console.log(moment().format('LLLL')); // 'Friday, June 24, 2016 1:42 AM' moment.locale('de'); // Use moment now that the locale has been properly set. console.log(moment().format('LLLL')); // 'Freitag, 24. Juni 2016 01:42' }) }); For more complicated use cases please read excellent explanation by @jrburke. 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. edit NuGet / Moment.js Install-Package Moment.js edit meteor / atmosphere / momentjs:moment meteor add momentjs:moment edit npm install moment var moment = require('moment'); moment().format(); Note: There is a bug that prevents moment.locale from being loaded. var moment = require('moment'); moment.locale('cs'); console.log(moment.locale()); // en Use the workaround below var moment = require('moment'); require('moment/locale/cs'); console.log(moment.locale()); // cs In order to include all the locales var moment = require('moment'); require("moment/min/locales.min"); moment.locale('cs'); console.log(moment.locale()); // cs edit npm install moment var moment = require('moment'); moment().format(); Note: By default, webpack bundles all Moment.js locales (in Moment.js 2.18.1, that's 160 minified KBs). To strip unnecessary locales and bundle only the used ones, add moment-locales-webpack-plugin: // webpack.config.js const MomentLocalesPlugin = require('moment-locales-webpack-plugin'); module.exports = { plugins: [// To strip all locales except "en" new MomentLocalesPlugin(), // Or: To strip all locales except "en", "es-us" and "ru" // ("en" is built into Moment and can't be removed) new MomentLocalesPlugin({ localesToKeep: ['es-us', 'ru'], }),], }; There are other resources to optimize Moment.js with webpack, for example this one. edit As of version 2.13.0, Moment includes a typescript definition file. Install via NPM npm install moment Import and use in your Typescript file const moment = require('moment'); let now = moment().format('LLLL'); Note: If you have trouble importing moment For Typescript 2.x try adding "moduleResolution": "node" in compilerOptions in your tsconfig.json file For Typescript 1.x try adding "allowSyntheticDefaultImports": true in compilerOptions in your tsconfig.json file and then use the syntax import moment from 'moment'; Locale Import To use moment.locale you first need to import the language you are targeting. import * as moment from 'moment'; import 'moment/locale/pt-br'; console.log(moment.locale()); // en moment.locale('fr'); console.log(moment.locale()); // fr moment.locale('pt-br'); console.log(moment.locale()); // pt-br edit To load moment, place it in the path specified by your System.config in the baseURL configuration. Then import it into your page. System.config({ baseURL: '/app' }); System.import('moment.js'); If you need moment to be loaded as global, you can do this with the meta configuration: System.config({ meta: { 'moment': { format: 'global' } } }); Alternatively, to provide Moment as a global to only a specific dependency, you can do this: System.config({ meta: { 'path/to/global-file.js': { globals: { moment: 'moment' } } } }); edit edit If you are having any troubles, the first place to check is the guides. If you don't find what you are looking for there, try asking a question on Stack Overflow with the momentjs tag. Note: More than half of the issues seen on Stack Overflow can be answered by this blog post. You can also use the GitHub issue tracker to find related issues or open a new issue. In addition, Moment has a Gitter where the internal team is frequently available. For general troubleshooting help, Stack Overflow is the preferred forum. Moment's maintainers are very active on Stack Overflow, as are several other knowledgeable users. The fastest response will be there. 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. Please read: moment(...) is local mode. Ambiguous input (without offset) is assumed to be local time. Unambiguous input (with offset) is adjusted to local time. moment.utc(...) is utc mode. Ambiguous input is assumed to be UTC. Unambiguous input is adjusted to UTC. moment.parseZone() keep the input zone passed in. Ambiguous input is assumed to be UTC. moment.tz(...) with the moment-timezone plugin can parse input in a specific time zone. Keep in mind that a time zone and a time zone offset are two different things. An offset of -08:00 doesn't necessarily mean you are in the US Pacific time zone. See the Parsing Guide for additional information. edit moment(); moment(undefined); // From 2.14.0 onward, also supported moment([]); moment({}); 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()). Note: From version 2.14.0, moment([]) and moment({}) also return now. They used to default to start-of-today before 2.14.0, but that was arbitrary so it was changed. Note: Function parameters default to undefined when not passed in. Moment treats moment(undefined) as moment(). Note: Moments are created at evaluation time, so moment().diff(moment()) may not always return 0. See this GitHub issue for more details. edit When creating a moment from a string, we first check if the string matches known ISO 8601 formats, we then check if the string matches the RFC 2822 Date time format before dropping to the fall back of 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 2013-02 # A month date part 2013-W06-5 # A week date part 2013-039 # An ordinal date part 20130208 # Basic (short) full date 201303 # Basic (short) year+month 2013 # Basic (short) year only 2013W065 # Basic (short) week, weekday 2013W06 # Basic (short) week only 2013050 # Basic (short) ordinal date (year + day-of-year) A time part can also be included, separated from the date part by a space or an uppercase T. 2013-02-08T09 # An hour time part separated by a T 2013-02-08 09 # An hour time part separated by a space 2013-02-08 09:30 # An hour and minute time part 2013-02-08 09:30:26 # An hour, minute, and second time part 2013-02-08 09:30:26.123 # An hour, minute, second, and millisecond time part 2013-02-08 24:00:00.000 # hour 24, minute, second, millisecond equal 0 means next day at midnight 20130208T080910,123 # Short date and time up to ms, separated by comma 20130208T080910.123 # Short date and time up to ms 20130208T080910 # Short date and time up to seconds 20130208T0809 # Short date and time up to minutes 20130208T08 # Short date and time, hours only 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, +-HH 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 2013-02-08 09:30:26.123+07 # +-HH Note: 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 The RFC 2822 date time format Before parsing a RFC 2822 date time the string is cleansed to remove any comments and/or newline characters. The additional characters are legal in the format but add nothing to creating a valid moment instance. After cleansing, the string is validated in the following space-separated sections, all using the English language: 6 Mar 17 21:22 UT 6 Mar 17 21:22:23 UT 6 Mar 2017 21:22:23 GMT 06 Mar 2017 21:22:23 Z Mon 06 Mar 2017 21:22:23 z Mon, 06 Mar 2017 21:22:23 +0000 Day of Week in three letters, followed by an optional comma. (optional) Day of Month (1 or 2 digit), followed by a three-letter month and 2 or 4 digit year Two-digit hours and minutes separated by a colon (:), followed optionally by another colon and seconds in 2-digits Timezone or offset in one of the following formats: UT : +0000 GMT : +0000 EST | CST | MST | PST | EDT | CDT | MDT | PDT : US time zones* A - I | K - Z : Military time zones* Time offset +/-9999 [*] See section 4.3 of the specification for details. The parser also confirms that the day-of-week (when included) is consistent with the date. edit moment(String, String); moment(String, String, String); 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 by default, so both of the following will return the same thing. moment("12-25-1995", "MM-DD-YYYY"); moment("12/25/1995", "MM-DD-YYYY"); You may get unexpected results when parsing both date and time. The below example may not parse as you expect: moment('24/12/2019 09:15:00', "DD MM YYYY hh:mm:ss"); You can use strict mode, which will identify the parsing error and set the Moment object as invalid: moment('24/12/2019 09:15:00', "DD MM YYYY hh:mm:ss", true); The parsing tokens are similar to the formatting tokens used in moment#format. Year, month, and day tokens Tokens are case-sensitive. InputExampleDescription YYYY20144 or 2 digit year. Note: Only 4 digit can be parsed on strict mode YY142 digit year Y-25Year with any number of digits and sign Q1..4Quarter of year. Sets month to first month in quarter. M MM1..12Month number MMM MMMMJan..DecemberMonth name in locale set by moment.locale() D DD1..31Day of month Do1st..31stDay of month with ordinal DDD DDDD1..365Day of year X1410715640.579Unix timestamp x1410715640579Unix ms timestamp YYYY from version 2.10.5 supports 2 digit years, and converts them to a year near 2000 (same as YY). Y was added in 2.11.1. It will match any number, signed or unsigned. It is useful for years that are not 4 digits or are before the common era. It can be used for any year. 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. Tokens are case-sensitive. InputExampleDescription gggg2014Locale 4 digit week year gg14Locale 2 digit week year w ww1..53Locale week of year e0..6Locale day of week ddd ddddMon...SundayDay name in locale set by moment.locale() GGGG2014ISO 4 digit week year GG14ISO 2 digit week year W WW1..53ISO week of year E1..7ISO day of week Locale aware formats 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. Tokens are case-sensitive. InputExampleDescription L09/04/1986Date (in local format) LLSeptember 4 1986Month name, day of month, year LLLSeptember 4 1986 8:30 PMMonth name, day of month, year, time LLLLThursday, September 4 1986 8:30 PMDay of week, month name, day of month, year, time LT8:30 PMTime (without seconds) LTS8:30:00 PMTime (with seconds) Hour, minute, second, millisecond, and offset tokens Tokens are case-sensitive. InputExampleDescription H HH0..23Hours (24 hour time) h hh1..12Hours (12 hour time used with a A.) k kk1..24Hours (24 hour time from 1 to 24) a Aam pmPost or ante meridiem (Note the one character a p are also considered valid) m mm0..59Minutes s ss0..59Seconds S SS SSS ... SSSSSSSSS0..999999999Fractional seconds Z ZZ+12:00Offset from UTC as +-HH:mm, +-HHmm, or Z From version 2.10.5: fractional second tokens length 4 up to 9 can parse any number of digits, but will only consider the top 3 (milliseconds). Use if you have the time printed with many fractional digits and want to consume the input. Note that the number of S characters provided is only relevant when parsing in strict mode. In standard mode, S, SS, SSS, SSSS are all equivalent, and interpreted as fractions of a second. For example, .12 is always 120 milliseconds, passing SS will not cause it to be interpreted as 12 milliseconds. 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. SSSSS ... SSSSSSSSS were added in version 2.10.5. k kk were added in version 2.13.0. Unless you specify a time zone offset, parsing a string will create a date in the current time zone. moment("2010-10-20 4:30", "YYYY-MM-DD HH:mm"); // parsed as 4:30 local time moment("2010-10-20 4:30 +0000", "YYYY-MM-DD HH:mm Z"); // parsed as 4:30 UTC Era Year related tokens Tokens are case-sensitive. InputExamplesDescription y .. yyyy5 +5 -500Years yo5th 1stOrdinal Years NADAbbr Era name NNADAbbr Era name NNNADAbbr Era name NNNNAnno DominiFull Era name NNNNNADNarrow Era name Era support was added in 2.25.0. The tokens/API are still in flux. Notes and gotchas If the moment that results from the parsed input does not exist, moment#isValid will return false. moment("2010 13", "YYYY MM").isValid(); // false (not a real month) moment("2010 11 31", "YYYY MM DD").isValid(); // false (not a real day) moment("2010 2 29", "YYYY MM DD").isValid(); // false (not a leap year) moment("2010 notamonth 29", "YYYY MMM DD").isValid(); // false (not a real month name) 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('2012 July', 'YYYY MMM', ['qj', 'en']); Moment's parser is very forgiving, and this can lead to undesired/unexpected behavior. For example, the following behavior can be observed: moment('2016 is a date', 'YYYY-MM-DD').isValid() //true, 2016 was matched Previous to 2.13.0 the parser exhibited the following behavior. This has been corrected. moment('I am spartacus', 'h:hh A').isValid(); //true - the 'am' matches the 'A' flag. 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, including delimeters. moment('It is 2012-05-25', 'YYYY-MM-DD').isValid(); // true moment('It is 2012-05-25', 'YYYY-MM-DD', true).isValid(); // false moment('2012-05-25', 'YYYY-MM-DD', true).isValid(); // true moment('2012.05.25', 'YYYY-MM-DD', true).isValid(); // false You can use both locale and strictness. moment('2012-10-14', 'YYYY-MM-DD', 'fr', true); Strict parsing is frequently the best parsing option. For more information about choosing strict vs forgiving parsing, see the parsing guide. Parsing two digit years By default, two digit years above 68 are assumed to be in the 1900's and years 68 or below are assumed to be in the 2000's. This can be changed by replacing the moment.parseTwoDigitYear method. The only argument of this method is a string containing the two years input by the user, and should return the year as an integer. moment.parseTwoDigitYear = function(yearString) { return parseInt(yearString) + 2000; } Parsing glued hour and minutes From version 2.11.0 parsing hmm, Hmm, hmmss and Hmmss is supported: moment("123", "hmm").format("HH:mm") === "01:23" moment("1234", "hmm").format("HH:mm") === "12:34" edit moment(String, String[], String, Boolean); 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"]); // uses the last format moment("05-06-1995", ["MM-DD-YYYY", "DD-MM-YYYY"]); // uses the first format You may also specify a locale and strictness argument. They work the same as the single format case. moment("29-06-1995", ["MM-DD-YYYY", "DD-MM-YYYY"], 'fr'); // uses 'fr' locale moment("29-06-1995", ["MM-DD-YYYY", "DD-MM-YYYY"], true); // uses strict parsing moment("05-06-1995", ["MM-DD-YYYY", "DD-MM-YYYY"], 'fr', true); // uses 'fr' locale and strict parsing 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. edit moment(String, moment.CUSTOM_FORMAT, [String], [Boolean]); moment(String, moment.HTML5_FMT.DATETIME_LOCAL, [String], [Boolean]); // from 2.20.0 moment(String, [..., moment.ISO_8601, ...], [String], [Boolean]); 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. moment("2010-01-01T05:06:07", moment.ISO_8601); moment("2010-01-01T05:06:07", ["YYYY", moment.ISO_8601]); As of version 2.20.0, the following HTML5 formats are available as constants in the moment object's HTML5_FMT property (moment.HTML5_FMT.*): ConstantFormatExampleInput Type DATETIME_LOCALYYYY-MM-DDTHH:mm2017-12-14T16:34 DATETIME_LOCAL_SECONDSYYYY-MM-DDTHH:mm:ss2017-12-14T16:34:10 DATETIME_LOCAL_MSYYYY-MM-DDTHH:mm:ss.SSS2017-12-14T16:34:10.234 DATEYYYY-MM-DD2017-12-14 TIMEHH:mm16:34 TIME_SECONDSHH:mm:ss16:34:10 TIME_MSHH:mm:ss.SSS16:34:10.234 WEEKGGGG-[W]WW2017-W50 MONTHYYYY-MM2017-12 edit moment({unit: value, ...}); moment({ hour:15, minute:10 }); moment({ y :2010, M :3, d :5, h :15, m :10, s :3, ms :123}); moment({ year :2010, month :3, day :5, hour :15, minute :10, second :3, millisecond :123}); moment({ years:2010, months:3, days:5, hours:15, minutes:10, seconds:3, milliseconds:123}); moment({ years:2010, months:3, date:5, hours:15, minutes:10, seconds:3, milliseconds:123}); moment({ years:'2010', months:'3', date:'5', hours:'15', minutes:'10', seconds:'3', milliseconds:'123'}); // from 2.11.0 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. String values (as shown on the last line) are supported from version 2.11.0. Note that like moment(Array) and new Date(year, month, date), months are 0 indexed. edit 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); Note: ECMAScript calls this a "Time Value" edit 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); Note: Despite Unix timestamps being UTC-based, this function creates a moment object in local mode. If you need UTC, then subsequently call .utc(), as in: var day = moment.unix(1318781876).utc(); edit 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 the Date object; further changes to the Date won't affect the Moment, and vice-versa. edit 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, 3:25:50.125 PM 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 time zone. 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, 12]).isValid(); // false (not a real month) moment([2010, 10, 31]).isValid(); // false (not a real day) moment([2010, 1, 29]).isValid(); // false (not a leap year) edit Microsoft Web API returns JSON dates in proper ISO-8601 format by default, but older ASP.NET technologies may return 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)/"); // 2007-12-28T23:11:57.056-07:00 edit All moments are mutable. If you want a clone of a moment, you can do so implicitly or explicitly. 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 edit moment.utc(); moment.utc(Number); moment.utc(Number[]); moment.utc(String); moment.utc(String, String); moment.utc(String, String[]); moment.utc(String, String, String); moment.utc(String, String, String[]); moment.utc(String, String, Boolean); moment.utc(String, String, String, Boolean); 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).valueOf() === 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 edit moment.parseZone() moment.parseZone(String) moment.parseZone(String, String) moment.parseZone(String, [String]) moment.parseZone(String, String, Boolean) moment.parseZone(String, String, String, Boolean) Moment's string parsing functions like moment(string) and moment.utc(string) accept offset information if provided, but convert the resulting Moment object to local or UTC time. In contrast, moment.parseZone() parses the string but keeps the resulting Moment object in a fixed-offset timezone with the provided offset in the string. moment.parseZone("2013-01-01T00:00:00-13:00").utcOffset(); // -780 ("-13:00" in total minutes) moment.parseZone('2013 01 01 05 -13:00', 'YYYY MM DD HH ZZ').utcOffset(); // -780 ("-13:00" in total minutes) moment.parseZone('2013-01-01-13:00', ['DD MM YYYY ZZ', 'YYYY MM DD ZZ']).utcOffset(); // -780 ("-13:00" in total minutes); It also allows you to pass locale and strictness arguments. moment.parseZone("2013 01 01 -13:00", 'YYYY MM DD ZZ', true).utcOffset(); // -780 ("-13:00" in total minutes) moment.parseZone("2013-01-01-13:00", 'YYYY MM DD ZZ', true).utcOffset(); // NaN (doesn't pass the strictness check) moment.parseZone("2013 01 01 -13:00", 'YYYY MM DD ZZ', 'fr', true).utcOffset(); // -780 (with locale and strictness argument) moment.parseZone("2013 01 01 -13:00", ['DD MM YYYY ZZ', 'YYYY MM DD ZZ'], 'fr', true).utcOffset(); // -780 (with locale and strictness argument alongside an array of formats) moment.parseZone is equivalent to parsing the string and using moment#utcOffset to parse the zone. var s = "2013-01-01T00:00:00-13:00"; moment(s).utcOffset(s); edit Moment applies stricter initialization rules than the Date constructor. new Date(2013, 25, 14).toString(); // "Sat Feb 14 2015 00:00:00 GMT-0500 (EST)" 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. In addition to the above, As of 2.13.0 the meridiem and parsedDateParts flags work together to determine date validity. meridiem: Indicates what meridiem (AM/PM) was parsed, if any. String. parsedDateParts: Returns an array of date parts parsed in descending order - i.e. parsedDateParts[0] === year. If no parts are present, but meridiem has value, date is invalid. Array. 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. Note: Validity is determined on moment creation. A modified moment (i.e. moment().hour(NaN)) will remain valid. 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: years months days hours minutes seconds milliseconds Note: In case of multiple wrong units the first one is returned (because days validity may depend on month, for example). If a moment is invalid, it behaves like a NaN in floating point operations. All of the following produce invalid moments: invalid.add(unit, value) another.add(invalid) invalid.clone() invalid.diff(another) invalid.endOf(unit) invalid.max(another) another.max(invalid) invalid.min(another) another.min(invalid) invalid.set(unit, value) invalid.startOf(unit) invalid.subtract(unit, value) The following produce a localized version of 'InvalidDate': invalid.format(anyFmt) results in 'Invalid Date' in the current locale invalid.from(another) another.from(invalid) invalid.fromNow(suffix) invalid.to(another) another.to(invalid) invalid.toNow(suffix) invalid.toISOString() (Before 2.18.0) invalid.toString() The following return false: invalid.isAfter(another) invalid.isAfter(invalid) another.isAfter(invalid) invalid.isBefore(another) invalid.isBefore(invalid) another.isBefore(invalid) invalid.isBetween(another, another) invalid.isBetween(invalid, invalid) invalid.isSame(another) invalid.isSame(invalid) another.isSame(invalid) invalid.isSameOrAfter(another) invalid.isSameOrAfter(invalid) another.isSameOrAfter(invalid) invalid.isSameOrBefore(another) invalid.isSameOrBefore(invalid) another.isSameOrBefore(invalid) And these return null or NaN with some structure: invalid.get(unit) returns null, as all other named getters invalid.toArray() === [NaN, NaN, NaN, NaN, NaN, NaN] invalid.toObject() has all values set to NaN invalid.toDate() returns an invalid Date object invalid.toJSON() returns null invalid.unix() returns null invalid.valueOf() returns null invalid.toISOString() returns null (As of 2.18.0) edit After a moment object is created, all of the inputs can be accessed with creationData() method: moment("2013-01-02", "YYYY-MM-DD", true).creationData() === { input: "2013-01-02", format: "YYYY-MM-DD", locale: Locale obj, isUTC: false, strict: true } edit 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, 5:10.20.000 moment({hour: 5, minute: 10, seconds: 20, milliseconds: 300}); // today, 5:10.20.300 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 day-of-month, 05:06:07.000 Defaulting to this year, if year is not specified: moment(3, "MM"); // this year, 3rd month (March) moment("Apr 4 05:06:07", "MMM DD hh:mm:ss"); // this year, 4th April, 05:06:07.000 Moment.js uses overloaded getters and setters. You may be familiar with this pattern from its 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).valueOf() === 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).valueOf() === 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. Note: From 2.19.0 passing NaN to any setter is a no-op. Before 2.19.0 it was invalidating the moment in a wrong way. edit moment().millisecond(Number); 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. edit moment().second(Number); 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. edit moment().minute(Number); 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 hour. edit moment().hour(Number); 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. edit moment().date(Number); 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. Note: if you chain multiple actions to construct a date, you should start from a year, then a month, then a day etc. Otherwise you may get unexpected results, like when day=31 and current month has only 30 days (the same applies to native JavaScript Date manipulation), the returned date will be the 30th of the current month (see month for more details). Bad: moment().date(day).month(month).year(year) Good: moment().year(year).month(month).date(day) 2.16.0 deprecated using moment().dates(). Use moment().date() instead. edit moment().day(Number|String); 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 value given is from 0 to 6, the resulting date will be within the current (Sunday-to-Saturday) week. If the range is exceeded, it will bubble up to other weeks. moment().day(-7); // last Sunday (0 - 7) moment().day(0); // this Sunday (0) 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"); edit moment().weekday(Number); 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 edit moment().isoWeekday(Number); moment().isoWeekday(); // Number Gets or sets the ISO day of the week with 1 being Monday and 7 being Sunday. As with moment#day, if the range is exceeded, it will bubble up to other weeks. moment().isoWeekday(1); // Monday moment().isoWeekday(7); // Sunday A day name is also supported. This is parsed in the moment's current locale. moment().isoWeekday("Sunday"); moment().isoWeekday("Monday"); edit moment().dayOfYear(Number); 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. edit moment().week(Number); 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. edit moment().isoWeek(Number); 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. edit moment().month(Number|String); 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"); // 2012-03-02 // after 2.1.0 moment([2012, 0, 31]).month(1).format("YYYY-MM-DD"); // 2012-02-29 2.16.0 deprecated using moment().months(). Use moment().month() instead. edit moment().quarter(); // Number moment().quarter(Number); moment().quarters(); // Number moment().quarters(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() // 1 moment('2013-04-01T00:00:00.000').quarter() // 2 moment('2013-07-01T00:00:00.000').subtract(1, 'ms').quarter() // 2 moment('2013-07-01T00:00:00.000').quarter() // 3 moment('2013-10-01T00:00:00.000').subtract(1, 'ms').quarter() // 3 moment('2013-10-01T00:00:00.000').quarter() // 4 moment('2014-01-01T00:00:00.000').subtract(1, 'ms').quarter() // 4 Sets the quarter (1 to 4). moment('2013-01-01T00:00:00.000').quarter(2) // '2013-04-01T00:00:00.000' moment('2013-02-05T05:06:07.000').quarter(2).format() // '2013-05-05T05:06:07-07:00' edit moment().year(Number); moment().year(); // Number moment().years(Number); moment().years(); // Number Gets or sets the year. Accepts numbers from -270,000 to 270,000. 2.6.0 deprecated using moment().years(). Use moment().year() instead. edit moment().weekYear(Number); 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. edit moment().isoWeekYear(Number); moment().isoWeekYear(); // Number Gets or sets the ISO week-year. edit Gets the number of weeks according to locale in the current moment's year. edit moment().isoWeeksInYear(); Gets the number of weeks in the current moment's year, according to ISO weeks. edit moment().get('year'); 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). edit moment().set(String, Int); 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). Object parsing was added in 2.9.0 edit moment.max(Moment[,Moment...]); moment.max(Moment[]); 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 var friends = fetchFriends(); /* [{name: 'Dan', birthday: '11.12.1977'}, {name: 'Mary', birthday: '11.12.1986'}, {name: 'Stephan', birthday: '11.01.1993'}]*/ var friendsBirthDays = friends.map(function(friend){ return moment(friend.birthday, 'DD.MM.YYYY'); }); moment.max(friendsBirthDays); // '11.01.1993' With no arguments the function returns a moment instance with the current time. From version 2.10.5, if an invalid moment is one of the arguments, the result is an invalid moment. moment.max(moment(), moment.invalid()).isValid() === false moment.max(moment.invalid(), moment()).isValid() === false moment.max([moment(), moment.invalid()]).isValid() === false moment.max([moment.invalid(), moment()]).isValid() === false edit moment.min(Moment[,Moment...]); moment.min(Moment[]); 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 moment.min([a, b]); // a With no arguments the function returns a moment instance with the current time. From version 2.10.5, if an invalid moment is one of the arguments, the result is an invalid moment. moment.min(moment(), moment.invalid()).isValid() === false moment.min(moment.invalid(), moment()).isValid() === false moment.min([moment(), moment.invalid()]).isValid() === false moment.min([moment.invalid(), moment()]).isValid() === false 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).hours(0).minutes(0).seconds(0); 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. edit moment().add(Number, String); 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 milliseconds 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 saving time. If you are adding years, months, weeks, or days, the original hour will always match the added hour. Adding a month will add the specified number of months to the date. moment([2010, 1, 28]); // February 28 moment([2010, 1, 28]).add(1, 'month'); // March 28 var m = moment(new Date(2011, 2, 12, 5, 0, 0)); // the day before DST in the US 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 before DST in the US m.hours(); // 5 m.add(24, 'hours').hours(); // 6 (but you may have to set the timezone first) 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'); As of 2.12.0 when decimal values are passed for days and months, they are rounded to the nearest integer. Weeks, quarters, and years are converted to days or months, and then rounded to the nearest integer. moment().add(1.5, 'months') == moment().add(2, 'months') moment().add(.7, 'years') == moment().add(8, 'months') //.7*12 = 8.4, rounded to 8 edit moment().subtract(Number, String); 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'); As of 2.12.0 when decimal values are passed for days and months, they are rounded to the nearest integer. Weeks, quarters, and years are converted to days or months, and then rounded to the nearest integer. moment().subtract(1.5, 'months') == moment().subtract(2, 'months') moment().subtract(.7, 'years') == moment().subtract(8, 'months') //.7*12 = 8.4, rounded to 8 Note that in order to make the operations moment.add(-.5, 'days') and moment.subtract(.5, 'days') equivalent, -.5, -1.5, -2.5, etc are rounded down. edit moment().startOf(String); Mutates the original moment by setting it to the start of a unit of time. moment().startOf('year'); // set to January 1st, 12:00 am this year moment().startOf('month'); // set to the first of this month, 12:00 am moment().startOf('quarter'); // set to the beginning of the current quarter, 1st day of months, 12:00 am moment().startOf('week'); // set to the first day of this week, 12:00 am moment().startOf('isoWeek'); // set to the first day of this week according to ISO 8601, 12:00 am moment().startOf('day'); // set to 12:00 am today moment().startOf('date'); // set to 12:00 am today moment().startOf('hour'); // set to now, but with 0 mins, 0 secs, and 0 ms moment().startOf('minute'); // set to now, but with 0 seconds and 0 milliseconds moment().startOf('second'); // same as moment().milliseconds(0); These shortcuts are essentially the same as the following. moment().startOf('year'); moment().month(0).date(1).hours(0).minutes(0).seconds(0).milliseconds(0); 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. Note: moment#startOf('date') was added as an alias for day in 2.13.0 edit 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 23:59:59.999 this year 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. edit moment().max(Moment|String|Number|Date|Array); 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)); edit moment().min(Moment|String|Number|Date|Array); 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); edit moment().local(); moment().local(Boolean); // from 2.8.0 Sets a flag on the original moment to use local time to display a moment

instead of the original moment's time. var a = moment.utc([2011, 0, 1, 8]); a.hours(); // 8 UTC a.local(); a.hours(); // 0 PST Local can also be used to convert out of a fixed offset mode: moment.parseZone('2016-05-03T22:15:01+02:00').local().format(); // "2016-05-03T15:15:01-05:00" Passing true will change the time zone without changing the current time. moment.parseZone('2016-05-03T22:15:01+02:00').local(true).format(); //"2016-05-03T22:15:01-05:00" See moment.utc() for more information on UTC mode. edit moment().utc(); moment().utc(Boolean); // from 2.8.0 Sets a flag on the original moment to use UTC to display a moment instead of the original moment's time. var a = moment([2011, 0, 1, 8]); a.hours(); // 8 PST a.utc(); a.hours(); // 16 UTC UTC can also be used to convert out of a fixed offset mode: moment.parseZone('2016-05-03T22:15:01+02:00').utc().format(); //"2016-05-03T20:15:01Z" Passing true will change the time zone without changing the current time. moment.parseZone('2016-05-03T22:15:01+02:00').utc(true).format(); //"2016-05-03T22:15:01Z" See moment.utc() for more information on UTC mode. edit moment().utcOffset(); moment().utcOffset(Number|String); moment().utcOffset(Number|String, Boolean); Get or set 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, etc.) Setting the UTC offset by supplying minutes. The offset is set on the moment object that utcOffset() is called on. If you are wanting to set the offset globally, try using moment-timezone. Note that once you set an offset, it's fixed and won't change on its own (i.e there are no DST rules). If you want an actual time zone -- 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 last match of +00 -00 +00:00 +0000 -00:00 -0000 Z, so you can even pass an ISO8601 formatted string with offset and the moment will be changed to that UTC offset. Note that if the string does not include 'Z', it must include the + or - character. moment().utcOffset("2013-03-07T07:00:00+08:00"); The utcOffset function has an optional second parameter which accepts a boolean value indicating whether to keep the existing time of day. Passing false (the default) will keep the same instant in Universal Time, but the local time will change. Passing true will keep the same local time, but at the expense of choosing a different point in Universal Time. One use of this feature is if you want to construct a moment with a specific time zone offset using only numeric input values: moment([2016, 0, 1, 0, 0, 0]).utcOffset(-5, true) // Equivalent to "2016-01-01T00:00:00-05:00" edit moment().zone(); moment().zone(Number|String); Note: This function has been deprecated in 2.9.0. Consider moment.fn.utcOffset instead. Get the time zone 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"); Once parsing and manipulation are done, you need some way to display the moment. edit moment().format(); 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(); // "2014-09-08T08:02:17-05:00" (ISO 8601, no fractional seconds) moment().format("dddd, MMMM Do YYYY, h:mm:ss a"); // "Sunday, February 14th 2010, 3:25:50 pm" moment().format("ddd, hA"); // "Sun, 3PM" moment().format("[Today is] dddd"); // "Today is Sunday" moment('gibberish').format('YYYY MM DD'); // "Invalid date" 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 Qo 1st 2nd 3rd 4th 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 YYYYYY -001970 -001971 ... +001907 +001971 Note: Expanded Years (Covering the full time value range of approximately 273,790 years forward or backward from 01 January, 1970) Y 1970 1971 ... 9999 +10000 +10001 Note: This complies with the ISO 8601 standard for dates past the year 9999 Era Year y 1 2 ... 2020 ... Era N, NN, NNN BC AD Note: Abbr era name NNNN Before Christ, Anno Domini Note: Full era name NNNNN BC AD Note: Narrow era name 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 k 1 2 ... 23 24 kk 01 02 ... 23 24 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 00 01 ... 98 99 SSS 000 001 ... 998 999 SSSS ... SSSSSSSSS 000[0..] 001[0..] ... 998[0..] 999[0..] Time Zone z or zz EST CST ... MST PST Note: as of 1.6.0, the z/zz format tokens have been deprecated from plain moment objects. Read more about it here. However, they *do* work if you are using a specific time zone with the moment-timezone addon. Z -07:00 -06:00 ... +06:00 +07:00 ZZ -0700 -0600 ... +0600 +0700 Unix Timestamp X 1360013296 Unix Millisecond Timestamp x 1360013296123 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. SSSS to SSSSSSSSS were added in 2.10.5. They display 3 significant digits and the rest is filled with zeros. k and kk were added in 2.13.0. 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 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 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. As of version 2.13.0, when in UTC mode, the default format is governed by moment.defaultFormatUtc which is in the format YYYY-MM-DDTHH:mm:ss[Z]. This returns Z as the offset, instead of +00:00. In certain instances, a local timezone (such as Atlantic/Reykjavik) may have a zero offset, and will be considered to be UTC. In such cases, it may be useful to set moment.defaultFormat and moment.defaultFormatUtc to use the same formatting. Changing the value of moment.defaultFormat will only affect formatting, and will not affect parsing. for example: moment.defaultFormat = "DD.MM.YYYY HH:mm"; // parse with .toDate() moment('20.07.2018 09:19').toDate() // Invalid date // format the date string with the new defaultFormat then parse moment('20.07.2018 09:19', moment.defaultFormat).toDate() // Fri Jul 20 2018 09:19:00 GMT+0300 edit moment().fromNow(); 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. Time is rounded to the nearest second. The breakdown of which string is displayed for each length of time is outlined in the table below. Range Key Sample Output 0 to 44 seconds s a few seconds ago unset ss 44 seconds ago 45 to 89 seconds m a minute ago 90 seconds to 44 minutes mm 2 minutes ago ... 44 minutes ago 45 to 89 minutes h an hour ago 90 minutes to 21 hours hh 2 hours ago ... 21 hours ago 22 to 35 hours d a day ago 36 hours to 25 days dd 2 days ago ... 25 days ago 26 to 45 days M a month ago 45 to 319 days MM 2 months ago ... 10 months ago 320 to 547 days (1.5 years) y a year ago 548 days+ yy 2 years ago ... 20 years ago Note: From version 2.10.3, if the target moment object is invalid the result is the localized Invalid date string. Note: The ss key was added in 2.18.0. It is an optional threshold. It will never display UNLESS the user manually sets the ss threshold. Until the ss threshold is set, it defaults to the value of the s threshold minus 1 (so, invisible to the user). edit moment().from(Moment|String|Number|Date|Array); 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" From version 2.10.3, if any of the endpoints are invalid the result is the localized Invalid date string. edit moment().toNow(); moment().toNow(Boolean); A common way of displaying time is handled by moment#toNow. This is sometimes called timeago or relative time. This is similar to moment.fromNow, but gives the opposite interval: a.fromNow() = - a.toNow(). This is similar to moment.to, but is special-cased for the current time. Use moment.to, if you want to control the two end points of the interval. moment([2007, 0, 29]).toNow(); // in 4 years If you pass true, you can get the value without the prefix. moment([2007, 0, 29]).toNow(); // in 4 years moment([2007, 0, 29]).toNow(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 44 seconds s in seconds 45 to 89 seconds m in a minute 90 seconds to 44 minutes mm in 2 minutes ... in 44 minutes 45 to 89 minutes h in an hour 90 minutes to 21 hours hh in 2 hours ... in 21 hours 22 to 35 hours d in a day 36 hours to 25 days dd in 2 days ... in 25 days 26 to 45 days M in a month 45 to 319 days MM in 2 months ... in 10 months 320 to 547 days (1.5 years) y in a year 548 days+ yy in 2 years ... in 20 years From version 2.10.3, if the target moment object is invalid the result is the localized Invalid date string. edit moment().to(Moment|String|Number|Date|Array); moment().to(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#to. var a = moment([2007, 0, 28]); var b = moment([2007, 0, 29]); a.to(b) // "in a day" 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.to(b); // "in a day" a.to([2007, 0, 29]); // "in a day" a.to(new Date(2007, 0, 29)); // "in a day" a.to("2007-01-29"); // "in a day" Like moment#toNow, 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.to(start); // "5 days ago" end.to(start, true); // "5 days" From version 2.10.3, if any of the endpoints are invalid the result is the localized Invalid date string. edit moment().calendar(); moment().calendar(referenceDay); moment().calendar(referenceDay, formats); // from 2.10.5 moment().calendar(formats); // from 2.25.0 Calendar time displays time relative to a given referenceDay (defaults to the start of today), but does so slightly differently than moment#fromNow. moment#calendar will format a date with different strings depending on how close to referenceDay's date (today by default) the date is. Last week Last Monday at 2:30 AM The day before Yesterday at 2:30 AM The same day Today at 2:30 AM The next day Tomorrow at 2:30 AM The next week Sunday at 2:30 AM Everything else 7/10/2011 These strings are localized, and can be customized. From 2.10.5 moment supports specifying calendar output formats per invocation: moment().calendar(null, { sameDay: '[Today]', nextDay: '[Tomorrow]', nextWeek: 'dddd', lastDay: '[Yesterday]', lastWeek: '[Last] dddd', sameElse: 'DD/MM/YYYY' }); sameElse is used as the format when the moment is more than a week away from the referenceDay Note: From version 2.14.0 the formats argument to calendar can be a callback that is executed within the moment context with a single argument now: moment().calendar(null, { sameDay: function (now) { if (this.isBefore(now)) { return '[Will Happen Today]'; } else { return '[Happened Today]'; } /* ... */ } }); Note: From version 2.25.0 you can only pass a formats argument, it could be an object of strings and functions: moment().calendar({ sameDay: '[Today]', nextDay: '[Tomorrow]', nextWeek: 'dddd', lastDay: '[Yesterday]', lastWeek: '[Last] dddd', sameElse: 'DD/MM/YYYY' }); moment().calendar({ sameDay: function (now) { if (this.isBefore(now)) { return '[Will Happen Today]'; } else { return '[Happened Today]'; } /* ... */ } }); edit moment().diff(Moment|String|Number|Date|Array); moment().diff(Moment|String|Number|Date|Array, String); moment().diff(Moment|String|Number|Date|Array, String, Boolean); 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 To get the duration of a difference between two moments, you can pass diff as an argument into moment#duration. See the docs on moment#duration for more info. 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 truncate the result to zero decimal places, returning an integer. If you want a floating point number, pass true as the third argument. Before 2.0.0, moment#diff returned a number rounded to the nearest integer, not a truncated number. var a = moment([2008, 9]); var b = moment([2007, 0]); a.diff(b, 'years'); // 1 a.diff(b, 'years', true); // 1.75 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 An 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. edit moment().valueOf(); +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. Note: ECMAScript calls this a "Time Value" edit 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. edit Get the number of days in the current month. moment("2012-02", "YYYY-MM").daysInMonth() // 29 moment("2012-01", "YYYY-MM").daysInMonth() // 31 edit To get a copy of the native Date object that Moment.js wraps, use moment#toDate. This will return a copy of the Date that the moment uses, so any changes to that Date will not cause moment to change. If you want to change the moment Date, see moment#manipulate or moment#set. moment#native has been replaced by moment#toDate and has been deprecated as of 1.6.0. edit This returns an array that mirrors the parameters from new Date(). moment().toArray(); // [2013, 1, 4, 14, 40, 16, 154]; edit When serializing an object to JSON, if there is a Moment object, it will be represented as an ISO8601 string, adjusted to UTC. JSON.stringify({ postDate : moment() }); // '{"postDate":"2013-02-04T22:44:30.652Z"}' If instead you would like an ISO8601 string that reflects the moment's utcOffset(), then you can modify the toJSON function like this: moment.fn.toJSON = function() { return this.format(); } This changes the behavior as follows: JSON.stringify({ postDate : moment() }); // '{"postDate":"2013-02-04T14:44:30-08:00"}' edit moment().toISOString(); moment().toISOString(keepOffset); // from 2.20.0 Formats a string to the ISO8601 standard. moment().toISOString() // 2013-02-04T22:44:30.652Z Note that .toISOString() returns a timestamp in UTC, even if the moment in question is in local mode. This is done to provide consistency with the specification for native JavaScript Date .toISOString(), as outlined in the ES2015 specification. From version 2.20.0, you may call .toISOString(true) to prevent UTC conversion. From version 2.8.4 the native Date.prototype.toISOString is used if available, for performance reasons. edit This returns an object containing year, month, day-of-month, hour, minute, seconds, milliseconds. moment().toObject() // { // years: 2015 // months: 6 // date: 26, // hours: 1, // minutes: 53, // seconds: 14, // milliseconds: 600 // } edit Returns an english string in a similar format to JS Date's .toString(). moment().toString() // "Sat Apr 30 2016 16:59:46 GMT-0500" edit Returns a machine readable string, that can be evaluated to produce the same moment. Because of the name its also used in node interactive shell to display objects. moment().inspect() // 'moment("2016-11-09T22:23:27.861")' moment.utc().inspect() // 'moment.utc("2016-11-10T06:24:10.638+00:00")' moment.parseZone('2016-11-10T06:24:12.958+05:00').inspect() // 'moment.parseZone("2016-11-10T06:24:12.958+05:00")' moment(new Date('nope')).inspect() // 'moment.invalid(/* Invalid Date */)' moment('blah', 'YYYY').inspect() // 'moment.invalid(/* blah */)' Note: This function is mostly intended for debugging, not all cases are handled precisely. edit moment().isBefore(Moment|String|Number|Date|Array); moment().isBefore(Moment|String|Number|Date|Array, String); Check if a moment is before another moment. The first argument will be parsed as a moment, if not already so. 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. As the second parameter determines the precision, and not just a single value to check, using day will check for year, month and day. moment('2010-10-20').isBefore('2010-12-31', 'year'); // false moment('2010-10-20').isBefore('2011-01-01', 'year'); // true 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 isoWeek day hour minute 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 it's possible that the moment created in isBefore is measurably after the one created in moment(), so the call would return true. edit moment().isSame(Moment|String|Number|Date|Array); moment().isSame(Moment|String|Number|Date|Array, String); Check if a moment is the same as another moment. The first argument will be parsed as a moment, if not already so. moment('2010-10-20').isSame('2010-10-20'); // true If you want to limit the granularity to a unit other than milliseconds, pass it 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 When including a second parameter, it will match all units equal or larger. Passing in month will check month and year. Passing in day will check day, month, and year. moment('2010-01-01').isSame('2011-01-01', 'month'); // false, different year moment('2010-01-01').isSame('2010-02-01', 'day'); // false, different month 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 isoWeek day hour minute second If the two moments have different timezones, the timezone of the first moment will be used for the comparison. // Note: Australia/Sydney is UTC+11:00 on these dates moment.tz("2018-11-09T10:00:00", "Australia/Sydney").isSame(moment.tz("2018-11-08T12:00:00", "UTC"), "day"); // false moment.tz("2018-11-08T12:00:00", "UTC").isSame(moment.tz("2018-11-09T10:00:00", "Australia/Sydney"), "day"); // true NOTE: moment().isSame() 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 isSame to perform the check, so the result would be true. But if the code runs slower it's possible that the moment created in isSame is measurably after the one created in moment(), so the call would return false. edit moment().isAfter(Moment|String|Number|Date|Array); moment().isAfter(Moment|String|Number|Date|Array, String); Check if a moment is after another moment. The first argument will be parsed as a moment, if not already so. 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. As the second parameter determines the precision, and not just a single value to check, using day will check for year, month and day. moment('2010-10-20').isAfter('2010-01-01', 'year'); // false 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 isoWeek day hour minute second If nothing is passed to moment#isAfter, it will default to the current time. moment().isAfter(); // false edit moment().isSameOrBefore(Moment|String|Number|Date|Array); moment().isSameOrBefore(Moment|String|Number|Date|Array, String); Check if a moment is before or the same as another moment. The first argument will be parsed as a moment, if not already so. moment('2010-10-20').isSameOrBefore('2010-10-21'); // true moment('2010-10-20').isSameOrBefore('2010-10-20'); // true moment('2010-10-20').isSameOrBefore('2010-10-19'); // false If you want to limit the granularity to a unit other than milliseconds, pass the units as the second parameter. As the second parameter determines the precision, and not just a single value to check, using day will check for year, month and day. moment('2010-10-20').isSameOrBefore('2009-12-31', 'year'); // false moment('2010-10-20').isSameOrBefore('2010-12-31', 'year'); // true moment('2010-10-20').isSameOrBefore('2011-01-01', 'year'); // true Like moment#isAfter and moment#isSame, any of the units of time that are supported for moment#startOf are supported for moment#isSameOrBefore: year month week isoWeek day hour minute second edit moment().isSameOrAfter(Moment|String|Number|Date|Array); moment().isSameOrAfter(Moment|String|Number|Date|Array, String); Check if a moment is after or the same as another moment. The first argument will be parsed as a moment, if not already so. moment('2010-10-20').isSameOrAfter('2010-10-19'); // true moment('2010-10-20').isSameOrAfter('2010-10-20'); // true moment('2010-10-20').isSameOrAfter('2010-10-21'); // false If you want to limit the granularity to a unit other than milliseconds, pass the units as the second parameter. As the second parameter determines the precision, and not just a single value to check, using day will check for year, month and day. moment('2010-10-20').isSameOrAfter('2011-12-31', 'year'); // false moment('2010-10-20').isSameOrAfter('2010-01-01', 'year'); // true moment('2010-10-20').isSameOrAfter('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#isSameOrAfter: year month week isoWeek day hour minute second edit //From 2.13.0 onward moment().isBetween(moment-like, moment-like); moment().isBetween(moment-like, moment-like, String); moment().isBetween(moment-like, moment-like, String, String); // where moment-like is Moment|String|Number|Date|Array //2.9.0 to 2.12.0 moment().isBetween(moment-like, moment-like); 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). The match is exclusive. The first two arguments will be parsed as moments, if not already so. moment('2010-10-20').isBetween('2010-10-19', '2010-10-25'); // true moment('2010-10-20').isBetween('2010-10-19', undefined); // true, since moment(undefined) evaluates as moment() Note that the order of the two arguments matter: the "smaller" date should be in the first argument. moment('2010-10-20').isBetween('2010-10-19', '2010-10-25'); // true moment('2010-10-20').isBetween('2010-10-25', '2010-10-19'); // false If you want to limit the granularity to a unit other than milliseconds, pass the units as the third parameter. moment('2010-10-20').isBetween('2010-01-01', '2012-01-01', 'year'); // false moment('2010-10-20').isBetween('2009-12-31', '2012-01-01', 'year'); // true Like moment#isSame, moment#isBefore, moment#isAfter any of the units of time that are supported for moment#startOf are supported for moment#isBetween. Year, month, week, isoWeek, day, hour, minute, and second. Version 2.13.0 introduces inclusivity. A [indicates inclusion of a value. A (indicates exclusion. If the inclusivity parameter is used, both indicators must be passed. moment('2016-10-30').isBetween('2016-10-30', '2016-12-30', undefined, '()'); //false moment('2016-10-30').isBetween('2016-10-30', '2016-12-30', undefined, '[)'); //true moment('2016-10-30').isBetween('2016-01-01', '2016-10-30', undefined, '()'); //false moment('2016-10-30').isBetween('2016-01-01', '2016-10-30', undefined, '(]'); //true moment('2016-10-30').isBetween('2016-10-30', '2016-10-30', undefined, '[]'); //true Note that in the event that the from and to parameters are the same, but the inclusivity parameters are different, false will preside. moment('2016-10-30').isBetween('2016-10-30', '2016-10-30', undefined, '(]'); //false If the inclusivity parameter is not specified, Moment will default to (). edit moment#isDST checks if the current moment is in daylight saving time. moment([2011, 2, 12]).isDST(); // false, March 12 2011 is not DST moment([2011, 2, 14]).isDST(); // true, March 14 2011 is DST // This example is for "en" locale: edit moment('2013-03-10 2:30', 'YYYY-MM-DD HH:mm').isDSTShifted() Note: As of version 2.14.0 this function is deprecated. It doesn't give the right answer after modifying the moment object. For more information refer to moment/3160 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(); //=> '2013-03-10T01:30:00-05:00' This is because daylight saving 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. edit 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 edit To check if a variable is a moment object, use moment.isMoment(). moment.isMoment() // false moment.isMoment(new Date()) // false moment.isMoment(moment()) // true From version 2.11.0, you can also test for a moment object by instanceof operator: moment() instanceof moment // true edit 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 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. edit // From 2.8.1 onward 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 (United States) 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_août_septembre_octobre_novembre_décembre'.split('_'), monthsShort : 'janv._févr._mars_avr._mai_juin_juil._août_sept._oct._nov._déc.'.split('_'), monthsParseExact : true, weekdays : 'dimanche_lundi_mardi_mercredi_jeudi_vendredi_samedi'.split('_'), weekdaysShort : 'dim._lun._mar._mer._jeu._ven._sam.'.split('_'), weekdaysMin : 'Di_Lu_Ma_Me_Je_Ve_Sa'.split('_'), weekdaysParseExact : true, longDateFormat : { LT : 'HH:mm', LTS : 'HH:mm:ss', L : 'DD/MM/YYYY', LL : 'D MMMM YYYY', LLL : 'D MMMM YYYY HH:mm', LLLL : 'dddd D MMMM YYYY HH:mm' }, 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 : 'un an', yy : '%d ans' }, dayOfMonthOrdinalParse : /\d{1,2}(er|e)/, ordinal : function (number) { return number + (number === 1 ? 'er' : 'e'); }, meridiemParse : /PD|MD/, isPM : function (input) { return input.charAt(0) === 'M'; }, // In case the meridiem units are not separated around 12, then implement // this function (look at locale/id.js for an example). // meridiemHour : function (hour, meridiem) { // return /* 0-23 hour, given meridiem token and hour 1-12 */ ; // }, meridiem : function (hours, minutes, isLower) { return hours < 12 ? 'PD' : 'MD'; }, week : { dow : 1, // Monday is the first day of the week. doy : 4 // Used to determine first week of the year. } }); Details about week.dow and week.doy can be found in the customization section. 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 As of 2.21.0, Moment will console.warn if the locale is unavailable. As of 2.8.0, changing the global locale doesn't affect existing instances. moment.locale('fr'); var m = moment(1316116057189); m.fromNow(); // il y a une heure moment.locale('en'); m.fromNow(); // il y a une heure 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' The logic works as follows -- the next locale is picked and tried as-is. If that fails, the code normally tries to chop the last bit (normally the country designation) and try again. However, if the next array element has the same or longer prefix as the one to be tried, the iteration continues. So for example if the array has the sequence "AA-BB", "AA-CC", "XX-YY" then first "AA-BB" is tried, then a naive solution would try "AA", but this one instead checks to see that "AA-CC" is actually more concrete than "AA", so it tries "AA-CC" next, and only after it fails (if it fails) it tries "AA", because "XX-YY" does not have "AA" as prefix. So in the end the following locales are tried in this order (assuming all fail so the next one is tried): "AA-BB", "AA-CC", "AA", "XX-YY", "XX" edit // From version 2.8.1 onward moment().locale(String|String[]|Boolean); // Deprecated version 2.8.1 moment().lang(String|String[]|Boolean); A global locale configuration can be problematic when passing around moments that may need to be formatted into different locale. moment.locale('en'); // default the locale to English var localLocale = moment(); localLocale.locale('fr'); // set this instance to use French localLocale.format('LLLL'); // dimanche 15 juillet 2012 11:01 moment().format('LLLL'); // Sunday, July 15 2012 11:01 AM moment.locale('es'); // change the global locale to Spanish localLocale.format('LLLL'); // dimanche 15 juillet 2012 11:01 moment().format('LLLL'); // Domingo 15 Julio 2012 11:01 localLocale.locale(['tq', 'fr']); // set this instance to the first localization found localLocale.format('LLLL'); // dimanche 15 juillet 2012 11:01 moment().format('LLLL'); // Sunday, July 15 2012 11:01 AM localLocale.locale(false); // reset the instance locale localLocale.format('LLLL'); // Domingo 15 Julio 2012 11:01 moment().format('LLLL'); // Domingo 15 Julio 2012 11:01 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.localeData().months(moment([2012, 0])) // "janvier" fr.locale('en'); fr.localeData().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 way it does in the global locale configuration. edit Loading locales in NodeJS is super easy. If there is a locale file in moment/locale/ named after that key, import it first, then call moment.locale to load it. var moment = require('moment'); //or // import moment from 'moment'; // import locale file(s) import 'moment/locale/fr'; moment.locale('fr'); moment(1316116057189).fromNow(); // il y a 6 ans To save the step of loading individual locales (i.e. just load them all), import the moment/min/moment-with-locales module instead. import moment from 'moment/min/moment-with-locales'; moment.locale('de'); moment(1316116057189).fromNow(); // vor 6 Jahren If you want your locale supported, create a pull request to the develop branch with the required locale and unit test files. edit // From 2.8.1 onward 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. Be sure to specify the charset to prevent encoding issues. moment.locale('fr'); // Set the default/global locale // ... There are minified versions of all locales together: To minimize HTTP requests, use our Grunt task to compile Moment with a custom list of locales: grunt transpile:fr,it If you are using JSPM as plugin manager, you should add the locale in your lib. import * as moment from 'moment'; import 'moment/locale/fr'; Note: Locale files are defined in UMD style, so they should work seamlessly in all environments. 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/src/locale/fr.js and moment/src/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! edit // From version 2.8.1 onward 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' As of version 2.12.0 it is possible to list all locales that have been loaded and are available to use: moment.locales() edit moment.months() 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' As of 2.13.0 you can pass a bool as the first parameter of the weekday functions. If true, the weekdays will be returned in locale specific order. For instance, in the Arabic locale, Saturday is the first day of the week, thus: moment.locale('ar'); moment.weekdays(true); // lists weekdays Saturday-Friday in Arabic moment.weekdays(true, 2); //will result in Monday in Arabic Note: Absent the locale specific parameter, 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' edit localeData = moment.localeData() localeData.months(Moment) localeData.months() localeData.monthsShort(Moment) localeData.monthsShort() localeData.monthsParse(String) localeData.weekdays(Moment) localeData.weekdays() localeData.weekdays(Boolean) ## Added 2.24.0, sorts weekdays by locale localeData.weekdaysShort(Moment) localeData.weekdaysShort() localeData.weekdaysShort(Boolean) ## Added 2.24.0, sorts weekdays by locale localeData.weekdaysMin(Moment) localeData.weekdaysMin() localeData.weekdaysMin(Boolean) ## Added 2.24.0, sorts weekdays by locale localeData.weekdaysParse(String) localeData.longDateFormat(String) localeData.isPM(String) localeData.meridiem(Number, Number, Boolean) localeData.calendar(String, Moment) localeData.relativeTime(Number, Boolean, String, Boolean) localeData.pastFuture(Number, String) localeData.ordinal(Number) localeData.preparse(String) localeData.postformat(String) localeData.week(Moment) 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 aMoment localeData.monthsParse(longOrShortMonthString); // returns month id (0 to 11) of input localeData.weekdays(aMoment); // full weekday name of aMoment localeData.weekdaysShort(aMoment); // short weekday name of aMoment localeData.weekdaysMin(aMoment); // min weekday name of aMoment localeData.weekdaysParse(minShortOrLongWeekdayString); // returns weekday id (0 to 6) of input localeData.longDateFormat(dateFormat); // returns the full format of abbreviated date-time formats LT, L, LL and so on localeData.isPM(amPmString); // returns true iff amPmString represents PM localeData.meridiem(hours, minutes, isLower); // returns am/pm string for particular time-of-day in upper/lower case localeData.calendar(key, aMoment); // returns a format that would be used for calendar representation. Key is one of 'sameDay', 'nextDay', 'lastDay', 'nextWeek', 'prevWeek', 'sameElse' localeData.relativeTime(number, withoutSuffix, key, isFuture); // returns relative time string, key is on of 's', 'm', 'mm', 'h', 'hh', 'd', 'dd', 'M', 'MM', 'y', 'yy'. Single letter when number is 1. localeData.pastFuture(diff, relTime); // convert relTime string to past or future string depending on diff localeData.ordinal(number); // convert number to ordinal string 1 -> 1st localeData.preparse(str); // called before parsing on every input string localeData.postformat(str); // called after formatting on every string localeData.week(aMoment); // returns week-of-year of aMoment localeData.invalidDate(); // returns a translation of 'Invalid date' localeData.firstDayOfWeek(); // 0-6 (Sunday to Saturday) localeData.firstDayOfYear(); // 0-15 Used to determine first week of the year. Details about firstDayOfYear can be found in the customization section. edit moment.locale('x-pseudo') As of version 2.13.0 moment optionally includes a pseudo locale. This locale will populate the dates with very obviously changed data. Pseudo locales can be useful when testing, as they make obvious what data has and has not been localized. Just include the pseudo-locale, and set moment's locale to x-pseudo. Text from Moment will be very easy to spot. moment.locale('x-pseudo'); moment().format('LLL'); //14 F~ébrú~árý 2010 15:25 moment().fromNow(); //'á ~féw ~sécó~ñds á~gó' moment().calendar(); //'T~ódá~ý át 02:00' Moment.js is very easy to customize. In general, you should create a locale setting with your customizations. moment.locale('en-my-settings', { // customizations. }); You can remove a previously defined locale by passing null as the second argument. The deleted locale will no longer be available for use. moment.locale('fr'); // 'fr' moment.locale('en'); // 'en' moment.locale('fr', null); moment.locale('fr'); // 'en' As of 2.12.0 it is possible to create a locale that inherits from a parent locale. moment.defineLocale('en-foo', { parentLocale: 'en', /* */ }); Properties that are not specified in the locale will be inherited from the parent locale. As of 2.16.0 it is possible to define a locale with a parent that hasn't itself been defined or loaded. moment.defineLocale('fakeLocale', {parentLocale:'xyz'}) As of 2.21.0 when attempting to create a moment with the newly defined locale, moment will attempt to lazy load the parent if it exists. Failing that it will default the parent to the global locale. As of 2.12.0 it is also possible to update a locale's properties. moment.updateLocale('en', { /**/ }); Any properties specified will be updated, while others will remain the same. This function does not affect moments that already exist. Note that calling updateLocale also changes the current global locale, to the locale that is updated; see this GitHub issue for more information. To revert an update use: moment.updateLocale('en', null); 2.12.0 deprecated using moment.locale() to change an existing locale. Use moment.updateLocale() instead. edit // From 2.12.0 onward moment.updateLocale('en', { months : String[] }); moment.updateLocale('en', { months : Function }); moment.updateLocale('en', { months : { format : String[], standalone : String[] } }); // From 2.11.0 moment.locale('en', { months : { format : String[], standalone : String[] } }); // From 2.8.1 to 2.11.2 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.updateLocale('en', { months : ["January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December"] }); 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.updateLocale('en', { months : function (momentToFormat, format) { // momentToFormat is the moment currently being formatted // format is the formatting string if (/^MMMM/.test(format)) { // if the format starts with 'MMMM' return nominative[momentToFormat.month()]; } else { return subjective[momentToFormat.month()]; } } }); From version 2.11.0 months can also be an object, specifying standalone and format forms (nominative and accusative). The regular expression that is run on the format to check whether to use the format form is /D[oD]?(\[[^\[\]]*\]|\s+)+MMMM?/. From version 2.14.0 a different one can be specified with the isFormat key. moment.updateLocale('en', { months : { format: 'sausio_vasario_kovo_balandžio_gegužės_birželio_liepos_rugpjūčio_rugsėjo_spalio_lapkričio_gruodžio'.split('_'), standalone: 'sausis_vasaris_kovas_balandis_gegužė_birželis_liepa_rugpjūtis_rugsėjis_spalis_lapkritis_gruodis'.split('_'), isFormat: /D[oD]?(\[[^\[\]]*\]|\s+)+MMMM?|MMMM?(\[[^\[\]]*\]|\s+)+D[oD]?/ // from 2.14.0 } }); edit // From 2.12.0 onward moment.updateLocale('en', { monthsShort : String[] }); moment.updateLocale('en', { monthsShort : Function }); moment.updateLocale('en', { monthsShort : { format: String[], standalone : String[] } }); // From 2.11.0 moment.locale('en', { monthsShort : { format: String[], standalone : String[] } }); // From 2.8.1 to 2.11.2 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.updateLocale('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.updateLocale('en', { monthsShort : function (momentToFormat, format) { if (/^MMMM/.test(format)) { return nominative[momentToFormat.month()]; } else { return subjective[momentToFormat.month()]; } } }); Note: From version 2.11.0, like Locale#months, Locale#monthsShort can be an object with standalone and format cases. moment.updateLocale('en', { monthsShort : { format: 'янв_фев_мар_апр_мая_июня_июля_авг_сен_окт_ноя_дек'.split('_'), standalone: 'янв_фев_март_апр_май_июнь_июль_авг_сен_окт_ноя_дек'.split('_') } }); edit // From version 2.12.0 onward moment.updateLocale('en', { weekdays : String[] }); moment.updateLocale('en', { weekdays : Function }); moment.updateLocale('en', { weekdays : { standalone : String[], format : String[], isFormat : RegExp } }); // From version 2.11.0 moment.locale('en', { weekdays : { standalone : String[], format : String[], isFormat : Boolean } }); // From version 2.8.1 to 2.11.2 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.updateLocale('en', { weekdays : ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"] }); Locale#weekdays can be a callback function as well. moment.updateLocale('en', { weekdays : function (momentToFormat, format) { return weekdays[momentToFormat.day()]; } }); Note: From version 2.11.0 format/standalone cases can be passed as well. isFormat will be used against the full format string to determine which form to use. moment.updateLocale('en', { weekdays : { standalone: 'Воскресенье_Понедельник_Вторник_Среда_Четверг_Пятница_Суббота'.split('_'), format: 'Воскресенье_Понедельник_Вторник_Среду_Четверг_Пятницу_Субботу'.split('_'), isFormat: /\[?[Вв] ?(?:прошлую|следующую|эту)? ?\] ?dddd/ } }); edit // From 2.12.0 onward moment.updateLocale('en', { weekdaysShort : String[] }); moment.updateLocale('en', { weekdaysShort : Function }); // From 2.8.1 to 2.11.2 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.updateLocale('en', { weekdaysShort : ["Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"] }); Locale#weekdaysShort can be a callback function as well. moment.updateLocale('en', { weekdaysShort : function (momentToFormat, format) { return weekdaysShort[momentToFormat.day()]; } }); edit // From 2.12.0 onward moment.updateLocale('en', { weekdaysMin : String[] }); moment.updateLocale('en', { weekdaysMin : Function }); // From 2.8.1 to 2.11.2 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.updateLocale('en', { weekdaysMin : ["Su", "Mo", "Tu", "We", "Th", "Fr", "Sa"] }); Locale#weekdaysMin can be a callback function as well. moment.updateLocale('en', { weekdaysMin : function (momentToFormat, format) { return weekdaysMin[momentToFormat.day()]; } }); edit // From 2.12.0 onward moment.updateLocale('en', { weekdaysMin : String[] }); moment.updateLocale('en', { weekdaysMin : Function }); // From 2.8.1 to 2.11.2 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.updateLocale('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.updateLocale('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" } }); edit // From 2.12.0 onward moment.updateLocale('en', { relativeTime : Object }); // From 2.8.1 to 2.11.2 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.updateLocale('en', { relativeTime : { future: "in %s", past: "%s ago", s : 'a few seconds', ss : '%d seconds', m: "a minute", mm: "%d minutes", h: "an hour", hh: "%d hours", d: "a day", dd: "%d days", w: "a week", ww: "%d weeks", 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 a 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. Note: Handling for w and ww was added in 2.25.0. edit // From 2.12.0 onward moment.updateLocale('en', { meridiem : Function }); // From 2.8.1 to 2.11.2 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 omitted, 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.updateLocale('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"; } } }); edit // From 2.12.0 onward moment.updateLocale('en', { meridiemParse : RegExp isPM : Function }); // From 2.8.1 to 2.11.2 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.updateLocale('en', { isPM : function (input) { return ((input + '').toLowerCase()[0] === 'p'); } }); To configure what strings should be parsed as input, set the meridiemParse property. moment.updateLocale('en', { meridiemParse : /[ap]\.?m?\.?/i }); edit // From 2.12.0 onward moment.updateLocale('en', { calendar : Object }); // From 2.8.1 to 2.11.2 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 and first argument a moment that depicts now. It should return a formatting string. function callback (now) { return '[hoy a la' + ((this.hours() !== 1) ? 's' : '') + '] LT'; } edit moment.calendarFormat = Function This lets you modify the tokens used by calendar. moment.calendarFormat = function (myMoment, now) { var diff = myMoment.diff(now, 'days', true); var nextMonth = now.clone().add(1, 'month'); var retVal = diff < -6 ? 'sameElse' : diff < -1 ? 'lastWeek' : diff < 0 ? 'lastDay' : diff < 1 ? 'sameDay' : diff < 2 ? 'nextDay' : diff < 7 ? 'nextWeek' : // introduce thisMonth and nextMonth (myMoment.month() === now.month() && myMoment.year() === now.year()) ? 'thisMonth' : (nextMonth.month() === myMoment.month() && nextMonth.year() === myMoment.year()) ? 'nextMonth' : 'sameElse'; return retVal; }; edit // From 2.12.0 onward moment.updateLocale('en', { ordinal : Function }); // From 2.8.1 to 2.11.2 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.updateLocale('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. edit moment.relativeTimeThreshold(unit); // getter moment.relativeTimeThreshold(unit, limit); // setter 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 unit is one of ss, s, m, h, d, w, M. unit meaning usage ss a few seconds least number of seconds to be counted in seconds, minus 1. Must be set after setting the `s` unit or without setting the `s` unit. 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 week. w weeks least number of weeks to be considered a month. Not used by default. M months least number of months to be considered a year. // Retrieve existing thresholds moment.relativeTimeThreshold('ss'); // 44 moment.relativeTimeThreshold('s'); // 45 moment.relativeTimeThreshold('m'); // 45 moment.relativeTimeThreshold('h'); // 22 moment.relativeTimeThreshold('d'); // 26 moment.relativeTimeThreshold('w'); // null (disabled) moment.relativeTimeThreshold('M'); // 11 // Set new thresholds moment.relativeTimeThreshold('s', 40); moment.relativeTimeThreshold('ss', 3); moment.relativeTimeThreshold('m', 40); moment.relativeTimeThreshold('h', 20); moment.relativeTimeThreshold('d', 25); moment.relativeTimeThreshold('w', 4); // enables weeks moment.relativeTimeThreshold('M', 10); Note: Week unit was added in 2.25.0. By default it is not used (set to null), but you can set it to non-null value, and also (optionally) set d lower, so it transitions from days to weeks earlier. Note: Retrieving thresholds was added in 2.8.1. Note: Retrieving and setting ss threshold was added in 2.18.0. edit moment.relativeTimeRounding(); // getter moment.relativeTimeRounding(fn); // setter duration.humanize rounds a possibly double value before supplying it to the relativeTime format string specified in the locale. To control the rounding you can use moment.relativeTimeRounding. var roundingDefault = moment.relativeTimeRounding(); // Round relative time evaluation down moment.relativeTimeRounding(Math.floor); moment.relativeTimeThreshold('s', 60); moment.relativeTimeThreshold('m', 60); moment.relativeTimeThreshold('h', 24); moment.relativeTimeThreshold('d', 7); moment.relativeTimeThreshold('w', 4); moment.relativeTimeThreshold('M', 12); var a = moment(); a.subtract({hours: 23, minutes: 59, seconds: 59}); a.toNow(); // == 'in 23 hours' 'Round down towards the nearest hour' // back to default moment.relativeTimeRounding(roundingDefault); You can even choose to do no rounding at all: var retainValue = function (value) { return value; }; moment.relativeTimeRounding(retainValue); var a = moment(); a.subtract({hours: 39}); a.toNow(); // == 'in 1.625 days', 'Round down towards the nearest year' edit moment.now = function () { return +new Date(); } If you want to change the time that Moment sees, you can specify a method that returns the number of milliseconds since the Unix epoch (January 1, 1970). The default is: moment.now = function () { return +new Date(); } This will be used when calling moment(), and the current date used when tokens are omitted from format(). In general, any method that needs the current time uses this under the hood. edit // From 2.12.0 onward moment.updateLocale('en', { week : { dow : Int, doy : Int } }); // From 2.8.1 to 2.11.2 moment.locale('en', { week : { dow : Int, doy : Int } }); // Deprecated in 2.8.1 moment.lang('en', { week : { dow : Int, doy : Int } }); Locale#week.dow should be an integer representing the first day of the week, 0 is Sunday, 1 is Monday, ..., 6 is Saturday. Locale#week.doy should be an integer. doy is used together with dow to determine the first week of the year. doy is calculated as 7 + dow - janX, where janX is the first day of January that must belong to the first week of the year. // ISO-8601, Europe moment.updateLocale("en", { week: { dow: 1, // First day of week is Monday doy: 4 // First week of year must contain 4 January (7 + 1 - 4) }}); // US, Canada moment.updateLocale("en", { week: { dow: 0, // First day of week is Sunday doy: 6 // First week of year must contain 1 January (7 + 0 - 1) }}); // Many Arab countries moment.updateLocale("en", { week: { dow: 6, // First day of week is Saturday doy: 12 // First week of year must contain 1 January (7 + 6 - 1) }}); // Also common moment.updateLocale("en", { week: { dow: 1, // First day of week is Monday doy: 7 // First week of year must contain 1 January (7 + 1 - 1) }}); edit moment.updateLocale('en', { eras: [{ since: '0001-01-01', until: +Infinity, offset: 1, name: 'Anno Domini', narrow: 'AD', abbr: 'AD' }, { until: -Infinity, since: '0000-12-31', offset: 1, name: 'Before Christ', narrow: 'BC', abbr: 'BC' }], }); Specify Eras for a particular locale. An era is a time interval with name and year numbering. Absolute year number (like 2020) can also be specified as 2020 AD: the 2020th year of the era AD. Similarly the absolute year number -0500 can be described as 501 BC, the 501st year from the BC era. eras: [{ since: '0001-01-01', // the start of the era until: +Infinity, // the end of the era, can be +/-Infinity offset: 1, // added to year to (mostly) avoid 0 era years name: 'Anno Domini',// full name of era narrow: 'AD', // narrow name of era abbr: 'AD' // abbreviated name of era }] since and until govern the direction of the era. As in the case of BC it grows toward -Infinity, thus since > until. For eras that increment toward +Infinity since < until. Parsing/formatting of eras is accomplished with yo, y* and N* tokens. Note: The era-related APIs are subject to change. edit // From 2.12.0 onward moment.updateLocale('en', { invalidDate : String }); // From 2.8.1 to 2.11.2 moment.locale('en', { invalidDate : String }); // Deprecated in 2.8.1 moment.lang('en', { invalidDate : String }); Locale#invalidDate should be a string. moment.updateLocale("es", { invalidDate: "Fecha invalida" }); Moment.js also has duration objects. Where a moment is defined as a single point in time, a duration is 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. As discussed here, the duration format for Moment.js differs very slightly from the specifications for ISO 8601 nominal duration and RFC 5545 duration. edit moment.duration(Number, String); moment.duration(Number); moment.duration(Object); moment.duration(String); moment.duration(String, String); // 2.25.0 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'); moment.duration('2', 'years'); // from 2.25.0 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 As of 2.3.0, moment also supports parsing ISO 8601 durations. moment.duration('P1Y2M3DT4H5M6S'); moment.duration('P1M'); As of 2.11.0, duration format strings with a space between days and rest is supported. moment.duration('7 23:59:59.999'); As of 2.13.0, mixed negative and positive signs are supported when parsing durations. moment.duration('PT-6H3M') As of 2.18.0, invalid durations are supported, similarly to invalid moments. To create an invalid duration you can pass NaN for a value of a unit. In upcoming releases expect invalid durations to cover more cases (like null values for units). moment.duration(NaN); moment.duration(NaN, 'days'); moment.duration.invalid(); edit moment.duration().clone(); Create a clone of a duration. Durations are mutable, just like moment objects, so this lets you get a snapshot, at some point in time. var d1 = moment.duration(); var d2 = d1.clone(); d1.add(1, 'second'); d1.asMilliseconds() !== d2.asMilliseconds(); edit moment.duration().humanize(); moment.duration().humanize(withSuffix); moment.duration().humanize(withSuffix, thresholds); // from 2.25.0 moment.duration().humanize(thresholds); // from 2.25.0 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 describing a duration a month (suffix-less). If you want an oriented duration in a month, a month ago (with suffix), pass in true as seen below. moment.duration(1, "minutes").humanize(true); // in a minute For suffixes before now, pass in a negative number. moment.duration(-1, "minutes").humanize(true); // a minute ago Invalid durations are humanized to the localized version of Invalid Date. moment.duration.invalid().humanize(); // Invalid Date Humanize output can be configured with relative time thresholds. To specify thresholds for a particular invocation of humanize, pass them as a sole argument or after suffix arg: moment.duration(-1, 'week').humanize(true, {d: 7, w: 4}); // a week ago moment.duration(-1, 'week').humanize({d: 7, w: 4}); // a week Note: Passing thresholds in humanize was added in 2.25.0. edit moment.duration().milliseconds(); 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 edit moment.duration().seconds(); 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 edit moment.duration().minutes(); 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. edit moment.duration().hours(); 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. edit moment.duration().days(); moment.duration().asDays(); As with the other getters for durations, moment.duration().days() gets the days (0 - 30). moment.duration().asDays() gets the length of the duration in days. edit moment.duration().weeks(); moment.duration().asWeeks(); As with the other getters for durations, moment.duration().weeks() gets the weeks (0 - 4). moment.duration().asWeeks() gets the length of the duration in weeks. Pay attention that unlike the other getters for duration, weeks are counted as a subset of the days, and are not taken off the days count. Note: The length of a duration in weeks is defined as 7 days. edit moment.duration().months(); 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. edit moment.duration().years(); 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. edit moment.duration().add(Number, String); moment.duration().add(Number); moment.duration().add(Duration); moment.duration().add(Object); Mutates the original duration by adding time. The same keys and shorthands used to create durations can be used here as the second argument. var a = moment.duration(1, 'd'); var b = moment.duration(2, 'd'); a.add(b).days(); // 3 Note that adding an invalid duration to any other duration results in an invalid duration. edit moment.duration().subtract(Number, String); moment.duration().subtract(Number); moment.duration().subtract(Duration); moment.duration().subtract(Object); Mutates the original duration by subtracting time. The same keys and shorthands used to create durations can be used here as the second argument. var a = moment.duration(3, 'd'); var b = moment.duration(2, 'd'); a.subtract(b).days(); // 1 Note that adding an invalid duration to any other duration results in an invalid duration. edit var duration = moment.duration(x.diff(y)) You can also use duration with moment#diff to get the duration between two moments. To do so, simply pass the moment#diff method into moment#duration as follows: var x = new moment() var y = new moment() var duration = moment.duration(x.diff(y)) // returns duration object with the duration between x and y See here for more information about moment#diff. edit moment.duration().as(String); 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'); Invalid durations return NaN for all units. edit moment.duration().get(String); 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'); Invalid durations return NaN for all units. edit moment.duration().toJSON(); When serializing a duration object to JSON, it will be represented as an ISO8601 string. JSON.stringify({ postDuration : moment.duration(5, 'm') }); // '{"postDuration":"PT5M"}' Invalid durations return Invalid Date as json representation. edit To check if a variable is a moment duration object, use moment.isDuration(). moment.isDuration() // false moment.isDuration(new Date()) // false moment.isDuration(moment()) // false moment.isDuration(moment.duration()) // true moment.isDuration(moment.duration(2, 'minutes')) // true edit moment.duration().toISOString(); Returns duration in string as specified by ISO 8601 standard. moment.duration(1, 'd').toISOString() // "P1D" Format PnYnMnDTnHnMnS description: Unit Meaning P _P_ stands for period. Placed at the start of the duration representation. Y Year M Month D Day T Designator that precedes the time components. H Hour M Minute S Second edit moment.duration().locale(); moment.duration().locale(String); You can get or set the locale of a duration using locale(...). The locale will affect the duration's string methods, like humanize(). See the intl section for more information on internationalization generally. moment.duration(1, "minutes").locale("en").humanize(); // a minute moment.duration(1, "minutes").locale("fr").humanize(); // une minute moment.duration(1, "minutes").locale("es").humanize(); // un minuto Suffixes in humanize() are also internationalized: moment.duration(1, "minutes").locale("en").humanize(true); // in a minute moment.duration(1, "minutes").locale("fr").humanize(true); // dans une minute moment.duration(1, "minutes").locale("es").humanize(true); // en un minuto moment.duration(-1, "minutes").locale("en").humanize(true); // a minute ago moment.duration(-1, "minutes").locale("fr").humanize(true); // il y a une minute moment.duration(-1, "minutes").locale("es").humanize(true); // hace un minuto Moment exposes some methods which may be useful to people extending the library or writing custom parsers. edit moment.normalizeUnits(String); Many of Moment's functions allow the caller to pass in aliases for unit enums. For example, all of the gets 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' edit 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(). Some other people have made plugins for Moment.js that may be useful to you. edit npm install moment-strftime 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. 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-0600 (MDT) More information and detailed docs can be found on GitHub at . edit npm install moment-jdateformatparser If you want to work with the java.text.DateFormat you can use this plugin. For example, moment("2013-12-24 14:30").formatWithJDF("dd.MM.yyyy"); // returns the formatted date "24.12.2013" moment().toJDFString("DD.MM.YYYY"); // returns the Java format pattern "dd.MM.yyyy" The repository is located at github.com/MadMG/moment-jdateformatparser. edit edit 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. edit npm install moment-precise-range-plugin 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 16:05:06"); // 1 year 2 months 3 days 4 hours 5 minutes 6 seconds moment.preciseDiff("2014-01-01 12:00:00", "2014-04-20 12:00:00"); // 3 months 19 days To obtain the raw numeric values rather than a string, pass the value true as the third argument to the method: moment.preciseDiff(m1, m2, true); // {years : 0, months : 1, days : 2, hours : 3, minutes : 4, seconds : 5, firstDateWasLater : false} edit npm install moment-isocalendar 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. edit npm install moment-jalaali 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 Jalaali date. m.format('jYYYY/jM/jD [is] YYYY/M/D'); // 1360/5/26 is 1981/8/17 The repository is located at github.com/behrang/moment-jalaali. edit If you want to work with Hijri calendar then you can use moment-hijri plugin. moment-hijri is a moment plugin for the Hijri lunar calendar based on Umm al-Qura calculations. This plugin is developed by Suhail Alkowaileet. When you install it, it will wrap moment and you will be able to parse Hijri dates. Here is a short example: m = moment('1410/8/28', 'iYYYY/iM/iD'); // Parse a Hijri date. m.format('iYYYY/iM/iD [is] YYYY/M/D'); // 1410/8/28 is 1990/3/25 The repository is located at github.com/xsoh/moment-hijri. edit edit 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(); // Length Interval 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"] 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').twitterLong(); // 5 hours Yes, it does smart pluralization. moment().subtract(1, 'hour').twitterLong(); // 1 hour Not short enough for you? moment().subtract(6, 'days').twitterShort(); // 6d edit If you ever have need for Fiscal, Calendar or Academic quarters, you can use the moment-fquarter plugin by @robgallen. At its 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 edit npm install moment-parseformat This plugin extracts the format of a date/time string. var format = moment.parseFormat('Thursday, February 6th, 2014 9:20pm'); // 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 edit This plugin will round date/time to a given interval. For example, require('moment-round'); var m = new moment(); // 2015-06-18 15:30:19 m.round(5, 'seconds'); // 2015-06-18 15:30:20 m.ceil(3, 'minutes'); // 2015-06-18 15:33:00 m.floor(16, 'hours'); // 2015-06-18 00:00:00 m.ceil(21, 'hours'); // 2015-06-18 21:00:00 m.ceil(20, 'hours'); // 2015-06-19 00:00:00 The repository is located at github.com/WebDevTmas/moment-round. edit bower install moment-transform moment-transform is a plugin that manipulated dates through patterns. You can use basic operations –set/add/subtract– on individual parts (hours, month, …) of a Moment instance. moment().transform('YYYY-MM-+01 00:00:00.000'); // Tonight at midnight moment().transform('14:30:00.000'); // Today, 2:30 pm moment().transform('YYYY-MM--30 00:00:00.000'); // 30 days ago Optional parameters lets you specify custom patterns and force strict pattern usage (non-alphabetic characters are not mandatory in passed string by default). moment().transform('+01MMYYYY', 'DD/MM/YYYY', false); // Tomorrow, same time moment().transform('+01MMYYYY', 'DD/MM/YYYY', true); // Invalid date You can see it live there while the repository is here. edit npm install moment-taiwan If you want to work with Taiwan calendar system , you can use Bradwoo8621's plugin moment-taiwan. When installed, it will wrap moment and moment will be able to format and parse Taiwan years. Here is a short example: m = moment('104/01/01', 'tYY/MM/DD') // Parse a Taiwan date m.format('tYY/MM/DD [is] YYYY/M/D') // 104/01/01 is 2015/01/01 m.twYear() // 104 The repository is located at github.com/bradwoo8621/moment-taiwan. edit npm install moment-duration-format This is a plugin that will allow comprehensive formatting of Moment Durations. For example, moment.duration(123, "minutes").format("h:mm"); // "2:03" The repository is located at github.com/jsmreese/moment-duration-format. edit This is a Moment.js plugin that allows the use of timers, which offer much more control than the native JavaScript timers. It's basically a rewrite of JavaScripts own setInterval and setTimeout. For example, var timer = moment.duration(5, "seconds").timer({loop: true}, function() { // Callback }); The repository is located at github.com/SeverinDK/moment-timer. edit npm install moment-business This is a Moment.js library that allows Moment operations for Western work weeks: 7 day weeks where Saturday and Sunday are non-work days. For example, import business from 'moment-business'; // true if the moment is Mon-Fri, false otherwise business.isWeekDay(someMoment); // Adds five work days to the Moment business.addWeekDays(someMoment, 5); The repository is located at github.com/jmeas/moment-business. edit If you want to format times in a short way, you can use the moment-shortformat plugin by @researchgate. It is based on and similar to the moment.twitter plugin but has a different output. moment().subtract(5, 'hours').short(); // 5h ago moment().add(5, 'hours').short(); // in 5h You can also disable the use of the relative time templates moment().subtract(1, 'hour').short(false); // 1h If the date is too far in the future or the past it will display like that moment().subtract(500, 'days').short(); // 5 Mar, 1970 edit npm install moment-feiertage --save This (moment-feiertage) is a Moment.js plugin to determine if a date is a German holiday. Holidays are taken from Wikipedia (de). It's a bit complicated to determine if a date is a holiday, because religious holidays vary every year and differ within the 16 German states. Made by DaniSchenk. var someDateInSomeStates = moment('2018-11-01').isHoliday(['BW', 'SH', 'TH']); /* returns { allStates: false, holidayName: 'Allerheiligen', holidayStates: ['BW'], testedStates: ['BW', 'SH', 'TH'] }*/ The repository is located at github.com/DaniSchenk/moment-feiertage.