Package: mondate (via r-universe)

August 22, 2024

Type Package

Title Keep track of dates in terms of months

Version 1.0

Date 2015-11-11

Author Dan Murphy

Maintainer Dan Murphy <chiefmurphy@gmail.com>

Description Keep track of dates in terms of months. Model dates as at close of business. Perform date arithmetic in units of ``months'' and ``years''. Allow ``infinite'' dates to model ``ultimate'' time spans.

License GPL (>= 2)

URL http://www.r-project.org, http://code.google.com/p/mondate/

LazyLoad yes

Imports methods, utils

Depends R (>= 3.0.0)

Suggests RUnit, zoo, knitr

VignetteBuilder knitr

Repository https://chiefmurph.r-universe.dev

RemoteUrl https://github.com/chiefmurph/mondate

RemoteRef HEAD

RemoteSha 67d2d11a5abdf94bbd7579f1f91d326f28da9276

Contents

add 2
Arith-methods 3
array-method 5
as.data.frame method 6
as.difftime 7
as.list method 8

as.mondate . . . 8
Coersion-from-mondate methods . . . 9
Combining-methods . . . 11
Compare-methods . . . 12
cut.mondate . . . 13
cutmondate-methods . . . 15
diff.mondate . . . 16
difftime-class . . . 17
displayFormat-methods . . . 17
format.mondate . . . 18
funcNULL-class . . . 19
get.mondate.displayFormats . . . 19
matrix-methods . . . 20
Miscellaneous-methods . . . 20
mondate-class . . . 22
mondate-methods . . . 24
names method . . . 28
print-methods . . . 29
seq.mondate . . . 29
seqmondate-methods . . . 31
subtract . . . 32
Summary-methods . . . 33
timeunits-methods . . . 33
year-month-day-methods . . . 34
YearQuartersFormat . . . 36
[-methods . . . 37

Index **38**

---

add *Add numerics to mondates using day-of-month logic*

Description

Add numerics to mondates. When units = "months", "years", or "quarters" and the `numeric` is a whole number the result has the same day of the month subject to the number of days in the month, thus abandoning `mondate`'s approach of representing days as fractional months. See also argument 'forcelastday'.

Usage

```
add(e1, e2, units, forcelastday = FALSE)
```

Arguments

e1	a mondate
e2	anumeric
units	Anything allowed by base:::as.difftime. In addition, can be "months" or "years". If missing, defaults to timeunits(e1).
forcelastday	If FALSE, the result will have the same day of the month subject to the number of days in the month. If TRUE, the day of the month of the result will be the last day of the month if e1 is on the last day of the month.

Value

A mondate.

Author(s)

Dan Murphy.

Examples

```
x <- mondate(0:12)
add(x, 1)             # The third date will the the 29th of March
x <- mondate.ymd(2013, 1:11, 15)
add(x, 1)             # Always the 15th of the month. Compare to ...
x + 1
stopifnot(add(x, 13, units = "months") == mondate.ymd(2014, 2:12, 15))
# Last day of February, not a leap year
x <- mondate("2-28-2014")
day(add(x, 1:4, units = "years")) # always the 28th of the month, vs. ...
day(add(x, 1:4, units = "years", forcelastday = TRUE)) # Feb. 29th in 2016
```

Arith-methods *Methods for Function Group Arith on mondate Objects*

Description

Arithmetic methods for class mondate. Includes three special "units between" methods.

Methods

signature(e1 = "mondate", e2 = "mondate") Performs arithmetic operations on two mondates. The result will be numeric with attribute timeunits belonging to the slot of the first argument (with a warning if the two arguments have different timeunits). The only operation that would seem useful for two mondates is subtraction.

signature(e1 = "mondate", e2 = "missing") Creates a mondate as 0 - e1.

signature(e1 = "mondate", e2 = "numeric")

signature(e1 = "numeric", e2 = "mondate")

`signature(e1 = "mondate", e2 = "array")`

`signature(e1 = "array", e2 = "mondate")` Performs arithmetic operations of a `numeric` on a `mondate` where the units of the `numeric` is taken from the `timeunits` slot of the `mondate`. The result will be a `mondate` with the same properties as the `mondate` in the function call.

The only operations that would seem useful for a `mondate` and a numeric are addition and subtraction.

Most of the time it is expected that `timeunits`="months" so that, for example, adding/subtracting a number to/from that `mondate` adds/subtracts that number of months. If the `mondate`'s `timeunits`="years" then the perceived intention is to add/subtract that number of years. To accomplish that, the `mondate`'s numeric value is divided by 12, the operation is performed, and the result converted to a `mondate`. If the `mondate`'s `timeunits`="days", then the `mondate`'s value is converted to the number of days since the start of the millennium, the operation is performed (e.g., the `numeric` number of days is added or subtracted), and the result converted to a `mondate`. (See the `convert` option of the `as.numeric("mondate")` function.)

`+signature(e1 = "mondate", e2 = "difftime")`

`-signature(e1 = "mondate", e2 = "difftime")` Use a difftime object to add and subtract secs, days, weeks, months, and years to or from a `mondate`.

`-signature(e1 = "mondate", e2 = "mondate")` Returns a difftime object equal to the signed number of units between `e1` and `e2`, where "units" = `timeunits(e1)`, with a warning if `e1` and `e2` have differing `timeunits`.

`-signature(e1 = "mondate", e2 = "Date")` Returns a difftime object equal to the signed number of units between `e1` and `e2` **- 1 day**, where "units" = `timeunits(e1)`.

`-signature(e1 = "mondate", e2 = "ANY")` Returns a difftime object equal to the signed number of units between `e1` and `as.Date(e2)` **- 1 day**, where "units" = `timeunits(e1)`.

`signature(e1 = "mondate", e2 = "missing")` Creates a `mondate` as `0 - e1`.

`MonthsBetween(from = "mondate", to = "mondate")`

`MonthsBetween(from = "ANY", to = "ANY")` Same as `abs(from - to)` in months.

`YearsBetween(from = "mondate", to = "mondate")`

`YearsBetween(from = "ANY", to = "ANY")` Same as `abs(from - to)` in years, which is also the number of months between divided by 12.

`DaysBetween(from = "mondate", to = "mondate")` Same as `abs(from - to)` in days, which is also the difference between the `as.Date` representation of `from` and `to`.

See Also

`Arith`

Examples

```
M <- mondate("1-1-2010") # will display in U.S. date format
M - 1:12 # the 1st of the month for all months in 2009
       # in reverse chronological order; inherits M's displayFormat

x <- mondate(matrix(12 * 1:4, 2, 2)) # 2x2 matrix of 2000-2003 year ends
x
```

```
y <- x + 12                          # one year later, also a matrix
y
y - x                       # 2x2 matrix of 12s, with an attribute ("months")
MonthsBetween(x, y)         # same, without the attribute
YearsBetween(x, y)
DaysBetween(x, y)

## Use difftime object to add, subtract secs, days, weeks, months, years
x <- mondate(1)             # January 31, 2000
y <- as.difftime(1, units = "days")
x + y
x - y
yw <- as.difftime(1, units = "weeks")
x + yw
x - yw
x + as.difftime(1, , "days")
x + as.difftime(86400, , "secs")

x <- mondate.ymd(2012, 2, 29)  # leap day
x + as.difftime(1, , "years")  # last day of February 2013, not a leap day
x - as.difftime(1, , "months") # not the last day of January

# Using a mondate as an "as of" date, measure elapsed time in months
# from the beginning of Date's
asof <- mondate("2015-12-31")
eventDate <- as.Date("2015-01-01")
asof - eventDate     # 12 months
asof - "2015-01-01"  # ditto
```

`array-method` *Array Methods*

Description

Apply array attributes to a `mondate`.

Methods

`array(data = NA, dim = length(data), dimnames = NULL)` Shapes `mondate` `data` as a matrix. Inherits the `mondate`'s other properties. See `array` for further details.

Examples

```
M <- mondate.mdy(12, 31, 2006:2011, timeunits = "years") # 6 year-ends
array(M, c(2,3))
```

`as.data.frame method` *Coerce a mondate to a Data Frame*

Description

Function to coerce a mondate for containment in a data.frame.

Usage

```
## S3 method for class 'mondate'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

`x`	a `mondate`.
`row.names`	`NULL` or a character vector giving the row names for the data frame. Missing values are not allowed.
`optional`	logical. See `base::as.data.frame` for details.
`...`	optional additional arguments to be passed to or from methods.

Details

Although users would normally call `data.frame`, the `as.data.frame` function is the underlying workhorse that enables a `mondate` to be stored in a `data.frame`. When array-shaped mondates are stored, the number of rows is retained and if the dimension > 2 it is "flattened" in the usual `data.frame` sense. The `mondate` method is fashioned after the `as.data.frame.Date` method.

See Also

`data.frame`, `as.data.frame`, `format`

Examples

```
YE <- mondate.mdy(12, 31, 2001:2005)
data.frame(yearend = YE, value = 1000 * 1.05^(1:5)) # 5% annual inflation
```

`as.difftime` *Difftime with units Months and Years*

Description

Expand difftime units to include months and years .

Usage

```
as.difftime(tim, format = "%X", units = "auto")
```

Arguments

`tim`	See base `as.difftime`. Must be `numeric` when `units` is "months" or "years".
`format`	See base `as.difftime`.
`units`	Anything allowed by `base::as.difftime`. In addition, can be "months" or "years" in which case `tim` must be `numeric`.

Details

Primarily used to facilitate adding months and years to `mondates`. See base `as.difftime`.

Value

See base `as.difftime`.

Author(s)

Dan Murphy.

See Also

Base `as.difftime`

Examples

```
x <- mondate(0:12)
y <- as.difftime(1, , "months")
x + y
x - y
```

as.list method *Construct a list from a mondate*

Description

Function to construct a list from a mondate.

Usage

```
## S3 method for class 'mondate'
as.list(x, ...)
```

Arguments

x	a mondate.
...	optional additional arguments to be passed to or from methods.

Details

Constructs a list from a mondate object. The length of the list equals the length of x. Each sublist contains an individual element of x. This method is most useful when a mondate is the X argument of one of the *apply functions. See an example of a recommended call to sapply in "Examples" below.

Examples

```
YE <- mondate.mdy(12, 31, 2011:2012)
sapply(YE, class)    # "mondate" "mondate"
sapply(YE, month)   # same as month(YE)
month(YE)
```

as.mondate *As.Mondate Method*

Description

Coerce an object to class mondate.

Usage

```
as.mondate(x, ...)
```

Arguments

x	an R object.
...	optional arguments passed to other methods.

Details

This is a convenience function that simply calls the appropriate `mondate` conversion method depending on the class of `x`.

Value

A `mondate` if coersion is successful.

Author(s)

Dan Murphy

See Also

`mondate-methods`

Examples

```
y <- as.Date("2012-12-31")
as.mondate(y)
```

`Coersion-from-mondate methods`
Coersion Methods for Mondates

Description

Methods to coerce a `mondate` to other R objects. Currently that includes numbers, characters, and three classes of dates.

Usage

```
 ## S3 method for class 'mondate'
as.character(x, format, ...)
 ## S3 method for class 'mondate'
as.Date(x, ...)
 ## S3 method for class 'mondate'
as.POSIXct(x, ...)
 ## S3 method for class 'mondate'
as.POSIXlt(x, ...)
 ## S4 method for signature 'mondate'
as.numeric(x, convert = FALSE, stripdim = FALSE,
               timeunits = c("months", "years", "days"),
               ...)
```

Arguments

x	a mondate
format	the format to give the Date representation of x
...	arguments passed to and from other methods
convert	See Methods
stripdim	See Methods
timeunits	See Methods

Methods

as.character(x = "mondate", format, ...) Coerce mondate to class character. Uses the format function.

format If missing the value is drawn from the displayFormat property of x.

... arguments passed to other methods (e.g., format).

as.numeric(x = "mondate", convert=FALSE, stripdim=FALSE, timeunits=c("months", "years", "days"), ...) Coerce mondate to class numeric.

convert: FALSE (the default) is equivalent to getDataPart. If TRUE the result will be converted to the number of years since the beginning of the millennium if timeunits="years"; to the number of days since the beginning of the millennium if timeunits="days". Also in the case that convert=TRUE the numeric returned will have "timeunits" as an attribute.

stripdim: FALSE (the default) retains the array attributes dim and dimnames. If TRUE the dimension attributes are stripped, which is the default behavior of base::as.numeric.

timeunits If missing the value is drawn from the property of the mondate.

as.Date(x = "mondate") Coerce mondate to class Date

as.POSIXlt(x = "mondate") Coerce mondate to class POSIXlt

as.POSIXct(x = "mondate") Coerce mondate to class POSIXct

Examples

```
(b<-mondate(1))              # end of first month of current millennium
as.numeric(b)                # 1
as.character(b)              # December 31, 2000 in date format of locale
as.character(b, format="%b-%Y")  # "Dec-2000"
as.numeric(b, convert=TRUE, timeunits="years") # converts to 1/12 "years"
(b<-mondate(1, timeunits="days")) # end of first day of millennium
as.numeric(b)                # 1/31
as.numeric(b, convert=TRUE)  # 1 (with a "days" attribute)
as.Date(b)                   # displays as "2000-01-31"
as.POSIXct(b)                # displays as "2000-01-31 UTC"
weekdays(as.POSIXct(b))      # January 31, 2000 was a "Saturday" (in English)
as.POSIXlt(b)$hour           # zero, as are ...$min and ...$sec
```

Combining-methods *Methods for Combining Mondates*

Description

Methods to combine mondates.

Usage

```
 cbindmondate(..., deparse.level = 1)
 rbindmondate(..., deparse.level = 1)
 ## S3 method for class 'mondate'
rep(x, ...)
 ## S4 method for signature 'mondate'
c(x, ..., recursive = FALSE)
```

Arguments

x	a mondate
deparse.level	see base::cbind
recursive	see base::c
...	arguments passed to and from other methods

Details

The package calls setGeneric("c-rbind").

Value

c-rbindmondate(...)

The cbindmondate and rbindmondate functions are similar to the base cbind and rbind functions, respectively, to combine the arguments. If all arguments are mondates then the result is converted to a mondate with displayFormat and timeunits properties equal to those of the first argument in .... If not all arguments are mondates then the result is a data.frame by virtue of the call cbind.data.frame(...).

A mondate (or a data.frame from c-rbindmondate when ... holds non-mondate arguments). For c and rep, a vector.

Methods

c(x = "mondate", ...) Combine mondates into a vector. ... any R object(s) that can be coerced to a mondate. The behavior mimics that of the base function. The result will be a mondate with properties equal to those of x.

rep(x = "mondate", ...) Replicates a mondate. The behavior mimics that of the base function. See rep for further details. The result will be a mondate with properties equal to those of x.

Examples

```
x <- mondate(1:6) # first 6 month-ends of the year 2000
c(x,x+6)          # all month-ends of 2000
c(0,x)            # result is "numeric", as determined by the first argument

M<-mondate.ymd(2001:2005,12,31) # 5 year-ends
names(M)<-LETTERS[1:5]
cbindmondate(M)                      # as a 5x1 matrix
rbindmondate(M,M)
begin_date <- M-12
cbindmondate(begin_date,end_date=M)  # 5 pairs of year boundary-dates. Columns
                              # are "automatically" named in the default case
                              # (all mondates with timeunits="months").
dayt <- as.Date("2010-6-30")
cbindmondate(x,mondate(dayt))        # column names show as 'x' and blank
cbindmondate(x=x,DateColumn=mondate("2010-6-30")) # both columns are named

rep(mondate("2010-2-14"), 3)

(M<-seq(from=mondate("1/1/2010"),length=2)) # Jan. and Feb. 1st
rep(M,3)                                    # three pairs
rep(M,each=3)                               # three Jan.'s, three Feb.'s
```

`Compare-methods` *Comparison Methods*

Description

Methods for the `Compare` group of functions.

Methods

`signature(e1 = "mondate", e2 = "mondate")` compares two `mondates`. The usual recycling rules apply to the shorter of the two `mondates`. The result will be logical. The usual rules apply as to the shape of the result.

See Also

`Compare`

Examples

```
A<-mondate.ymd(2001:2003,12,31) # three year ends
B<-mondate.ymd(2001:2003, 6,30) # three mid-years
B<A                             # c(TRUE, TRUE, TRUE)
```

cut.mondate *Convert a mondate Object to a Factor*

Description

Method for cut applied to mondate objects.

Usage

```
 ## S3 method for class 'mondate'
cut(x, breaks, labels = NULL,
      include.lowest = FALSE, right = TRUE,
      start.on.monday = TRUE, startmonth = NULL, startyear = NULL,
      attr.breaks = FALSE, ...)
```

Arguments

x	a mondate
breaks	a vector of cut points or number giving the number of intervals which x is to be cut into or an interval specification, one of "day", "week", "month", "quarter" or "year", optionally preceded by an integer and a space, or followed by "s" (pluralized).
labels	labels for the levels of the resulting category. By default, labels are constructed from the right-hand end of the intervals (which are included for the default value of right). If labels = FALSE, simple integer codes are returned instead of a factor.
include.lowest	logical, indicating if an 'x[i]' equal to the lowest (or highest, for right = FALSE) 'breaks' value should be included. (Note: Beginning with version 0.11 of mondate, !include.lowest is invalid when breaks is character.)
right	logical, indicating if the intervals should be closed on the right (and open on the left) or vice versa.
start.on.monday	logical. If breaks = "weeks", should the week start on Mondays or Sundays?
startmonth	If not NULL, the number of the month (1 = January, ..., 12 = December) corresponding to the beginning of the year/fiscal-year. If NULL, the resulting intervals begin or end depending on the minimum (!right) or maximum (right) value, respectively, of x. This argument only has an impact when character breaks defines multi-month intervals.
startyear	If not NULL, the year corresponding to the left-most level. If NULL, the resulting intervals begin or end depending on the minimum (!right) or maximum (right) value, respectively, of x. This argument only has an impact when character breaks defines month or multi-month intervals.
attr.breaks	logical. If TRUE the result has a "breaks" attribute which is a mondate whose pairwise values determine a covering of x. Most helpful when breaks = "days", "weeks", "months", "quarters", or "years".
...	optional arguments passed to or from other methods.

Details

This method converts a mondate to a factor corresponding to time intervals as determined by the argument breaks. In the "end-of-business-day" spirit of the mondate package, the intervals (levels) are labeled by the last day in the interval. If right = FALSE the levels are labeled by the left endpoint of the interval, which is **the day immediately preceding the beginning of the period**. For methods that work in the spirit of Date and POSIXt – i.e., a time interval is better labeled by the first day of the period – see cutmondate.

For numeric breaks – which case also includes mondates – the method calls cut.default and the intervals encoded in the levels are converted to date format using mondate logic. In the spirit of mondate, the default right = TRUE ensures that the intervals are left-open/right-closed. The default include.lowest = TRUE ensures that min(x) is included in the first interval.

For breaks = "days" or "weeks", the method calls cut(as.Date(x)). For breaks = "months", "quarters", and "years", numeric breaks are determined from the range of x.

If breaks is preceded by an integer, call it step, then the period of the first level is determined by min(x) and subsequent "day", "week", "month", "quarter", and "year" periods are determined sequentially per seq(min(x), max(x), by = step).

When attr.breaks = TRUE, the result will have a "breaks" attribute (attr(., "breaks")) which pairwise "cover" x. Such "breaks" are suitable be use by 'hist', for example.

In all cases, the formats of the dates representing the levels are inherited from displayFormat(x). Accordingly, if such resulting potential levels would be non-unique, cut.mondate resorts to the fall-back scheme of cut.default, i.e., "labels such as "Range3" will be used."

Value

A factor is returned, unless labels = FALSE which returns the integer level codes.

Author(s)

Dan Murphy. Many thanks to the R-development team for cut and cut.Date.

See Also

cut, cut.Date, and cutmondate for methods that work better with Dates.

Examples

```
# days
x <- mondate.ymd(2013, 1, 1:7)
cut(x, breaks = "days", include.lowest = TRUE)
cut(x, breaks = "days", include.lowest = TRUE)

# weeks
x <- mondate.ymd(2013, 1, 1:31)  # days in January 2013
# labeled by the first 5 Sundays of 2013
cut(x, breaks = "weeks", include.lowest = TRUE, right = TRUE)

x <- mondate.ymd(2015, 1, 5:25)  # days in January 2013
# labeled by the 1st and 3rd Mondays of 2015
```

```
cut(x, breaks = "2 weeks", include.lowest = TRUE, right = FALSE)

# months
x <- mondate.ymd(2013, 1:12, 15) # 15th day of each month in 2013
# labeled by the left endpoint of the interval = last day of the month
cut(x, breaks = "months", include.lowest = TRUE)
# labeled by the right endpoint of the interval = last day of the prior month
cut(x, breaks = "months", right = FALSE, include.lowest = TRUE)
cut(x, breaks = "2 months", include.lowest = TRUE)

# quarters
x <- mondate.ymd(2013, 1:12, 15)          # 15th day of each month in 2013
cut(x, "quarters", include.lowest = TRUE) # labeled by last day of the quarter
cut(x, "2 quarters", include.lowest = TRUE)

# years
m <- mondate(0:12)
cut(m, "years", include.lowest = TRUE)  # labeled by last day of the year 2000
cut(m, "years", right = FALSE, include.lowest = TRUE) # labeled by first day of 2000
displayFormat(m) <- "%Y"
cut(m, "years", include.lowest = TRUE)  # labeled by just the year of the date

# numeric scalar breaks

# Four intervals starting with 2015-01-15 (exclusive) and ending with
# 2015-12-15 (inclusive)
# First factor value is NA because include.lowest is FALSE (default)
x <- mondate.ymd(2015, 1:12, 15)
cut(x, breaks = 4)
# Now 2015-01-15 will be included in the first left-closed interval.
cut(x, breaks = 4, include.lowest = TRUE)

# specific break boundaries -- month-ends
cut(x, breaks = mondate("2014-12-31") + 0:12) # levels show interval boundaries
# Compare with next, where levels are labeled by interval endpoints
cut(x, breaks = "month", include.lowest = TRUE)
```

`cutmondate-methods` *Methods to Generate Date Cuts*

Description

Methods to cut sequences of mondate, Date, and POSIXt objects. This is essentially a wrapper for the `cut.mondate` method with more intuitive defaults for the arguments depending on the object being cut.

Methods

`signature(x = "mondate")` calls the cut.mondate function

`signature(x = c("Date", "character"))` checks for `breaks` = "months", "years", or "quarters". If so, calls the cut.mondate function using x = mondate(x, displayFormat = " otherwise, simply passes the arguments on to the cut.Date method. The display argument to "mondate" ensures that the levels will display according to the default "Date" format.

`signature(x = c("Date", "missing"))` if `breaks` is omitted, "months" is assumed.

`signature(x = c("Date", "ANY"))` if `breaks` is anything else, the entire argument list is passed on to cut.Date.

`signature(x = "POSIXt")` cutmondate is called again with x = as.Date(x).

`diff.mondate` *'diff' for package mondate*

Description

Returns suitably lagged and iterated differences of an object of class **mondate**.

Usage

```
## S3 method for class 'mondate'
diff(x, lag = 1L, differences = 1L, ...)
```

Arguments

`x`	a mondate vector or matrix containing the values to be differenced.
`lag`	an integer indicating which lag to use.
`differences`	an integer indicating the order of the difference.
`...`	further arguments to be passed to or from methods.

Details

See the `diff` function in `base`.

Value

If `x` is a vector of length `n` and `differences=1`, then the computed result is equal to the successive differences `x[(1+lag):n] - x[1:(n-lag)]`.

If `difference` is larger than one this algorithm is applied recursively to `x`. Note that the returned value is a vector which is shorter than `x`.

If `x` is a matrix then the difference operations are carried out on each column separately.

Author(s)

Dan Murphy

See Also

diff

Examples

```
evalDate<-mondate(c(12+12*1:10))  # year ends 2001, ..., 2010
diff(evalDate)                    # vector of length 9 holding the number 12,
                                  # with "timeunits" attribute = "months"
```

---

difftime-class *Class "difftime"*

Description

Register old-style (a.k.a. 'S3') class as a formally defined class.

Author(s)

Dan Murphy

---

displayFormat-methods *Methods to Access 'displayFormat' Property*

Description

Methods to get and set the displayFormat value of a mondate.

Usage

```
## S4 method for signature 'mondate'
displayFormat(x)
## S4 method for signature 'ANY'
displayFormat(x)
## S4 replacement method for signature 'mondate'
displayFormat(x)<-value
```

Arguments

x	a mondate.
value	For the "get" method, a character string indicating the date format with which to display the mondate. Choices are currently 1. "%m/%d/%Y" 2. "%m-%d-%Y" 3. "%Y-%m-%d" 4. "%Y/%m/%d" If x is not a mondate, the "get" value returned is NULL.

Note

The mondateDisplayFormat versions have been deprecated.

Examples

```
x<-mondate("2010-6-30")            # The middle of 2010
displayFormat(x)                   # "%Y-%m-%d"
displayFormat(x) <- "%m/%d/%Y"
x                                  # x now displays as 06/30/2010
```

---

format.mondate *Format a mondate*

Description

Function to format a mondate into its character representation according to the displayFormat property.

Usage

```
## S3 method for class 'mondate'
format(x, ...)
```

Arguments

x	a mondate.
...	further arguments passed to or from other methods.

Details

For more details see format and especially strptime.

Value

character representation of the mondate.

See Also

strptime.

Examples

```
(b<-mondate(1))   # end of first month of millennium
format(b)         # "01/31/2000" -- with quotes -- in the U.S. locale
format(b, format="%Y-%m-%d")  # "2000-12-31"
```

funcNULL-class *Class "funcNULL"*

Description

A class representing a function for special mondate formatting if necessary or NULL if not.

Author(s)

Dan Murphy

get.mondate.displayFormats *get or set the vector of formats to use when converting character to mondate*

Description

Methods to access the vector of displayFormats used by default.

Usage

```
get.mondate.displayFormats()
set.mondate.displayFormats(x, clear = FALSE)
```

Arguments

x	a character value(s) holding additional formats to be used for converting character to mondate
clear	logical indicating if the current formats should be "cleared" and completely replaced by the value(s) in x. If FALSE, the value(s) in x are appended to the current set.

Value

A vector of formats.

Author(s)

Dan Murphy

Examples

```
get.mondate.displayFormats()
# Attempting to convert a date given in the French format dd/mm/yyyy actually
# works because the European format %Y/%m/%d successfully translates
# that character string in the same way as.Date translates it.
m <- mondate("31/08/2015")
m
year(m) # 31 .. probably an unreasonable value
d <- as.Date("31/08/2015")
d
year(d)
# Ensure the appropriate format is found first by
# reprioritizing it in the list
set.mondate.displayFormats(c(Frb = "%d/%m/%Y", get.mondate.displayFormats()),
                           clear = TRUE)
m <- mondate("31/08/2015")
m
year(m) # 2015
```

---

matrix-methods *Matrix Methods for Mondate's*

Description

Apply matrix attributes to a mondate.

Methods

`matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE, dimnames = NULL)` Shapes mondate data as a matrix. Inherits the mondate's other properties. See `matrix` for further details.

Examples

```
m <- mondate.mdy(12, 31, 2001:2006) # 6 year-ends
matrix(m)                           # a one-column matrix
matrix(m, 2, byrow=TRUE)            # a two-row matrix stored in row-order
```

Miscellaneous-methods *Miscellaneous Methods for mondate's*

Description

Miscellaneous mondate methods.

Usage

```
 ## S3 method for class 'mondate'
mean(x, trim = 0, na.rm = FALSE, ...)
 ## S3 method for class 'mondate'
unique(x, ...)
 ## S3 method for class 'mondate'
quarters(x, abbreviate)
```

Arguments

x	a mondate
trim	see base::mean
na.rm	see base::mean
abbreviate	logical. Should the names be abbreviated?
...	arguments passed to and from other methods

Methods

`mean(x = "mondate", ...)` Calculate the mean date of mondates. Arguments `trim` and `na.rm` have the usual meaning (see `base::mean`).

`pmean(... = "mondate")` Calculate the "parallel" mean date of mondates. Arguments in `...` must all be mondates. Result will be a mondate with properties equal to those of the first mondate in `....`

`unique(x = "mondate", ...)` Returns a mondate but with duplicate elements/rows removed. For an explanation of the arguments in `...`, see `base::unique`.

`quarters(x = "mondate", abbreviate)` Returns a character vector of "Q1" to "Q4". See quarters.

Examples

```
(M<-mondate.mdy(12,31,2001:2003))
mean(M)                          # the middle value, Dec. 31, 2002
(M<-c(M,mondate.mdy(12,31,2004))) # tack on another yearend
mean(M)                          # mid-year 2003
mean(M,12)                       # 12 is coerced to Dec. 31, 2000, so the
                                 # mean is again Dec. 31, 2002

x <- mondate.ymd(2001:2005,12)   # five year ends
y <- x-12                        # one year earlier
pmean(x,y)                       # 2001-06-30 ... 2005-06-30

unique(M,M)                      # just M
(M<-matrix(M,nrow=2))            # now a matrix
rbind(M,M)                       # 2 M's, stacked
unique(rbind(M,M))               # M again, still a matrix

m <- mondate.ymd(2013, 1:12)     # end of the months of 2013
quarters(m)
```

`mondate-class` *Class "mondate"*

Description

A `mondate` represents a date as a `numeric` equalling the number of months since the beginning of the current millennium (the "mondate epoch"). Somewhat arbitrarily, and at the risk of reopening a decade-old debate, "the beginning of the current millennium" is defined as the instant between December 31, 1999 and January 1, 2000.

The need for a "mondate" class arises in the area of actuarial analysis, and other areas of financial modeling that need to reconcile to a company's book of accounts. Its motivation is based on the following presumptions:

1. Business accounting-wise, the closing of the books for a month, quarter, and year are the important milestones for measuring time.
2. For accountants – and actuaries – it is usually not important to measure events on an hourly basis.
3. All events that occur during a business day, up to and including the closing of the books for a day, are all "accounted for" as having occurred "at the same time."

To appreciate the difficulty in measuring the passage of time in days, note that there are typically three fewer days in the first half of the year (January 1 through June 30) than there are in the second half. Yet accountants will say that on June 30th the year is half over. For another example, note that – with the exception of July/August and December/January – the same days of the month for two consecutive months are not one "month" apart if measured in days because, with those exceptions, consecutive months contain differing numbers of days, so which of the two months do you choose as the yardstick? Since changes in accounts over the course of a month, quarter and year are the amounts by which financial results are measured, it is important to be able to measure the passage of time where a year is comprised of twelve months of equal "accounting weight."

That gives rise to a date as measured in months, coined "mondate".

A `mondate` is simply a real number whose fractional part represents the fraction of the month as of the end of the day. E.g., the fractional part of January 1st = 1/31; the fractional part of February 1st = 1/28 or 1/29, depending on the year. A `mondate` which is a whole number (i.e., no fractional part) corresponds to a month that is fully completed, whose subsequent month has not yet begun; i.e., the instant in time between one month and the next.

The length of time (in months) between `mondate x` and `mondate y` is simply `y-x`. An interval of time is represented in a mathematical sense by the half-open/half-closed interval **(x,y]**. For example, calendar year 2009 is the interval

(2008-12-31, 2009-12-31]

i.e., all events *after* the close of business **2008-12-31** and *through and including* the close of business **2009-12-31**. The mondate vector

c(mondate("2008-12-31"), mondate("2009-12-31"))

could be used to represent such an interval.

A mondate allows "infinite dates", which are helpful constructs for modeling "ultimate" events in actuarial, longitudinal, and time series analyses (see the mondate method for signature "numeric").

A mondate has two important properties (in S4 parlance, "slots"). The first is displayFormat which is the format of the date when the mondate is printed or shown. The other is timeunits which is the "units" reported when date arithmetic is performed. The default units is "months", but "years" and "days" are also allowed, in which case the difference of two dates, say, would be reported in "years" or "days", respectively.

Objects from the Class

Objects can be created by calls of the form mondate(x, ...).

Slots

.Data: Object of class "numeric" or a numeric array

displayFormat: Object of class "character". The is the format of the date when displayed. Currently, there are four choices:

1. "
2. "
3. "
4. "

Currently, the default displayFormat for a "United States" locale is mm/dd/YYYY (" in all other locales YYYY-mm-dd (" The default can be changed to another format using options(mondate.displayFormat = myFormat). See Examples.

timeunits: Object of class "character" There are three options:

"months": the default

"years": although it is assumed that "month" is the fundamental unit of time, sometimes it is more convenient to report the result of date arithmetic in terms of "years", for example, in actuarial analyses when events are measured over the course of years, not months. Of course, one "year" = twelve "months".

"days": mostly for convenience when the number of days between events needs to be reported. Of course, unlike with "years", there is no simple relationship between "days" and "months" – it depends on the month and year of the date. If the *fundamental* unit of time for a particular problem is "days", not "months", then a different date class (e.g., class Date) might be a better tool for modeling the passage of time for that problem.

The default can be changed to a different unit using the options(mondate.timeunits = myUnits) command. See Examples.

formatFUN: Object of class "funcNULL" Use this slot to store a function to format a mondate. See Examples.

Extends

Class c("numeric","array"), from data part.

Author(s)

Dan Murphy

References

For information about how month-based time measurement can help with standardizing rates of return, see Damien Laker, "Time Calculations for Annualizing Returns: the Need for Standardization," *The Journal of Performance Measurement*, Summer 2008, pp. 45-54.

See Also

yearmon in the zoo package.

Examples

```
# See the \code{mondate-methods} section for an
# explanation of the \code{mondate} method below.

# January 1, 2010 in US displayFormat
mondate("1-1-2010")

# generate 10 random dates in calendar year 2000; will be
# displayed in local format
mondate(runif(10,0,12))

# Change the default format so that the character representation of the date
#   sorts in chronological order.
options(mondate.displayFormat = "%Y-%m-%d")
# January
mondate(runif(10,0,12))

# generate 10 random dates in calendar year 2010;
# date arithmetic results will be reported in units of "years"
mondate(10+runif(10),timeunits="years")
```

---

mondate-methods *Create an instance of the mondate class*

Description

All purpose **mondate** constructor / coercer.

Usage

```
mondate(x,
  displayFormat = getOption("mondate.default.displayFormat",
                            default = .get.default.displayFormat()),
  timeunits = getOption("mondate.timeunits",
                        default = .get.default.timeunits()),
```

```
      ...)
    ## S4 method for signature 'mondate'
mondate(x, displayFormat = x@displayFormat,
                  timeunits = x@timeunits, formatFUN = x@formatFUN, ...)
    ## S4 method for signature 'numeric'
mondate(x, displayFormat, timeunits, ...)
    ## S4 method for signature 'Date'
mondate(x, displayFormat, timeunits, ...)
    ## S4 method for signature 'POSIXt'
mondate(x, displayFormat, timeunits, ...)
    ## S4 method for signature 'character'
mondate(x, displayFormat = "keep", timeunits, format, ...)
    ## S4 method for signature 'array'
mondate(x, displayFormat, timeunits, ...)
    ## S4 method for signature 'missing'
mondate(x, displayFormat, timeunits, ...)
    ## S4 method for signature 'ANY'
mondate(x, displayFormat, timeunits, ...)
```

Arguments

`x`	an R object to convert to a `mondate`. Can be another `mondate`, a character representing a date, a date, a numeric, or an object which converts to a numeric with as.numeric(x). More details below.
`displayFormat`	character string representing the date format with which to display the `mondate`. The default.displayFormat is determined at the time an instance is created according to Sys.getlocale("LC_TIME"): if it contains the words "United States", the default will be "%m/%d/%Y" (MM/DD/YYYY), otherwise "%Y-%m-%d" (YYYY-MM-DD). Other choices are "%m-%d-%Y" and "%Y/%m/%d". See "Details" section for how to change defaults.
`timeunits`	character string "months" (default), "years", or "days" indicating the units in which date arithmetic will be carried out.
`formatFUN`	format function for converting a mondate to character. In case of conversion from `mondate`, default is to inherit the value.
`format`	format string for converting a character to a Date (using `as.Date(x, format, ...)`) from which the mondate value is determined.
`...`	arguments to be passed to other methods.

Details

Package users can change the default values of `displayFormat` and `timeunits` using `options()` with the names "mondate.displayFormat" and "mondate.timeunits", respectively. Warning! Use with care! No checks are performed if and when the options are established. It is up to the user to ensure the new defaults are valid – `displayFormat` must be appropriate for formatting dates in R and `timeunits` must be one of "months", "years", or "days". See an example below.

Methods

`signature(x = "mondate")` For `mondate` x, this could be a way to copy a `mondate` and perhaps change the `mondate`'s `displayFormat` or `timeunits` slots in the process. For any class that extends `mondate`, use of this method will return the underlying mondate class without additional slots (if any) of the subclass.

`signature(x = "numeric")` For `numeric` x, the simplest case is when `timeunits` = "months", in which case the value of x and properties `displayFormat` and `timeunits` are simply stored. If `timeunits` = "years" then it is presumed that the value of x represents the number of years since the beginning of the millennium, in which case the value of x is multiplied by 12 and then stored. If `timeunits` = "days" then it is presumed that the value of x represents the number of days since the beginning of the millennium, in which case the value is calculated using `as.Date`. Note that infinite values of `x` are allowed, helpful in actuarial ("at ultimate") longitudinal, and time series modeling.

`signature(x = "Date")`

`signature(x = "POSIXt")` For a date x, `as.POSIXlt` is used to convert to an ISO standard date, from which the number of months of that day since the beginning of the millennium is calculated.

`signature(x = "character")` If `format` is provided, then that format is used to attempt to convert the character value to a date. Otherwise, characters are converted to dates using the first format found in the set of valid formats that successfully converts the first non-NA entry in `x`, and that format is retained as the `displayFormat` of the result unless the user explicitly provides a value for `displayFormat`. The current set of valid formats is "%m/%d/%Y", "%m-%d-%Y", "%Y-%m-%d", and "%Y/%m/%d". If any entries of `x` do not convert successfully, those entries get the value `NA` and a warning is issued. Finally, if `format` is not provided and none of the valid formats successfully converts `x` to a date, then as a last resort the character string is attempted to be coerced to a `numeric` and then to a `mondate`.

`signature(x = "factor")` The `character` method is run on `as.character(x)`.

`signature(x = "array")` If an object `x` is an `array`, then this method enables the `mondate` to inherit its shape. After that, other "signatures" take over.

`signature(x = "missing")` Enables the call `mondate()` to work. Useful for prototypes, e.g. Body of method is simply `new("mondate")`.

`signature(x = "ANY")` For any other class of x an attempt will be made to convert to `Date` ("`as.Date(x)`"). If unsuccessful, an attempt will be made to convert to `numeric`; if successful, a warning will be issued to check the results relative to the `numeric` conversion, otherwise execution will be `stopped`.

See Also

`POSIXt`, `yearmon`, `yearqtr`

Examples

```
mondate("1-31-2010") # Jan. 31, 2010
mondate(60)          # 60 months after 12/31/1999, so Dec. 31, 2004
dat <- as.Date("2010-1-31")
(M <- mondate(dat))    # Jan. 31, 2010
```

```
x <- 12 * 1:6
mondate(x)          # first 6 yearends in 2000's
y <- x + 12
mondate(cbind(x,y))  # bounding dates of first 6 years of millennium
(y <- mondate(1:6,timeunits="years")) # first 6 yearends, 'years' timeunits
# The results of date arithmetic on y will be displayed in "years".
# E.g., the differences of y can be calculated as:
tail(y,-1) - head(y,-1)# vector of five 1's, with "timeunits" attribute = "years"
as.numeric(x)
as.numeric(y)        # the underlying numeric representations are the same

# Demonstrating "infinite" dates
y <- c(y,Inf)
y                    # last element shows as Inf
tail(y,-1) - head(y,-1)# last element is now infinity

# The zoo examples point out a difference between zoo and mondate.
#   zoo assumes that the zero-th part of a month or quarter is the first
#   day of the month or quarter, whereas mondate assumes that it is
#   the instant before the first day of the month or quarter.
# Since frac=0 is zoo's as.Date coersion default, a month or quarter in
#   zoo's sense converts to the end of the first day rather than
#   the beginning.
library(zoo)
x <- ts(1:10, frequency = 4, start = c(1959, 2)) # starting 2nd qtr of 1959
x
# There is no method for class 'ts' so x is coerced (successfully)
#   because that class has an as.Date method, but with a warning.
# The result is a vector of length 10 representing the close of business
#   at the end of the first day of each of the given quarters.
mondate(x)

# The yearmon class will identify any day in June 2010 with that month.
as.yearmon("2010-6-15")
mondate(as.yearmon("2010-6-15"))         # end of first day of June 2010
mondate(as.yearmon("2010-6-15", frac=1)) # end of  last day of June 2010
mondate(as.yearqtr("2010-2", frac=1))    # same

# The if missing, displayFormat will be determined from the character input
x <- mondate("2010-12-31")
x               # x displays in the input European format
# The provided, displayFormat must match the format of the character input
#   or NA's will result.
mondate("2010-12-31", displayFormat = "%m-%d-%Y") # results in NA

# Always display x using just the year
x <- mondate(as.Date("2012-3-1"), displayFormat="%Y")
x                # shows as the year 2012, but month and day are nevertheless retained
month(x)         # 3
day(x)           # 1

# Change the default displayFormat to only display the year and month
options(mondate.displayFormat = "%Y-%m")
```

```
y <- mondate(as.Date("2013-12-31"))
y
# mondate: timeunits="months"
# [1] 2013-12
# Previous mondate instances retain their display formats:
x
# mondate: timeunits="months"
# [1] 2012
```

`names method` *Assign names to a mondate.*

Description

Function to assign names to a mondate.

Usage

```
## S3 replacement method for class 'mondate'
names(x) <- value
```

Arguments

`x`	a `mondate`.
`value`	the names to assign to `x`

Details

Assigns the names attribute to the .Data part of `x`.

Examples

```
YE <- mondate.mdy(12, 31, 2011:2012)
names(YE) <- c("A", "B")
```

print-methods *Methods to Display a Mondate*

Description

Methods to display a mondate in an R session.

Usage

```
    ## S3 method for class 'mondate'
print(x, ...)
    ## S4 method for signature 'mondate'
show(object)
```

Arguments

x	a mondate
object	a mondate
...	arguments passed to and from other methods

Methods

print(x = "mondate", ...) Print the date in its character representation using the displayFormat property. Uses the noquote function. Arguments in ... are passed to base::print.

show(object = "mondate") Same as print above, but with a "header" showing the timeunits property.

seq.mondate *Mondate Sequence Generation*

Description

Generate regular mondate sequences.

Usage

```
## S3 method for class 'mondate'
seq(from, to, by, ..., right = TRUE)
```

Arguments

from	coercible to a mondate. May be "missing".
to	coercible to a mondate. May be "missing".
by	increment of the sequence. If numeric, a sequence is generated using the underlying numeric value(s) of the coerced argument(s) and converted to a mondate object (see seq). If "days" or "weeks", a sequence is generated using "Date" objects coerced from the mondate-coerced arguments (see seq.Date). If "months", "years", or "quarters", by is converted to 1, 12, or 3, respectively, and the numeric case continues. Furthermore, the characters can optionally be preceded by a (positive or negative) integer and a space, and the singular form may also be used.
...	optional arguments passed to seq or, if by "day" or "week", seq.Date
right	in the case when by specifies units of multiples of months ("months", "years", or "quarters") should increments be considered as attaching to the end of the day (the mondate default). If FALSE, the increment attaches to the beginning of the day (see "Examples")

Details

For more details about sequence generation, see seq.

If from and to are both provided, the displayFormat and timeunits properties are taken from from, without a warning if from's properties differ from to's.

Value

A mondate vector with slots (including displayFormat, timeunits, formatFUN) from argument from, if provided, otherwise from argument to.

Author(s)

Dan Murphy

See Also

seq, seq.Date and seqmondate

Examples

```
x <- mondate.ymd(2010, 1)   # January 31, 2014
y <- mondate.ymd(2010, 12)  # December 31, 2014
seq(from = x, to = y, by = 1)  # all month-ends in 2014
# 8 quarter-ends beginning 1st quarter 2009; US displayFormat
seq(mondate("3/31/2009"), by = 3, length.out = 8)
# 8 quarter-ends ending year-end 2009; R's date format
seq(to = mondate("2009-12-31"), by = 3, length.out = 8)
#
# Use of RIGHT = FALSE in seq.mondate is depracated.
# Instead, to generate sequences of
```

```
#   Dates corresponding to the first days of the months
# use the seq.Date method.
seqmondate(as.Date("2014-01-01"), by = "months", length = 12)
# To generate the last day of a sequence of months,
#   use the default right = TRUE:
(m <- seq.mondate(as.Date("2014-01-31"), by = "months", length = 12))
# Coerce back to Date if necessary:
as.Date(m)
# Note how the Date method yields a different sequence, not corresponding to
#   the last day of the month.
seq(as.Date("2014-01-31"), by = "months", length = 12)
```

---

seqmondate-methods *Methods to Generate Date Sequences*

Description

Methods to generate date sequences. This is essentially a wrapper for `seq.mondate` that requires `from` and `to` to be of the same class – when both are specified – and returns a sequence of that class. The primary purpose is to generate sequences in units of "months" (the default value of the unnamed argument `by`).

Arguments

`from`

`to`	objects of class `mondate`, `Date`, `POSIXlt`, or `POSIXct`. May be "missing". If both present, must be of the same class.
`...`	optional arguments passed to `seq.mondate`

Value

A sequence of the same class as `from\to`.

Methods

`signature(from = "mondate", to = "mondate", ...)`

`signature(from = "Date", to = "Date", ...)`

`signature(from = "POSIXlt", to = "POSIXlt", ...)`

`signature(from = "POSIXct", to = "POSIXct", ...)`

`signature(from = "ANY", to = "ANY", ...)`

Examples

```
# 13 month-end dates starting with the end of 2014 and
#   ending with the end of 2015
seqmondate(mondate.ymd(2014), mondate.ymd(2015))

# In most situations, seq.Date and seqmondate return identical sequences,
# as when 'from' is at the beginning of the month.
s1 <- seq(as.Date("2015-01-01"), as.Date("2015-12-01"), by = "month")
s2 <- seqmondate(as.Date("2015-01-01"), as.Date("2015-12-01"))
stopifnot(identical(s1, s2))

# But when 'from' is near the end of a long month, seq.Date, seq.POSIXct, etc
# can step into subsequent months (as documented in ?seq.POSIXt).
seq(as.POSIXct("2015-01-31"), by = "month", length.out = 12)
# ... contrasted with ...
seqmondate(as.POSIXct("2015-01-31"), by = "month", length.out = 12)
```

---

subtract *Subtract numerics from mondates using day-of-month logic*

Description

Subtract numerics from mondates. When units = "months", "years", or "quarters" and the `numeric` is a whole number the result has the same day of the month subject to the number of days in the month, thus abandoning `mondate`'s approach of representing days as fractional months. See also argument 'forcelastday'.

Usage

```
subtract(e1, e2, units, forcelastday = FALSE)
```

Arguments

`e1`	a `mondate`
`e2`	an`umeric`
`units`	Anything allowed by `base:::as.difftime`. In addition, can be "months" or "years". If missing, defaults to `timeunits(e1)`.
`forcelastday`	If FALSE, the result will have the same day of the month subject to the numbere of days in the month. If TRUE, the day of the month of the result will be the last day of the month if `e1` is on the last day of the month.

Value

A `mondate`.

Author(s)

Dan Murphy.

Examples

```
x <- mondate(0:12)
subtract(x, 1)             # The third date will the the 29th of March
x <- mondate.ymd(2013, 2:12, 15)
subtract(x, 1)             # Always the 15th of the month. Compare to ...
x - 1
stopifnot(subtract(x, 1, units = "months") == mondate.ymd(2013, 1:11, 15))
```

Summary-methods *Summary Methods*

Description

Methods for the Summary group of functions.

Methods

signature(x = "mondate") summarizes a mondate.

The result will be a mondate with the same displayFormat and timeunits properties. The usual rules apply as to the shape of the result.

Examples

```
A<-mondate.ymd(2001:2010,12,31) # ten yearends
min(A)                  # December 31, 2001
max(A)                  # December 31, 2010
range(A)
```

---

timeunits-methods *Methods to Access 'timeunits' Property*

Description

Methods to get and set the timeunits value of a mondate.

Usage

```
## S4 method for signature 'mondate'
timeunits(x)
## S4 method for signature 'ANY'
timeunits(x)
## S4 replacement method for signature 'mondate'
timeunits(x)<-value
```

Arguments

x	a mondate.
value	For the "get" method, a character string indicating the units with which to measure time as a result of operations on a mondate. Choices are currently 1. "months" 2. "years" 3. "days" If x is not a mondate, the "get" value returned is NULL.

Note

The mondateTimeunits versions have been deprecated.

Examples

```
x <- mondate("2010-6-30") # The middle of 2010
timeunits(x)              # "months"
y <- x + 12               # One year (12 months) later.
timeunits(y)              # "months"
y - x                     # Result is 12 months
timeunits(y) <- "years"
y - x                     # Result is 1 year, with a warning because
                          # x@timeunits != y@timeunits. Units of result
                          # comes from the first argument.
timeunits(y) <- "days"
suppressWarnings(y - x)   # Result is 365 days -- warning suppressed
```

---

year-month-day-methods

Useful Methods for Class mondate

Description

The methods in this section facilitate year-, month-, day-, and quarter-number extraction from objects that represent dates, as well as mondate construction using the year, month, and day numbers of the date (as those numbers would be seen in the character representation of the date, i.e.., "January" = 1 and the first day of a month = 1).

Usage

```
year(x, ...)
month(x, ...)
day(x, ...)
quarter(x, ...)
ymd(x)
```

```
mondate.mdy(m, d, y, ...)

mondate.ymd(y, m, d, ...)
```

Arguments

x	an object of class mondate, Date, POSIXt, character, or array
m	the month: 1, ..., 12. If "missing" and d is also "missing", m=12 by default.
d	the day: 1, ..., 31, depending on the month. If "missing" in the case of mondate.ymd, the last day of the month is inserted.
y	the four-digit year.
...	For year, month, and day, not used. For mondate.mdy and mondate.ymd arguments to be passed to the mondate constructor, e.g., displayFormat and timeunits.

Value

year returns the year numeric (e.g., 2000).

month returns the month numeric 1, ..., 12.

day returns the numeric day of the month.

quarter returns the numeric calendar quarter of the year: 1 for January - March, 2 for April - June, etc.

ymd returns a matrix with the number of rows equal to the length of x, with appropriately-named columns 1 through 3 holding the year, month, and day, respectively and with "rownames" equal to names(x).

mondate.mdy and mondate.ymd return mondates with the given month, day, and year. Arguments m, d, and y may have length > 1; the usual recycling rules apply.

The mondate.ymd function has a bit more functionality. If d is "missing" in a call to mondate.ymd then the date returned will be the last day of month m in year y. If m is also missing, then the date returned will be the last day of the year. If any of y, m, or d are NA the result will be an all-NA vector. And if y=+-Inf then the result will be an "infinite" mondate vector, overriding other rules included herein. The length of a vector result is determined using the usual recycling rules as though a valid, finite scalar had been inserted instead of NA or +-Inf. See the **Examples** section.

Examples

```
M <- mondate.mdy(6,30,2008)
year(M)     # 2008
month(M)    # 6
day(M)      # 30

mondate.ymd(2008,6,30)  # ditto
mondate.ymd(2008,6)     # ditto; 'day' argument is "missing" so
                        # returns the last day of the month
mondate.ymd(2008,1:12)  # all month-ends of 2008, a leapyear
# year-ends 2001 through 2013, displaying only the 4-digit year when shown
```

```
mondate.ymd(2001:2013, displayFormat = "%Y")
mondate.ymd(2010:2012, NA) # NA mondate vector of length 3
mondate.ymd(Inf,11,31)  # An infinite mondate even though only 30 days in
                        # November

x <- mondate.ymd(2013, 1:12) # month-ends in 2013
# Give x some shape
dim(x) <- 3:4
dimnames(x) <- list(A = letters[1:3], B = LETTERS[1:4])
# Return the quarter numbers in an array with the same shape and dimnames
quarter(x)
```

YearQuartersFormat *Formatting Functions for mondate Objects*

Description

Functions to format a mondate into its character representation according to the displayFormat property.

Usage

```
YearQuartersFormat(x)
```

Arguments

x	a mondate or a Date or a POSIXt.

Details

YearQuartersFormat is an example of a special formatting function that can be provided to a mondate object when created. It will represent the date as YYYYQ* where * is 1-4. See Examples

Examples

```
b <- mondate(1:12, formatFUN = YearQuartersFormat)   # end of first 12 months of 2000
b
```

`[-methods` *Extraction Method for Mondates*

Description

Methods to extract portions of a `mondate`

Usage

```
## S3 method for class 'mondate'
head(x, ...)
## S3 method for class 'mondate'
tail(x, ...)
```

Arguments

`x`	a `mondate`.
`...`	See the `base` functions for details.

Details

Whether the `mondate` `x` is shaped as a `vector` or a `matrix`, the `head` and `rbind` methods will behave just as they would if `x` were `numeric`.

Methods

`[(x = "mondate")` See `[` for more details.

Examples

```
(m<-structure(mondate.ymd(2001:2010,12,31),names=LETTERS[1:10]))
m[1]
m[2:5]
head(m)
tail(m,2)

(M<-cbind(m-12,m,m+12, deparse.level=2))  # a matrix
M[1:5,1:2]    # '[' works with matrix mondates
head(M,2)  # as does 'head'
```

Index

∗ **~format**
  get.mondate.displayFormats, 19
∗ **arith**
  Arith-methods, 3
∗ **chron**
  Coersion-from-mondate methods, 9
∗ **classes**
  difftime-class, 17
  funcNULL-class, 19
  mondate-class, 22
∗ **methods**
  [-methods, 37
  Arith-methods, 3
  array-method, 5
  Coersion-from-mondate methods, 9
  Combining-methods, 11
  Compare-methods, 12
  cut.mondate, 13
  cutmondate-methods, 15
  displayFormat-methods, 17
  matrix-methods, 20
  Miscellaneous-methods, 20
  mondate-methods, 24
  print-methods, 29
  seqmondate-methods, 31
  Summary-methods, 33
  timeunits-methods, 33
  year-month-day-methods, 34
+,mondate,difftime-method *(*Arith-methods*)*, 3
+-methods *(*Arith-methods*)*, 3
-,mondate,ANY-method *(*Arith-methods*)*, 3
-,mondate,Date-method *(*Arith-methods*)*, 3
-,mondate,difftime-method *(*Arith-methods*)*, 3
-,mondate,missing-method *(*Arith-methods*)*, 3
-,mondate,mondate-method *(*Arith-methods*)*, 3
-methods *(*Arith-methods*)*, 3
[,mondate-method *(*[-methods*)*, 37
[-methods, 37

add, 2
Arith, *4*
Arith,array,mondate-method *(*Arith-methods*)*, 3
Arith,mondate,array-method *(*Arith-methods*)*, 3
Arith,mondate,mondate-method *(*Arith-methods*)*, 3
Arith,mondate,numeric-method *(*Arith-methods*)*, 3
Arith,numeric,mondate-method *(*Arith-methods*)*, 3
Arith-methods, 3
array, *5*
array *(*array-method*)*, 5
array,mondate-method *(*array-method*)*, 5
array-method, 5
array-methods *(*array-method*)*, 5
as.character.mondate *(*Coersion-from-mondate methods*)*, 9
as.data.frame, *6*
as.data.frame method, 6
as.data.frame.mondate *(*as.data.frame method*)*, 6
as.Date.mondate *(*Coersion-from-mondate methods*)*, 9
as.difftime, *7*, 7
as.list method, 8
as.list.mondate *(*as.list method*)*, 8
as.mondate, 8
as.numeric,mondate-method *(*Coersion-from-mondate methods*)*, 9
as.POSIXct.mondate *(*Coersion-from-mondate

methods), 9
as.POSIXlt.mondate (Coersion-from-mondate methods), 9

c,mondate-method (Combining-methods), 11
c-methods (Combining-methods), 11
cbindmondate (Combining-methods), 11
Coersion-from-mondate methods, 9
Combining-methods, 11
Compare, *12*
Compare,mondate,mondate-method (Compare-methods), 12
Compare-methods, 12
cut, *14*
cut (cut.mondate), 13
cut.Date, *14*
cut.mondate, 13
cutmondate, *14*
cutmondate (cutmondate-methods), 15
cutmondate,Date,ANY-method (cutmondate-methods), 15
cutmondate,Date,character-method (cutmondate-methods), 15
cutmondate,Date,missing-method (cutmondate-methods), 15
cutmondate,mondate,ANY-method (cutmondate-methods), 15
cutmondate,POSIXt,ANY-method (cutmondate-methods), 15
cutmondate-methods, 15

data.frame, *6*
day (year-month-day-methods), 34
day,array-method (year-month-day-methods), 34
day,character-method (year-month-day-methods), 34
day,Date-method (year-month-day-methods), 34
day,mondate-method (year-month-day-methods), 34
day,POSIXt-method (year-month-day-methods), 34
day-methods (year-month-day-methods), 34
DaysBetween (Arith-methods), 3
DaysBetween,mondate,mondate-method (Arith-methods), 3
DaysBetween-methods (Arith-methods), 3
diff, *17*
diff.mondate, 16
difftime-class, 17
displayFormat (displayFormat-methods), 17
displayFormat,ANY-method (displayFormat-methods), 17
displayFormat,mondate-method (displayFormat-methods), 17
displayFormat-methods, 17
displayFormat<- (displayFormat-methods), 17
displayFormat<-,mondate-method (displayFormat-methods), 17
displayFormat<--methods (displayFormat-methods), 17

format, *6*, *18*
format.mondate, 18
funcNULL-class, 19

get.mondate.displayFormats, 19

head ([-methods), 37

matrix, *20*
matrix (matrix-methods), 20
matrix,mondate-method (matrix-methods), 20
matrix-methods, 20
mean.mondate (Miscellaneous-methods), 20
Miscellaneous-methods, 20
mondate (mondate-methods), 24
mondate,ANY-method (mondate-methods), 24
mondate,array-method (mondate-methods), 24
mondate,character-method (mondate-methods), 24
mondate,Date-method (mondate-methods), 24
mondate,factor-method (mondate-methods), 24
mondate,matrix-method (mondate-methods), 24
mondate,missing-method (mondate-methods), 24
mondate,mondate-method (mondate-methods), 24
mondate,numeric-method (mondate-methods), 24

mondate,POSIXt-method
(mondate-methods), 24
mondate-class, 22
mondate-methods, 24
mondate.mdy (year-month-day-methods), 34
mondate.ymd (year-month-day-methods), 34
month (year-month-day-methods), 34
month,array-method
(year-month-day-methods), 34
month,character-method
(year-month-day-methods), 34
month,Date-method
(year-month-day-methods), 34
month,mondate-method
(year-month-day-methods), 34
month,POSIXt-method
(year-month-day-methods), 34
month-methods (year-month-day-methods), 34
MonthsBetween (Arith-methods), 3
MonthsBetween,ANY,ANY-method
(Arith-methods), 3
MonthsBetween,mondate,mondate-method
(Arith-methods), 3
MonthsBetween-methods (Arith-methods), 3

names method, 28
names<-.mondate (names method), 28

pmean (Miscellaneous-methods), 20
pmean,mondate-method
(Miscellaneous-methods), 20
pmean-methods (Miscellaneous-methods), 20
POSIXt, *26*
print-methods, 29
print.mondate (print-methods), 29

quarter (year-month-day-methods), 34
quarter,array-method
(year-month-day-methods), 34
quarter,character-method
(year-month-day-methods), 34
quarter,Date-method
(year-month-day-methods), 34
quarter,mondate-method
(year-month-day-methods), 34
quarter,POSIXt-method
(year-month-day-methods), 34
quarter-methods
(year-month-day-methods), 34
quarters, *21*
quarters.mondate
(Miscellaneous-methods), 20

rbindmondate (Combining-methods), 11
rep, *11*
rep.mondate (Combining-methods), 11

seq, *30*
seq (seq.mondate), 29
seq.Date, *30*
seq.mondate, 29, *31*
seqmondate, *30*
seqmondate (seqmondate-methods), 31
seqmondate,ANY,ANY-method
(seqmondate-methods), 31
seqmondate,ANY,missing-method
(seqmondate-methods), 31
seqmondate,Date,Date-method
(seqmondate-methods), 31
seqmondate,Date,missing-method
(seqmondate-methods), 31
seqmondate,missing,ANY-method
(seqmondate-methods), 31
seqmondate,missing,Date-method
(seqmondate-methods), 31
seqmondate,missing,mondate-method
(seqmondate-methods), 31
seqmondate,missing,POSIXct-method
(seqmondate-methods), 31
seqmondate,missing,POSIXlt-method
(seqmondate-methods), 31
seqmondate,mondate,missing-method
(seqmondate-methods), 31
seqmondate,mondate,mondate-method
(seqmondate-methods), 31
seqmondate,POSIXct,missing-method
(seqmondate-methods), 31
seqmondate,POSIXct,POSIXct-method
(seqmondate-methods), 31
seqmondate,POSIXlt,missing-method
(seqmondate-methods), 31
seqmondate,POSIXlt,POSIXlt-method
(seqmondate-methods), 31
seqmondate-methods, 31
set.mondate.displayFormats
(get.mondate.displayFormats),

19
show,mondate-method *(print-methods)*, 29
strptime, *18*
subtract, 32
Summary,mondate-method
(Summary-methods), 33
Summary-methods, 33

tail *([-methods)*, 37
timeunits *(timeunits-methods)*, 33
timeunits,ANY-method
(timeunits-methods), 33
timeunits,mondate-method
(timeunits-methods), 33
timeunits-methods, 33
timeunits<- *(timeunits-methods)*, 33
timeunits<-,mondate-method
(timeunits-methods), 33
timeunits<--methods
(timeunits-methods), 33

unique.mondate *(Miscellaneous-methods)*,
20

year *(year-month-day-methods)*, 34
year,array-method
(year-month-day-methods), 34
year,character-method
(year-month-day-methods), 34
year,Date-method
(year-month-day-methods), 34
year,mondate-method
(year-month-day-methods), 34
year,POSIXt-method
(year-month-day-methods), 34
year-methods *(year-month-day-methods)*,
34
year-month-day-methods, 34
yearmon, *24*, *26*
yearqtr, *26*
YearQuartersFormat, 36
YearsBetween *(Arith-methods)*, 3
YearsBetween,ANY,ANY-method
(Arith-methods), 3
YearsBetween,mondate,mondate-method
(Arith-methods), 3
YearsBetween-methods *(Arith-methods)*, 3
ymd *(year-month-day-methods)*, 34