Package 'datetime'

July 22, 2025

Type Package

Title Nominal Dates, Times, and Durations

Version 0.1.4

Author Tim Bergsma

Maintainer Tim Bergsma <bergsmat@gmail.com>

Description Provides methods for working with nominal dates, times, and durations. Base R has sophisticated facilities for handling time, but these can give unexpected results if, for example, timezone is not handled properly. This package provides a more casual approach to support cases which do not require rigorous treatment. It systematically deconstructs the concepts origin and timezone, and de-emphasizes the display of seconds. It also converts among nominal durations such as seconds, hours, days, and weeks. See '?datetime' and '?duration' for examples. Adapted from 'metrumrg' <http://r-forge.r-project.org/R/?group_id=1215>.

License GPL-3

Suggests chron, SASxport

Encoding UTF-8

LazyData true

NeedsCompilation no

Repository CRAN

Date/Publication 2018-12-13 18:40:03 UTC

Contents

`as.second` *Interconvert Nominal Time Units*

Description

Perform standard conversions among various common time units.

Usage

```
as.second(x, ...)
as.minute(x, ...)
as.hour(x, ...)
as.day(x, ...)
as.week(x, ...)
as.month(x, ...)
as.year(x, ...)
```

Arguments

`x`	numeric
`...`	ignored

Details

The functions listed above are generic. Methods exist, either explictly or implicitly, for objects classified to represent second, minute, hour, day, week, month, and year. All these objects are subclasses of "duration"; `as.<n>.duration` serves wherever explicit methods are omitted. For each generic, methods exist for class "numeric". For each class, methods exist for the generics "format" and "print".

The strategy for time unit conversion is to classify a numeric vector according to some unit, and then coerce that object to some other class. Thus, `as.day(7)` is not particularly interesting, but `as.week(as.day(7))` yields 1.

Conversions use 60 seconds per minute, 60 minutes per hour, 24 hours per day, 7 days per week, 28 days per month, and 365.25 days per year. Currently, no other relationships are specified. Note that 12 nominal months does not make a full year. This is experimental, and may change in future versions.

The duration classes are also subclasses of `timeline`, which exists to support addition and subtraction of durations and timepoints. See examples here, and at `timeline`. You cannot add two timepoints, nor can you subtract a timepoint from a non-timepoint. When one argument is a timepoint, the other is coerced using `as.second`, and the result is the timepoint class. For two durations, the second value is coerced to the class of the first, with a message, if necessary. Otherwise, if only one argument is a duration, the other is coerced to that class. Negative durations are allowed.

Value

an S3 "num" object with class `c(n, 'duration','timeline','numeric')`, where 'n' is "second", "minute", "hour", "day", "week", "month", or "year", as implied.

Author(s)

Tim Bergsma

See Also

- as.datetime
- timeline
- c.timeline

Examples

```
as.year(as.month(12))
as.year(as.day(365.25))
as.second(as.year(1))
as.month(2) + as.week(2)
as.week(2) + as.month(1)
as.month(2) - as.week(2)
as.week(2) - as.month(1)
as.week(2) + 1
as.week(2) - 1
2 + as.week(1)
2 - as.week(1)
class(c(as.day(1),as.day(2)))
class(as.day(1:5)[3])
class(as.day(1:5)[[3]])
class(seq(from=as.day(2),to=as.day(6)))
class(rep(as.day(1),5))
```

---

c.timeline *Support for timepoint Classes*

Description

These functions support classes timepoint, timeline, time, date, and datetime (and related functions). They are mostly S3 methods for base R generics.

Usage

```
## S3 method for class 'timeline'
c(..., recursive = FALSE)
## S3 method for class 'timeline'
x[..., drop = TRUE]
## S3 replacement method for class 'timepoint'
x[...] <- value
## S3 method for class 'timeline'
x[[..., drop = TRUE]]
## S3 method for class 'timepoint'
```

```
as.character(x, ...)
as.chartime(x, ...)
## S3 method for class 'numeric'
as.chartime(x, format, mark=TRUE,...)
## S3 method for class 'chartime'
as.numeric(x, format,...)
## S3 method for class 'timepoint'
print(x, ...)
## S3 method for class 'timeline'
rep(x, ...)
## S3 method for class 'timeline'
seq(from, to, by, length.out, along.with, ...)
```

Arguments

...	arguments to c, or passed to other functions
recursive	same meaning as for c
x	object of class timepoint
drop	same meaning as for '[' and '[['
value	value to be assigned, as for '[<-'
format	input or output format describing character time (see strftime)
mark	boolean: mark times with dangling seconds using '+'
from	as for seq.default
to	as for seq.default
by	as for seq.default
length.out	as for seq.default
along.with	as for seq.default

Details

Normally you shouldn't need to worry about these functions. c and the '[' variants exist just so that class information is not lost on invocation of the generic. as.character.timepoint and print.timepoint just call format. chartime variants are used internally by other functions. seq.timeline requires from. If an interval cannot be calculated from supplied arguments, by is set to 1 hour for time or 1 day for date or datetime.

Value

print	an invisible object with same class as x
as.chartime	generic: does not return
as.chartime.numeric	character (time)
as.numeric.chartime	numeric (seconds)
as.character.timepoint	character (time)
others	object with same class as x

Author(s)

Tim Bergsma

See Also

- timepoint
- seq.default
- strftime

Examples

```
#as.data.frame
data.frame(
dt=as.datetime(seq(from=0,by=86500,length.out=3)),
d=as.date(seq(from=0,by=86400,length.out=3)),
t=as.time(c(60,120,180))
)
#                  dt          d     t
# 1  1970-01-01 00:00 1970-01-01 00:01
# 2 1970-01-02 00:01+ 1970-01-02 00:02
# 3 1970-01-03 00:03+ 1970-01-03 00:03

#combine
c(as.time(0),as.time(60))
# 00:00 00:01
c(as.date(0),as.date(86400))
# 1970-01-01 1970-01-02
c(as.datetime(0),as.datetime(86500))
# 1970-01-01T00:00  1970-01-02T00:01+

#subset
as.time(c('08:00','09:00'))[2]
# 09:00
as.date(c('2008-01-01','2008-01-04'))[2]
# 2008-01-04
as.datetime(c('2008-01-01T12:00','2008-01-04T12:30'))[2]
# 2008-01-04 12:30

#element selection
as.time(c('08:00','09:00'))[[2]]
# 09:00
as.date(c('2008-01-01','2008-01-04'))[[2]]
# 2008-01-04
as.datetime(c('2008-01-01T12:00','2008-01-04T12:30'))[[2]]
# 2008-01-04 12:30

#assignment
a <- as.time(seq(60,300, by=60))
a#00:01 00:02 00:03 00:04 00:05
a[5] <- 60
a#00:01 00:02 00:03 00:04 00:01
```

```
a[3] <- NA
a#00:01 00:02 <NA>  00:04 00:01

#identity
as.time(as.time(0))
# 00:00
as.date(as.date(0))
# 1970-01-01
as.datetime(as.datetime(0))
# 1970-01-01T00:00

#repetition
rep(as.time(86340),2)
# 23:59 23:59

#sequence
seq(from=as.time('00:00'),length.out=3)
seq(from=as.time('00:00'),by=as.time('00:05'),length.out=3)
seq(from=as.time('00:00'),by=as.time('00:05'),along.with=integer(3))
seq(from=as.time('00:00'),to=as.time('06:00'))
seq(from=as.time('00:00'),to=as.time('06:00'),by=as.time('02:00'))
seq(from=as.time('00:00'),to=as.time('06:00'),length.out=4)
```

`timepoint` *Temporal Classes with Selective Defaults*

Description

`timepoint` is an abstract superclass of `time`, `date`, and `datetime`. These latter are convenience classes that store timepoint information as seconds since the start of 1970-01-01. They rely on `POSIXlt` and `POSIXct`, giving full access to the format constructs of `strftime`. However, the concepts of 'origin' and 'timezone' are deconstructed (fixed to 1970-01-01 and GMT). Default formats are suitably chosen for inputs (`as.character` methods) and outputs (`format` methods) and may be overridden. By default, `format` will append a '+' symbol to timepoints with dangling seconds (not multiples of 60): seconds are not displayed by default but still operate (perhaps dangerously) in comparisons.

Usage

```
as.time(x, ...)
## S3 method for class 'character'
as.time(x, format = '%H:%M',...)
## S3 method for class 'numeric'
as.time(x,...)
## S3 method for class 'time'
as.time(x, ...)
## S3 method for class 'times'
as.time(x, ...)
```

```
as.date(x, ...)
## S3 method for class 'character'
as.date(x, format = '%Y-%m-%d',...)
## S3 method for class 'numeric'
as.date(x,...)
## S3 method for class 'Date'
as.date(x,...)
## S3 method for class 'date'
as.date(x,...)
## S3 method for class 'dates'
as.date(x,...)
as.datetime(x, ...)
## S3 method for class 'character'
as.datetime(x, format = '%Y-%m-%dT%H:%M',...)
## S3 method for class 'numeric'
as.datetime(x,...)
## S3 method for class 'date'
as.datetime(x, y = 0,...)
## S3 method for class 'datetime'
as.datetime(x, ...)
## S3 method for class 'POSIXct'
as.datetime(x, ...)
## S3 method for class 'POSIXlt'
as.datetime(x, ...)
## S3 method for class 'chron'
as.datetime(x, ...)
## S3 method for class 'time'
format(x, format = '%H:%M', mark=TRUE,...)
## S3 method for class 'date'
format(x, format = '%Y-%m-%d', mark=TRUE,...)
## S3 method for class 'datetime'
format(x, format = '%Y-%m-%dT%H:%M', mark=TRUE,...)
## S3 method for class 'timepoint'
unique(x, incomparables=FALSE,...)
## S3 method for class 'timepoint'
Summary(..., na.rm=FALSE)
## S3 method for class 'timepoint'
xtfrm(x,...)
```

Arguments

x	character time as per format, numeric seconds since 1970-01-01, or timepoint subclass
...	other arguments, usually ignored
y	optional time for constructing datetime from date: anything coercible with `as.second()`
format	character, as per `strftime`
mark	boolean: mark times with dangling seconds using '+'

incomparables	passed to unique
na.rm	passed to Summary

Details

Creating a timepoint object with these methods ultimately calls one of the .numeric methods, each of which round their first argument to zero places. This means that all comparisons are based on whole numbers, and therefore not subject to rounding errors.

Seconds that are not multiples of 60 can be stored in time and datetime objects, but will not be displayed by default (see above). date can only store numbers of seconds that correspond to midnight. To add time, explicitly create an datetime object using as.datetime.date.

The timepoint classes are all subclasses of numeric, so numeric operations are generally available.

The timepoint classes support NA, Inf, -Inf, as.data.frame, seq, subset, element selection, element assignment, and interconversion.

The timepoint classes are also subclasses timeline, which exists to support addition and subtraction of durations and timepoints. See examples.

- You cannot add two timepoints.
- You cannot subtract a timepoint from a non-timepoint.
- For the difference of two timepoints, the arguments and result are coerced with as.second.
- When one argument is a timepoint, the other is coerced using as.second, and the result is the timepoint class.
- For two durations, the second value is coerced to the class of the first, with a message, if necessary.
- If only one argument is a duration, the other is coerced to that class.

Value

format	character
as.time	object with class c('time','timepoint','numeric')
as.date	object with class c('date','timepoint','numeric')
as.datetime	object with class c('datetime','timepoint','numeric')

Author(s)

Tim Bergsma

See Also

- duration
- c.timeline

Examples

```
#numeric to timepoint
as.time(0)
# 00:00
as.time(1)
# 00:00+
as.time(-1)
# 23:59+
as.time(60)
# 00:01
as.time(-60)
# 23:59
as.time(86400)
# 00:00
as.time(86460)
# 00:01
as.date(0)
# 1970-01-01
as.date(1)
# 1970-01-01
as.date(-1)
# 1969-12-31
as.date(-86400)
# 1969-12-31
as.date(-86401)
# 1969-12-30
as.datetime(0)
# 1970-01-01T00:00
as.datetime(60)
# 1970-01-01T00:01
as.datetime(61)
# 1970-01-01T00:01+
as.datetime(-1)
# 1969-12-31T23:59+

#character to timepoint
as.time('00:00')
# 00:00
as.time('23:59')
# 23:59
as.time('23:59:00')
# 23:59
as.time('23:59:01')
# 23:59
as.time('23:59:01',format='%H:%M:%S')
# 23:59+
as.time('24:00')
# 00:00
as.date('1970-01-02')
# 1970-01-02
as.date('01/02/1970',format='%m/%d/%Y')
# 1970-01-02
```

```
as.datetime('01/02/1970 12:30',format='%m/%d/%Y %H:%M')
# 1970-01-02 12:30
as.datetime('01/02/1970 12:30:15',format='%m/%d/%Y %H:%M:%S')
# 1970-01-02 12:30+

#timepoint to numeric
as.numeric(as.time(0))
# 0
as.numeric(as.time(1))
# 1
as.numeric(as.time(-1))
# 86399
as.numeric(as.time(60))
# 60
as.numeric(as.time(-60))
# 86340
as.numeric(as.time(86400))
# 0
as.numeric(as.time(86460))
# 60
as.numeric(as.date(0))
# 0
as.numeric(as.date(1))
# 0
as.numeric(as.date(-1))
# -86400
as.numeric(as.date(-86400))
# -86400
as.numeric(as.date(-86401))
# -172800
as.numeric(as.datetime(0))
# 0
as.numeric(as.datetime(60))
# 60
as.numeric(as.datetime(61))
# 61
as.numeric(as.datetime(-1))
# -1
as.numeric(as.time('00:00'))
# 0
as.numeric(as.time('23:59'))
# 86340
as.numeric(as.time('23:59:00'))
# 86340
as.numeric(as.time('23:59:01'))
# 86340
as.numeric(as.time('23:59:01',format='%H:%M:%S'))
# 86341
as.numeric(as.time('24:00'))
# 0
as.numeric(as.date('1970-01-02'))
# 86400
as.numeric(as.date('01/02/1970',format='%m/%d/%Y'))
```

```
# 86400
as.numeric(as.datetime('01/02/1970 12:30',format='%m/%d/%Y %H:%M'))
# 131400
as.numeric(as.datetime('01/02/1970 12:30:15',format='%m/%d/%Y %H:%M:%S'))
# 131415

#timepoint to character
as.character(as.time(0))
# '00:00'
as.character(as.date(0))
# '1970-01-01'
as.character(as.datetime(0))
# '1970-01-01T00:00'

#non-default printout
format(as.time(30000),format='%H')
# '08'
format(as.date('1970-01-01'),format='%d%b%y')
# '01Jan70'
format(as.datetime('1970-01-02T23:30'),format='[%d/%m/%y %H:%M:%S]')
# '[02/01/70 23:30:00]'
format(as.time(1))
# '00:00+'
format(as.time(1),mark=FALSE)
# '00:00'

#sequence
seq(from=as.time('8:00'),to=as.time('12:00'))
# 08:00 09:00 10:00 11:00 12:00
seq(from=as.date('2008-01-01'),to=as.date('2008-01-04'))
# 2008-01-01 2008-01-02 2008-01-03 2008-01-04
seq(from=as.datetime('2008-01-01T12:00'),to=as.datetime('2008-01-04T12:30'))
# 2008-01-01 12:00 2008-01-02 12:00 2008-01-03 12:00 2008-01-04 12:00

#interconversion
as.time(as.date('2008-10-14'))
# 00:00
as.time(as.datetime('2008-10-14T08:00'))
# 08:00
as.date(as.time('23:59'))
# 1970-01-01
as.date(as.datetime('2008-10-14T08:00'))
# 2008-10-14
as.datetime(as.time(6000000))
# 1970-01-01T10:40
as.datetime(as.date('2008-10-14'))
# 2008-10-14 00:00
as.datetime(as.date('2008-10-14'),y=as.time('00:30'))
# 2008-10-14 00:30

#intercoversion from other systems
as.date(as.Date('1970-01-01'))
# 1970-01-01
```

```
as.datetime(as.POSIXct('1970-01-01',tz='GMT'))
# 1970-01-01T00:00
as.datetime(as.POSIXlt('1970-01-01',tz='GMT'))
# 1970-01-01T00:00
if(require(chron)) as.time(times('12:30:00'))
# 12:30
as.date(dates('01/01/1970'))
# 1970-01-01
if(require(chron))as.datetime(chron(dates='01/01/1970',times='12:30:00'))
# 1970-01-01T12:30
as.date.sasdate(0)
# 1960-01-01
as.time(as.numeric(NA))
# <NA>

#infinity
as.time(Inf)
# Inf
as.date(Inf)
# Inf
as.datetime(Inf)
# Inf
as.time(-Inf)
# -Inf
as.datetime(Inf) + (Inf)
# Inf
as.datetime(Inf) + (-Inf)
# <NA>

#comparison
as.time('08:00') < as.time('09:00')
# TRUE
as.date('1970-01-01') > as.date('2008-01-01')
# FALSE
as.datetime('1970-01-01 08:00') > as.date('1970-01-01')
# TRUE

#summary
max(as.date(c('1970-01-01','1980-01-01','1975-01-01')))
# 1980-01-01

#operations
as.datetime(0) + as.second(60)
# 1970-01-01T00:01
as.second(60) + as.datetime(0)
# 1970-01-01T00:01
try(as.datetime(60) + as.datetime(60))
# Error in `+.timeline`(as.datetime(60), as.datetime(60)) :
#   addition is undefined for two timepoints
as.datetime(0) + 60
# 1970-01-01 00:01
60 + as.datetime(0)
# 1970-01-01T00:01
```

```
as.minute(1) + as.datetime(0)
# 1970-01-01T00:01
as.datetime(0) - as.second(60)
# 1969-12-31T23:59
as.datetime(0) - 60
# 1969-12-31T23:59
as.datetime(60) - as.datetime(0)
# 60
try(as.second(60) - as.datetime(60))
# Error in `-.timeline`(as.second(60), as.datetime(60)) :
#   subtracting a timepoint from non-timepoint is undefined
try(60 - as.datetime(60))
# Error in `-.timeline`(as.second(60), as.datetime(60)) :
#   subtracting a timepoint from non-timepoint is undefined
as.second(10) * 6
# 60
as.datetime(0) * 2 # meaningless, but not prevented
# 1970-01-01T00:00

#unary operations
-as.time(1)
# 23:59+
+as.time(1)
# 00:00+

#sorting
sort(unique(as.time(c(180,120,60))))
# 00:01 00:02 00:03
```

`toSAS.datetime` *Convert Timepoint to SAS Format*

Description

Convert timepoint objects to SAS format for writing XPT files

Usage

```
## S3 method for class 'datetime'
toSAS(x, format="", format.info=NULL)
## S3 method for class 'date'
toSAS(x, format="", format.info=NULL)
## S3 method for class 'time'
toSAS(x, format="", format.info=NULL)
```

Arguments

`x`	subclass of timepoint
`format`	SAS format name
`format.info`	Table of SAS format information

Details

SASxport defines `toSAS` and calls it on each column when writing XPT files. The `datetime` method returns the integer number of seconds since the start of 1960-01-01. The `date` method returns the integer number of days since 1960-01-01.The `time` method returns the number of seconds since midnight.

Value

numeric

Author(s)

Tim Bergsma

See Also

- `timepoint`

Examples

```
if(require(SASxport)) toSAS(as.datetime('1960-01-01T00:00')) # 0
if(require(SASxport)) toSAS(as.date('1960-01-02')) # 1
if(require(SASxport)) toSAS(as.time('00:01')) # 60
```

Index