```
//! moment.js ;(function (global, factory) { typeof exports === 'object' && typeof module !== 'undefined' ? module.exports = factory() : typeof define === 'function' && define.amd ? define(factory) : global.moment = factory() }(this, (function () { 'use strict'; var hookCallback; function hooks () { return hookCallback.apply(null, arguments); } // This is done to register the method called with moment() // without creating circular dependencies. function setHookCallback (callback) { hookCallback = callback; } function isArray(input) { return input instanceof Array || Object.prototype.toString.call(input) === '[object Array]'; } function isObject(input) { // IE8 will treat undefined and null as object if it wasn't for // input != null return input != null && Object.prototype.toString.call(input) === '[object Object]'; } function isObjectEmpty(obj) { if (Object.getOwnPropertyNames) { return (Object.getOwnPropertyNames(obj).length === 0); } else { var k; for (k in obj) { if (obj.hasOwnProperty(k)) { return false; } } return true; } } function isUndefined(input) { return input === void 0; } function isNumber(input) { return typeof input === 'number' || Object.prototype.toString.call(input) === '[object Number]'; } function isDate(input) { return input instanceof Date || Object.prototype.toString.call(input) === '[object Date]'; } function map(arr, fn) { var res = [], i; for (i = 0; i < arr.length; ++i) { res.push(fn(arr[i], i)); } return res; } function hasOwnProp(a, b) { return Object.prototype.hasOwnProperty.call(a, b); } function extend(a, b) { for (var i in b) { if (hasOwnProp(b, i)) { a[i] = b[i]; } } if (hasOwnProp(b, 'toString')) { a.toString = b.toString; } if (hasOwnProp(b, 'valueOf')) { a.valueOf = b.valueOf; } return a; } function createUTC (input, format, locale, strict) { return createLocalOrUTC(input, format, locale, strict, true).utc(); } function defaultParsingFlags() { // We need to deep clone this object. return { empty : false, unusedTokens : [], unusedInput : [], overflow : -2, charsLeftOver : 0, nullInput : false, invalidMonth : null, invalidFormat : false, userInvalidated : false, iso : false, parsedDateParts : [], meridiem : null, rfc2822 : false, weekdayMismatch : false }; } function getParsingFlags(m) { if (m._pf == null) { m._pf = defaultParsingFlags(); } return m._pf; } var some; if (Array.prototype.some) { some = Array.prototype.some; } else { some = function (fun) { var t = Object(this); var len = t.length >>> 0; for (var i = 0; i < len; i++) { if (i in t && fun.call(this, t[i], i, t)) { return true; } } return false; }; } function isValid(m) { if (m._isValid == null) { var flags = getParsingFlags(m); var parsedParts = some.call(flags.parsedDateParts, function (i) { return i != null; }); var isNowValid = !isNaN(m._d.getTime()) && flags.overflow < 0 && !flags.empty && !flags.invalidMonth && !flags.invalidWeekday && !flags.weekdayMismatch && !flags.nullInput && !flags.invalidFormat && !flags.userInvalidated && (!flags.meridiem || (flags.meridiem && parsedParts)); if (m._strict) { isNowValid = isNowValid && flags.charsLeftOver === 0 && flags.unusedTokens.length === 0 && flags.bigHour === undefined; } if (Object.isFrozen == null || !Object.isFrozen(m)) { m._isValid = isNowValid; } else { return isNowValid; } } return m._isValid; } function createInvalid (flags) { var m = createUTC(NaN); if (flags != null) { extend(getParsingFlags(m), flags); } else { getParsingFlags(m).userInvalidated = true; } return m; } // Plugins that add properties should also add the key here (null value), // so we can properly clone ourselves. var momentProperties = hooks.momentProperties = []; function copyConfig(to, from) { var i, prop, val; if (!isUndefined(from._isAMomentObject)) { to._isAMomentObject = from._isAMomentObject; } if (!isUndefined(from._i)) { to._i = from._i; } if (!isUndefined(from._f)) { to._f = from._f; } if (!isUndefined(from._l)) { to._l = from._l; } if (!isUndefined(from._strict)) { to._strict = from._strict; } if (!isUndefined(from._tzm)) { to._tzm = from._tzm; } if (!isUndefined(from._isUTC)) { to._isUTC = from._isUTC; } if (!isUndefined(from._offset)) { to._offset = from._offset; } if (!isUndefined(from._pf)) { to._pf = getParsingFlags(from); } if (!isUndefined(from._locale)) { to._locale = from._locale; } if (momentProperties.length > 0) { for (i = 0; i < momentProperties.length; i++) { prop = momentProperties[i]; val = from[prop]; if (!isUndefined(val)) { to[prop] = val; } } } return to; } var updateInProgress = false; // Moment prototype object function Moment(config) { copyConfig(this, config); this._d = new Date(config._d != null ? config._d.getTime() : NaN); if (!this.isValid()) { this._d = new Date(NaN); } // Prevent infinite loop in case updateOffset creates new moment // objects. if (updateInProgress === false) { updateInProgress = true; hooks.updateOffset(this); updateInProgress = false; } } function isMoment (obj) { return obj instanceof Moment || (obj != null && obj._isAMomentObject != null); } function absFloor (number) { if (number < 0) { // -0 -> 0 return Math.ceil(number) || 0; } else { return Math.floor(number); } } function toInt(argumentForCoercion) { var coercedNumber = +argumentForCoercion, value = 0; if (coercedNumber !== 0 && isFinite(coercedNumber)) { value = absFloor(coercedNumber); } return value; } // compare two arrays, return the number of differences function compareArrays(array1, array2, dontConvert) { var len = Math.min(array1.length, array2.length), lengthDiff = Math.abs(array1.length - array2.length), diffs = 0, i; for (i = 0; i < len; i++) { if ((dontConvert && array1[i] !== array2[i]) || (!dontConvert && toInt(array1[i]) !== toInt(array2[i]))) { diffs++; } } return diffs + lengthDiff; } function warn(msg) { if (hooks.suppressDeprecationWarnings === false && (typeof console !== 'undefined') && console.warn) { console.warn('Deprecation warning: ' + msg); } } function deprecate(msg, fn) { var firstTime = true; return extend(function () { if (hooks.deprecationHandler != null) { hooks.deprecationHandler(null, msg); } if (firstTime) { var args = []; var arg; for (var i = 0; i < arguments.length; i++) { arg = ''; if (typeof arguments[i] === 'object') { arg += '\n[' + i + '] '; for (var key in arguments[0]) { arg += key + ': ' + arguments[0][key] + ', '; } arg = arg.slice(0, -2); // Remove trailing comma and space } else { arg = arguments[i]; } args.push(arg); } warn(msg +
```

'\nArguments: ' + Array.prototype.slice.call(args).join('') + '\n' + (new Error()).stack); firstTime = false; } return fn.apply(this, arguments); }, fn); } var deprecations = {}; function deprecateSimple(name, msg) { if (hooks.deprecationHandler != null) { hooks.deprecationHandler(name, msg); } if (!deprecations[name]) { warn(msg); deprecations[name] = true; } } hooks.suppressDeprecationWarnings = false; hooks.deprecationHandler = null; function isFunction(input) { return input instanceof Function || Object.prototype.toString.call(input) === '[object Function]'; } function set (config) { var prop, i; for (i in config) { prop = config[i]; if (isFunction(prop)) { this[i] = prop; } else { this['_' + i] = prop; } } this._config = config; // Lenient ordinal parsing accepts just a number in addition to // number + (possibly) stuff coming from _dayOfMonthOrdinalParse. // TODO: Remove "ordinalParse" fallback in next major release. this._dayOfMonthOrdinalParseLenient = new RegExp((this._dayOfMonthOrdinalParse.source || this._ordinalParse.source) + '|' + (/\d{1,2}/).source); } function mergeConfigs(parentConfig, childConfig) { var res = extend({}, parentConfig), prop; for (prop in childConfig) { if (hasOwnProp(childConfig, prop)) { if (isObject(parentConfig[prop]) && isObject(childConfig[prop])) { res[prop] = {}; extend(res[prop], parentConfig[prop]); extend(res[prop], childConfig[prop]); } else if (childConfig[prop] != null) { res[prop] = childConfig[prop]; } else { delete res[prop]; } } } for (prop in parentConfig) { if (hasOwnProp(parentConfig, prop) && !hasOwnProp(childConfig, prop) && isObject(parentConfig[prop])) { // make sure changes to properties don't modify parent config res[prop] = extend({}, res[prop]); } } return res; } function Locale(config) { if (config != null) { this.set(config); } } var keys; if (Object.keys) { keys = Object.keys; } else { keys = function (obj) { var i, res = []; for (i in obj) { if (hasOwnProp(obj, i)) { res.push(i); } } return res; }; } var defaultCalendar = { sameDay : '[Today at] LT', nextDay : '[Tomorrow at] LT', nextWeek : 'dddd [at] LT', lastDay : '[Yesterday at] LT', lastWeek : '[Last] dddd [at] LT', sameElse : 'L' }; function calendar (key, mom, now) { var output = this._calendar[key] || this._calendar['sameElse']; return isFunction(output) ? output.call(mom, now) : output; } var defaultLongDateFormat = { LTS : 'h:mm:ss A', LT : 'h:mm A', L : 'MM/DD/YYYY', LL : 'MMMM D, YYYY', LLL : 'MMMM D, YYYY h:mm A', LLLL : 'dddd, MMMM D, YYYY h:mm A' }; function longDateFormat (key) { var format = this._longDateFormat[key], formatUpper = this._longDateFormat[key.toUpperCase()]; if (format || !formatUpper) { return format; } this._longDateFormat[key] = formatUpper.replace(/MMMM|MM|DD|dddd/g, function (val) { return val.slice(1); }); return this._longDateFormat[key]; } var defaultInvalidDate = 'Invalid date'; function invalidDate () { return this._invalidDate; } var defaultOrdinal = '%d'; var defaultDayOfMonthOrdinalParse = /\d{1,2}/; function ordinal (number) { return this._ordinal.replace('%d', number); } var defaultRelativeTime = { 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', M : 'a month', MM : '%d months', y : 'a year', yy : '%d years' }; function relativeTime (number, withoutSuffix, string, isFuture) { var output = this._relativeTime[string]; return (isFunction(output)) ? output(number, withoutSuffix, string, isFuture) : output.replace(/%d/i, number); } function pastFuture (diff, output) { var format = this._relativeTime[diff > 0 ? 'future' : 'past']; return isFunction(format) ? format(output) : format.replace(/%s/i, output); } var aliases = {}; function addUnitAlias (unit, shorthand) { var lowerCase = unit.toLowerCase(); aliases[lowerCase] = aliases[lowerCase + 's'] = aliases[shorthand] = unit; } function normalizeUnits(units) { return typeof units === 'string' ? aliases[units] || aliases[units.toLowerCase()] : undefined; } function normalizeObjectUnits(inputObject) { var normalizedInput = {}, normalizedProp, prop; for (prop in inputObject) { if (hasOwnProp(inputObject, prop)) { normalizedProp = normalizeUnits(prop); if (normalizedProp) { normalizedInput[normalizedProp] = inputObject[prop]; } } } return normalizedInput; } var priorities = {}; function addUnitPriority(unit, priority) { priorities[unit] = priority; } function getPrioritizedUnits(unitsObj) { var units = []; for (var u in unitsObj) { units.push({unit: u, priority: priorities[u]}); } units.sort(function (a, b) { return a.priority - b.priority; }); return units; } function zeroFill(number, targetLength, forceSign) { var absNumber = '' + Math.abs(number), zerosToFill = targetLength - absNumber.length, sign = number >= 0; return (sign ? (forceSign ? '+' : '') : '-') + Math.pow(10, Math.max(0, zerosToFill)).toString().substr(1) + absNumber; } var formattingTokens = /(\[[^\[]*\])|(\\)?([Hh]mm(ss)?|Mo|MM?M?M?|Do|DDDo|DD?D?D?|ddd?d?|do?|w[o|w]?|W[o|W]?|Qo?|YYYYYY|YYYYY|YYYY|YY|gg(ggg?)?|GG(GGG?)?|e|E|a|A|hh?|HH?|kk?|mm?|ss?|S{1,9}|x|X|zz?|ZZ?|.)/g; var localFormattingTokens = /(\[[^\[]*\])|(\\)?(LTS|LT|LL?L?L?|l{1,4})/g; var formatFunctions = {}; var formatTokenFunctions = {}; // token: 'M' // padded: ['MM', 2] // ordinal: 'Mo' // callback: function () { this.month() + 1 } function addFormatToken (token, padded, ordinal, callback) { var func = callback; if (typeof callback === 'string') { func = function () { return this[callback](); }; } if (token) { formatTokenFunctions[token] = func; } if (padded) { formatTokenFunctions[padded[0]] = function () { return zeroFill(func.apply(this, arguments), padded[1], padded[2]); }; } if (ordinal) { formatTokenFunctions[ordinal] = function () { return this.localeData().ordinal(func.apply(this, arguments), token); }; } } function removeFormattingTokens(input) { if (input.match(/\[[\s\S]/)) { return input.replace(/^\[|\]$/g, ''); } return input.replace(/\\/g, ''); } function makeFormatFunction(format) { var array = format.match(formattingTokens), i, length; for (i = 0, length = array.length; i < length; i++) { if (formatTokenFunctions[array[i]]) { array[i] = formatTokenFunctions[array[i]]; } else { array[i] = removeFormattingTokens(array[i]); } } return function (mom) { var output = '', i;

for (i = 0; i < length; i++) { output += isFunction(array[i]) ? array[i].call(mom, format) : array[i]; } return output; }; } // format date using native date object function formatMoment(m, format) { if (!m.isValid()) { return m.localeData().invalidDate(); } format = expandFormat(format, m.localeData()); formatFunctions[format] = formatFunctions[format] || makeFormatFunction(format); return formatFunctions[format](m); } function expandFormat(format, locale) { var i = 5; function replaceLongDateFormatTokens(input) { return locale.longDateFormat(input) || input; } localFormattingTokens.lastIndex = 0; while (i >= 0 && localFormattingTokens.test(format)) { format = format.replace(localFormattingTokens, replaceLongDateFormatTokens); localFormattingTokens.lastIndex = 0; i -= 1; } return format; } var match1 = /\d/; // 0 - 9 var match2 = /\d\d/; // 00 - 99 var match3 = /\d{3}/; // 000 - 999 var match4 = /\d{4}/; // 0000 - 9999 var match6 = /[+-]?\d{6}/; // -999999 - 999999 var match1to2 = /\d\d?/; // 0 - 99 var match3to4 = /\d\d\d\d?/; // 999 - 9999 var match5to6 = /\d\d\d\d\d\d?/; // 99999 - 999999 var match1to3 = /\d{1,3}/; // 0 - 999 var match1to4 = /\d{1,4}/; // 0 - 9999 var match1to6 = /[+-]?\d{1,6}/; // -999999 - 999999 var matchUnsigned = /\d+/; // 0 - inf var matchSigned = /[+-]?\d+/; // -inf - inf var matchOffset = /Z|[+-]\d\d:?\d\d/gi; // +00:00 -00:00 +0000 -0000 or Z var matchShortOffset = /Z|[+-]\d\d(?::?\d\d)?/gi; // +00 -00 +00:00 -00:00 +0000 -0000 or Z var matchTimestamp = /[+-]?\d+(\.\d{1,3})?/; // 123456789 123456789.123 // any word (or two) characters or numbers including two/three word month in arabic. // includes scottish gaelic two word and hyphenated months var matchWord = /[0-9]{0,256}['a-z\u00A0-\u05FF\u0700-\uD7FF\uF900-\uFDCF\uFDF0-\uFF07\uFF10-\uFFEF]{1,256}|[\u0600-\u06FF\/]{1,256}(\s*?[\u0600-\u06FF]{1,256}){1,2}/i; var regexes = {}; function addRegexToken (token, regex, strictRegex) { regexes[token] = isFunction(regex) ? regex : function (isStrict, localeData) { return (isStrict && strictRegex) ? strictRegex : regex; }; } function getParseRegexForToken (token, config) { if (!hasOwnProp(regexes, token)) { return new RegExp(unescapeFormat(token)); } return regexes[token](config._strict, config._locale); } // Code from http://stackoverflow.com/questions/3561493/is-there-a-regexp-escape-function-in-javascript function unescapeFormat(s) { return regexEscape(s.replace('\\', '').replace(/\\(\[)|\\(\])|\[([^\]\[]*)\]|\\(.)/g, function (matched, p1, p2, p3, p4) { return p1 || p2 || p3 || p4; })); } function regexEscape(s) { return s.replace(/[-\/\\^$*+?.()|[\]{}]/g, '\\$&'); } var tokens = {}; function addParseToken (token, callback) { var i, func = callback; if (typeof token === 'string') { token = [token]; } if (isNumber(callback)) { func = function (input, array) { array[callback] = toInt(input); }; } for (i = 0; i < token.length; i++) { tokens[token[i]] = func; } } function addWeekParseToken (token, callback) { addParseToken(token, function (input, array, config, token) { config._w = config._w || {}; callback(input, config._w, config, token); }); } function addTimeToArrayFromToken(token, input, config) { if (input != null && hasOwnProp(tokens, token)) { tokens[token](input, config._a, config, token); } } var YEAR = 0; var MONTH = 1; var DATE = 2; var HOUR = 3; var MINUTE = 4; var SECOND = 5; var MILLISECOND = 6; var WEEK = 7; var WEEKDAY = 8; // FORMATTING addFormatToken('Y', 0, 0, function () { var y = this.year(); return y <= 9999 ? '' + y : '+' + y; }); addFormatToken(0, ['YY', 2], 0, function () { return this.year() % 100; }); addFormatToken(0, ['YYYY', 4], 0, 'year'); addFormatToken(0, ['YYYYY', 5], 0, 'year'); addFormatToken(0, ['YYYYYY', 6, true], 0, 'year'); // ALIASES addUnitAlias('year', 'y'); // PRIORITIES addUnitPriority('year', 1); // PARSING addRegexToken('Y', matchSigned); addRegexToken('YY', match1to2, match2); addRegexToken('YYYY', match1to4, match4); addRegexToken('YYYYY', match1to6, match6); addRegexToken('YYYYYY', match1to6, match6); addParseToken(['YYYYY', 'YYYYYY'], YEAR); addParseToken('YYYY', function (input, array) { array[YEAR] = input.length === 2 ? hooks.parseTwoDigitYear(input) : toInt(input); }); addParseToken('YY', function (input, array) { array[YEAR] = hooks.parseTwoDigitYear(input); }); addParseToken('Y', function (input, array) { array[YEAR] = parseInt(input, 10); }); // HELPERS function daysInYear(year) { return isLeapYear(year) ? 366 : 365; } function isLeapYear(year) { return (year % 4 === 0 && year % 100 !== 0) || year % 400 === 0; } // HOOKS hooks.parseTwoDigitYear = function (input) { return toInt(input) + (toInt(input) > 68 ? 1900 : 2000); }; // MOMENTS var getSetYear = makeGetSet('FullYear', true); function getIsLeapYear () { return isLeapYear(this.year()); } function makeGetSet (unit, keepTime) { return function (value) { if (value != null) { set$1(this, unit, value); hooks.updateOffset(this, keepTime); return this; } else { return get(this, unit); } }; } function get (mom, unit) { return mom.isValid() ? mom._d['get' + (mom._isUTC ? 'UTC' : '') + unit]() : NaN; } function set$1 (mom, unit, value) { if (mom.isValid() && !isNaN(value)) { if (unit === 'FullYear' && isLeapYear(mom.year()) && mom.month() === 1 && mom.date() === 29) { mom._d['set' + (mom._isUTC ? 'UTC' : '') + unit](value, mom.month(), daysInMonth(value, mom.month())); } else { mom._d['set' + (mom._isUTC ? 'UTC' : '') + unit](value); } } } // MOMENTS function stringGet (units) { units = normalizeUnits(units); if (isFunction(this[units])) { return this[units](); } return this; } function stringSet (units, value) { if (typeof units === 'object') { units = normalizeObjectUnits(units); var prioritized = getPrioritizedUnits(units); for (var i = 0; i < prioritized.length; i++) { this[prioritized[i].unit](units[prioritized[i].unit]); } } else { units = normalizeUnits(units); if (isFunction(this[units])) { return this[units](value); } } return this; } function mod(n, x) { return ((n % x) + x) % x; } var indexOf; if (Array.prototype.indexOf) { indexOf =

```
Array.prototype.indexOf; } else { indexOf = function (o) { // I know var i; for (i = 0; i < this.length; ++i) { if (this[i] === o) { return i; } } return -1; }; } function daysInMonth(year, month) { if (isNaN(year) || isNaN(month)) { return NaN; } var modMonth = mod(month, 12); year += (month - modMonth) / 12; return modMonth === 1 ? (isLeapYear(year) ? 29 : 28) : (31 - modMonth % 7 % 2); } // FORMATTING addFormatToken('M', ['MM', 2], 'Mo', function () { return this.month() + 1; }); addFormatToken('MMM', 0, 0, function (format) { return this.localeData().monthsShort(this, format); }); addFormatToken('MMMM', 0, 0, function (format) { return this.localeData().months(this, format); }); // ALIASES addUnitAlias('month', 'M'); // PRIORITY addUnitPriority('month', 8); // PARSING addRegexToken('M', match1to2); addRegexToken('MM', match1to2, match2); addRegexToken('MMM', function (isStrict, locale) { return locale.monthsShortRegex(isStrict); }); addRegexToken('MMMM', function (isStrict, locale) { return locale.monthsRegex(isStrict); }); addParseToken(['M', 'MM'], function (input, array) { array[MONTH] = toInt(input) - 1; }); addParseToken(['MMM', 'MMMM'], function (input, array, config, token) { var month = config._locale.monthsParse(input, token, config._strict); // if we didn't find a month name, mark the date as invalid. if (month != null) { array[MONTH] = month; } else { getParsingFlags(config).invalidMonth = input; } }); // LOCALES var MONTHS_IN_FORMAT = /D[oD]?(\[[^\[\]]*\]|\s)+MMMM?/; var defaultLocaleMonths = 'January_February_March_April_May_June_July_August_September_October_November_December'.split('_'); function localeMonths (m, format) { if (!m) { return isArray(this._months) ? this._months : this._months['standalone']; } return isArray(this._months) ? this._months[m.month()] : this._months[(this._months.isFormat || MONTHS_IN_FORMAT).test(format) ? 'format' : 'standalone'][m.month()]; } var defaultLocaleMonthsShort = 'Jan_Feb_Mar_Apr_May_Jun_Jul_Aug_Sep_Oct_Nov_Dec'.split('_'); function localeMonthsShort (m, format) { if (!m) { return isArray(this._monthsShort) ? this._monthsShort : this._monthsShort['standalone']; } return isArray(this._monthsShort) ? this._monthsShort[m.month()] : this._monthsShort[MONTHS_IN_FORMAT.test(format) ? 'format' : 'standalone'][m.month()]; } function handleStrictParse(monthName, format, strict) { var i, ii, mom, llc = monthName.toLocaleLowerCase(); if (!this._monthsParse) { // this is not used this._monthsParse = []; this._longMonthsParse = []; this._shortMonthsParse = []; for (i = 0; i < 12; ++i) { mom = createUTC([2000, i]); this._shortMonthsParse[i] = this.monthsShort(mom, '').toLocaleLowerCase(); this._longMonthsParse[i] = this.months(mom, '').toLocaleLowerCase(); } } if (strict) { if (format === 'MMM') { ii = indexOf.call(this._shortMonthsParse, llc); return ii !== -1 ? ii : null; } else { ii = indexOf.call(this._longMonthsParse, llc); return ii !== -1 ? ii : null; } } else { if (format === 'MMM') { ii = indexOf.call(this._shortMonthsParse, llc); if (ii !== -1) { return ii; } ii = indexOf.call(this._longMonthsParse, llc); return ii !== -1 ? ii : null; } else { ii = indexOf.call(this._longMonthsParse, llc); if (ii !== -1) { return ii; } ii = indexOf.call(this._shortMonthsParse, llc); return ii !== -1 ? ii : null; } } } function localeMonthsParse (monthName, format, strict) { var i, mom, regex; if (this._monthsParseExact) { return handleStrictParse.call(this, monthName, format, strict); } if (!this._monthsParse) { this._monthsParse = []; this._longMonthsParse = []; this._shortMonthsParse = []; } // TODO: add sorting // Sorting makes sure if one month (or abbr) is a prefix of another // see sorting in computeMonthsParse for (i = 0; i < 12; i++) { // make the regex if we don't have it already mom = createUTC([2000, i]); if (strict && !this._longMonthsParse[i]) { this._longMonthsParse[i] = new RegExp('^' + this.months(mom, '').replace('.', '') + '$', 'i'); this._shortMonthsParse[i] = new RegExp('^' + this.monthsShort(mom, '').replace('.', '') + '$', 'i'); } if (!strict && !this._monthsParse[i]) { regex = '^' + this.months(mom, '') + '|^' + this.monthsShort(mom, ''); this._monthsParse[i] = new RegExp(regex.replace('.', ''), 'i'); } // test the regex if (strict && format === 'MMMM' && this._longMonthsParse[i].test(monthName)) { return i; } else if (strict && format === 'MMM' && this._shortMonthsParse[i].test(monthName)) { return i; } else if (!strict && this._monthsParse[i].test(monthName)) { return i; } } } // MOMENTS function setMonth (mom, value) { var dayOfMonth; if (!mom.isValid()) { // No op return mom; } if (typeof value === 'string') { if (/^\d+$/.test(value)) { value = toInt(value); } else { value = mom.localeData().monthsParse(value); // TODO: Another silent failure? if (!isNumber(value)) { return mom; } } } dayOfMonth = Math.min(mom.date(), daysInMonth(mom.year(), value)); mom._d['set' + (mom._isUTC ? 'UTC' : '') + 'Month'](value, dayOfMonth); return mom; } function getSetMonth (value) { if (value != null) { setMonth(this, value); hooks.updateOffset(this, true); return this; } else { return get(this, 'Month'); } } function getDaysInMonth () { return daysInMonth(this.year(), this.month()); } var defaultMonthsShortRegex = matchWord; function monthsShortRegex (isStrict) { if (this._monthsParseExact) { if (!hasOwnProp(this, '_monthsRegex')) { computeMonthsParse.call(this); } if (isStrict) { return this._monthsShortStrictRegex; } else { return this._monthsShortRegex; } } else { if (!hasOwnProp(this, '_monthsShortRegex')) { this._monthsShortRegex = defaultMonthsShortRegex; } return this._monthsShortStrictRegex && isStrict ? this._monthsShortStrictRegex : this._monthsShortRegex; } } var defaultMonthsRegex = matchWord; function monthsRegex (isStrict) { if (this._monthsParseExact) { if (!hasOwnProp(this, '_monthsRegex')) { computeMonthsParse.call(this); } if (isStrict) { return this._monthsStrictRegex; } else { return this._monthsRegex; } } else { if (!hasOwnProp(this, '_monthsRegex')) {
```

this._monthsRegex = defaultMonthsRegex; } return this._monthsStrictRegex && isStrict ? this._monthsStrictRegex : this._monthsRegex; } } function computeMonthsParse () { function cmpLenRev(a, b) { return b.length - a.length; } var shortPieces = [], longPieces = [], mixedPieces = [], i, mom; for (i = 0; i < 12; i++) { // make the regex if we don't have it already mom = createUTC([2000, i]); shortPieces.push(this.monthsShort(mom, '')); longPieces.push(this.months(mom, '')); mixedPieces.push(this.months(mom, '')); mixedPieces.push(this.monthsShort(mom, '')); } // Sorting makes sure if one month (or abbr) is a prefix of another it // will match the longer piece. shortPieces.sort(cmpLenRev); longPieces.sort(cmpLenRev); mixedPieces.sort(cmpLenRev); for (i = 0; i < 12; i++) { shortPieces[i] = regexEscape(shortPieces[i]); longPieces[i] = regexEscape(longPieces[i]); } for (i = 0; i < 24; i++) { mixedPieces[i] = regexEscape(mixedPieces[i]); } this._monthsRegex = new RegExp('^(' + mixedPieces.join('|') + ')', 'i'); this._monthsShortRegex = this._monthsRegex; this._monthsStrictRegex = new RegExp('^(' + longPieces.join('|') + ')', 'i'); this._monthsShortStrictRegex = new RegExp('^(' + shortPieces.join('|') + ')', 'i'); } function createDate (y, m, d, h, M, s, ms) { // can't just apply() to create a date: // https://stackoverflow.com/q/181348 var date = new Date(y, m, d, h, M, s, ms); // the date constructor remaps years 0-99 to 1900-1999 if (y < 100 && y >= 0 && isFinite(date.getFullYear())) { date.setFullYear(y); } return date; } function createUTCDate (y) { var date = new Date(Date.UTC.apply(null, arguments)); // the Date.UTC function remaps years 0-99 to 1900-1999 if (y < 100 && y >= 0 && isFinite(date.getUTCFullYear())) { date.setUTCFullYear(y); } return date; } // start-of-first-week - start-of-year function firstWeekOffset(year, dow, doy) { var // first-week day -- which january is always in the first week (4 for iso, 1 for other) fwd = 7 + dow - doy, // first-week day local weekday -- which local weekday is fwd fwdlw = (7 + createUTCDate(year, 0, fwd).getUTCDay() - dow) % 7; return -fwdlw + fwd - 1; } // https://en.wikipedia.org/wiki/ISO_week_date#Calculating_a_date_given_the_year.2C_week_number_and_weekday function dayOfYearFromWeeks(year, week, weekday, dow, doy) { var localWeekday = (7 + weekday - dow) % 7, weekOffset = firstWeekOffset(year, dow, doy), dayOfYear = 1 + 7 * (week - 1) + localWeekday + weekOffset, resYear, resDayOfYear; if (dayOfYear <= 0) { resYear = year - 1; resDayOfYear = daysInYear(resYear) + dayOfYear; } else if (dayOfYear > daysInYear(year)) { resYear = year + 1; resDayOfYear = dayOfYear - daysInYear(year); } else { resYear = year; resDayOfYear = dayOfYear; } return { year: resYear, dayOfYear: resDayOfYear }; } function weekOfYear(mom, dow, doy) { var weekOffset = firstWeekOffset(mom.year(), dow, doy), week = Math.floor((mom.dayOfYear() - weekOffset - 1) / 7) + 1, resWeek, resYear; if (week < 1) { resYear = mom.year() - 1; resWeek = week + weeksInYear(resYear, dow, doy); } else if (week > weeksInYear(mom.year(), dow, doy)) { resWeek = week - weeksInYear(mom.year(), dow, doy); resYear = mom.year() + 1; } else { resYear = mom.year(); resWeek = week; } return { week: resWeek, year: resYear }; } function weeksInYear(year, dow, doy) { var weekOffset = firstWeekOffset(year, dow, doy), weekOffsetNext = firstWeekOffset(year + 1, dow, doy); return (daysInYear(year) - weekOffset + weekOffsetNext) / 7; } // FORMATTING addFormatToken('w', ['ww', 2], 'wo', 'week'); addFormatToken('W', ['WW', 2], 'Wo', 'isoWeek'); // ALIASES addUnitAlias('week', 'w'); addUnitAlias('isoWeek', 'W'); // PRIORITIES addUnitPriority('week', 5); addUnitPriority('isoWeek', 5); // PARSING addRegexToken('w', match1to2); addRegexToken('ww', match1to2, match2); addRegexToken('W', match1to2); addRegexToken('WW', match1to2, match2); addWeekParseToken(['w', 'ww', 'W', 'WW'], function (input, week, config, token) { week[token.substr(0, 1)] = toInt(input); }); // HELPERS // LOCALES function localeWeek (mom) { return weekOfYear(mom, this._week.dow, this._week.doy).week; } var defaultLocaleWeek = { dow : 0, // Sunday is the first day of the week. doy : 6 // The week that contains Jan 1st is the first week of the year. }; function localeFirstDayOfWeek () { return this._week.dow; } function localeFirstDayOfYear () { return this._week.doy; } // MOMENTS function getSetWeek (input) { var week = this.localeData().week(this); return input == null ? week : this.add((input - week) * 7, 'd'); } function getSetISOWeek (input) { var week = weekOfYear(this, 1, 4).week; return input == null ? week : this.add((input - week) * 7, 'd'); } // FORMATTING addFormatToken('d', 0, 'do', 'day'); addFormatToken('dd', 0, 0, function (format) { return this.localeData().weekdaysMin(this, format); }); addFormatToken('ddd', 0, 0, function (format) { return this.localeData().weekdaysShort(this, format); }); addFormatToken('dddd', 0, 0, function (format) { return this.localeData().weekdays(this, format); }); addFormatToken('e', 0, 0, 'weekday'); addFormatToken('E', 0, 0, 'isoWeekday'); // ALIASES addUnitAlias('day', 'd'); addUnitAlias('weekday', 'e'); addUnitAlias('isoWeekday', 'E'); // PRIORITY addUnitPriority('day', 11); addUnitPriority('weekday', 11); addUnitPriority('isoWeekday', 11); // PARSING addRegexToken('d', match1to2); addRegexToken('e', match1to2); addRegexToken('E', match1to2); addRegexToken('dd', function (isStrict, locale) { return locale.weekdaysMinRegex(isStrict); }); addRegexToken('ddd', function (isStrict, locale) { return locale.weekdaysShortRegex(isStrict); }); addRegexToken('dddd', function (isStrict, locale) { return locale.weekdaysRegex(isStrict); }); addWeekParseToken(['dd', 'ddd', 'dddd'], function (input, week, config, token) { var weekday = config._locale.weekdaysParse(input, token, config._strict); // if we didn't get a weekday name, mark the date as invalid if (weekday != null) { week.d = weekday; } else { getParsingFlags(config).invalidWeekday = input; } }); addWeekParseToken(['d', 'e', 'E'], function (input, week, config, token) { week[token] = toInt(input); }); // HELPERS

```
function parseWeekday(input, locale) { if (typeof input !== 'string') { return input; } if (!isNaN(input)) { return parseInt(input, 10); } input = locale.weekdaysParse(input); if (typeof input === 'number') { return input; } return null; } function parseIsoWeekday(input, locale) { if (typeof input === 'string') { return locale.weekdaysParse(input) % 7 || 7; } return isNaN(input) ? null : input; } // LOCALES var defaultLocaleWeekdays = 'Sunday_Monday_Tuesday_Wednesday_Thursday_Friday_Saturday'.split('_'); function localeWeekdays (m, format) { if (!m) { return isArray(this._weekdays) ? this._weekdays : this._weekdays['standalone']; } return isArray(this._weekdays) ? this._weekdays[m.day()] : this._weekdays[this._weekdays.isFormat.test(format) ? 'format' : 'standalone'][m.day()]; } var defaultLocaleWeekdaysShort = 'Sun_Mon_Tue_Wed_Thu_Fri_Sat'.split('_'); function localeWeekdaysShort (m) { return (m) ? this._weekdaysShort[m.day()] : this._weekdaysShort; } var defaultLocaleWeekdaysMin = 'Su_Mo_Tu_We_Th_Fr_Sa'.split('_'); function localeWeekdaysMin (m) { return (m) ? this._weekdaysMin[m.day()] : this._weekdaysMin; } function handleStrictParse$1(weekdayName, format, strict) { var i, ii, mom, llc = weekdayName.toLocaleLowerCase(); if (!this._weekdaysParse) { this._weekdaysParse = []; this._shortWeekdaysParse = []; this._minWeekdaysParse = []; for (i = 0; i < 7; ++i) { mom = createUTC([2000, 1]).day(i); this._minWeekdaysParse[i] = this.weekdaysMin(mom, '').toLocaleLowerCase(); this._shortWeekdaysParse[i] = this.weekdaysShort(mom, '').toLocaleLowerCase(); this._weekdaysParse[i] = this.weekdays(mom, '').toLocaleLowerCase(); } } if (strict) { if (format === 'dddd') { ii = indexOf.call(this._weekdaysParse, llc); return ii !== -1 ? ii : null; } else if (format === 'ddd') { ii = indexOf.call(this._shortWeekdaysParse, llc); return ii !== -1 ? ii : null; } else { ii = indexOf.call(this._minWeekdaysParse, llc); return ii !== -1 ? ii : null; } } else { if (format === 'dddd') { ii = indexOf.call(this._weekdaysParse, llc); if (ii !== -1) { return ii; } ii = indexOf.call(this._shortWeekdaysParse, llc); if (ii !== -1) { return ii; } ii = indexOf.call(this._minWeekdaysParse, llc); return ii !== -1 ? ii : null; } else if (format === 'ddd') { ii = indexOf.call(this._shortWeekdaysParse, llc); if (ii !== -1) { return ii; } ii = indexOf.call(this._weekdaysParse, llc); if (ii !== -1) { return ii; } ii = indexOf.call(this._minWeekdaysParse, llc); return ii !== -1 ? ii : null; } else { ii = indexOf.call(this._minWeekdaysParse, llc); if (ii !== -1) { return ii; } ii = indexOf.call(this._weekdaysParse, llc); if (ii !== -1) { return ii; } ii = indexOf.call(this._shortWeekdaysParse, llc); return ii !== -1 ? ii : null; } } } function localeWeekdaysParse (weekdayName, format, strict) { var i, mom, regex; if (this._weekdaysParseExact) { return handleStrictParse$1.call(this, weekdayName, format, strict); } if (!this._weekdaysParse) { this._weekdaysParse = []; this._minWeekdaysParse = []; this._shortWeekdaysParse = []; this._fullWeekdaysParse = []; } for (i = 0; i < 7; i++) { // make the regex if we don't have it already mom = createUTC([2000, 1]).day(i); if (strict && !this._fullWeekdaysParse[i]) { this._fullWeekdaysParse[i] = new RegExp('^' + this.weekdays(mom, '').replace('.', '\\.?') + '$', 'i'); this._shortWeekdaysParse[i] = new RegExp('^' + this.weekdaysShort(mom, '').replace('.', '\\.?') + '$', 'i'); this._minWeekdaysParse[i] = new RegExp('^' + this.weekdaysMin(mom, '').replace('.', '\\.?') + '$', 'i'); } if (!this._weekdaysParse[i]) { regex = '^' + this.weekdays(mom, '') + '|^' + this.weekdaysShort(mom, '') + '|^' + this.weekdaysMin(mom, ''); this._weekdaysParse[i] = new RegExp(regex.replace('.', ''), 'i'); } // test the regex if (strict && format === 'dddd' && this._fullWeekdaysParse[i].test(weekdayName)) { return i; } else if (strict && format === 'ddd' && this._shortWeekdaysParse[i].test(weekdayName)) { return i; } else if (strict && format === 'dd' && this._minWeekdaysParse[i].test(weekdayName)) { return i; } else if (!strict && this._weekdaysParse[i].test(weekdayName)) { return i; } } } // MOMENTS function getSetDayOfWeek (input) { if (!this.isValid()) { return input != null ? this : NaN; } var day = this._isUTC ? this._d.getUTCDay() : this._d.getDay(); if (input != null) { input = parseWeekday(input, this.localeData()); return this.add(input - day, 'd'); } else { return day; } } function getSetLocaleDayOfWeek (input) { if (!this.isValid()) { return input != null ? this : NaN; } var weekday = (this.day() + 7 - this.localeData()._week.dow) % 7; return input == null ? weekday : this.add(input - weekday, 'd'); } function getSetISODayOfWeek (input) { if (!this.isValid()) { return input != null ? this : NaN; } // behaves the same as moment#day except // as a getter, returns 7 instead of 0 (1-7 range instead of 0-6) // as a setter, sunday should belong to the previous week. if (input != null) { var weekday = parseIsoWeekday(input, this.localeData()); return this.day(this.day() % 7 ? weekday : weekday - 7); } else { return this.day() || 7; } } var defaultWeekdaysRegex = matchWord; function weekdaysRegex (isStrict) { if (this._weekdaysParseExact) { if (!hasOwnProp(this, '_weekdaysRegex')) { computeWeekdaysParse.call(this); } if (isStrict) { return this._weekdaysStrictRegex; } else { return this._weekdaysRegex; } } else { if (!hasOwnProp(this, '_weekdaysRegex')) { this._weekdaysRegex = defaultWeekdaysRegex; } return this._weekdaysStrictRegex && isStrict ? this._weekdaysStrictRegex : this._weekdaysRegex; } } var defaultWeekdaysShortRegex = matchWord; function weekdaysShortRegex (isStrict) { if (this._weekdaysParseExact) { if (!hasOwnProp(this, '_weekdaysRegex')) { computeWeekdaysParse.call(this); } if (isStrict) { return this._weekdaysShortStrictRegex; } else { return this._weekdaysShortRegex; } } else { if (!hasOwnProp(this, '_weekdaysShortRegex')) { this._weekdaysShortRegex = defaultWeekdaysShortRegex; } return this._weekdaysShortStrictRegex && isStrict ? this._weekdaysShortStrictRegex : this._weekdaysShortRegex; } } var defaultWeekdaysMinRegex = matchWord; function weekdaysMinRegex (isStrict) { if
```

(this._weekdaysParseExact) { if (!hasOwnProp(this, '_weekdaysRegex')) { computeWeekdaysParse.call(this); } if (isStrict) { return this._weekdaysMinStrictRegex; } else { return this._weekdaysMinRegex; } } else { if (!hasOwnProp(this, '_weekdaysMinRegex')) { this._weekdaysMinRegex = defaultWeekdaysMinRegex; } return this._weekdaysMinStrictRegex && isStrict ? this._weekdaysMinStrictRegex : this._weekdaysMinRegex; } } function computeWeekdaysParse () { function cmpLenRev(a, b) { return b.length - a.length; } var minPieces = [], shortPieces = [], longPieces = [], mixedPieces = [], i, mom, minp, shortp, longp; for (i = 0; i < 7; i++) { // make the regex if we don't have it already mom = createUTC([2000, 1]).day(i); minp = this.weekdaysMin(mom, ''); shortp = this.weekdaysShort(mom, ''); longp = this.weekdays(mom, ''); minPieces.push(minp); shortPieces.push(shortp); longPieces.push(longp); mixedPieces.push(minp); mixedPieces.push(shortp); mixedPieces.push(longp); } // Sorting makes sure if one weekday (or abbr) is a prefix of another it // will match the longer piece. minPieces.sort(cmpLenRev); shortPieces.sort(cmpLenRev); longPieces.sort(cmpLenRev); mixedPieces.sort(cmpLenRev); for (i = 0; i < 7; i++) { shortPieces[i] = regexEscape(shortPieces[i]); longPieces[i] = regexEscape(longPieces[i]); mixedPieces[i] = regexEscape(mixedPieces[i]); } this._weekdaysRegex = new RegExp('^(' + mixedPieces.join('|') + ')', 'i'); this._weekdaysShortRegex = this._weekdaysRegex; this._weekdaysMinRegex = this._weekdaysRegex; this._weekdaysStrictRegex = new RegExp('^(' + longPieces.join('|') + ')', 'i'); this._weekdaysShortStrictRegex = new RegExp('^(' + shortPieces.join('|') + ')', 'i'); this._weekdaysMinStrictRegex = new RegExp('^(' + minPieces.join('|') + ')', 'i'); } // FORMATTING function hFormat() { return this.hours() % 12 || 12; } function kFormat() { return this.hours() || 24; } addFormatToken('H', ['HH', 2], 0, 'hour'); addFormatToken('h', ['hh', 2], 0, hFormat); addFormatToken('k', ['kk', 2], 0, kFormat); addFormatToken('hmm', 0, 0, function () { return '' + hFormat.apply(this) + zeroFill(this.minutes(), 2); }); addFormatToken('hmmss', 0, 0, function () { return '' + hFormat.apply(this) + zeroFill(this.minutes(), 2) + zeroFill(this.seconds(), 2); }); addFormatToken('Hmm', 0, 0, function () { return '' + this.hours() + zeroFill(this.minutes(), 2); }); addFormatToken('Hmmss', 0, 0, function () { return '' + this.hours() + zeroFill(this.minutes(), 2) + zeroFill(this.seconds(), 2); }); function meridiem (token, lowercase) { addFormatToken(token, 0, 0, function () { return this.localeData().meridiem(this.hours(), this.minutes(), lowercase); }); } meridiem('a', true); meridiem('A', false); // ALIASES addUnitAlias('hour', 'h'); // PRIORITY addUnitPriority('hour', 13); // PARSING function matchMeridiem (isStrict, locale) { return locale._meridiemParse; } addRegexToken('a', matchMeridiem); addRegexToken('A', matchMeridiem); addRegexToken('H', match1to2); addRegexToken('h', match1to2); addRegexToken('k', match1to2); addRegexToken('HH', match1to2, match2); addRegexToken('hh', match1to2, match2); addRegexToken('kk', match1to2, match2); addRegexToken('hmm', match3to4); addRegexToken('hmmss', match5to6); addRegexToken('Hmm', match3to4); addRegexToken('Hmmss', match5to6); addParseToken(['H', 'HH'], HOUR); addParseToken(['k', 'kk'], function (input, array, config) { var kInput = toInt(input); array[HOUR] = kInput === 24 ? 0 : kInput; }); addParseToken(['a', 'A'], function (input, array, config) { config._isPm = config._locale.isPM(input); config._meridiem = input; }); addParseToken(['h', 'hh'], function (input, array, config) { array[HOUR] = toInt(input); getParsingFlags(config).bigHour = true; }); addParseToken('hmm', function (input, array, config) { var pos = input.length - 2; array[HOUR] = toInt(input.substr(0, pos)); array[MINUTE] = toInt(input.substr(pos)); getParsingFlags(config).bigHour = true; }); addParseToken('hmmss', function (input, array, config) { var pos1 = input.length - 4; var pos2 = input.length - 2; array[HOUR] = toInt(input.substr(0, pos1)); array[MINUTE] = toInt(input.substr(pos1, 2)); array[SECOND] = toInt(input.substr(pos2)); getParsingFlags(config).bigHour = true; }); addParseToken('Hmm', function (input, array, config) { var pos = input.length - 2; array[HOUR] = toInt(input.substr(0, pos)); array[MINUTE] = toInt(input.substr(pos)); }); addParseToken('Hmmss', function (input, array, config) { var pos1 = input.length - 4; var pos2 = input.length - 2; array[HOUR] = toInt(input.substr(0, pos1)); array[MINUTE] = toInt(input.substr(pos1, 2)); array[SECOND] = toInt(input.substr(pos2)); }); // LOCALES function localeIsPM (input) { // IE8 Quirks Mode & IE7 Standards Mode do not allow accessing strings like arrays // Using charAt should be more compatible. return ((input + '').toLowerCase().charAt(0) === 'p'); } var defaultLocaleMeridiemParse = /[ap]\.?m?\.?/i; function localeMeridiem (hours, minutes, isLower) { if (hours > 11) { return isLower ? 'pm' : 'PM'; } else { return isLower ? 'am' : 'AM'; } } // MOMENTS // Setting the hour should keep the time, because the user explicitly // specified which hour they want. So trying to maintain the same hour (in // a new timezone) makes sense. Adding/subtracting hours does not follow // this rule. var getSetHour = makeGetSet('Hours', true); var baseConfig = { calendar: defaultCalendar, longDateFormat: defaultLongDateFormat, invalidDate: defaultInvalidDate, ordinal: defaultOrdinal, dayOfMonthOrdinalParse: defaultDayOfMonthOrdinalParse, relativeTime: defaultRelativeTime, months: defaultLocaleMonths, monthsShort: defaultLocaleMonthsShort, week: defaultLocaleWeek, weekdays: defaultLocaleWeekdays, weekdaysMin: defaultLocaleWeekdaysMin, weekdaysShort: defaultLocaleWeekdaysShort, meridiemParse: defaultLocaleMeridiemParse }; // internal storage for locale config files var locales = {}; var localeFamilies = {}; var globalLocale; function normalizeLocale(key) { return key ? key.toLowerCase().replace('_', '-') : key; } // pick the locale from the array // try ['en-au', 'en-gb'] as 'en-au', 'en-gb', 'en', as in move through the list trying each // substring from most specific to least, but move to the next array item if it's a more

specific variant than the current root function chooseLocale(names) { var i = 0, j, next, locale, split; while (i < names.length) { split = normalizeLocale(names[i]).split('-'); j = split.length; next = normalizeLocale(names[i + 1]); next = next ? next.split('-') : null; while (j > 0) { locale = loadLocale(split.slice(0, j).join('-')); if (locale) { return locale; } if (next && next.length >= j && compareArrays(split, next, true) >= j - 1) { //the next array item is better than a shallower substring of this one break; } j--; } i++; } return globalLocale; } function loadLocale(name) { var oldLocale = null; // TODO: Find a better way to register and load all the locales in Node if (!locales[name] && (typeof module !== 'undefined') && module && module.exports) { try { oldLocale = globalLocale._abbr; var aliasedRequire = require; aliasedRequire('./locale/' + name); getSetGlobalLocale(oldLocale); } catch (e) {} } return locales[name]; } // This function will load locale and then set the global locale. If // no arguments are passed in, it will simply return the current global // locale key. function getSetGlobalLocale (key, values) { var data; if (key) { if (isUndefined(values)) { data = getLocale(key); } else { data = defineLocale(key, values); } if (data) { // moment.duration._locale = moment._locale = data; globalLocale = data; } else { if ((typeof console !== 'undefined') && console.warn) { //warn user if arguments are passed but the locale could not be set console.warn('Locale ' + key + ' not found. Did you forget to load it?'); } } } return globalLocale._abbr; } function defineLocale (name, config) { if (config !== null) { var locale, parentConfig = baseConfig; config.abbr = name; if (locales[name] != null) { deprecateSimple('defineLocaleOverride', 'use moment.updateLocale(localeName, config) to change ' + 'an existing locale. moment.defineLocale(localeName, ' + 'config) should only be used for creating a new locale ' + 'See http://momentjs.com/guides/#/warnings/define-locale/ for more info.'); parentConfig = locales[name]._config; } else if (config.parentLocale != null) { if (locales[config.parentLocale] != null) { parentConfig = locales[config.parentLocale]._config; } else { locale = loadLocale(config.parentLocale); if (locale != null) { parentConfig = locale._config; } else { if (!localeFamilies[config.parentLocale]) { localeFamilies[config.parentLocale] = []; } localeFamilies[config.parentLocale].push({ name: name, config: config }); return null; } } } locales[name] = new Locale(mergeConfigs(parentConfig, config)); if (localeFamilies[name]) { localeFamilies[name].forEach(function (x) { defineLocale(x.name, x.config); }); } // backwards compat for now: also set the locale // make sure we set the locale AFTER all child locales have been // created, so we won't end up with the child locale set. getSetGlobalLocale(name); return locales[name]; } else { // useful for testing delete locales[name]; return null; } } function updateLocale(name, config) { if (config != null) { var locale, tmpLocale, parentConfig = baseConfig; // MERGE tmpLocale = loadLocale(name); if (tmpLocale != null) { parentConfig = tmpLocale._config; } config = mergeConfigs(parentConfig, config); locale = new Locale(config); locale.parentLocale = locales[name]; locales[name] = locale; // backwards compat for now: also set the locale getSetGlobalLocale(name); } else { // pass null for config to unupdate, useful for tests if (locales[name] != null) { if (locales[name].parentLocale != null) { locales[name] = locales[name].parentLocale; } else if (locales[name] != null) { delete locales[name]; } } } return locales[name]; } // returns locale data function getLocale (key) { var locale; if (key && key._locale && key._locale._abbr) { key = key._locale._abbr; } if (!key) { return globalLocale; } if (!isArray(key)) { //short-circuit everything else locale = loadLocale(key); if (locale) { return locale; } key = [key]; } return chooseLocale(key); } function listLocales() { return keys(locales); } function checkOverflow (m) { var overflow; var a = m._a; if (a && getParsingFlags(m).overflow === -2) { overflow = a[MONTH] < 0 || a[MONTH] > 11 ? MONTH : a[DATE] < 1 || a[DATE] > daysInMonth(a[YEAR], a[MONTH]) ? DATE : a[HOUR] < 0 || a[HOUR] > 24 || (a[HOUR] === 24 && (a[MINUTE] !== 0 || a[SECOND] !== 0 || a[MILLISECOND] !== 0)) ? HOUR : a[MINUTE] < 0 || a[MINUTE] > 59 ? MINUTE : a[SECOND] < 0 || a[SECOND] > 59 ? SECOND : a[MILLISECOND] < 0 || a[MILLISECOND] > 999 ? MILLISECOND : -1; if (getParsingFlags(m)._overflowDayOfYear && (overflow < YEAR || overflow > DATE)) { overflow = DATE; } if (getParsingFlags(m)._overflowWeeks && overflow === -1) { overflow = WEEK; } if (getParsingFlags(m)._overflowWeekday && overflow === -1) { overflow = WEEKDAY; } getParsingFlags(m).overflow = overflow; } return m; } // Pick the first defined of two or three arguments. function defaults(a, b, c) { if (a != null) { return a; } if (b != null) { return b; } return c; } function currentDateArray(config) { // hooks is actually the exported moment object var nowValue = new Date(hooks.now()); if (config._useUTC) { return [nowValue.getUTCFullYear(), nowValue.getUTCMonth(), nowValue.getUTCDate()]; } return [nowValue.getFullYear(), nowValue.getMonth(), nowValue.getDate()]; } // convert an array to a date. // the array should mirror the parameters below // note: all values past the year are optional and will default to the lowest possible value. // [year, month, day , hour, minute, second, millisecond] function configFromArray (config) { var i, date, input = [], currentDate, expectedWeekday, yearToUse; if (config._d) { return; } currentDate = currentDateArray(config); //compute day of the year from weeks and weekdays if (config._w && config._a[DATE] == null && config._a[MONTH] == null) { dayOfYearFromWeekInfo(config); } //if the day of the year is set, figure out what it is if (config._dayOfYear != null) { yearToUse = defaults(config._a[YEAR], currentDate[YEAR]); if (config._dayOfYear > daysInYear(yearToUse) || config._dayOfYear === 0) { getParsingFlags(config)._overflowDayOfYear = true; } date = createUTCDate(yearToUse, 0, config._dayOfYear); config._a[MONTH] = date.getUTCMonth();

```
config._a[DATE] = date.getUTCDate(); } // Default to current date. // * if no year, month, day of month are given, default to today // * if day of month is given, default month and year // * if month is given, default only year // * if year is given, don't default anything for (i = 0; i < 3 && config._a[i] == null; ++i) { config._a[i] = input[i] = currentDate[i]; } // Zero out whatever was not defaulted, including time for (; i < 7; i++) { config._a[i] = input[i] = (config._a[i] == null) ? (i === 2 ? 1 : 0) : config._a[i]; } // Check for 24:00:00.000 if (config._a[HOUR] === 24 && config._a[MINUTE] === 0 && config._a[SECOND] === 0 && config._a[MILLISECOND] === 0) { config._nextDay = true; config._a[HOUR] = 0; } config._d = (config._useUTC ? createUTCDate : createDate).apply(null, input); expectedWeekday = config._useUTC ? config._d.getUTCDay() : config._d.getDay(); // Apply timezone offset from input. The actual utcOffset can be changed // with parseZone. if (config._tzm != null) { config._d.setUTCMinutes(config._d.getUTCMinutes() - config._tzm); } if (config._nextDay) { config._a[HOUR] = 24; } // check for mismatching day of week if (config._w && typeof config._w.d !== 'undefined' && config._w.d !== expectedWeekday) { getParsingFlags(config).weekdayMismatch = true; } } function dayOfYearFromWeekInfo(config) { var w, weekYear, week, weekday, dow, doy, temp, weekdayOverflow; w = config._w; if (w.GG != null || w.W != null || w.E != null) { dow = 1; doy = 4; // TODO: We need to take the current isoWeekYear, but that depends on // how we interpret now (local, utc, fixed offset). So create // a now version of current config (take local/utc/offset flags, and // create now). weekYear = defaults(w.GG, config._a[YEAR], weekOfYear(createLocal(), 1, 4).year); week = defaults(w.W, 1); weekday = defaults(w.E, 1); if (weekday < 1 || weekday > 7) { weekdayOverflow = true; } } else { dow = config._locale._week.dow; doy = config._locale._week.doy; var curWeek = weekOfYear(createLocal(), dow, doy); weekYear = defaults(w.gg, config._a[YEAR], curWeek.year); // Default to current week. week = defaults(w.w, curWeek.week); if (w.d != null) { // weekday -- low day numbers are considered next week weekday = w.d; if (weekday < 0 || weekday > 6) { weekdayOverflow = true; } } else if (w.e != null) { // local weekday -- counting starts from begining of week weekday = w.e + dow; if (w.e < 0 || w.e > 6) { weekdayOverflow = true; } } else { // default to begining of week weekday = dow; } } if (week < 1 || week > weeksInYear(weekYear, dow, doy)) { getParsingFlags(config)._overflowWeeks = true; } else if (weekdayOverflow != null) { getParsingFlags(config)._overflowWeekday = true; } else { temp = dayOfYearFromWeeks(weekYear, week, weekday, dow, doy); config._a[YEAR] = temp.year; config._dayOfYear = temp.dayOfYear; } } // iso 8601 regex // 0000-00-00 0000-W00 or 0000-W00-0 + T + 00 or 00:00 or 00:00:00 or 00:00:00.000 + +00:00 or +0000 or +00) var extendedIsoRegex = /^\s*((?:[+-]\d{6}|\d{4})-(?:\d\d-\d\d|W\d\d-\d|W\d\d|\d\d\d|\d\d))(?:(T| )(\d\d(?::\d\d(?::\d\d(?:[.,]\d+)?)?)?)([\+\-]\d\d(?::?\d\d)?|\s*Z)?)?$/; var basicIsoRegex = /^\s*((?:[+-]\d{6}|\d{4})(?:\d\d\d|W\d\d\d|W\d\d|\d\d\d|\d\d))(?:(T| )(\d\d(?:\d\d(?:\d\d(?:[.,]\d+)?)?)?)([\+\-]\d\d(?::?\d\d)?|\s*Z)?)?$/; var tzRegex = /Z|[+-]\d\d(?::?\d\d)?/; var isoDates = [ ['YYYYYY-MM-DD', /[+-]\d{6}-\d\d-\d\d/], ['YYYY-MM-DD', /\d{4}-\d\d-\d\d/], ['GGGG-[W]WW-E', /\d{4}-W\d\d-\d/], ['GGGG-[W]WW', /\d{4}-W\d\d/, false], ['YYYY-DDD', /\d{4}-\d{3}/], ['YYYY-MM', /\d{4}-\d\d/, false], ['YYYYYYMMDD', /[+-]\d{10}/], ['YYYYMMDD', /\d{8}/], // YYYYMM is NOT allowed by the standard ['GGGG[W]WWE', /\d{4}W\d{3}/], ['GGGG[W]WW', /\d{4}W\d{2}/, false], ['YYYYDDD', /\d{7}/] ]; // iso time formats and regexes var isoTimes = [ ['HH:mm:ss.SSSS', /\d\d:\d\d:\d\d\.\d+/], ['HH:mm:ss,SSSS', /\d\d:\d\d:\d\d,\d+/], ['HH:mm:ss', /\d\d:\d\d:\d\d/], ['HH:mm', /\d\d:\d\d/], ['HHmmss.SSSS', /\d\d\d\d\d\d\.\d+/], ['HHmmss,SSSS', /\d\d\d\d\d\d,\d+/], ['HHmmss', /\d\d\d\d\d\d/], ['HHmm', /\d\d\d\d/], ['HH', /\d\d/] ]; var aspNetJsonRegex = /^\/?Date\((\-?\d+)/i; // date from iso format function configFromISO(config) { var i, l, string = config._i, match = extendedIsoRegex.exec(string) || basicIsoRegex.exec(string), allowTime, dateFormat, timeFormat, tzFormat; if (match) { getParsingFlags(config).iso = true; for (i = 0, l = isoDates.length; i < l; i++) { if (isoDates[i][1].exec(match[1])) { dateFormat = isoDates[i][0]; allowTime = isoDates[i][2] !== false; break; } } if (dateFormat == null) { config._isValid = false; return; } if (match[3]) { for (i = 0, l = isoTimes.length; i < l; i++) { if (isoTimes[i][1].exec(match[3])) { // match[2] should be 'T' or space timeFormat = (match[2] || ' ') + isoTimes[i][0]; break; } } if (timeFormat == null) { config._isValid = false; return; } } if (!allowTime && timeFormat != null) { config._isValid = false; return; } if (match[4]) { if (tzRegex.exec(match[4])) { tzFormat = 'Z'; } else { config._isValid = false; return; } } config._f = dateFormat + (timeFormat || '') + (tzFormat || ''); configFromStringAndFormat(config); } else { config._isValid = false; } } // RFC 2822 regex: For details see https://tools.ietf.org/html/rfc2822#section-3.3 var rfc2822 = /^(?:(Mon|Tue|Wed|Thu|Fri|Sat|Sun),?\s)?(\d{1,2})\s(Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)\s(\d{2,4})\s(\d\d):(\d\d)(?::(\d\d))?\s(?:(UT|GMT|[ECMP][SD]T)|([Zz])|([+-]\d{4}))$/; function extractFromRFC2822Strings(yearStr, monthStr, dayStr, hourStr, minuteStr, secondStr) { var result = [ untruncateYear(yearStr), defaultLocaleMonthsShort.indexOf(monthStr), parseInt(dayStr, 10), parseInt(hourStr, 10), parseInt(minuteStr, 10) ]; if (secondStr) { result.push(parseInt(secondStr, 10)); } return result; } function untruncateYear(yearStr) { var year = parseInt(yearStr, 10); if (year <= 49) { return 2000 + year; } else if (year <= 999) { return 1900 + year; } return year; } function preprocessRFC2822(s) { // Remove comments and folding whitespace and replace multiple-spaces with a single space return s.replace(/\([^)]*\)|[\n\t]/g, ' ').replace(/(\s\s+)/g, ' ').replace(/^\s\s*/, '').replace(/\s\s*$/,
```

```
"); } function checkWeekday(weekdayStr, parsedInput, config) { if (weekdayStr) { // TODO: Replace the vanilla JS Date object with an independent day-of-week check. var weekdayProvided = defaultLocaleWeekdaysShort.indexOf(weekdayStr), weekdayActual = new Date(parsedInput[0], parsedInput[1], parsedInput[2]).getDay(); if (weekdayProvided !== weekdayActual) { getParsingFlags(config).weekdayMismatch = true; config._isValid = false; return false; } } return true; } var obsOffsets = { UT: 0, GMT: 0, EDT: -4 * 60, EST: -5 * 60, CDT: -5 * 60, CST: -6 * 60, MDT: -6 * 60, MST: -7 * 60, PDT: -7 * 60, PST: -8 * 60 }; function calculateOffset(obsOffset, militaryOffset, numOffset) { if (obsOffset) { return obsOffsets[obsOffset]; } else if (militaryOffset) { // the only allowed military tz is Z return 0; } else { var hm = parseInt(numOffset, 10); var m = hm % 100, h = (hm - m) / 100; return h * 60 + m; } } // date and time from ref 2822 format function configFromRFC2822(config) { var match = rfc2822.exec(preprocessRFC2822(config._i)); if (match) { var parsedArray = extractFromRFC2822Strings(match[4], match[3], match[2], match[5], match[6], match[7]); if (!checkWeekday(match[1], parsedArray, config)) { return; } config._a = parsedArray; config._tzm = calculateOffset(match[8], match[9], match[10]); config._d = createUTCDate.apply(null, config._a); config._d.setUTCMinutes(config._d.getUTCMinutes() - config._tzm); getParsingFlags(config).rfc2822 = true; } else { config._isValid = false; } } // date from iso format or fallback function configFromString(config) { var matched = aspNetJsonRegex.exec(config._i); if (matched !== null) { config._d = new Date(+matched[1]); return; } configFromISO(config); if (config._isValid === false) { delete config._isValid; } else { return; } configFromRFC2822(config); if (config._isValid === false) { delete config._isValid; } else { return; } // Final attempt, use Input Fallback hooks.createFromInputFallback(config); } hooks.createFromInputFallback = deprecate( 'value provided is not in a recognized RFC2822 or ISO format. moment construction falls back to js Date(), ' + 'which is not reliable across all browsers and versions. Non RFC2822/ISO date formats are ' + 'discouraged and will be removed in an upcoming major release. Please refer to ' + 'http://momentjs.com/guides/#/warnings/js-date/ for more info.', function (config) { config._d = new Date(config._i + (config._useUTC ? ' UTC' : '')); } ); // constant that refers to the ISO standard hooks.ISO_8601 = function () {}; // constant that refers to the RFC 2822 form hooks.RFC_2822 = function () {}; // date from string and format string function configFromStringAndFormat(config) { // TODO: Move this to another part of the creation flow to prevent circular deps if (config._f === hooks.ISO_8601) { configFromISO(config); return; } if (config._f === hooks.RFC_2822) { configFromRFC2822(config); return; } config._a = []; getParsingFlags(config).empty = true; // This array is used to make a Date, either with `new Date` or `Date.UTC` var string = '' + config._i, i, parsedInput, tokens, token, skipped, stringLength = string.length, totalParsedInputLength = 0; tokens = expandFormat(config._f, config._locale).match(formattingTokens) || []; for (i = 0; i < tokens.length; i++) { token = tokens[i]; parsedInput = (string.match(getParseRegexForToken(token, config)) || [])[0]; // console.log('token', token, 'parsedInput', parsedInput, // 'regex', getParseRegexForToken(token, config)); if (parsedInput) { skipped = string.substr(0, string.indexOf(parsedInput)); if (skipped.length > 0) { getParsingFlags(config).unusedInput.push(skipped); } string = string.slice(string.indexOf(parsedInput) + parsedInput.length); totalParsedInputLength += parsedInput.length; } // don't parse if it's not a known token if (formatTokenFunctions[token]) { if (parsedInput) { getParsingFlags(config).empty = false; } else { getParsingFlags(config).unusedTokens.push(token); } addTimeToArrayFromToken(token, parsedInput, config); } else if (config._strict && !parsedInput) { getParsingFlags(config).unusedTokens.push(token); } } // add remaining unparsed input length to the string getParsingFlags(config).charsLeftOver = stringLength - totalParsedInputLength; if (string.length > 0) { getParsingFlags(config).unusedInput.push(string); } // clear _12h flag if hour is <= 12 if (config._a[HOUR] <= 12 && getParsingFlags(config).bigHour === true && config._a[HOUR] > 0) { getParsingFlags(config).bigHour = undefined; } getParsingFlags(config).parsedDateParts = config._a.slice(0); getParsingFlags(config).meridiem = config._meridiem; // handle meridiem config._a[HOUR] = meridiemFixWrap(config._locale, config._a[HOUR], config._meridiem); configFromArray(config); checkOverflow(config); } function meridiemFixWrap (locale, hour, meridiem) { var isPm; if (meridiem == null) { // nothing to do return hour; } if (locale.meridiemHour != null) { return locale.meridiemHour(hour, meridiem); } else if (locale.isPM != null) { // Fallback isPm = locale.isPM(meridiem); if (isPm && hour < 12) { hour += 12; } if (!isPm && hour === 12) { hour = 0; } return hour; } else { // this is not supposed to happen return hour; } } // date from string and array of format strings function configFromStringAndArray(config) { var tempConfig, bestMoment, scoreToBeat, i, currentScore; if (config._f.length === 0) { getParsingFlags(config).invalidFormat = true; config._d = new Date(NaN); return; } for (i = 0; i < config._f.length; i++) { currentScore = 0; tempConfig = copyConfig({}, config); if (config._useUTC != null) { tempConfig._useUTC = config._useUTC; } tempConfig._f = config._f[i]; configFromStringAndFormat(tempConfig); if (!isValid(tempConfig)) { continue; } // if there is any input that was not parsed add a penalty for that format currentScore += getParsingFlags(tempConfig).charsLeftOver; //or tokens currentScore += getParsingFlags(tempConfig).unusedTokens.length * 10; getParsingFlags(tempConfig).score = currentScore; if (scoreToBeat == null || currentScore < scoreToBeat) { scoreToBeat = currentScore; bestMoment = tempConfig; } } extend(config, bestMoment || tempConfig);
```

} function configFromObject(config) { if (config._d) { return; } var i = normalizeObjectUnits(config._i); config._a = map([i.year, i.month, i.day || i.date, i.hour, i.minute, i.second, i.millisecond], function (obj) { return obj && parseInt(obj, 10); }); configFromArray(config); } function createFromConfig (config) { var res = new Moment(checkOverflow(prepareConfig(config))); if (res._nextDay) { // Adding is smart enough around DST res.add(1, 'd'); res._nextDay = undefined; } return res; } function prepareConfig (config) { var input = config._i, format = config._f; config._locale = config._locale || getLocale(config._l); if (input === null || (format === undefined && input === '')) { return createInvalid({nullInput: true}); } if (typeof input === 'string') { config._i = input = config._locale.preparse(input); } if (isMoment(input)) { return new Moment(checkOverflow(input)); } else if (isDate(input)) { config._d = input; } else if (isArray(format)) { configFromStringAndArray(config); } else if (format) { configFromStringAndFormat(config); } else { configFromInput(config); } if (!isValid(config)) { config._d = null; } return config; } function configFromInput(config) { var input = config._i; if (isUndefined(input)) { config._d = new Date(hooks.now()); } else if (isDate(input)) { config._d = new Date(input.valueOf()); } else if (typeof input === 'string') { configFromString(config); } else if (isArray(input)) { config._a = map(input.slice(0), function (obj) { return parseInt(obj, 10); }); configFromArray(config); } else if (isObject(input)) { configFromObject(config); } else if (isNumber(input)) { // from milliseconds config._d = new Date(input); } else { hooks.createFromInputFallback(config); } } function createLocalOrUTC (input, format, locale, strict, isUTC) { var c = {}; if (locale === true || locale === false) { strict = locale; locale = undefined; } if ((isObject(input) && isObjectEmpty(input)) || (isArray(input) && input.length === 0)) { input = undefined; } // object construction must be done this way. // https://github.com/moment/moment/issues/1423 c._isAMomentObject = true; c._useUTC = c._isUTC = isUTC; c._l = locale; c._i = input; c._f = format; c._strict = strict; return createFromConfig(c); } function createLocal (input, format, locale, strict) { return createLocalOrUTC(input, format, locale, strict, false); } var prototypeMin = deprecate('moment().min is deprecated, use moment.max instead. http://momentjs.com/guides/#/warnings/min-max/', function () { var other = createLocal.apply(null, arguments); if (this.isValid() && other.isValid()) { return other < this ? this : other; } else { return createInvalid(); } }); var prototypeMax = deprecate('moment().max is deprecated, use moment.min instead. http://momentjs.com/guides/#/warnings/min-max/', function () { var other = createLocal.apply(null, arguments); if (this.isValid() && other.isValid()) { return other > this ? this : other; } else { return createInvalid(); } }); // Pick a moment m from moments so that m[fn](other) is true for all // other. This relies on the function fn to be transitive. // // moments should either be an array of moment objects or an array, whose // first element is an array of moment objects. function pickBy(fn, moments) { var res, i; if (moments.length === 1 && isArray(moments[0])) { moments = moments[0]; } if (!moments.length) { return createLocal(); } res = moments[0]; for (i = 1; i < moments.length; ++i) { if (!moments[i].isValid() || moments[i][fn](res)) { res = moments[i]; } } return res; } // TODO: Use [].sort instead? function min () { var args = [].slice.call(arguments, 0); return pickBy('isBefore', args); } function max () { var args = [].slice.call(arguments, 0); return pickBy('isAfter', args); } var now = function () { return Date.now ? Date.now() : +(new Date()); }; var ordering = ['year', 'quarter', 'month', 'week', 'day', 'hour', 'minute', 'second', 'millisecond']; function isDurationValid(m) { for (var key in m) { if (!(indexOf.call(ordering, key) !== -1 && (m[key] == null || !isNaN(m[key])))) { return false; } } var unitHasDecimal = false; for (var i = 0; i < ordering.length; ++i) { if (m[ordering[i]]) { if (unitHasDecimal) { return false; // only allow non-integers for smallest unit } if (parseFloat(m[ordering[i]]) !== toInt(m[ordering[i]])) { unitHasDecimal = true; } } } return true; } function isValid$1() { return this._isValid; } function createInvalid$1() { return createDuration(NaN); } function Duration (duration) { var normalizedInput = normalizeObjectUnits(duration), years = normalizedInput.year || 0, quarters = normalizedInput.quarter || 0, months = normalizedInput.month || 0, weeks = normalizedInput.week || 0, days = normalizedInput.day || 0, hours = normalizedInput.hour || 0, minutes = normalizedInput.minute || 0, seconds = normalizedInput.second || 0, milliseconds = normalizedInput.millisecond || 0; this._isValid = isDurationValid(normalizedInput); // representation for dateAddRemove this._milliseconds = +milliseconds + seconds * 1e3 + // 1000 minutes * 6e4 + // 1000 * 60 hours * 1000 * 60 * 60; //using 1000 * 60 * 60 instead of 36e5 to avoid floating point rounding errors https://github.com/moment/moment/issues/2978 // Because of dateAddRemove treats 24 hours as different from a // day when working around DST, we need to store them separately this._days = +days + weeks * 7; // It is impossible to translate months into days without knowing // which months you are are talking about, so we have to store // it separately. this._months = +months + quarters * 3 + years * 12; this._data = {}; this._locale = getLocale(); this._bubble(); } function isDuration (obj) { return obj instanceof Duration; } function absRound (number) { if (number < 0) { return Math.round(-1 * number) * -1; } else { return Math.round(number); } } // FORMATTING function offset (token, separator) { addFormatToken(token, 0, 0, function () { var offset = this.utcOffset(); var sign = '+'; if (offset < 0) { offset = -offset; sign = '-'; } return sign + zeroFill(~~(offset / 60), 2) + separator + zeroFill(~~(offset) % 60, 2); }); } offset('Z', ':'); offset('ZZ', ''); // PARSING addRegexToken('Z', matchShortOffset); addRegexToken('ZZ', matchShortOffset); addParseToken(['Z', 'ZZ'], function (input, array, config) { config._useUTC = true; config._tzm = offsetFromString(matchShortOffset, input); }); // HELPERS

```
// timezone chunker // '+10:00' > ['10', '00'] // '-1530' > ['-15', '30'] var chunkOffset = /([\+\-]|\d\d)/gi; function offsetFromString(matcher, string) { var matches = (string || '').match(matcher); if (matches === null) { return null; } var chunk = matches[matches.length - 1] || []; var parts = (chunk + '').match(chunkOffset) || ['-', 0, 0]; var minutes = + (parts[1] * 60) + toInt(parts[2]); return minutes === 0 ? 0 : parts[0] === '+' ? minutes : -minutes; } // Return a moment from input, that is local/utc/zone equivalent to model. function cloneWithOffset(input, model) { var res, diff; if (model._isUTC) { res = model.clone(); diff = (isMoment(input) || isDate(input) ? input.valueOf() : createLocal(input).valueOf()) - res.valueOf(); // Use low-level api, because this fn is low-level api. res._d.setTime(res._d.valueOf() + diff); hooks.updateOffset(res, false); return res; } else { return createLocal(input).local(); } } function getDateOffset (m) { // On Firefox.24 Date#getTimezoneOffset returns a floating point. // https://github.com/moment/moment/pull/1871 return -Math.round(m._d.getTimezoneOffset() / 15) * 15; } // HOOKS // This function will be called whenever a moment is mutated. // It is intended to keep the offset in sync with the timezone. hooks.updateOffset = function () {}; // MOMENTS // keepLocalTime = true means only change the timezone, without // affecting the local hour. So 5:31:26 +0300 --[utcOffset(2, true)]--> // 5:31:26 +0200 It is possible that 5:31:26 doesn't exist with offset // +0200, so we adjust the time as needed, to be valid. // // Keeping the time actually adds/subtracts (one hour) // from the actual represented time. That is why we call updateOffset // a second time. In case it wants us to change the offset again // _changeInProgress == true case, then we have to adjust, because // there is no such time in the given timezone. function getSetOffset (input, keepLocalTime, keepMinutes) { var offset = this._offset || 0, localAdjust; if (!this.isValid()) { return input != null ? this : NaN; } if (input != null) { if (typeof input === 'string') { input = offsetFromString(matchShortOffset, input); if (input === null) { return this; } } else if (Math.abs(input) < 16 && !keepMinutes) { input = input * 60; } if (!this._isUTC && keepLocalTime) { localAdjust = getDateOffset(this); } this._offset = input; this._isUTC = true; if (localAdjust != null) { this.add(localAdjust, 'm'); } if (offset !== input) { if (!keepLocalTime || this._changeInProgress) { addSubtract(this, createDuration(input - offset, 'm'), 1, false); } else if (!this._changeInProgress) { this._changeInProgress = true; hooks.updateOffset(this, true); this._changeInProgress = null; } } return this; } else { return this._isUTC ? offset : getDateOffset(this); } } function getSetZone (input, keepLocalTime) { if (input != null) { if (typeof input !== 'string') { input = -input; } this.utcOffset(input, keepLocalTime); return this; } else { return -this.utcOffset(); } } function setOffsetToUTC (keepLocalTime) { return this.utcOffset(0, keepLocalTime); } function setOffsetToLocal (keepLocalTime) { if (this._isUTC) { this.utcOffset(0, keepLocalTime); this._isUTC = false; if (keepLocalTime) { this.subtract(getDateOffset(this), 'm'); } } return this; } function setOffsetToParsedOffset () { if (this._tzm != null) { this.utcOffset(this._tzm, false, true); } else if (typeof this._i === 'string') { var tZone = offsetFromString(matchOffset, this._i); if (tZone != null) { this.utcOffset(tZone); } else { this.utcOffset(0, true); } } return this; } function hasAlignedHourOffset (input) { if (!this.isValid()) { return false; } input = input ? createLocal(input).utcOffset() : 0; return (this.utcOffset() - input) % 60 === 0; } function isDaylightSavingTime () { return ( this.utcOffset() > this.clone().month(0).utcOffset() || this.utcOffset() > this.clone().month(5).utcOffset() ); } function isDaylightSavingTimeShifted () { if (!isUndefined(this._isDSTShifted)) { return this._isDSTShifted; } var c = {}; copyConfig(c, this); c = prepareConfig(c); if (c._a) { var other = c._isUTC ? createUTC(c._a) : createLocal(c._a); this._isDSTShifted = this.isValid() && compareArrays(c._a, other.toArray()) > 0; } else { this._isDSTShifted = false; } return this._isDSTShifted; } function isLocal () { return this.isValid() ? !this._isUTC : false; } function isUtcOffset () { return this.isValid() ? this._isUTC : false; } function isUtc () { return this.isValid() ? this._isUTC && this._offset === 0 : false; } // ASP.NET json date format regex var aspNetRegex = /^(\-|\+)?(?:(\d*)[. ])?(\d+)\:(\d+)(?:\:(\d+)(\.\d*)?)?$/; // from http://docs.closure-library.googlecode.com/git/closure_goog_date_date.js.source.html // somewhat more in line with 4.4.3.2 2004 spec, but allows decimal anywhere // and further modified to allow for strings containing both week and day var isoRegex = /^(-|\+)?P(?:([-+]?[0-9,.]*)Y)?(?:([-+]?[0-9,.]*)M)?(?:([-+]?[0-9,.]*)W)?(?:([-+]?[0-9,.]*)D)?(?:T(?:([-+]?[0-9,.]*)H)?(?:([-+]?[0-9,.]*)M)?(?:([-+]?[0-9,.]*)S)?)?$/; function createDuration (input, key) { var duration = input, // matching against regexp is expensive, do it on demand match = null, sign, ret, diffRes; if (isDuration(input)) { duration = { ms : input._milliseconds, d : input._days, M : input._months }; } else if (isNumber(input)) { duration = {}; if (key) { duration[key] = input; } else { duration.milliseconds = input; } } else if (!!(match = aspNetRegex.exec(input))) { sign = (match[1] === '-') ? -1 : 1; duration = { y : 0, d : toInt(match[DATE]) * sign, h : toInt(match[HOUR]) * sign, m : toInt(match[MINUTE]) * sign, s : toInt(match[SECOND]) * sign, ms : toInt(absRound(match[MILLISECOND] * 1000)) * sign // the millisecond decimal point is included in the match }; } else if (!!(match = isoRegex.exec(input))) { sign = (match[1] === '-') ? -1 : (match[1] === '+') ? 1 : 1; duration = { y : parseIso(match[2], sign), M : parseIso(match[3], sign), w : parseIso(match[4], sign), d : parseIso(match[5], sign), h : parseIso(match[6], sign), m : parseIso(match[7], sign), s : parseIso(match[8], sign) }; } else if (duration == null) {// checks for null or undefined duration = {}; } else if (typeof duration === 'object' && ('from' in duration || 'to' in duration)) { diffRes = momentsDifference(createLocal(duration.from), createLocal(duration.to)); duration = {}; duration.ms = diffRes.milliseconds; duration.M = diffRes.months; } ret = new
```

```
Duration(duration); if (isDuration(input) && hasOwnProp(input, '_locale')) { ret._locale = input._locale; } return ret; } createDuration.fn = Duration.prototype; createDuration.invalid = createInvalid$1; function parseIso (inp, sign) { // We'd normally use ~~inp for this, but unfortunately it also // converts floats to ints. // inp may be undefined, so careful calling replace on it. var res = inp && parseFloat(inp.replace(',', '.')); // apply sign while we're at it return (isNaN(res) ? 0 : res) * sign; } function positiveMomentsDifference(base, other) { var res = {milliseconds: 0, months: 0}; res.months = other.month() - base.month() + (other.year() - base.year()) * 12; if (base.clone().add(res.months, 'M').isAfter(other)) { --res.months; } res.milliseconds = +other - +(base.clone().add(res.months, 'M')); return res; } function momentsDifference(base, other) { var res; if (!(base.isValid() && other.isValid())) { return {milliseconds: 0, months: 0}; } other = cloneWithOffset(other, base); if (base.isBefore(other)) { res = positiveMomentsDifference(base, other); } else { res = positiveMomentsDifference(other, base); res.milliseconds = -res.milliseconds; res.months = -res.months; } return res; } // TODO: remove 'name' arg after deprecation is removed function createAdder(direction, name) { return function (val, period) { var dur, tmp; //invert the arguments, but complain about it if (period !== null && !isNaN(+period)) { deprecateSimple(name, 'moment().' + name + '(period, number) is deprecated. Please use moment().' + name + '(number, period). ' + 'See http://momentjs.com/guides/#/warnings/add-inverted-param/ for more info.'); tmp = val; val = period; period = tmp; } val = typeof val === 'string' ? +val : val; dur = createDuration(val, period); addSubtract(this, dur, direction); return this; }; } function addSubtract (mom, duration, isAdding, updateOffset) { var milliseconds = duration._milliseconds, days = absRound(duration._days), months = absRound(duration._months); if (!mom.isValid()) { // No op return; } updateOffset = updateOffset == null ? true : updateOffset; if (months) { setMonth(mom, get(mom, 'Month') + months * isAdding); } if (days) { set$1(mom, 'Date', get(mom, 'Date') + days * isAdding); } if (milliseconds) { mom._d.setTime(mom._d.valueOf() + milliseconds * isAdding); } if (updateOffset) { hooks.updateOffset(mom, days || months); } } var add = createAdder(1, 'add'); var subtract = createAdder(-1, 'subtract'); function getCalendarFormat(myMoment, now) { var diff = myMoment.diff(now, 'days', true); return diff < -6 ? 'sameElse' : diff < -1 ? 'lastWeek' : diff < 0 ? 'lastDay' : diff < 1 ? 'sameDay' : diff < 2 ? 'nextDay' : diff < 7 ? 'nextWeek' : 'sameElse'; } function calendar$1 (time, formats) { // We want to compare the start of today, vs this. // Getting start-of-today depends on whether we're local/utc/offset or not. var now = time || createLocal(), sod = cloneWithOffset(now, this).startOf('day'), format = hooks.calendarFormat(this, sod) || 'sameElse'; var output = formats && (isFunction(formats[format]) ? formats[format].call(this, now) : formats[format]); return this.format(output || this.localeData().calendar(format, this, createLocal(now))); } function clone () { return new Moment(this); } function isAfter (input, units) { var localInput = isMoment(input) ? input : createLocal(input); if (!(this.isValid() && localInput.isValid())) { return false; } units = normalizeUnits(!isUndefined(units) ? units : 'millisecond'); if (units === 'millisecond') { return this.valueOf() > localInput.valueOf(); } else { return localInput.valueOf() < this.clone().startOf(units).valueOf(); } } function isBefore (input, units) { var localInput = isMoment(input) ? input : createLocal(input); if (!(this.isValid() && localInput.isValid())) { return false; } units = normalizeUnits(!isUndefined(units) ? units : 'millisecond'); if (units === 'millisecond') { return this.valueOf() < localInput.valueOf(); } else { return this.clone().endOf(units).valueOf() < localInput.valueOf(); } } function isBetween (from, to, units, inclusivity) { inclusivity = inclusivity || '()'; return (inclusivity[0] === '(' ? this.isAfter(from, units) : !this.isBefore(from, units)) && (inclusivity[1] === ')' ? this.isBefore(to, units) : !this.isAfter(to, units)); } function isSame (input, units) { var localInput = isMoment(input) ? input : createLocal(input), inputMs; if (!(this.isValid() && localInput.isValid())) { return false; } units = normalizeUnits(units || 'millisecond'); if (units === 'millisecond') { return this.valueOf() === localInput.valueOf(); } else { inputMs = localInput.valueOf(); return this.clone().startOf(units).valueOf() <= inputMs && inputMs <= this.clone().endOf(units).valueOf(); } } function isSameOrAfter (input, units) { return this.isSame(input, units) || this.isAfter(input,units); } function isSameOrBefore (input, units) { return this.isSame(input, units) || this.isBefore(input,units); } function diff (input, units, asFloat) { var that, zoneDelta, output; if (!this.isValid()) { return NaN; } that = cloneWithOffset(input, this); if (!that.isValid()) { return NaN; } zoneDelta = (that.utcOffset() - this.utcOffset()) * 6e4; units = normalizeUnits(units); switch (units) { case 'year': output = monthDiff(this, that) / 12; break; case 'month': output = monthDiff(this, that); break; case 'quarter': output = monthDiff(this, that) / 3; break; case 'second': output = (this - that) / 1e3; break; // 1000 case 'minute': output = (this - that) / 6e4; break; // 1000 * 60 case 'hour': output = (this - that) / 36e5; break; // 1000 * 60 * 60 case 'day': output = (this - that - zoneDelta) / 864e5; break; // 1000 * 60 * 60 * 24, negate dst case 'week': output = (this - that - zoneDelta) / 6048e5; break; // 1000 * 60 * 60 * 24 * 7, negate dst default: output = this - that; } return asFloat ? output : absFloor(output); } function monthDiff (a, b) { // difference in months var wholeMonthDiff = ((b.year() - a.year()) * 12) + (b.month() - a.month()), // b is in (anchor - 1 month, anchor + 1 month) anchor = a.clone().add(wholeMonthDiff, 'months'), anchor2, adjust; if (b - anchor < 0) { anchor2 = a.clone().add(wholeMonthDiff - 1, 'months'); // linear across the month adjust = (b - anchor) / (anchor - anchor2); } else { anchor2 = a.clone().add(wholeMonthDiff + 1, 'months'); // linear across the month adjust = (b - anchor) / (anchor2 - anchor); } //check for negative zero, return zero if negative zero return -(wholeMonthDiff + adjust) || 0; }
```

```
hooks.defaultFormat = 'YYYY-MM-DDTHH:mm:ssZ'; hooks.defaultFormatUtc = 'YYYY-MM-DDTHH:mm:ss[Z]'; function toString () { return this.clone().locale('en').format('ddd MMM DD YYYY HH:mm:ss [GMT]ZZ'); } function toISOString(keepOffset) { if (!this.isValid()) { return null; } var utc = keepOffset !== true; var m = utc ? this.clone().utc() : this; if (m.year() < 0 || m.year() > 9999) { return formatMoment(m, utc ? 'YYYYYY-MM-DD[T]HH:mm:ss.SSS[Z]' : 'YYYYYY-MM-DD[T]HH:mm:ss.SSSZ'); } if (isFunction(Date.prototype.toISOString)) { // native implementation is ~50x faster, use it when we can if (utc) { return this.toDate().toISOString(); } else { return new Date(this.valueOf() + this.utcOffset() * 60 * 1000).toISOString().replace('Z', formatMoment(m, 'Z')); } } return formatMoment(m, utc ? 'YYYY-MM-DD[T]HH:mm:ss.SSS[Z]' : 'YYYY-MM-DD[T]HH:mm:ss.SSSZ'); } /** * Return a human readable representation of a moment that can * also be evaluated to get a new moment which is the same * * @link https://nodejs.org/dist/latest/docs/api/util.html#util_custom_inspect_function_on_objects */ function inspect () { if (!this.isValid()) { return 'moment.invalid(/* ' + this._i + ' */)'; } var func = 'moment'; var zone = ''; if (!this.isLocal()) { func = this.utcOffset() === 0 ? 'moment.utc' : 'moment.parseZone'; zone = 'Z'; } var prefix = '[' + func + '("]'; var year = (0 <= this.year() && this.year() <= 9999) ? 'YYYY' : 'YYYYYY'; var datetime = '-MM-DD[T]HH:mm:ss.SSS'; var suffix = zone + '")]'; return this.format(prefix + year + datetime + suffix); } function format (inputString) { if (!inputString) { inputString = this.isUtc() ? hooks.defaultFormatUtc : hooks.defaultFormat; } var output = formatMoment(this, inputString); return this.localeData().postformat(output); } function from (time, withoutSuffix) { if (this.isValid() && ((isMoment(time) && time.isValid()) || createLocal(time).isValid())) { return createDuration({to: this, from: time}).locale(this.locale()).humanize(!withoutSuffix); } else { return this.localeData().invalidDate(); } } function fromNow (withoutSuffix) { return this.from(createLocal(), withoutSuffix); } function to (time, withoutSuffix) { if (this.isValid() && ((isMoment(time) && time.isValid()) || createLocal(time).isValid())) { return createDuration({from: this, to: time}).locale(this.locale()).humanize(!withoutSuffix); } else { return this.localeData().invalidDate(); } } function toNow (withoutSuffix) { return this.to(createLocal(), withoutSuffix); } // If passed a locale key, it will set the locale for this // instance. Otherwise, it will return the locale configuration // variables for this instance. function locale (key) { var newLocaleData; if (key === undefined) { return this._locale._abbr; } else { newLocaleData = getLocale(key); if (newLocaleData != null) { this._locale = newLocaleData; } return this; } } var lang = deprecate( 'moment().lang() is deprecated. Instead, use moment().localeData() to get the language configuration. Use moment().locale() to change languages.', function (key) { if (key === undefined) { return this.localeData(); } else { return this.locale(key); } } ); function localeData () { return this._locale; } function startOf (units) { units = normalizeUnits(units); // the following switch intentionally omits break keywords // to utilize falling through the cases. switch (units) { case 'year': this.month(0); /* falls through */ case 'quarter': case 'month': this.date(1); /* falls through */ case 'week': case 'isoWeek': case 'day': case 'date': this.hours(0); /* falls through */ case 'hour': this.minutes(0); /* falls through */ case 'minute': this.seconds(0); /* falls through */ case 'second': this.milliseconds(0); } // weeks are a special case if (units === 'week') { this.weekday(0); } if (units === 'isoWeek') { this.isoWeekday(1); } // quarters are also special if (units === 'quarter') { this.month(Math.floor(this.month() / 3) * 3); } return this; } function endOf (units) { units = normalizeUnits(units); if (units === undefined || units === 'millisecond') { return this; } // 'date' is an alias for 'day', so it should be considered as such. if (units === 'date') { units = 'day'; } return this.startOf(units).add(1, (units === 'isoWeek' ? 'week' : units)).subtract(1, 'ms'); } function valueOf () { return this._d.valueOf() - ((this._offset || 0) * 60000); } function unix () { return Math.floor(this.valueOf() / 1000); } function toDate () { return new Date(this.valueOf()); } function toArray () { var m = this; return [m.year(), m.month(), m.date(), m.hour(), m.minute(), m.second(), m.millisecond()]; } function toObject () { var m = this; return { years: m.year(), months: m.month(), date: m.date(), hours: m.hours(), minutes: m.minutes(), seconds: m.seconds(), milliseconds: m.milliseconds() }; } function toJSON () { // new Date(NaN).toJSON() === null return this.isValid() ? this.toISOString() : null; } function isValid$2 () { return isValid(this); } function parsingFlags () { return extend({}, getParsingFlags(this)); } function invalidAt () { return getParsingFlags(this).overflow; } function creationData() { return { input: this._i, format: this._f, locale: this._locale, isUTC: this._isUTC, strict: this._strict }; } // FORMATTING addFormatToken(0, ['gg', 2], 0, function () { return this.weekYear() % 100; }); addFormatToken(0, ['GG', 2], 0, function () { return this.isoWeekYear() % 100; }); function addWeekYearFormatToken (token, getter) { addFormatToken(0, [token, token.length], 0, getter); } addWeekYearFormatToken('gggg', 'weekYear'); addWeekYearFormatToken('ggggg', 'weekYear'); addWeekYearFormatToken('GGGG', 'isoWeekYear'); addWeekYearFormatToken('GGGGG', 'isoWeekYear'); // ALIASES addUnitAlias('weekYear', 'gg'); addUnitAlias('isoWeekYear', 'GG'); // PRIORITY addUnitPriority('weekYear', 1); addUnitPriority('isoWeekYear', 1); // PARSING addRegexToken('G', matchSigned); addRegexToken('g', matchSigned); addRegexToken('GG', match1to2, match2); addRegexToken('gg', match1to2, match2); addRegexToken('GGGG', match1to4, match4); addRegexToken('gggg', match1to4, match4); addRegexToken('GGGGG', match1to6, match6); addRegexToken('ggggg', match1to6, match6); addWeekParseToken(['gggg', 'ggggg', 'GGGG', 'GGGGG'], function (input, week, config, token) { week[token.substr(0, 2)] = toInt(input); }); addWeekParseToken(['gg', 'GG'], function (input, week, config,
```

token) { week[token] = hooks.parseTwoDigitYear(input); }); // MOMENTS function getSetWeekYear (input) { return getSetWeekYearHelper.call(this, input, this.week(), this.weekday(), this.localeData()._week.dow, this.localeData()._week.doy); } function getSetISOWeekYear (input) { return getSetWeekYearHelper.call(this, input, this.isoWeek(), this.isoWeekday(), 1, 4); } function getISOWeeksInYear () { return weeksInYear(this.year(), 1, 4); } function getWeeksInYear () { var weekInfo = this.localeData()._week; return weeksInYear(this.year(), weekInfo.dow, weekInfo.doy); } function getSetWeekYearHelper(input, week, weekday, dow, doy) { var weeksTarget; if (input == null) { return weekOfYear(this, dow, doy).year; } else { weeksTarget = weeksInYear(input, dow, doy); if (week > weeksTarget) { week = weeksTarget; } return setWeekAll.call(this, input, week, weekday, dow, doy); } } function setWeekAll(weekYear, week, weekday, dow, doy) { var dayOfYearData = dayOfYearFromWeeks(weekYear, week, weekday, dow, doy), date = createUTCDate(dayOfYearData.year, 0, dayOfYearData.dayOfYear); this.year(date.getUTCFullYear()); this.month(date.getUTCMonth()); this.date(date.getUTCDate()); return this; } // FORMATTING addFormatToken('Q', 0, 'Qo', 'quarter'); // ALIASES addUnitAlias('quarter', 'Q'); // PRIORITY addUnitPriority('quarter', 7); // PARSING addRegexToken('Q', match1); addParseToken('Q', function (input, array) { array[MONTH] = (toInt(input) - 1) * 3; }); // MOMENTS function getSetQuarter (input) { return input == null ? Math.ceil((this.month() + 1) / 3) : this.month((input - 1) * 3 + this.month() % 3); } // FORMATTING addFormatToken('D', ['DD', 2], 'Do', 'date'); // ALIASES addUnitAlias('date', 'D'); // PRIORITY addUnitPriority('date', 9); // PARSING addRegexToken('D', match1to2); addRegexToken('DD', match1to2, match2); addRegexToken('Do', function (isStrict, locale) { // TODO: Remove "ordinalParse" fallback in next major release. return isStrict ? (locale._dayOfMonthOrdinalParse || locale._ordinalParse) : locale._dayOfMonthOrdinalParseLenient; }); addParseToken(['D', 'DD'], DATE); addParseToken('Do', function (input, array) { array[DATE] = toInt(input.match(match1to2)[0]); }); // MOMENTS var getSetDayOfMonth = makeGetSet('Date', true); // FORMATTING addFormatToken('DDD', ['DDDD', 3], 'DDDo', 'dayOfYear'); // ALIASES addUnitAlias('dayOfYear', 'DDD'); // PRIORITY addUnitPriority('dayOfYear', 4); // PARSING addRegexToken('DDD', match1to3); addRegexToken('DDDD', match3); addParseToken(['DDD', 'DDDD'], function (input, array, config) { config._dayOfYear = toInt(input); }); // HELPERS // MOMENTS function getSetDayOfYear (input) { var dayOfYear = Math.round((this.clone().startOf('day') - this.clone().startOf('year')) / 864e5) + 1; return input == null ? dayOfYear : this.add((input - dayOfYear), 'd'); } // FORMATTING addFormatToken('m', ['mm', 2], 0, 'minute'); // ALIASES addUnitAlias('minute', 'm'); // PRIORITY addUnitPriority('minute', 14); // PARSING addRegexToken('m', match1to2); addRegexToken('mm', match1to2, match2); addParseToken(['m', 'mm'], MINUTE); // MOMENTS var getSetMinute = makeGetSet('Minutes', false); // FORMATTING addFormatToken('s', ['ss', 2], 0, 'second'); // ALIASES addUnitAlias('second', 's'); // PRIORITY addUnitPriority('second', 15); // PARSING addRegexToken('s', match1to2); addRegexToken('ss', match1to2, match2); addParseToken(['s', 'ss'], SECOND); // MOMENTS var getSetSecond = makeGetSet('Seconds', false); // FORMATTING addFormatToken('S', 0, 0, function () { return ~~(this.millisecond() / 100); }); addFormatToken(0, ['SS', 2], 0, function () { return ~~(this.millisecond() / 10); }); addFormatToken(0, ['SSS', 3], 0, 'millisecond'); addFormatToken(0, ['SSSS', 4], 0, function () { return this.millisecond() * 10; }); addFormatToken(0, ['SSSSS', 5], 0, function () { return this.millisecond() * 100; }); addFormatToken(0, ['SSSSSS', 6], 0, function () { return this.millisecond() * 1000; }); addFormatToken(0, ['SSSSSSS', 7], 0, function () { return this.millisecond() * 10000; }); addFormatToken(0, ['SSSSSSSS', 8], 0, function () { return this.millisecond() * 100000; }); addFormatToken(0, ['SSSSSSSSS', 9], 0, function () { return this.millisecond() * 1000000; }); // ALIASES addUnitAlias('millisecond', 'ms'); // PRIORITY addUnitPriority('millisecond', 16); // PARSING addRegexToken('S', match1to3, match1); addRegexToken('SS', match1to3, match2); addRegexToken('SSS', match1to3, match3); var token; for (token = 'SSSS'; token.length <= 9; token += 'S') { addRegexToken(token, matchUnsigned); } function parseMs(input, array) { array[MILLISECOND] = toInt(('0.' + input) * 1000); } for (token = 'S'; token.length <= 9; token += 'S') { addParseToken(token, parseMs); } // MOMENTS var getSetMillisecond = makeGetSet('Milliseconds', false); // FORMATTING addFormatToken('z', 0, 0, 'zoneAbbr'); addFormatToken('zz', 0, 0, 'zoneName'); // MOMENTS function getZoneAbbr () { return this._isUTC ? 'UTC' : ''; } function getZoneName () { return this._isUTC ? 'Coordinated Universal Time' : ''; } var proto = Moment.prototype; proto.add = add; proto.calendar = calendar$1; proto.clone = clone; proto.diff = diff; proto.endOf = endOf; proto.format = format; proto.from = from; proto.fromNow = fromNow; proto.to = to; proto.toNow = toNow; proto.get = stringGet; proto.invalidAt = invalidAt; proto.isAfter = isAfter; proto.isBefore = isBefore; proto.isBetween = isBetween; proto.isSame = isSame; proto.isSameOrAfter = isSameOrAfter; proto.isSameOrBefore = isSameOrBefore; proto.isValid = isValid$2; proto.lang = lang; proto.locale = locale; proto.localeData = localeData; proto.max = prototypeMax; proto.min = prototypeMin; proto.parsingFlags = parsingFlags; proto.set = stringSet; proto.startOf = startOf; proto.subtract = subtract; proto.toArray = toArray; proto.toObject = toObject; proto.toDate = toDate; proto.toISOString = toISOString; proto.inspect = inspect; proto.toJSON = toJSON; proto.toString = toString; proto.unix = unix; proto.valueOf = valueOf; proto.creationData = creationData; proto.year = getSetYear;

```
proto.isLeapYear = getIsLeapYear; proto.weekYear = getSetWeekYear; proto.isoWeekYear = getSetISOWeekYear; proto.quarter = proto.quarters = getSetQuarter; proto.month = getSetMonth; proto.daysInMonth = getDaysInMonth; proto.week = proto.weeks = getSetWeek; proto.isoWeek = proto.isoWeeks = getSetISOWeek; proto.weeksInYear = getWeeksInYear; proto.isoWeeksInYear = getISOWeeksInYear; proto.date = getSetDayOfMonth; proto.day = proto.days = getSetDayOfWeek; proto.weekday = getSetLocaleDayOfWeek; proto.isoWeekday = getSetISODayOfWeek; proto.dayOfYear = getSetDayOfYear; proto.hour = proto.hours = getSetHour; proto.minute = proto.minutes = getSetMinute; proto.second = proto.seconds = getSetSecond; proto.millisecond = proto.milliseconds = getSetMillisecond; proto.utcOffset = getSetOffset; proto.utc = setOffsetToUTC; proto.local = setOffsetToLocal; proto.parseZone = setOffsetToParsedOffset; proto.hasAlignedHourOffset = hasAlignedHourOffset; proto.isDST = isDaylightSavingTime; proto.isLocal = isLocal; proto.isUtcOffset = isUtcOffset; proto.isUtc = isUtc; proto.isUTC = isUtc; proto.zoneAbbr = getZoneAbbr; proto.zoneName = getZoneName; proto.dates = deprecate('dates accessor is deprecated. Use date instead.', getSetDayOfMonth); proto.months = deprecate('months accessor is deprecated. Use month instead', getSetMonth); proto.years = deprecate('years accessor is deprecated. Use year instead', getSetYear); proto.zone = deprecate('moment().zone is deprecated, use moment().utcOffset instead. http://momentjs.com/guides/#/warnings/zone/', getSetZone); proto.isDSTShifted = deprecate('isDSTShifted is deprecated. See http://momentjs.com/guides/#/warnings/dst-shifted/ for more information', isDaylightSavingTimeShifted); function createUnix (input) { return createLocal(input * 1000); } function createInZone () { return createLocal.apply(null, arguments).parseZone(); } function preParsePostFormat (string) { return string; } var proto$1 = Locale.prototype; proto$1.calendar = calendar; proto$1.longDateFormat = longDateFormat; proto$1.invalidDate = invalidDate; proto$1.ordinal = ordinal; proto$1.preparse = preParsePostFormat; proto$1.postformat = preParsePostFormat; proto$1.relativeTime = relativeTime; proto$1.pastFuture = pastFuture; proto$1.set = set; proto$1.months = localeMonths; proto$1.monthsShort = localeMonthsShort; proto$1.monthsParse = localeMonthsParse; proto$1.monthsRegex = monthsRegex; proto$1.monthsShortRegex = monthsShortRegex; proto$1.week = localeWeek; proto$1.firstDayOfYear = localeFirstDayOfYear; proto$1.firstDayOfWeek = localeFirstDayOfWeek; proto$1.weekdays = localeWeekdays; proto$1.weekdaysMin = localeWeekdaysMin; proto$1.weekdaysShort = localeWeekdaysShort; proto$1.weekdaysParse = localeWeekdaysParse; proto$1.weekdaysRegex = weekdaysRegex; proto$1.weekdaysShortRegex = weekdaysShortRegex; proto$1.weekdaysMinRegex = weekdaysMinRegex; proto$1.isPM = localeIsPM; proto$1.meridiem = localeMeridiem; function get$1 (format, index, field, setter) { var locale = getLocale(); var utc = createUTC().set(setter, index); return locale[field](utc, format); } function listMonthsImpl (format, index, field) { if (isNumber(format)) { index = format; format = undefined; } format = format || ''; if (index != null) { return get$1(format, index, field, 'month'); } var i; var out = []; for (i = 0; i < 12; i++) { out[i] = get$1(format, i, field, 'month'); } return out; } // () // (5) // (fmt, 5) // (fmt) // (true) // (true, 5) // (true, fmt, 5) // (true, fmt) function listWeekdaysImpl (localeSorted, format, index, field) { if (typeof localeSorted === 'boolean') { if (isNumber(format)) { index = format; format = undefined; } format = format || ''; } else { format = localeSorted; index = format; localeSorted = false; if (isNumber(format)) { index = format; format = undefined; } format = format || ''; } var locale = getLocale(), shift = localeSorted ? locale._week.dow : 0; if (index != null) { return get$1(format, (index + shift) % 7, field, 'day'); } var i; var out = []; for (i = 0; i < 7; i++) { out[i] = get$1(format, (i + shift) % 7, field, 'day'); } return out; } function listMonths (format, index) { return listMonthsImpl(format, index, 'months'); } function listMonthsShort (format, index) { return listMonthsImpl(format, index, 'monthsShort'); } function listWeekdays (localeSorted, format, index) { return listWeekdaysImpl(localeSorted, format, index, 'weekdays'); } function listWeekdaysShort (localeSorted, format, index) { return listWeekdaysImpl(localeSorted, format, index, 'weekdaysShort'); } function listWeekdaysMin (localeSorted, format, index) { return listWeekdaysImpl(localeSorted, format, index, 'weekdaysMin'); } getSetGlobalLocale('en', { dayOfMonthOrdinalParse: /\d{1,2}(th|st|nd|rd)/, ordinal : function (number) { var b = number % 10, output = (toInt(number % 100 / 10) === 1) ? 'th' : (b === 1) ? 'st' : (b === 2) ? 'nd' : (b === 3) ? 'rd' : 'th'; return number + output; } }); // Side effect imports hooks.lang = deprecate('moment.lang is deprecated. Use moment.locale instead.', getSetGlobalLocale); hooks.langData = deprecate('moment.langData is deprecated. Use moment.localeData instead.', getLocale); var mathAbs = Math.abs; function abs () { var data = this._data; this._milliseconds = mathAbs(this._milliseconds); this._days = mathAbs(this._days); this._months = mathAbs(this._months); data.milliseconds = mathAbs(data.milliseconds); data.seconds = mathAbs(data.seconds); data.minutes = mathAbs(data.minutes); data.hours = mathAbs(data.hours); data.months = mathAbs(data.months); data.years = mathAbs(data.years); return this; } function addSubtract$1 (duration, input, value, direction) { var other = createDuration(input, value); duration._milliseconds += direction * other._milliseconds; duration._days += direction * other._days; duration._months += direction * other._months; return duration._bubble(); } // supports only 2.0-style add(1, 's') or add(duration) function add$1 (input, value) { return addSubtract$1(this, input, value, 1); } // supports only 2.0-style subtract(1, 's') or subtract(duration) function subtract$1 (input, value) { return addSubtract$1(this, input, value, -1); } function absCeil (number) { if (number < 0) { return Math.floor(number); }
```

```
else { return Math.ceil(number); } } function bubble () { var milliseconds = this._milliseconds; var days = this._days; var months = this._months; var data = this._data; var seconds, minutes, hours, years, monthsFromDays; // if we have a mix of positive and negative values, bubble down first // check: https://github.com/moment/moment/issues/2166 if (!((milliseconds >= 0 && days >= 0 && months >= 0) || (milliseconds <= 0 && days <= 0 && months <= 0))) { milliseconds += absCeil(monthsToDays(months) + days) * 864e5; days = 0; months = 0; } // The following code bubbles up values, see the tests for // examples of what that means. data.milliseconds = milliseconds % 1000; seconds = absFloor(milliseconds / 1000); data.seconds = seconds % 60; minutes = absFloor(seconds / 60); data.minutes = minutes % 60; hours = absFloor(minutes / 60); data.hours = hours % 24; days += absFloor(hours / 24); // convert days to months monthsFromDays = absFloor(daysToMonths(days)); months += monthsFromDays; days -= absCeil(monthsToDays(monthsFromDays)); // 12 months -> 1 year years = absFloor(months / 12); months %= 12; data.days = days; data.months = months; data.years = years; return this; } function daysToMonths (days) { // 400 years have 146097 days (taking into account leap year rules) // 400 years have 12 months === 4800 return days * 4800 / 146097; } function monthsToDays (months) { // the reverse of daysToMonths return months * 146097 / 4800; } function as (units) { if (!this.isValid()) { return NaN; } var days; var months; var milliseconds = this._milliseconds; units = normalizeUnits(units); if (units === 'month' || units === 'year') { days = this._days + milliseconds / 864e5; months = this._months + daysToMonths(days); return units === 'month' ? months : months / 12; } else { // handle milliseconds separately because of floating point math errors (issue #1867) days = this._days + Math.round(monthsToDays(this._months)); switch (units) { case 'week' : return days / 7 + milliseconds / 6048e5; case 'day' : return days + milliseconds / 864e5; case 'hour' : return days * 24 + milliseconds / 36e5; case 'minute' : return days * 1440 + milliseconds / 6e4; case 'second' : return days * 86400 + milliseconds / 1000; // Math.floor prevents floating point math errors here case 'millisecond': return Math.floor(days * 864e5) + milliseconds; default: throw new Error('Unknown unit ' + units); } } } // TODO: Use this.as('ms')? function valueOf$1 () { if (!this.isValid()) { return NaN; } return ( this._milliseconds + this._days * 864e5 + (this._months % 12) * 2592e6 + toInt(this._months / 12) * 31536e6 ); } function makeAs (alias) { return function () { return this.as(alias); }; } var asMilliseconds = makeAs('ms'); var asSeconds = makeAs('s'); var asMinutes = makeAs('m'); var asHours = makeAs('h'); var asDays = makeAs('d'); var asWeeks = makeAs('w'); var asMonths = makeAs('M'); var asYears = makeAs('y'); function clone$1 () { return createDuration(this); } function get$2 (units) { units = normalizeUnits(units); return this.isValid() ? this[units + 's']() : NaN; } function makeGetter(name) { return function () { return this.isValid() ? this._data[name] : NaN; }; } var milliseconds = makeGetter('milliseconds'); var seconds = makeGetter('seconds'); var minutes = makeGetter('minutes'); var hours = makeGetter('hours'); var days = makeGetter('days'); var months = makeGetter('months'); var years = makeGetter('years'); function weeks () { return absFloor(this.days() / 7); } var round = Math.round; var thresholds = { ss: 44, // a few seconds to seconds s : 45, // seconds to minute m : 45, // minutes to hour h : 22, // hours to day d : 26, // days to month M : 11 // months to year }; // helper function for moment.fn.from, moment.fn.fromNow, and moment.duration.fn.humanize function substituteTimeAgo(string, number, withoutSuffix, isFuture, locale) { return locale.relativeTime(number || 1, !!withoutSuffix, string, isFuture); } function relativeTime$1 (posNegDuration, withoutSuffix, locale) { var duration = createDuration(posNegDuration).abs(); var seconds = round(duration.as('s')); var minutes = round(duration.as('m')); var hours = round(duration.as('h')); var days = round(duration.as('d')); var months = round(duration.as('M')); var years = round(duration.as('y')); var a = seconds <= thresholds.ss && ['s', seconds] || seconds < thresholds.s && ['ss', seconds] || minutes <= 1 && ['m'] || minutes < thresholds.m && ['mm', minutes] || hours <= 1 && ['h'] || hours < thresholds.h && ['hh', hours] || days <= 1 && ['d'] || days < thresholds.d && ['dd', days] || months <= 1 && ['M'] || months < thresholds.M && ['MM', months] || years <= 1 && ['y'] || ['yy', years]; a[2] = withoutSuffix; a[3] = +posNegDuration > 0; a[4] = locale; return substituteTimeAgo.apply(null, a); } // This function allows you to set the rounding function for relative time strings function getSetRelativeTimeRounding (roundingFunction) { if (roundingFunction === undefined) { return round; } if (typeof(roundingFunction) === 'function') { round = roundingFunction; return true; } return false; } // This function allows you to set a threshold for relative time strings function getSetRelativeTimeThreshold (threshold, limit) { if (thresholds[threshold] === undefined) { return false; } if (limit === undefined) { return thresholds[threshold]; } thresholds[threshold] = limit; if (threshold === 's') { thresholds.ss = limit - 1; } return true; } function humanize (withSuffix) { if (!this.isValid()) { return this.localeData().invalidDate(); } var locale = this.localeData(); var output = relativeTime$1(this, !withSuffix, locale); if (withSuffix) { output = locale.pastFuture(+this, output); } return locale.postformat(output); } var abs$1 = Math.abs; function sign(x) { return ((x > 0) - (x < 0)) || +x; } function toISOString$1() { // for ISO strings we do not use the normal bubbling rules: // * milliseconds bubble up until they become hours // * days do not bubble at all // * months bubble up until they become years // This is because there is no context-free conversion between hours and days // (think of clock changes) // and also not between days and months (28-31 days per month) if (!this.isValid()) { return this.localeData().invalidDate(); } var seconds = abs$1(this._milliseconds) / 1000; var days = abs$1(this._days); var months = abs$1(this._months); var minutes, hours, years; //
```

3600 seconds -> 60 minutes -> 1 hour minutes = absFloor(seconds / 60); hours = absFloor(minutes / 60); seconds %= 60; minutes %= 60; // 12 months -> 1 year years = absFloor(months / 12); months %= 12; // inspired by https://github.com/dordille/moment-isoduration/blob/master/moment.isoduration.js var Y = years; var M = months; var D = days; var h = hours; var m = minutes; var s = seconds ? seconds.toFixed(3).replace(/\.?0+$/, '') : ''; var total = this.asSeconds(); if (!total) { // this is the same as C#'s (Noda) and python (isodate)... // but not other JS (goog.date) return 'P0D'; } var totalSign = total < 0 ? '-' : ''; var ymSign = sign(this._months) !== sign(total) ? '-' : ''; var daysSign = sign(this._days) !== sign(total) ? '-' : ''; var hmsSign = sign(this._milliseconds) !== sign(total) ? '-' : ''; return totalSign + 'P' + (Y ? ymSign + Y + 'Y' : '') + (M ? ymSign + M + 'M' : '') + (D ? daysSign + D + 'D' : '') + ((h || m || s) ? 'T' : '') + (h ? hmsSign + h + 'H' : '') + (m ? hmsSign + m + 'M' : '') + (s ? hmsSign + s + 'S' : ''); } var proto$2 = Duration.prototype; proto$2.isValid = isValid$1; proto$2.abs = abs; proto$2.add = add$1; proto$2.subtract = subtract$1; proto$2.as = as; proto$2.asMilliseconds = asMilliseconds; proto$2.asSeconds = asSeconds; proto$2.asMinutes = asMinutes; proto$2.asHours = asHours; proto$2.asDays = asDays; proto$2.asWeeks = asWeeks; proto$2.asMonths = asMonths; proto$2.asYears = asYears; proto$2.valueOf = valueOf$1; proto$2._bubble = bubble; proto$2.clone = clone$1; proto$2.get = get$2; proto$2.milliseconds = milliseconds; proto$2.seconds = seconds; proto$2.minutes = minutes; proto$2.hours = hours; proto$2.days = days; proto$2.weeks = weeks; proto$2.months = months; proto$2.years = years; proto$2.humanize = humanize; proto$2.toISOString = toISOString$1; proto$2.toString = toISOString$1; proto$2.toJSON = toISOString$1; proto$2.locale = locale; proto$2.localeData = localeData; proto$2.toIsoString = deprecate('toIsoString() is deprecated. Please use toISOString() instead (notice the capitals)', toISOString$1); proto$2.lang = lang; // Side effect imports // FORMATTING addFormatToken('X', 0, 0, 'unix'); addFormatToken('x', 0, 0, 'valueOf'); // PARSING addRegexToken('x', matchSigned); addRegexToken('X', matchTimestamp); addParseToken('X', function (input, array, config) { config._d = new Date(parseFloat(input, 10) * 1000); }); addParseToken('x', function (input, array, config) { config._d = new Date(toInt(input)); }); // Side effect imports hooks.version = '2.22.2'; setHookCallback(createLocal); hooks.fn = proto; hooks.min = min; hooks.max = max; hooks.now = now; hooks.utc = createUTC; hooks.unix = createUnix; hooks.months = listMonths; hooks.isDate = isDate; hooks.locale = getSetGlobalLocale; hooks.invalid = createInvalid; hooks.duration = createDuration; hooks.isMoment = isMoment; hooks.weekdays = listWeekdays; hooks.parseZone = createInZone; hooks.localeData = getLocale; hooks.isDuration = isDuration; hooks.monthsShort = listMonthsShort; hooks.weekdaysMin = listWeekdaysMin; hooks.defineLocale = defineLocale; hooks.updateLocale = updateLocale; hooks.locales = listLocales; hooks.weekdaysShort = listWeekdaysShort; hooks.normalizeUnits = normalizeUnits; hooks.relativeTimeRounding = getSetRelativeTimeRounding; hooks.relativeTimeThreshold = getSetRelativeTimeThreshold; hooks.calendarFormat = getCalendarFormat; hooks.prototype = proto; // currently HTML5 input type only supports 24-hour formats hooks.HTML5_FMT = { DATETIME_LOCAL: 'YYYY-MM-DDTHH:mm', // DATETIME_LOCAL_SECONDS: 'YYYY-MM-DDTHH:mm:ss', // DATETIME_LOCAL_MS: 'YYYY-MM-DDTHH:mm:ss.SSS', // DATE: 'YYYY-MM-DD', // TIME: 'HH:mm', // TIME_SECONDS: 'HH:mm:ss', // TIME_MS: 'HH:mm:ss.SSS', // WEEK: 'YYYY-[W]WW', // MONTH: 'YYYY-MM' // }; return hooks; }));

	Ф.И.О.	Дата	Подпись	МК ID: 19	Дата создания в системе: 27.11.2018	1.11.2018 : 2.36.15
Дубл.						
Взам.						
Подп.						
						1 1
Разраб.						
Разраб.				вапавп		
				gdfgdf		КДИ

Н.контр					
М01	Сталь Р6М5 ДСТУ ГОСТ 7.1:2006				

	Код	ЕВ	МД	ЕН	Н.расх.	КИМ	КОД загот.	Профиль и размеры	КД	МЗ
М02		кг	33443	1			Пруток	Ø3443x34344	1	

А	цех	Уч.	Рм	Опер	Код, наименование операции	Обозначение документа										
Б	Код, наименование оборудования					см	Проф.	Р	УТ	КР	КОИд	ЕН	ОП	К шт.	Тп.з.	Т шт.

	Ф.И.О.	Дата	Подпись
Дубл.			
Взам.			
Подп.			

1 1

вапавп

	Код	ЕВ	МД	ЕН	Н.расх.	КИМ	КОД загот.	Профиль и размеры	КД	МЗ
М02		кг	33443	1			Пруток	Ø3443x34344	1	

А	цех	Уч.	Рм	Опер	Код, наименование операции	Обозначение документа

Б	Код, наименование оборудования	см	Проф.	Р	УТ	КР	КОИд	ЕН	ОП	К шт.	Тп.з.	Т шт.