Package ‘chron’
August 18, 2020

Version  2.3-56
Title    Chronological Objects which can Handle Dates and Times
Description Provides chronological objects which can handle dates and times.
Depends R (>= 2.12.0)
Imports graphics, stats
Suggests scales, ggplot2
Enhances zoo
License GPL-2
NeedsCompilation yes

Author David James [aut] (S original),
    Kurt Hornik [aut, trl, cre] (<https://orcid.org/0000-0003-4198-9911>),
    Gabor Grothendieck [ctb],
    R Core Team [ctb]
Maintainer Kurt Hornik <Kurt.Hornik@R-project.org>
Repository CRAN
Date/Publication 2020-08-18 10:54:50 UTC

R topics documented:

  chron ................................................................. 2
  cut.dates .................................................................. 4
  dates ........................................................................ 5
  day.of.week ................................................................ 6
  days ........................................................................... 7
  format.chron ............................................................... 8
  hours ........................................................................ 9
  is.holiday .................................................................. 10
  origin ....................................................................... 11
  scale ........................................................................ 12
  seq.dates ................................................................. 12
  trunc.times .............................................................. 13
  yearmon ................................................................. 14
Create a Chronological Object

Description

Create chronological objects which represent dates and times of day.

Usage

chron(dates., times., format = c(dates = "m/d/y", times = "h:m:s"),
       out.format, origin.)

Arguments

dates. character or numeric vector specifying dates. If character, dates. are assumed to be in one of the date formats below; if numeric, dates. are assumed to be Julian dates, i.e., number of days since origin..
times. optional character or numeric vector specifying times of day. If character, times. are assumed to be in one of the time formats below; if numeric, times. are assumed to be fractions of a day.
format vector or list specifying the input format of the input. The format can be either strings specifying one of the recognized formats below or a list of user-supplied functions to convert dates from character into Julian dates and vice versa. The dates format can be any permutation of the characters "d", "m", or "y" delimited by a separator (possibly null), e.g., "m/d/y", "d-m-y", "ymd", are all valid; the format can also be permutations of the words "day", "month" and "year" (with non-null separator), which produces the month name, e.g., "month day year" produces "April 20 1992", "day mon year" produces "20 Apr 1992". The times format can be any permutation of "h", "m", and "s" separated by any one non-special character. The default is "h:m:s".
out.format vector or list specifying date and time format for printing and output. Default is same as format.
origin. a vector specifying the date with respect to which Julian dates are computed. Default is c(month = 1, day = 1, year = 1970); you may set the option chron.origin to specify your own default, e.g., options(chron.origin = c(month=1,day=1,year=1990)).

Value

An object of class "times" if only times. were specified, "dates" if only dates., or "chron" if both dates. and times. were supplied. All these inherit from class "times".

These objects represent dates and times of day, and allow the following arithmetic and summaries: subtraction d1-d2, constant addition d1+constants, all logical comparisons, summaries min(), max(), and range() (which drop NAs by default); constants specify days (fractions are converted...
to time-of-day, e.g., 2.5 represents 2 days and 12 hours. Operations such as sorting, differencing, etc., are automatically handled.

There are methods for `as.character()`, `as.numeric()`, `cut()`, `is.na()`, `print()`, `summary()`, `plot()`, `lines()`, `lag()`, and the usual subsetting functions `[,]`<. The functions `days()`, `months()`, `quarters()`, `years()`, `weeks()`, `weekdays()`, `hours()`, `minutes()`, and `seconds()` take any `chron` object as input and extract the corresponding time interval. `cut()` is used to create ordered factors from `chron` objects. Chronological objects may be used with the modeling software.

If `x` is character then it will be converted using `as.POSIXct` (with the format argument, if any, passed to as.POSIXct) and `tz = "GMT"` and then converted to `chron`. If `x` is numeric and format is not specified then it will be converted to `chron` using `chron(x)`. If `x` is numeric and format is specified then `x` will be converted to character and then processed using `as.POSIXct` as discussed above. If the format is specified as `NULL` it will be treated the same as if it were missing.

The current implementation of `chron` objects does not handle time zones nor daylight savings time.

### See Also

`dates`, `times`, `julian.default`, `cut.dates`, `seq.dates`.

### Examples

```r
dts <- dates(c("02/27/92", "02/27/92", "01/14/92", "02/28/92", "02/01/92"))
dts
# [1] 02/27/92 02/27/92 01/14/92 02/28/92 02/01/92
tms
x <- chron(dates = dts, times = tms)
x
# [1] (02/27/92 23:03:19) (02/27/92 22:29:56) (01/14/92 01:03:30)
# [4] (02/28/92 18:21:03) (02/01/92 16:56:26)

# We can add or subtract scalars (representing days) to dates or
# chron objects:
c(dts[1], dts[1] + 10)
# [1] 02/27/92 03/08/92
dts[1] - 31
# [1] 01/27/92

# We can substract dates which results in a times object that
# represents days between the operands:
# Time in days:
# [1] 44

# Logical comparisons work as expected:
dts[1] > "01/25/92"
# [1] 02/27/92 02/27/92 02/28/92 02/01/92
dts > dts[3]
# [1] TRUE TRUE FALSE TRUE TRUE
```
# Summary operations which are sensible are permitted and work as expected:

```r
range(dts)
# [1] 01/14/92 02/28/92
diff(x)
# Time in days:
# [1] -0.02319444 -44.89335648 45.72052083 -27.05876157
sort(dts)[1:3]
# [1] 01/14/92 02/01/92 02/27/92
```

---

**cut.dates**

*Create a Factor from a Chron or Dates Object*

**Description**

Divide the range of a chron or dates object `x` into intervals and code the values in `x` according to which interval they fall.

**Usage**

```r
## S3 method for class 'dates'
cut(x, breaks, labels, start.on.monday = TRUE, ...)
```

**Arguments**

- `x`: chron or dates object (see `chron`), character dates such as "10/04/91" or Julian dates.
- `breaks`: either a vector of break points (a `dates` vector, character dates such as "10/04/91" or Julian dates), a constant specifying number of equally spaced intervals extending from `min(x) - 1` to `max(x) + 1`, or one of the strings in `c("days", "weeks", "months", "year")` specifying a time period.
- `labels`: character labels for intervals.
- `start.on.monday`: should weeks be assumed to start on Mondays? Default is `TRUE`. Set to `FALSE` if weeks start on Sundays; for other days of the week specify the corresponding number: Sunday == 0, Monday == 1, Tuesday == 2, ..., Saturday == 6.
- `...`: further arguments to be passed to or from methods.

**Value**

an ordered factor whose levels represent the various time intervals.

**See Also**

`seq.dates`
**Examples**

```r
# days from 07/01/92 thru 07/15/92 fell into 3 Monday-started weeks
cut(dates("07/01/92") + 0:14, "weeks")
# [1] week 1 week 1 week 1 week 1 week 2 week 2 week 2
# [9] week 2 week 2 week 2 week 2 week 3 week 3 week 3

dts <- dates(c("02/27/92", "02/27/92", "01/14/92",
"02/28/92", "02/01/92"))
cut(dts, "months")
# [1] Feb 92 Feb 92 Jan 92 Feb 92 Feb 92
boxplot(runif(5) ~ cut(dts, "months"))
```

---

**dates**

*Generate Dates and Times Components from Input*

**Description**

Create objects which represent dates or times.

**Usage**

```r
dates(x, ...)
times(x, ...)
```

**Arguments**

- `x`  
  a chron object, a character vector, or a numeric vector specifying time. If character, it must be in a format recognized by `chron()`. If numeric, it specifies Julian dates, i.e., number of days since an origin.

- `...`  
  parameters for `chron()`.

**Value**

An object of class `dates` or `times`, depending of the function called.

These functions return objects inheriting from `dates` and `times`, respectively. They call `chron()` if `x` does not belong to any of the chronological classes.

**See Also**

- `chron`
- `times`
- `seq.dates`
- `cut.dates`
Examples

dts <- dates(c("02/27/92", "02/27/92", "01/14/92", "02/28/92", "02/01/92"))
dts
# [1] 02/27/92 02/27/92 01/14/92 02/28/92 02/01/92
class(dts)

x <- chron(dates = c("02/27/92", "02/27/92", "01/14/92", "02/28/92"),
dates(x)
# [1] 02/27/92 02/27/92 01/14/92 02/28/92

day.of.week

Convert between Julian and Calendar Dates

Description

Utility functions to convert between Julian dates (numbers of days since an origin, by default 1970-01-01) and calendar dates given by year, month, and day within the month.

Usage

## Default S3 method:
julian(x, d, y, origin., ...)
month.day.year(jul, origin.)
leap.year(y)
day.of.week(month, day, year)

Arguments

x, month  vector of month numbers.
d, day     vector of day numbers.
y, year    vector of years.
jul        vector of Julian Dates, i.e., number of days since origin.
origin.    vector specifying the origin as month, day, and year. If missing, it defaults to getOption("chron.origin") if this is non-null, otherwise c(month = 1, day = 1, year = 1970).
...        further arguments to be passed to or from methods.

Value

A vector of Julian dates (number of days since origin.) when julian() is called, or a list with members month, day, year corresponding to the input Julian dates if month.day.year() is called. leap.year() returns a logical vector indicating whether the corresponding year is a leap year. day.of.week() returns a number between 0 and 6 to specify day of the week–0 refers to Sunday.

These functions were taken from Becker, Chambers, and Wilks (1988), and were slightly modified to take chron and dates objects; some also take the extra argument origin..
days

See Also
chron, dates, times

Examples

julian(1, 1, 1970)
# [1] 0
unlist(month.day.year(0))
# month  day  year
# 1 1 1970

Return Various Periods from a Chron or Dates Object

Description
Given a chron or dates object, extract the year, quarter, month, day (within the month) or weekday (days within the week) of the date it represents.

Usage
days(x)

## Default S3 method:
weekdays(x, abbreviate = TRUE)
## Default S3 method:
months(x, abbreviate = TRUE)
## Default S3 method:
quarters(x, abbreviate = TRUE)
years(x)

Arguments

x an object inheriting from class "dates", or coercible to such via as.chron.
abbreviate should abbreviated names be returned? Default is TRUE.

Details
Note that months, quarters and weekdays are generics defined in package base which also provides methods for objects of class "Date" as generated, e.g., by Sys.Date. These methods return character rather than factor variables as the default methods in chron do. To take advantage of the latter, Date objects can be converted to dates objects using as.chron, see the examples.

Value

an ordered factor corresponding to days, weekdays, months, quarters, or years of x for the respective function.
See Also

`is.weekend, is.holiday`

Examples

```r
dts <- dates("07/01/78") + trunc(50 * rnorm(30))
plot(weekdays(dts))
plot(months(dts))

## The day in the current timezone as a Date object.
Dt <- Sys.Date()
## Using the months method for Date objects.
months(Dt)
## Using the months default method.
months(as.chron(Dt))
```

---

**format.chron**  
*Format a chron object*

**Description**

Format a chron object.

**Usage**

```r
## S3 method for class 'chron'
format(x, format = att$format, origin. = att$origin,
       sep = " ", simplify, enclosed = c("(", ")"), ...)
```

**Arguments**

- `x` A chron object.
- `format` As in `chron` or a single character string with percent codes as detailed in `strptime`.
- `origin.` As in `chron`.
- `sep` A character string that separates the dates and times in the output. Ignored if percent codes are used in `format`.
- `simplify` As in `format.dates`. Ignored if percent codes are used in `format`.
- `enclosed` A character vector of length 2 containing the strings that begin and end each output date/time. Ignored if percent codes are used in `format`.
- `...` Not currently used.

**Value**

A character vector.
hours

See Also
chron, format, dates, strptime.

Examples
format(chron(0, 0), c("yy/m/d", "h:m:s"), sep = "T", enclose = c(""", ""))
format(chron(0, 0), "%Y-%m-%dT%H:%M:%S") # same

---

hours Return Hours, Minutes, or Seconds from a Times Object

Description
Given a chron or times object, extract the hours, minutes or seconds of the time it represents.

Usage
hours(x)
minutes(x)
seconds(x)

Arguments
x an object inheriting from class "times", or coercible to such via as.chron.

Value
the corresponding time period as an ordered factor.

See Also
chron, dates, times

Examples
x <- chron(dates = c("02/27/92", "02/27/92", "01/14/92", "02/28/92"),
h <- hours(x)
y <- runif(4)
boxplot(y ~ h)
is.holiday

Find Weekends and Holidays in a Chron or Dates Object

Description

Determine the date represented by a chron or dates object is a weekend or a holiday.

Usage

is.weekend(x)

is.holiday(x, holidays)

Arguments

x

an object inheriting from "dates", or coercible to "chron".

holidays

optional "chron" or "dates" object listing holidays. If argument holidays is missing, either the value of the object .Holidays (if it exists) or NULL will be used.

Value

a logical object indicating whether the corresponding date is a weekend in the case of is.weekend() or a holiday in the case of is.holiday().

See Also

days, weekdays.default, months.default, quarters.default, years; chron, dates, cut.dates, seq.dates

Examples

dts <- dates("01/01/98") + trunc(365 * runif(50))
table(is.weekend(dts))

.Holidays
# New Year Memorial Indepen. Labor day Thanksgiving Christmas
# 01/01/92 05/25/92 07/04/92 09/07/92 11/26/92 12/25/92
# NOTE: Only these 6 holidays from 1992 are defined by default!
description

Get and set chron origin.

Usage

origin(x)
origin(x) <- value

Arguments

x a chron object
value month, day, year vector

Details

origin extracts the origin of its argument. origin<- sets the origin of its argument.

Value

The origin function returns a month, day, year vector.

Note

The default origin for chron objects is January 1, 1970 and it is recommended that that origin be used. Some chron functions may not perform properly with a non-default origin.

See Also

chron

Examples

ch <- as.chron(Sys.Date())
origin(ch)
Description

chron_trans is a ggplot2 transformer for chron. scale_x_chron and scale_y_chron are ggplot2 scales.

Usage

chron_trans(format = "%Y-%m-%d", n = 5)
scale_x_chron(..., format = "%Y-%m-%d", n = 5)
scale_y_chron(..., format = "%Y-%m-%d", n = 5)

Arguments

format format string as described in strptime.
n Approximate number of axis ticks.
... Passed to scale_x_continuous.

Examples

if(require("ggplot2")) {
  dd <- data.frame(tt = chron(1:10), value = 101:110)
  p <- ggplot(dd, aes(tt, value)) +
       geom_point() + scale_x_chron(format = "%m-%d")
  print(p)
}

Description

Generate a regular sequence of dates.

Usage

## S3 method for class 'dates'
seq(from, to, by = "days", length., ...)
trunc.times

Arguments

from
 starting date; it can be a chron or dates object, a character string, e.g., "05/23/91", or a Julian date.
to
 ending date, like from.
by
 either a numeric value or one of the valid strings "days", "weeks", "months", or "years".
length.
 optional number of elements in the sequence.
...
 further arguments to be passed to or from methods.

Value

a sequence with values (from, from + by, from + 2*by, ..., to) of class class(from) and origin(origin(from)). Note that from must be less than or equal to the argument to.

See Also

chron, dates, cut.dates

Examples

seq.dates("01/01/92", "12/31/92", by = "months")
# [1] 01/01/92 02/01/92 03/01/92 04/01/92 05/01/92 06/01/92
# [7] 07/01/92 08/01/92 09/01/92 10/01/92 11/01/92 12/01/92

day.of.the.month <- seq.dates("02/29/92", by = "month", length = 15)
day.of.the.month
# [1] 02/29/92 03/31/92 04/30/92 05/31/92 06/30/92 07/31/92
# [7] 08/31/92 09/30/92 10/31/92 11/30/92 12/31/92 01/31/93
# [13] 02/28/93 03/31/93 04/30/93

trunc.times

Truncate times Objects

Description

Truncate times objects.

Usage

## S3 method for class 'times'
trunc(x, units = "days", eps = 1e-10, ...)

13
Arguments

- **x**: a "times" object.
- **units**: Can be one of "days", "hours", "minutes", "seconds" or an unambiguous abbreviated version of any of those.
- **eps**: Comparison tolerance. Times are considered equal if their absolute difference is less than eps.
- **...**: further arguments to be passed to or from methods.

Details

The time is truncated to the second, minute, hour or day or to the value specified.

There is also a round method which has the same arguments as the `trunc` method, and performs rounding instead of truncation.

Value

An object of class "times".

See Also

- `trunc` for the generic function and default methods.

Examples

```r
tt <- times(c("12:13:14", "15:46:17"))
trunc(tt, "minutes")
trunc(tt, "min")
trunc(tt, times("00:01:00"))
trunc(tt, "00:01:00")
trunc(tt, 1/(24*60))

tt2 <- structure(c(3.0, 3.1, 3.5, 3.9), class = "times")
trunc(tt2, "day")
trunc(tt2)
```

Description

These functions can be used to convert the times of "ts" series with frequency of 12 or 4 or objects of "yearmon" and "yearqtr" class, as defined in the "zoo" package, to chron dates.
yearmon

Usage

```r
## S3 method for class 'yearmon'
as.chron(x, frac = 0, holidays = FALSE, ...)
## S3 method for class 'yearqtr'
as.chron(x, frac = 0, holidays = FALSE, ...)
## S3 method for class 'ts'
as.chron(x, frac = 0, holidays = FALSE, ...)
```

Arguments

- `x`: an object of class "yearmon" or "yearqtr" or "ts" objects, or a numeric vector interpreted "in years" and fractions of years.
- `frac`: Number between zero and one inclusive representing the fraction of the way through the month or quarter.
- `holidays`: If TRUE or a vector of chron dates, indicated holidays and weekends are excluded so the return value will be a non-holiday weekday.
- `...`: Other arguments passed to `chron`.

Details

The "yearmon" and "yearqtr" classes are defined in package `zoo`. If `holidays` is TRUE or a vector of dates then the `is.holiday` function is used to determine whether days are holidays.

The method for ts objects converts the times corresponding to `time(x)` to chron. The ts series must have a frequency that is a divisor of 12.

Value

Returns a chron object.

See Also

`is.holiday`, `ts`

Examples

```r
## Monthly time series data.
as.chron(AirPassengers)
as.chron(time(AirPassengers))
## convert to first day of the month that is not a weekend or holiday
as.chron(AirPassengers, frac = 0, holidays = TRUE)
## convert to last day of the month
as.chron(AirPassengers, frac = 1)
## convert to last day of the month that is not a weekend or holiday
as.chron(AirPassengers, frac = 1, holidays = TRUE)
## convert to last weekday of the month
as.chron(AirPassengers, frac = 1, holidays = c())

## Quarterly time series data.
as.chron(presidents)
as.chron(time(presidents))
```
Index

* chron
  chron, 2
  cut.dates, 4
dates, 5
day.of.week, 6
days, 7
hours, 9
is.holiday, 10
origin, 11
seq.dates, 12
trunc.times, 13

* misc
  format.chron, 8

* ts
  yearmon, 14
.Holidays (is.holiday), 10
.[.times (dates), 5
[<-.dates (dates), 5
[<-.times (dates), 5
[[.times (dates), 5

cut.dates, 3, 4, 5, 10, 13
Date, 7
dates, 3, 5, 7, 9, 10, 13
day.of.week, 6
days, 7, 10
diff.times (dates), 5
factor, 7
floor.dates (dates), 5
format.chron, 8
format.dates, 8, 9
format.times (dates), 5
format<-.times (dates), 5
hist.times (dates), 5
hours, 9
identify.times (dates), 5
is.chron (chron), 2
is.holiday, 8, 10, 15
is.na.times (dates), 5
is.weekend, 8
is.weekend (is.holiday), 10
julian.default, 3
leap.year (day.of.week), 6
MATH.dates (dates), 5
Math.dates (dates), 5
mean.times (dates), 5
minutes (hours), 9
month.day.year (day.of.week), 6
months.default, 10
months.default (days), 7
Ops.dates (dates), 5

INDEX

Ops.times(dates), 5
origin, 11
origin<-(origin), 11

plot.times(dates), 5
points.times(dates), 5
print.chron(chron), 2
print.dates(dates), 5
print.times(dates), 5

quantile.times(dates), 5
quarters.default, 10
quarters.default(dates), 7

round.times(trunc.times), 13

scale, 12
scale_x_chron(scale), 12
scale_x_continuous, 12
scale_y_chron(scale), 12
seconds(hours), 9

seq.dates, 3–5, 10, 12
strptime, 8, 9, 12
Summary.dates(dates), 5
Summary.times(dates), 5
summary.times(dates), 5
Sys.Date, 7

times, 3, 5, 7, 9
times(dates), 5
trunc, 14
trunc.dates(dates), 5
trunc.times, 13
ts, 15

weekdays.default, 10
weekdays.default(dates), 7

yearmon, 14, 15
yearqtr, 15
years, 10
years(dates), 7