Package 'timetools'

March 22, 2025

Type Package

Title Seasonal/Sequential (Instants/Durations, Even or not) Time Series

Version 1.15.4

Date 2025-03-22

Imports methods

Author Vladislav Navel [aut, cre]

Maintainer Vladislav Navel <vnavel@yahoo.fr>

Description Objects to manipulate sequential and seasonal time series. Sequential time series based on time instants and time duration are handled. Both can be regularly or unevenly spaced (overlapping duration are allowed). Only POSIX* format are used for dates and times. The following classes are provided : 'POSIXcti', 'POSIXctp', 'TimeIntervalDataFrame', 'TimeInstantDataFrame', 'SubtimeDataFrame' ; methods to switch from a class to another and to modify the time support of series (hourly time series to daily time series for instance) are also defined. Tools provided can be used for instance to handle environmental monitoring data (not always produced on a regular time base).

License GPL

LazyLoad yes

URL https://sourceforge.net/projects/timetools/

Encoding UTF-8

NeedsCompilation yes

RoxygenNote 7.2.3

Repository CRAN

Date/Publication 2025-03-22 17:40:02 UTC

Contents

`timetools-package` *Seasonal/Sequential (Instants/Duration, Even or not) Time Series*

Description

Objects to manipulate sequential and seasonal time series. Sequential time series based on time instants and time duration are handled. Both can be regularly or unevenly spaced (overlapping duration are allowed).

Only POSIX* format are used for dates and times.

The following classes are provided : POSIXcti, POSIXctp, TimeIntervalDataFrame, TimeInstantDataFrame, SubtimeDataFrame ; methods to switch from a class to another and to modify the time support of series (hourly time series to daily time series for instance) are also defined.

Tools provided can be used for instance to handle environmental monitoring data (not always produced on a regular time base).

Author(s)

Vladislav Navel <vnavel@yahoo.fr>

See Also

TimeInstantDataFrame, TimeIntervalDataFrame, changeSupport, SubtimeDataFrame, POSIXcti, POSIXctp

Examples

```
ti1 <- TimeIntervalDataFrame (
c('2010-01-01', '2010-02-01'), c('2010-02-01', '2010-02-02'),
'UTC', data.frame(ex1=1:2) )

ti2 <- TimeIntervalDataFrame (
c('2010-01-01', '2010-02-01', '2010-02-02'), NULL,
'UTC', data.frame(ex1=1:2) )

all.equal (ti1, ti2)

ti3 <- TimeIntervalDataFrame (
c('2010-01-01', '2010-01-02', '2010-01-04'), NULL,
'UTC', data.frame(ex3=c(6, 1.5)))

# weighted mean over a period of 3 days with at least 75% of
# coverage (NA is retunr if not)
ti3
d <- POSIXctp(unit='day')
changeSupport (ti3, 3L*d, 0.75)


ti4 <- TimeIntervalDataFrame (
c('2010-01-01', '2010-01-02', '2010-01-04',
  '2010-01-07', '2010-01-09', '2010-01-10'), NULL,
'UTC', data.frame(ex4=c(6, 1.5, 5, 3, NA)))

# weighted mean over a period of 3 days with at least 75% of
# coverage (NA is retunr if not) or 50%
ti4
changeSupport (ti4, 3L*d, 0.75)
changeSupport (ti4, 3L*d, 0.5)
```

---

as.data.frame *Convert an object to a data.frame*

Description

Convert an object to a data.frame.

Usage

```
  ## S3 method for class 'TimeInstantDataFrame'
as.data.frame(x, row.names=NULL, optional=FALSE,
   include.dates=FALSE, ...)
  ## S3 method for class 'TimeIntervalDataFrame'
as.data.frame(x, row.names=NULL, optional=FALSE,
   include.dates=FALSE, ...)
  ## S3 method for class 'SubtimeDataFrame'
as.data.frame(x, row.names=NULL, optional=FALSE,
   include.dates=FALSE, ...)
```

Arguments

x	TimeIntervalDataFrame, TimeInstantDataFrame or SubtimeDataFrame
row.names	'NULL' or a character vector giving the row names for the data frame. Missing values are not allowed.
optional	logical. If 'TRUE', setting row names and converting column names (to syntactic names: see 'make.names') is optional.
include.dates	should time properties be included in the data.frame as a column ? (or 2 columns for TimeIntervalDataFrame)
...	additional arguments to be passed to or from methods.

Value

a data.frame

---

changeSupport *Function to change time support of TimeIntervalDataFrame*

Description

Methods that allows to aggregate AND disaggregate homogeneous AND heterogeneous time data.

Usage

```
  changeSupport(from, to, min.coverage, FUN = NULL,
    weights.arg = NULL, split.from = FALSE,
    merge.from = TRUE, ...)

  ## S4 method for signature 'TimeIntervalDataFrame,POSIXctp,numeric'
changeSupport(from, to, min.coverage, FUN=NULL,
 weights.arg=NULL, split.from=FALSE,
 merge.from=TRUE, ...)
  ## S4 method for signature
## 'TimeIntervalDataFrame,TimeIntervalDataFrame,numeric'
changeSupport(from, to, min.coverage,
```

```
FUN=NULL, weights.arg=NULL,
split.from=FALSE, merge.from=TRUE, ...)
  ## S4 method for signature 'TimeIntervalDataFrame,character,numeric'
changeSupport(from, to, min.coverage, FUN=NULL,
 weights.arg=NULL, split.from=FALSE,
 merge.from=TRUE, ...)
```

Arguments

from	TimeIntervalDataFrame for which the time support is to change
to	an object indicating the new support, see specific sections
min.coverage	a numeric between 0 and 1 indicating the percentage of valid values over each interval to allow an aggregation. NA is returned if the percentage is not reach. In changeSupport, when values are aggregated, intervals are not allowed to overlap. When a function (FUN) has a na.rm argument, the na.rm=TRUE behaviour is met if na.rm is set to TRUE and min.coverage to 0 (zero) ; the na.rm=FALSE behaviour is met if na.rm is set to FALSE whatever is the value of min.coverage. If min.coverage is as.numeric(NA), the function FUN is apply on all data within the interval, without checking if there is any overlapping. In this case, the result of the transformation must be analysed carefully.
FUN	function use to aggregate data of from. By default mean if 'from' is homogeneous. weighted.mean otherwise.
weights.arg	if FUN has a 'weight' argument, this parameter must be a character naming the weight argument. For instance, if FUN is weighted.mean, then weights.arg is 'w'.
...	arguments for FUN or for other methods
split.from	logical indicating if data in 'from' can be used for several intervals of the new time support (see 'details').
merge.from	logical indicating if data in 'from' can be merged over interval of the new time support.

Details

Agreggating homogeneous data is for example to calculate daily means of time series from hourly time series.

Agreggating heterogeneous data is for example to calculate annual means of time series from monthly time series (because each month doesn't have identical weight).

In above cases, the min.coverage allows to control if means should be calculated or not : for the monthly case, if there are NA values and the time coverage of 'not NA' values is lower min.coverage the result will be NA ; if time coverage is higher than min.coverage, the annual mean will be 'estimated' by the mean of available data.

Disaggregating data is more 'artificial' and is disabled by default (with the split.from argument). This argument is also used to precise if one value can be use for aggregation in more than one interval in the resulting TimeIntervalDataFrame (for sliding intervals for instance). Here are some examples of time disaggregration :

- A weekly mean can be dispatched over the days of the week. By default, the value attributed to each day is the value of the week, but this can be changed by using a special function (FUN argument).
- The value of a variable is known from monday at 15 hours to tuesday at 15 hours and from tuesday at 15 hours to wednesday at 15 hours. To 'evaluate' the value of the variable for tuesday can be estimated by doing a weigthed mean between the two values. Weights are determined by the intersection between each interval and tuesday. Here weights will be 0.625 (15/24) and 0.375 (9/24) (In this case, disaggregation is combined with a 'reaggregation').

These are 'trivial' examples but many other usage can be found for these methods. Other functions than weighted.mean or mean can be used. The Qair package (in its legislative part) gives several examples of usage (this package is not available on CRAN but see 'references' to know where you can find it).

Value

TimeIntervalDataFrame

from=TimeIntervalDataFrame, to=TimeIntervalDataFrame

to is a TimeIntervalDataFrame. The method will try to adapt data of from over interval of to. The returned object is the to TimeIntervalDataFrame with new columns corresponding of those of from.

If merge.from is TRUE, values affected for each interval of to will be calculated with all data in the interval. If split.from is TRUE, values partially in the interval will also be used for calculation.

If merge.from is FALSE, values affected for each interval of to will be the one inside this interval. If several values are inside the interval, NA will be affected. If split.from is TRUE, a value partially inside the interval is considered as being inside it. So if there is no other values in the interval, this value will be affected, else NA will be affected.

from=TimeIntervalDataFrame, to=character

to is one of 'year', 'month', 'day', 'hour', 'minute' or 'second'. It defines the period (POSIXctp) to use to build the new TimeIntervalDataFrame on which from will be aggregated (or disaggregated).

So first, an 'empty' (no data) TimeIntervalDataFrame is created, and then, the aggregation is done accordingly to the 'from=TimeIntervalDataFrame, to=TimeIntervalDataFrame' section.

from=TimeIntervalDataFrame, to=POSIXctp

to is period (see POSIXctp). It defines the base of the new TimeIntervalDataFrame on which from will be aggregated (or disaggregated).

So first, an 'empty' (no data) TimeIntervalDataFrame is created, and then, the aggregation is done accordingly to the 'from=TimeIntervalDataFrame, to=TimeIntervalDataFrame' section.

References

Qair-package : https://sourceforge.net/projects/packagerqair/

See Also

TimeIntervalDataFrame, POSIXcti

Examples

```
ti3 <- TimeIntervalDataFrame (
       c('2010-01-01', '2010-01-02', '2010-01-04'), NULL,
       'UTC', data.frame(ex3=c(6, 1.5)))

# weighted mean over a period of 3 days with at least 75% of
# coverage (NA is retunr if not)
ti3
d <- POSIXctp(unit='day')
changeSupport (ti3, 3L*d, 0.75)

ti4 <- TimeIntervalDataFrame (
c('2010-01-01', '2010-01-02', '2010-01-04',
  '2010-01-07', '2010-01-09', '2010-01-10'), NULL,
         'UTC', data.frame(ex4=c(6, 1.5, 5, 3, NA)))

# weighted mean over a period of 3 days with at least 75% of
# coverage (NA is retunr if not) or 50%
ti4
changeSupport (ti4, 3L*d, 0.75)
changeSupport (ti4, 3L*d, 0.5)

# use of split.from
ti1 <- RegularTimeIntervalDataFrame('2011-01-01', '2011-02-01', 'hour')
ti1$value <- 1:nrow(ti1)
# we can calculate sliding mean over periods of 24 hours.
# first lets build the corresponding TimeIntervalDataFrame
ti2 <- RegularTimeIntervalDataFrame('2011-01-01', '2011-02-01', 'hour', 'day')
# if we try to 'project' ti1 over ti2 it won't work :
summary (changeSupport (ti1[1:200,], ti2[1:200,], 0))
# all data are NA because 'spliting' is not enabled. Let's enable it :
summary (changeSupport (ti1[1:200,], ti2[1:200,], 0, split.from=TRUE))
```

---

compute.lim *Calculate limits for plotting*

Description

This function return a 2 elements vectors (numeric) which can be use as graph limits (xlim, ylim, rlim, etc.)

Usage

```
compute.lim(x, na.rm = FALSE)
```

Arguments

x	'numeric' for which limits must be calculated
na.rm	boolean should NA values be removed before calculation ?

Value

numeric of length 2

---

continuous *Test if a time object is continuous/set an time object continuous.*

Description

For Time objects.

Usage

```
continuous(x, ...)
continuous(x) <- value
```

Arguments

x	object to test
value	logical indicating whether x must be 'continuify' or not.
...	arguments to or from other methods

Details

For objects based on time intervals. After ordering intervals, test if the end of an interval is the start of the next interval. If any interval overlap another one, it returns FALSE.

If not any interval overlap another, and the object is not continuous, the object can be set 'continuous' with

`continuous(obj) <- TRUE`

Intervals will be added such as the object can pass the test describe below. The data is filled with NA values.

Value

Logical indicating if the object is continuous or not.

or

The object set continuous.

See Also

TimeIntervalDataFrame, POSIXcti

duration *Extract duration of a Time object*

Description

For Time objects.

Usage

```
duration(x, ...)
```

Arguments

x	object from which get the duration
...	arguments to or from other methods

Value

For time intervals it returns a vector of integers indicating, for each time interval, the duration of the interval in seconds.

For time periods it returns a vector of integers indicating the duration of each time period using its own time unit. For instance :

```
duration(POSIXctp(1:2, c('month', 'year'))) > 1 2
```

See Also

POSIXcti, POSIXctp

homogeneous *Test if a time object is homogeneous*

Description

For objects based on time intervals (`POSIXcti`). Test if intervals of the object are 'homogeneous' : if the period of each interval is the same.

Usage

```
homogeneous(x, ...)
```

Arguments

x	object to test for homogeneity
...	arguments to or from other methods

Value

logical indicating if 'x' is homogeneous or not.

See Also

TimeIntervalDataFrame, POSIXcti

`interval` *Extract time intervals of a time object.*

Description

For objects based on time intervals. Return `POSIXcti`.

Usage

```
interval(x, ...)
```

Arguments

`x`	object from which get time intervals
`...`	arguments to or from other methods

See Also

TimeIntervalDataFrame, POSIXcti

`ops.numeric` *define generic function to compare anything to a numeric*

Description

define generic function to compare anything to a numeric

origin *1970-01-01 GMT*

Description

Origin is the date-time for 1970-01-01 GMT in POSIXct format. This date-time is the origin for the numbering system used by POSIXct, POSIXlt, chron, and Date classes.

Usage

```
origin
```

Format

POSIXt[1:1], format: "1970-01-01 01:00:00"

Details

The original implementation of this 'object' is in the lubridate package.

Author(s)

Garrett Grolemund "grolemund at rice.edu", Hadley Wickham "h.wickham at gmail.com"

Examples

```
origin
# "1970-01-01 GMT"
```

overlapping *Test if any interval of a time intervals object intersect another*

Description

For objects based on time intervals. Test if any interval overlap another one. Because the test can be ressource consuming, it stops at the first case encountered that does not satisfy this condition. The two indices corresponding are printed.

Usage

```
  overlapping(x, idx, ...)

  ## S4 method for signature 'TimeIntervalDataFrame,ANY'
overlapping(x, idx, ...)
  ## S4 method for signature 'TimeIntervalDataFrame,logical'
overlapping(x, idx, ...)
```

Arguments

x	object to test for overlapping
idx	Logical set to TRUE if indexes of all overlapping intervals are to retrieve.
...	arguments to or from other methods

Value

logical indicating if 'x' has any overlapping time interval.

See Also

TimeIntervalDataFrame, POSIXcti

---

period *Test or extract different properties of Time objects*

Description

For objects based on time intervals. Return POSIXctp of the object if it is homogeneous and continuous.

Usage

```
period(x, ...)
```

Arguments

x	object from which get the period
...	arguments to or from other methods

Value

a time period object if possible. An error occur if not.

See Also

TimeIntervalDataFrame, POSIXctp

POSIXcti *Class* "POSIXcti"

Description

S4 class that defines 'time interval' objects.

Usage

```
POSIXcti(start, end, timezone='UTC', ...)
as.POSIXcti(from, ...)

  ## S3 method for class 'POSIXcti'
x[i, ...]
  ## S3 replacement method for class 'POSIXcti'
x[i] <- value
  ## S3 method for class 'POSIXcti'
c(...)
  ## S3 method for class 'POSIXcti'
split(x, f, drop=FALSE, ...)
  ## S3 method for class 'POSIXcti'
rep(x, ...)
  ## S3 method for class 'POSIXcti'
unique(x, incomparables=FALSE, ...)
  ## S3 method for class 'POSIXcti'
i1 %intersect% i2

  ## S3 method for class 'POSIXcti'
start(x, ...)
  ## S3 method for class 'POSIXcti'
end(x, ...)
  ## S4 method for signature 'POSIXcti'
duration(x, ...)

  ## S4 method for signature 'POSIXcti'
length(x)

  ## S3 method for class 'POSIXcti'
print(x, ...)
  ## S3 method for class 'POSIXcti'
format(x, format = "%Y-%m-%d %H:%M:%S", ...)
  ## S3 method for class 'POSIXcti'
summary(object, ...)
  ## S3 method for class 'POSIXcti'
head(x, ...)
  ## S3 method for class 'POSIXcti'
tail(x, ...)
```

```
  ## S4 method for signature 'POSIXcti'
show(object)

  ## S4 method for signature 'POSIXcti,POSIXcti'
match(x, table, nomatch = NA_integer_, incomparables=NULL)
  ## S4 method for signature 'POSIXcti,POSIXcti'
x %in% table

  ## S3 method for class 'POSIXcti'
Ops(e1, e2)

  ## S3 method for class 'POSIXcti'
i1 %included% i2

  ## S4 method for signature 'POSIXcti,POSIXctp'
e1 + e2
  ## S4 method for signature 'POSIXctp,POSIXcti'
e1 + e2
  ## S4 method for signature 'POSIXcti,POSIXctp'
e1 - e2
```

Arguments

start	POSIXct object indicating the beginning of the time interval.
end	POSIXct object indicating the end of the time interval.
timezone	character indicating the timezone in which the time interval is set. See timezone.
from	Object to convert to a time interval (actually works only for NA).
x	POSIXcti object on which the method has to be applied.
i	index (logical or numeric) of the time interval objects.
value	New POSIXcti object.
f	factor used to group the POSIXcti elements.
drop	Argument specific to the split method. See link[base]{split} documentation.
format	character indicating the format to use to represent the time interval. See section 'Text representation' below for further details.
object	POSIXcti object on which the method has to be applied.
table	The values (POSIXcti vector) to be matched against. See match for further details.
nomatch	The value to be returned i nthe case when no match is found. See match for further details.
incomparables	A vector of values that cannot be matched. See match for further details.
e1	For (<, <=, !=, ==, >=, >) POSIXcti to compare ; otherwise a POSIXcti to shift by a time period (POSIXctp) or a POSIXctp by which a POSIXcti has to be shift.
e2	For (<, <=, !=, ==, >=, >) POSIXcti to compare ; otherwise a POSIXcti to shift by a time period (POSIXctp) or a POSIXctp by which a POSIXcti has to be shift.

i1	POSIXcti to test/intersect. See %included% and %intersect% for further details.
i2	POSIXcti to test/intersect. See %included% and %intersect% for further details.
...	More arguments.

Objects from the Class

Objects of this class represent time intervals. One object is actually a vector of time intervals and so can have a length of one for a single time interval or a length of 'n' for 'n' time intervals.

Slots

start: Object of class "POSIXct" corresponding to the beginning of the interval.

duration: integer indicating in seconds the duration of intervals.

Accessing to POSIXcti properties

A POSIXcti has several properties. Because a POSIXcti is a vector of time intervals, the class has a length function. Other properties are time properties : start, end and duration allow to access to the corresponding properties. The duration of a time interval is the number of seconds for which the interval last.

Manipulating POSIXcti

Manipulating POSIXcti means acting on POSIXctis like on classical vectors. Methods available for this task allow to extract or replace parts of a POSIXcti (with the usual '[' operator), and to concatenate (c) or split POSIXcti (split). A unique and a rep method are defined to uniquify or repeat elements of a POSIXcti.

match and %in% methods have also been defined to find POSIXctp objects among others.

Last, the %intersect% method allow to intersect two POSIXcti.

Text representation

To represent a POSIXcti available functions are print, summary, head, tail, show and format.

The five first functions work the same way that their generic definition.

POSIXcti are formatted by pasting the character strings representing both start and end of each intervals. Start and end's format can be specified with the format argument according to the basic format.POSIXct function.

Testing two POSIXcti

To test two POSIXcti the different operators of comparison are used. One more is defined : %included%. If the POSIXcti compared have a different length, the shorter is recycled over the longer so the resulting vector (a logical vector) has length equal to the longer object.

Comparisons are made element by element. The result for a single comparison is given there :

e1 < e2 TRUE if end(e1) <= start(e2).

e1 <= e2 TRUE if `start(e1) <= start(e2) & end(e1) <= end(e2)`.

e1 != e2 TRUE if `start(e1) != start(e2) | duration(e1) != duration(e2)`.

e1 == e2 TRUE if `start(e1) == start(e2) & duration(e1) == duration(e2)`.

e1 >= e2 TRUE if `start(e1) >= start(e2) & end(e1) >= end(e2)`.

e1 > e2 TRUE if `end(e1) >= start(e2)`.

i1 %included% i2 TRUE if `start(i1) >= start(i2) & end(i1) <= end(i2)`.

Mathematical operations on POSIXcti

'Mathematical' operations are actually 'time lagging' for `POSIXcti`. A `time period` is added/removed to both start and end of intervals. The available operations are :

- POSIXcti + POSIXctp
- POSIXctp + POSIXcti
- POSIXcti - POSIXctp

See Also

`POSIXct`, `TimeIntervalDataFrame`, `POSIXst`, `POSIXctp`

Examples

```
# time interval : january of year 2013
jan <- POSIXcti('2013-01-01', '2013-02-01')
jan

# the complete year
y2013 <- POSIXcti('2013-01-01', '2014-01-01')
y2013

# is jan in 2013 ?
jan %included% y2013

# intersection
jan %intersect% y2013
```

POSIXctp *Class* "POSIXctp"

Description

Class that defines 'periods of time' objects such as 'one month', 'two months', 'three hours', 'four minutes', etc.

Usage

```
POSIXctp(duration, unit)
as.POSIXctp(from, ...)

  ## S3 method for class 'POSIXctp'
x[i, ...]
  ## S3 replacement method for class 'POSIXctp'
x[i] <- value
  ## S3 method for class 'POSIXctp'
c(...)
  ## S3 method for class 'POSIXctp'
split(x, f, drop=FALSE, ...)
  ## S3 method for class 'POSIXctp'
rep(x, ...)
  ## S3 method for class 'POSIXctp'
unique(x, incomparables=FALSE, ...)

  ## S4 method for signature 'POSIXctp'
unit(x, ...)
  ## S4 replacement method for signature 'POSIXctp'
unit(object) <- value
  ## S4 method for signature 'POSIXctp'
duration(x, ...)

  ## S4 method for signature 'POSIXctp'
length(x)

  ## S3 method for class 'POSIXctp'
print(x, ...)
  ## S3 method for class 'POSIXctp'
format(x, ...)
  ## S3 method for class 'POSIXctp'
summary(object, ...)
  ## S3 method for class 'POSIXctp'
head(x, ...)
  ## S3 method for class 'POSIXctp'
tail(x, ...)
  ## S4 method for signature 'POSIXctp'
show(object)

  ## S4 method for signature 'POSIXctp,POSIXctp'
match(x, table, nomatch = NA_integer_, incomparables=NULL)
  ## S4 method for signature 'POSIXctp,ANY'
match(x, table, nomatch = NA_integer_, incomparables=NULL)
  ## S4 method for signature 'POSIXctp,ANY'
x %in% table

  ## S3 method for class 'POSIXctp'
```

```
Ops(e1, e2)

  ## S4 method for signature 'numeric,POSIXctp'
e1 * e2
  ## S4 method for signature 'POSIXctp,numeric'
e1 * e2
  ## S4 method for signature 'POSIXctp,POSIXctp'
e1 + e2
  ## S4 method for signature 'POSIXctp,POSIXctp'
e1 - e2

  ## S4 method for signature 'POSIXct,POSIXctp'
e1 + e2
  ## S4 method for signature 'POSIXctp,POSIXct'
e1 + e2
  ## S4 method for signature 'POSIXct,POSIXctp'
e1 - e2

  ## S4 method for signature 'POSIXctp'
as.numeric(x, ...)
```

Arguments

duration	A vector integer indicating the duration of period (2 for 2 months, 1 for 1 year, etc). If a vector of numeric is given, it will coerced to an integer. Can be missing, see details below.
unit	A vector of factors defined by POSIXt.units() or a vector of character corresponding to the previous factors. See details below.
from	Object to convert to a period of time (actually works only for NA).
x	POSIXctp object on which the method has to be applied.
i	index (logical or numeric) of the POSIXctp objects.
value	New POSIXctp object.
f	factor used to group the POSIXctp elements.
drop	Argument specific to the split method. See link[base]{split} documentation.
object	POSIXctp object on which the method has to be applied.
table	The values (POSIXctp) to be matched against. See match for further details.
nomatch	The value to be returned i nthe case when no match is found. See match for further details.
incomparables	A vector of values that cannot be matched. See match for further details.
e1	POSIXctp, numeric or POSIXct. See details.
e2	POSIXctp, numeric or POSIXct. See details.
...	More arguments.

Objects from the Class

Objects of this class are used to represent periods of times such as 'one hour', 'two seconds', 'three years', etc. Partial periods of time are not allowed ('1.5 hours' will be coerced to an integer value using `as.integer`).

POSIXctp ('p' stands for'period') has only one unit. So 'one hour and 2 seconds' is not defined.

One object is actually a vector of periods of time and so can have a length of one for a single period of time or a length of 'n' for 'n' periods of time.

Slots

`duration`**:** integer corresponding to the length of the period.

`unit`**:** factor indicating the time unit of the period. See `POSIXt.units` to know available units.

Accessing to POSIXctp properties

A POSIXctp has several properties. Because a POSIXctp is a vector of periods of time, the class has a `length` function. Other properties are time properties : `unit` and `duration` allow to access to the corresponding properties. The `duration` of a period of time is an integer corresponding of the time that the period last, in its time unit. The unit of a period of time is an ordered factor as the one defined by `POSIXt.units`.

A `POSIXctp` can be converted to another time unit base (for instance 2 hours make 120 minutes). For that purpose the function `unit<-` is defined. The conversion will be effective only if the new unit can be exactly defined as a multiple of the old one ('hour' to 'second', ok ; 'year' to 'month', ok ; 'month' to 'minute' , NOT ok ; etc. When conversion can not be done, the result has its unit unchanged.

Manipulating POSIXctp

Manipulating `POSIXctp` means acting on `POSIXctps` like on classical vectors. Methods available for this task allow to extract or replace parts of a `POSIXctp` (with the usual '[' operator), and to concatenate (`c`) or split `POSIXctp` (`split`). A `unique` and a `rep` method are defined to uniquify or repeat elements of a `POSIXctp`.

`match` and `%in%` methods have also been defined to find `POSIXctp` objects among others.

Text representation

To represent a `POSIXctp` available functions are `print`, `summary`, `head`, `tail`, `show` and `format`.

The five first functions work the same way that their generic definition.

`POSIXctp` are formatted by pasting their duration (`integer`) with their unit (and with an 's' if relevant).

Testing two POSIXctp

To test two POSIXctp the different operators of comparison are used. If the `POSIXctp` compared have a different length, the shorter is recycled over the longer so the resulting vector (a logical vector) has length equal to the longer object.

Comparisons are made element by element. For a single comparison, first elements are converted into the same unit. If this is not possible, FALSE is returned if the test is '==', TRUE if the test is '!=', NA otherwise (elements can not be compared) ; else duration of elements are compared and the result of this comparison is returned.

Mathematical operations on POSIXctp

`POSIXctp` can be added (or subtracted) to different type of objects : to other `POSIXctp`, to `POSIXct`, to `POSIXcti` and to `POSIXst`. `POSIXctp` can also be multiplied by `numeric`.

For all operations, if the two arguments have a different length, the shorter is recycled over the longer so the resulting vector (a logical vector) has length equal to the longer object.

Basic mathematical operation for `POSIXctp` are (negative periods of time can be defined !!) :

- integer * POSIXctp
- POSIXctp * integer
- POSIXctp + POSIXctp
- POSIXctp - POSIXctp

When `POSIXctp`s do not have the same unit, an attempt is made to convert one to the unit of the other, if it successes the operation is done otherwise NA is returned.

Mathematical operations with `POSIXct`, `POSIXcti` and `POSIXst` are actually time lagging. A `POSIXct` to which a `POSIXctp` is added is lagged by the time periods indicated ; for a `POSIXcti`, start and end are lagged by the time periods. For `POSIXst`, units of the `POSIXst` must be identical : the object is then lagged by the time periods (if the result is higher than the maximum the result is recycled at the beginning. For instance : `saturday + 2 days = monday`).

- POSIXct + POSIXctp
- POSIXctp + POSIXct
- POSIXct - POSIXctp
- .
- POSIXcti + POSIXctp
- POSIXctp + POSIXcti
- POSIXcti - POSIXctp
- .
- POSIXst + POSIXctp
- POSIXctp + POSIXst
- POSIXst - POSIXctp

Changing class

`POSIXctp` can be converted to `numeric` with the `as.numeric` method. The duration of the object is returned.

See Also

`POSIXct`, `POSIXcti`, `POSIXst`

Examples

```
showClass("POSIXctp")
```

POSIXst *Class* "POSIXst"

Description

Class to define POSIXst object such as hours of day, seconds of year, etc.

Usage

```
POSIXst(x, unit, of = NULL, tz = "UTC", ...)
  ## Default S3 method:
 POSIXst(x, unit, of = NULL, tz = "UTC", ...)
  ## S3 method for class 'integer'
 POSIXst(x, unit, of = NULL, tz = "UTC", ...)
  ## S3 method for class 'numeric'
 POSIXst(x, unit, of = NULL, tz = "UTC", ...)
  ## S3 method for class 'POSIXct'
 POSIXst(x, unit, of = NULL, tz = attributes(x)$tzone, ...)
  ## S3 method for class 'POSIXlt'
 POSIXst(x, unit, of = NULL, tz = attributes(x)$tzone, ...)
  ## S3 method for class 'TimeInstantDataFrame'
 POSIXst(x, unit, of = NULL, tz = timezone(x), ...)
  ## S3 method for class 'TimeIntervalDataFrame'
 POSIXst(x, unit, of = NULL, tz = timezone(x), ..., cursor = NULL)

year(x, ...)
month(x, ...)
day(x, of, ...)
hour(x, of, ...)
minute(x, of, ...)
second(x, of, ...)

  ## S3 method for class 'POSIXst'
x[i]
  ## S3 replacement method for class 'POSIXst'
x[i] <- value
  ## S3 method for class 'POSIXst'
c(...)
  ## S3 method for class 'POSIXst'
split(x, f, drop=FALSE, ...)
  ## S3 method for class 'POSIXst'
rep(x, ...)
  ## S3 method for class 'POSIXst'
seq(from, to, ...)
```

```
  ## S3 method for class 'POSIXst'
unique(x, incomparables=FALSE, ...)
  ## S3 method for class 'POSIXst'
duplicated(x, incomparables=FALSE, ...)

  ## S3 method for class 'POSIXst'
unit(x, ...)
  ## S3 method for class 'POSIXst'
of(x, ...)
  ## S3 method for class 'POSIXst'
timezone(object)

  ## S4 method for signature 'POSIXst'
length(x)

  ## S3 method for class 'POSIXst'
print(x, ...)
  ## S3 method for class 'POSIXst'
format(x, format, ...)
  ## S3 method for class 'POSIXst'
summary(object, ...)
  ## S3 method for class 'POSIXst'
head(x, ...)
  ## S3 method for class 'POSIXst'
tail(x, ...)
  ## S4 method for signature 'POSIXst'
show(object)

  ## S4 method for signature 'POSIXst,POSIXst'
match(x, table, nomatch = NA_integer_, incomparables=NULL)
  ## S4 method for signature 'POSIXst,ANY'
match(x, table, nomatch = NA_integer_, incomparables=NULL)
  ## S4 method for signature 'POSIXst,ANY'
x %in% table

  ## S3 method for class 'POSIXst'
Ops(e1, e2)

  ## S4 method for signature 'POSIXst,POSIXctp'
e1 + e2
  ## S4 method for signature 'POSIXctp,POSIXst'
e1 + e2
  ## S4 method for signature 'POSIXst,POSIXctp'
e1 - e2

  ## S4 method for signature 'POSIXst,POSIXst'
e1 - e2
```

```
  ## S4 method for signature 'POSIXst'
as.numeric(x, ...)
```

Arguments

x	object to convert into POSIXst or POSIXst object on which the method has to be applied.
unit	indicates the subtime part to extract ('year', 'month', 'day', 'hour', 'minute', 'second')
of	used to specify the main period from which the is to extract ('year', 'month', day', 'hour', 'minute'). Not used for 'unit in c('year', 'month')'.
tz	if needed, specifies the timezone of POSIXst
cursor	for TimeIntervalDataFrame, if not NULL, the object is first coerced to a TimeInstantDataFrame using the as.TimeInstantDataFrame method.
i	index (logical or numeric) of the POSIXst objects.
value	New POSIXst object.
f	factor used to group the POSIXst elements.
drop	Argument specific to the split method. See link[base]{split} documentation.
object	POSIXst object on which the method has to be applied.
format	Character string to precise the desired format. See section 'Text representation' below for details.
table	The values (POSIXst) to be matched against. See match for further details.
nomatch	The value to be returned i nthe case when no match is found. See match for further details.
incomparables	A vector of values that cannot be matched. See match for further details.
e1	POSIXst or POSIXctp. See details.
e2	POSIXst or POSIXctp. See details.
from, to	starting and end values to sequence, see seq
...	More arguments.

Objects from the Class

Objects of this class are used to represent subtimes. A subtime (or a 'POSIXst', 'st' stand for Sub and Time) is a subdivision of time :

- second of a minute ;
- second of an hour ;
- second of a day ;
- second of a week ;
- second of a month ;

- second of a year ;
- minute of an hour ;
- minute of a day ;
- ...
- minute of a year ;
- ...
- month of year ;
- year AD (after death).

A POSIXst is a kind of time object composed of 2 units and a positional integer. The main unit can be accessed via the 'of' function ; the sub unit can be accessed via the 'unit' function. The positional integer correspond to the value of the subtime object. Consequently, a subtime `st` is the `valueth unit(st)` of `of(st)`.

The range of valid values for each kind of POSIXst is defined accordingly to the `DateTimeClasses` definitions. For instance valid values for seconds of hour are 0 to 61, valid values for day of week are 0 to 6, etc.

To define POSIXst objects see POSIXst section below.

One object is actually a vector of subtimes and so can have a length of one for a single subtime or a length of 'n' for 'n' subtimes.

Last, a POSIXst object has a 'timezone' slots. This is defined for compatibility reason with POSIXct object and also with `TimeInstantDataFrame`, `TimeIntervalDataFrame` and `SubtimeDataFrame`.

Slots

`subtime`**:** Object of class `"integer"` corresponding to the actual value of each subtime.

`unit`**:** factor representing a time unit. It represents the subdvision of time (in 'second of year' it corresponds to 'second'). See `POSIXt.units`.

`of`**:** factor representing a time unit. It represents the main time unit (in 'second of year' it corresponds to 'year'). See `POSIXt.units`.

`timezone`**:** Object of class `"character"` indicating the timezone of the POSIXst object.

POSIXst

`POSIXst` objects can be created from various other class objects. For this purpose the `POSIXst` method has been defined.

First, if 'x' is missing, an empty factor with the appropriated levels (according to 'unit and of') is returned.

A `POSIXst` can be created from an integer or a numeric. If so, 'unit' and 'of' must be supplied (see arguments section above). The 'tz' argument can be supplied (numeric will be converted to an integer). In both cases, values of the integer/numeric must be in the right range (see `DateTimeClasses`.

A `POSIXst` can be created from a `POSIXct` or `POSIXlt` object. In this case, the subtime (POSIXst) is extracted in the units indicated by 'unit' and 'of' arguments. The 'tz' argument indicates the timezone of the resulting object (it doesn't do any conversion on the POSIX(l,c)t objects).

Finally, POSIXst can be extracted from TimeInstantDataFrame and TimeIntervalDataFrame. For the first type of object, the method is applied to the time instants (which are POSIXct). For the latter, because a time interval can contains several POSIXst of one kind (for instance a day contains all 'hours of day'), the result of this method TimeIntervalDataFrame is a list of POSIXst. Each element of the list contains the POSIXsts asked for corresponding to each row of the TimeIntervalDataFrame object. If 'cursor' is supplied, the TimeIntervalDataFrame is first converted to a TimeInstantDataFrame (see as.TimeInstantDataFrame for details).

year(...), month(...), day(...), hour(...), minute(...) and second(...)

year, month, day, hour, minute and second are methods defined to extract the adequat information from a time object. These functions are wrappers to POSIXst.

Each of these methods call POSIXst replacing the unit argument with its own name : minute(x, of='day') will call POSIXst(x, unit='minute', of='day')

Accessing to POSIXst properties

A POSIXst has several properties. Because a POSIXst is a vector of subtimes, the class has a length function. Other properties are time properties : unit, of and timezone allow to access to the corresponding properties. The 'unit' and 'of' of a subtime is an ordered factor as the one defined by POSIXt.units.

For more informations on timezone, see the page of the manual.

Manipulating POSIXst

Manipulating POSIXst means acting on POSIXsts like on classical vectors. Methods available for this task allow to extract or replace parts of a POSIXst (with the usual '[' operator), and to concatenate (c) or split POSIXst (split). A unique, a duplicated, a rep and a seq methods are defined to uniquify, repeat or sequence elements of a POSIXst.

match and %in% methods have also been defined to find POSIXst objects among others.

Text representation

To represent a POSIXst available functions are print, summary, head, tail, show and format.

The five first functions work the same way that their generic definition.

'POSIXst' are formatted according to the format argument which must respect the following rules.

- %v value
- %s subtime unit (slot 'unit')
- %m main unit (slot 'of')
- %a Abbreviated weekday name in the current locale.
- %A Full weekday name in the current locale.
- %b Abbreviated month name in the current locale.
- %B Full month name in the current locale.
- %r timezone
- %p place of subtime (ie the string part of 1st, 2nd, 10th, etc.)

Testing two POSIXst

To test two POSIXst the different operators of comparison are used. If the POSIXst compared have a different length, the shorter is recycled over the longer so the resulting vector (a logical vector) has length equal to the longer object.

Comparisons are made element by element. Two POSIXst with a different 'unit' or a different 'of' are different (TRUE if test is '!=', FALSE if '==' NA otherwise). If they have identical 'unit' and 'of' the comparison is made over subtime slots.

Mathematical operations on POSIXst

POSIXst can be added and subtracted to POSIXctp. POSIXst can also be subtracted (and only subtracted) to POSIXst.

For all operations, if the two arguments have a different length, the shorter is recycled over the longer so the resulting vector (a logical vector) has length equal to the longer object.

Mathematical operations with POSIXctp are actually time lagging. Units of the POSIXctp must be identical to the POSIXst's : the object is then lagged by the time periods (if the result is higher than the maximum the result is recycled at the beginning. For instance : saturday + 2 days = monday).

- POSIXst + POSIXctp
- POSIXctp + POISXst
- POSIXst - POSIXctp

Subtracting a POSIXst to another result in a POSIXctp. For instance wednesday - monday = 2 days.

Changing class

POSIXst can be converted to numeric with the as.numeric method. The subtime slot of the object is returned.

See Also

POSIXct, POSIXcti, POSIXctp, TimeIntervalDataFrame,

Examples

```
showClass("POSIXst")
```

---

regular *Test if a time object is regular*

Description

Test for regularity of a time object.

Usage

```
regular(x, ...)
```

Arguments

x	object to test, from which get or set a property
...	arguments to or from other methods

Details

Test if the object is regular. A TimeInstantDataFrame is regular if all instants are equally spaced. A TimeIntervalDataFrame is regular if it is homogeneous and all interval's starts are equally spaced.

Value

boolean indicating if x is regular or not

See Also

TimeIntervalDataFrame, TimeInstantDataFrame

---

split *Divide into Groups and Reassemble (Time*DataFrame objects and POSIXct*)*

Description

'split' divides the data in the vector 'x' into the groups defined by 'f'. The replacement forms replace values corresponding to such a division. Here are listed 'split' methods defined for Time objects defined in the timetools package `POSIXst`, `POSIXcti`, etc.). See sections below for complete list of methods.

Usage

```
  ## S4 method for signature 'ANY,POSIXctp'
split(x, f, drop = FALSE, ...)
  ## S4 method for signature 'ANY,POSIXcti'
split(x, f, drop = FALSE, ...)
  ## S4 method for signature 'ANY,POSIXst'
split(x, f, drop = FALSE, ...)

  ## S4 method for signature 'TimeIntervalDataFrame,TimeIntervalDataFrame'
split(x, f, ..., split.x=FALSE, keep.f=TRUE)
  ## S4 method for signature 'TimeIntervalDataFrame,POSIXcti'
split(x, f, ..., split.x=FALSE)
  ## S4 method for signature 'TimeIntervalDataFrame,POSIXctp'
split(x, f, ...,  split.x=FALSE)
```

Arguments

x	data frame containing values to be divided into groups. TimeIntervalDataFrame as data.frame.
f	Can be of different kind. Is used to defined the grouping. See details below.
drop	logical indicating if levels that do not occur should be dropped (if f is a factor or a list).
...	further potential arguments passed to methods.
split.x	logical indicating if data in x that are over several intervals or not completely included in one interval of f must be 'cut' to fit to new intervals (TRUE) or ignored (FALSE).
keep.f	logical indicating if f values must be kept on the resulting list.

Details

For each new split method defined in timetools a short description is given there.

Value

The value returned from 'split' is a list of vectors containing the values for the groups. The components of the list are named by the levels of f (after converting to a factor). The class of each element of the list is the one of the initial x structure.

Split over POSIX*t* objects

Since POSIXst, POSIXctp and POSIXcti objects are similar to vector it must be possible to split other type of objects against those ones.

split(x, 'POSIXctp', drop=FALSE, ...) 'POSIXctp' is first cast as character (using format) and then the split is done.

split(x, 'POSIXcti', drop=FALSE, ...) 'POSIXcti' is first cast as character (using format) and then the split is done. The ... argument is used to specify the format if needed.

split(x, 'POSIXst', drop=FALSE, ...) 'POSIXst' is first cast as numeric and then the split is done.

split('TimeIntervalDataFrame', 'TimeIntervalDataFrame', ..., split.x = FALSE, keep.f = TRUE)

Split a TimeIntervalDataFrame into another TimeIntervalDataFrame.

The method takes each time interval of the first TimeIntervalDataFrame (TitDF) and searches with which time intervals of the second it intersects.

Each time interval of the first TItDF can intersect with none, one or several time intervals of the second TItDF. The arguments 'split.x' is defined to tell the method what to do :

- if the time interval in the first TItDF (ti1) doesn't match any in the second TItDF, nothing to do
- if it (ti1) matches one in the second TItDF (ti2) and is included inside it, it (ti1) is entirely taken in the final result

- if it (ti1) intersects one and only one (ti2) inside the second TItDF, (ti1) is truncated to be included inside (ti2) if 'split.x' is TRUE and (ti1) is removed if 'split.x' is FALSE
- if it (ti1) is over several time intervals of the second TItDF (ti2.a, ti2.b, etc.) :
  - if 'split.x' is TRUE, (ti1) is truncated into each ti2.x to be included inside each one
  - if 'split.x' is FALSE, (ti1) is removed.

split('TimeIntervalDataFrame', 'POSIXctp', ..., split.x = FALSE)

Split a TimeIntervalDataFrame into a time period (of length 1). A TimeIntervalDataFrame is created (cf TimeIntervalDataFrame constructor) and the the above method is called.

split('TimeIntervalDataFrame', 'POSIXcti', ..., split.x = FALSE)

Split a TimeIntervalDataFrame into time intervals (POSIXcti). It is exactly the same as splitting a TimeIntervalDataFrame into another except that 'f' has not data.

So a TimeIntervalDataFrame is created according to 'f' and the the above method is called.

See Also

split, TimeIntervalDataFrame-class, POSIXcti, POSIXst-class, POSIXctp-class

---

SubtimeDataFrame *Class* "SubtimeDataFrame"

Description

Class to hold subtime data such a day of week, month of year, etc.

Usage

```
SubtimeDataFrame(when, data = NULL, ...)

as.SubtimeDataFrame(x, unit, of, ...)
  ## S3 method for class 'TimeInstantDataFrame'
as.SubtimeDataFrame(x, unit, of, FUN=NULL, ...)
  ## S3 method for class 'TimeIntervalDataFrame'
as.SubtimeDataFrame(x, unit, of, FUN=NULL, cursor=NULL, ...)

  ## S4 method for signature 'SubtimeDataFrame'
x$name
  ## S4 replacement method for signature 'SubtimeDataFrame'
x$name <- value
  ## S3 method for class 'SubtimeDataFrame'
x[i, j, drop=FALSE]
  ## S3 replacement method for class 'SubtimeDataFrame'
x[i, j] <- value
  ## S4 method for signature 'SubtimeDataFrame'
```

```
x [[i, j, ...]]
  ## S3 replacement method for class 'SubtimeDataFrame'
x[[i, j]] <- value

  ## S3 method for class 'SubtimeDataFrame'
merge(x, y, by, all=TRUE, sort=FALSE, ...)
  ## S3 method for class 'SubtimeDataFrame'
split(x, f, drop=FALSE, ...)
  ## S4 method for signature 'SubtimeDataFrame'
lapply(X, FUN, ...)

  ## S4 method for signature 'SubtimeDataFrame'
timezone(object)
  ## S4 method for signature 'SubtimeDataFrame'
when(x, ...)
  ## S3 method for class 'SubtimeDataFrame'
unit(x, ...)
  ## S3 method for class 'SubtimeDataFrame'
of(x, ...)

  ## S4 method for signature 'SubtimeDataFrame'
dim(x)
  ## S4 method for signature 'SubtimeDataFrame'
length(x)
  ## S4 method for signature 'SubtimeDataFrame'
names(x)
  ## S4 replacement method for signature 'SubtimeDataFrame'
names(x) <- value
  ## S4 method for signature 'SubtimeDataFrame'
ncol(x)
  ## S4 method for signature 'SubtimeDataFrame'
nrow(x)
  ## S3 method for class 'SubtimeDataFrame'
row.names(x)
  ## S3 replacement method for class 'SubtimeDataFrame'
row.names(x) <- value

  ## S3 method for class 'SubtimeDataFrame'
print(x, ...)
  ## S3 method for class 'SubtimeDataFrame'
summary(object, ...)
  ## S3 method for class 'SubtimeDataFrame'
head(x, ...)
  ## S3 method for class 'SubtimeDataFrame'
tail(x, ...)
  ## S4 method for signature 'SubtimeDataFrame'
show(object)
```

```
  ## S3 method for class 'SubtimeDataFrame'
plot(
    x, y=NULL, type='p', lty=1:6, lwd=1, pch=1:25, col=NULL,
    xlim=NULL, ylim=NULL, log='', main='', sub='', xlab='', ylab='',
    ann=par('ann'), axes=TRUE, asp=NA, as.is=TRUE, format=NULL, ...)
  ## S3 method for class 'SubtimeDataFrame'
points(
    x, y=NULL, type='p', lty=1:6, lwd=1, pch=1:25, col=NULL, as.is=TRUE, ...)
  ## S3 method for class 'SubtimeDataFrame'
lines(
    x, y=NULL, type='l', lty=1:6, lwd=1, pch=1:25, col=NULL, as.is=TRUE, ...)
  ## S3 method for class 'SubtimeDataFrame'
barplot(height, format='', ...)
```

Arguments

when	POSIXst.
data	a data.frame with as much rows as needed for the created object. Can be NULL (hence the data.frame has zero column and as much rows as needed).
x	object to convert to a TimeInstantDataFrame or SubtimeDataFrame object (to modify, to extract or to test)
unit	indicates the subtime part to extract ('year', 'month', 'day', 'hour', 'minute', 'second').
of	used to specify the main period from which the is to extract ('year', 'month', 'day', 'hour', 'minute'). Not used for 'unit in c('year', 'month')'.
FUN	function to use for the aggregation (if wanted, see 'details')
cursor	For TimeIntervalDataFrame, it indicates where the TimeInstant must be taken. If 0, start of each intervals is taken as instant ; if 1 end of each intervals is taken as instant. Any other value will determine a weigthed instant between start and end (actually, value higher than 1 or lower than 0 will give instant outside this range).
i	indices specifying elements to extract or replace. (See Extract)
j	indices specifying elements to extract or replace. (See Extract)
name	A litteral character string or a name. (See Extract)
drop	Used for compatibility with data.frame methods.
value	New value for the object designated (data.frame, names, row.names, etc.).
y	SubtimeDataFrame to merge with x
all	logical; see merge
by	specifications of the columns used for merging.
sort	logical; if TRUE the resulting merged SubtimeDataFrame is ordered according to 'when' values.
f	a 'factor' in the sense that 'as.factor(f)' defines the grouping, or a list of such factors in which case their interaction is used for the grouping. See split.
X	a SubtimeDataFrame on which the FUN must be applied.

object	SubtimeDataFrame object (to modify, to extract or to test)
type	plotting argument, see plot.default
lty	plotting argument, see plot.default
lwd	plotting argument, see plot.default
pch	plotting argument, see plot.default
col	plotting argument, see plot.default
xlim	plotting argument, see plot.default
ylim	plotting argument, see plot.default
log	plotting argument, see plot.default
main	plotting argument, see plot.default
sub	plotting argument, see plot.default
xlab	plotting argument, see plot.default
ylab	plotting argument, see plot.default
ann	plotting argument, see plot.default
axes	plotting argument, see plot.default
asp	plotting argument, see plot.default
as.is	should data be represented incrementally (the first row is given an x-value of 1, the second of 2, ... the last of n)(TRUE) or should data be grouped by their subtime (all value corresponding to monday are drawn at an x-value of 1) (FALSE) ?
height	plotting argument, see barplot
format	for barplot see barplot, otherwise a string to format the x-labels according to the format method of POSIXst objects (see 'Text representation' of POSIXst).
...	More arguments.

Objects from the Class

Formally, the class consists of a data.frame and, for each row, a POSIXst (or subtime). This class is provided to deal with subtime data. This class is compatible with TimeIntervalDataFrame and TimeInstantDataFrame.

The construction of the class allows to manipulate objects as if they were data.frame (see 'Access to data' and 'Access to data properties'). Several functions are also available to access to time properties (see 'Access to time properties').

Methods are also available to facilitate the representations of instances of that class : see 'graphic representation' and 'text representation'.

Finally, some specific methods allow to easily deal with aggregation of data over time properties (day, hour, week, special or specific time).

SubtimeDataFrame constructors

Objects can be created by calls of the form

- `new("SubtimeDataFrame", ...)` ...argument must be replaced by named arguments corresponding to slots of a SubtimeDataFrame (see below). See also `new`.
- `SubtimeDataFrame (when, data=NULL, ...)` Arguments of the function correspond to object slots.
- `as.SubtimeDataFrame (from, representation, cursor=NULL, FUN=mean, ...)` Converting object to SubtimeDataFrame. Conversion from a TimeIntervalDataFrame to a SubtimeDataFrame can be direct or after aggregation.

  For a direct conversion (where date are only replaced by the desired subtime), FUN must be NULL.

  For an aggregated conversion, the function to use must be indicated by the FUN arg and all arguments to pass to this function can be given (namely).

Slots

`when`**:** Object of class `"POSIXst"` corresponding to the instant of each row of the `data.frame`.

`data`**:** Object of class `"data.frame"` data contained by the object.

Accessing to and manipulating data

The SubtimeDataFrame class is defined to works like the data.frame class with the difference that a subtime (`POSIXst`) is attached to each rows of the data.frame. Thus to access and manipulate data of a TimeInstantDataFrame the following methods are defined : '$', '$<-', '[', '[<-', '[[', '[[<-'. See `Extract` for details.

Other methods have been defined to allow some operations over TimeInstantDataFrame :

- merge to join two (or more) SubtimeDataFrame (see `merge`),
- a SubtimeDataFrame can be splitted exactly the same way that a data.frame can (see `split` in the base package),
- a function can be applied over each column of a TimeIntervalDataFrame via the lapply function provided that the function return one value (in this case the resulting value is a `TimeIntervalDataFrame` beginning at the first instant of the object and ending at the latest one), or as much values as the number of rows of the object (in this case the SubtimeDataFrame given in argument in returned with the new values calculated).

Because a SubtimeDataFrame works more or less like a data.frame, the following methods work on a SubtimeDataFrame : `dim`, `length`, `names`, `names<-`, `ncol`, `nrow`, `row.names`, `row.names<-`.

Access to time properties

A SubtimeDataFrame can be tested for a few time properties :

timezone gives or sets the timezone of the SubtimeDataFrame ;

when returns a POSIXst object with the time instant of the SubtimeDataFrame.

unit returns the unit of the time instant of the SubtimeDataFrame.

of returns the 'of' of the time instant of the SubtimeDataFrame.

Graphic representation

To plot a SubtimeDataFrame available functions are plot, lines, points and barplot.

These functions works more or less like their generic definition.

Text representation

To represent a SubtimeDataFrame available functions are print, summary, head, tail and show.

See Also

TimeInstantDataFrame, TimeIntervalDataFrame, POSIXst

Examples

```
showClass("SubtimeDataFrame")
```

---

tapply *Apply a Function Over a time properties*

Description

Apply a function over a Time*DataFrame that is first splitted into several sets according to time properties specified by INDEX.

Usage

```
  ## S4 method for signature 'TimeIntervalDataFrame,TimeIntervalDataFrame'
tapply(X, INDEX, FUN, ...,
       min.coverage=1, weights.arg=NULL, merge.X=TRUE, split.X=FALSE,
       keep.INDEX=TRUE,default=NA, simplify=TRUE)

  ## S4 method for signature 'TimeIntervalDataFrame,POSIXctp'
tapply(X, INDEX, FUN, ...,
       min.coverage=1, weights.arg=NULL, merge.X=TRUE, split.X=FALSE,
      default=NA, simplify=TRUE)

  ## S4 method for signature 'TimeIntervalDataFrame,POSIXcti'
tapply(X, INDEX, FUN, ...,
       min.coverage=1, weights.arg=NULL, merge.X=TRUE, split.X=FALSE,
      default=NA, simplify=TRUE)
```

Arguments

X	a TimeIntervalDataFrame or a TimeInstantDataFrame
INDEX	an object corresponding to or containing a time properties. Classes available depend on X. See sections below to know all (X, INDEX) combination defined.

FUN	the function to be applied.
...	optional arguments to 'FUN'.
simplify	if FALSE a list of 'Time*DataFrame' is returned ; if TRUE 'tapply' try to reduce the list to a single 'Time*DataFrame'.
default	argument inherited from the **base** function, currently unused; see tapply.
min.coverage	a numeric between 0 and 1 indicating the percentage of valid values over each interval to allow an aggregation. NA is returned if the percentage is not reach. In that configuration (min.coverage between 0 and 1, overlapping intervals are not allowed). When a function (FUN) has a na.rm argument, the na.rm=TRUE behaviour is met if na.rm is set to TRUE and min.coverage to 0 (zero) ; the na.rm=FALSE behaviour is met if na.rm is set to FALSE whatever is the value of min.coverage. If min.coverage is set to NA, time coverage of the resulting interval is not checked. Moreover, overlapping of X intervals is not checked. Thus the aggregation is done according to 'weights.arg' argument (if given).
weights.arg	if FUN has a 'weight' argument, this parameter must be a character naming the weight argument. For instance, if FUN is weighted.mean, then weights.arg is 'w'.
merge.X	logical indicating if data in 'X' can be merged over interval of the new time support.
split.X	logical indicating if data in 'X' that are over several intervals of 'INDEX' must be 'cut' to fit to new intervals (TRUE) or ignored (FALSE).
keep.INDEX	logical indicating if INDEX values must be kept on the resulting list.

Details

These functions are equivalent to old changeSupport methods. Instead of having the core splitting algorithm in it, it uses the split methods. Be aware that default parameters values between the two families ('changeSupport' and 'tapply') are not necessarily the same.

Users are encouraged to use 'tapply' instead of 'changeSupport' since new versions of 'changeSupport' are only wrappers to tapply.

signature(TimeIntervalDataFrame, TimeIntervalDataFrame)

split TimeIntervalDataFrame over another TimeIntervalDataFrame and then apply a function over each elements of the list.

signature(TimeIntervalDataFrame, POSIXctp)

split a TimeIntervalDataFrame against regular time intervals with a period defined by INDEX (a POSIXctp). Then a function is applied over each elements of the list.

signature(TimeIntervalDataFrame, POSIXcti)

split TimeIntervalDataFrame against specified intervals (POSIXcti). and then apply a function over each elements of the list.

See Also

tapply, TimeIntervalDataFrame-class, TimeInstantDataFrame-class, SubtimeDataFrame-class, changeSupport, POSIXcti-class, POSIXst-class, POSIXctp-class

---

TimeInstantDataFrame *Class* "TimeInstantDataFrame"

Description

Class to hold time data that ARE 'instantaneous'.

Usage

```
  TimeInstantDataFrame(when, timezone = "UTC", data = NULL, sort=FALSE, ...)

  as.TimeInstantDataFrame(from, ...)
  ## S3 method for class 'TimeIntervalDataFrame'
as.TimeInstantDataFrame(from, cursor = NULL, ...)

  RegularTimeInstantDataFrame(from, to, by, timezone = "UTC", data = NULL)

  ## S4 method for signature 'TimeInstantDataFrame'
x$name
  ## S4 replacement method for signature 'TimeInstantDataFrame'
x$name <- value
  ## S3 method for class 'TimeInstantDataFrame'
x[i, j, drop=FALSE]
  ## S3 replacement method for class 'TimeInstantDataFrame'
x[i, j] <- value
  ## S4 method for signature 'TimeInstantDataFrame'
x [[i, j, ...]]
  ## S3 replacement method for class 'TimeInstantDataFrame'
x[[i, j]] <- value

  ## S3 method for class 'TimeInstantDataFrame'
rbind(...)
  ## S3 method for class 'TimeInstantDataFrame'
merge(x, y, by, all=TRUE, tz='UTC', sort=TRUE, ...)
  ## S3 method for class 'TimeInstantDataFrame'
split(x, f, drop=FALSE, ...)
  ## S4 method for signature 'TimeInstantDataFrame'
lapply(X, FUN, ...)

  ## S4 method for signature 'TimeInstantDataFrame'
regular(x, ...)
  ## S4 method for signature 'TimeInstantDataFrame'
timezone(object)
```

```
  ## S4 replacement method for signature 'TimeInstantDataFrame'
timezone(object) <- value
  ## S4 method for signature 'TimeInstantDataFrame'
when(x, ...)

  ## S4 method for signature 'TimeInstantDataFrame'
dim(x)
  ## S4 method for signature 'TimeInstantDataFrame'
length(x)
  ## S4 method for signature 'TimeInstantDataFrame'
names(x)
  ## S4 replacement method for signature 'TimeInstantDataFrame'
names(x) <- value
  ## S4 method for signature 'TimeInstantDataFrame'
ncol(x)
  ## S4 method for signature 'TimeInstantDataFrame'
nrow(x)
  ## S3 method for class 'TimeInstantDataFrame'
row.names(x)
  ## S3 replacement method for class 'TimeInstantDataFrame'
row.names(x) <- value

  ## S3 method for class 'TimeInstantDataFrame'
print(x, tz=NULL, ...)
  ## S3 method for class 'TimeInstantDataFrame'
summary(object, ...)
  ## S3 method for class 'TimeInstantDataFrame'
head(x, tz, ...)
  ## S3 method for class 'TimeInstantDataFrame'
tail(x, tz, ...)
  ## S4 method for signature 'TimeInstantDataFrame'
show(object)

  ## S3 method for class 'TimeInstantDataFrame'
plot(x, y=NULL, type="p",
lty=1:6, lwd=1, pch=1:25, col=NULL,
xlim=NULL, ylim=NULL, log, main, sub, xlab, ylab,
ann=par("ann"), axes=TRUE, asp=NA, ...)
  ## S3 method for class 'TimeInstantDataFrame'
points(x, y=NULL, type="p",
lty=1:6, lwd=1, pch=1:25, col=NULL, ...)
  ## S3 method for class 'TimeInstantDataFrame'
lines(x, y=NULL, type="l",
lty=1:6, lwd=1, pch=1:25, col=NULL, ...)
  ## S3 method for class 'TimeInstantDataFrame'
barplot(height, format='', ...)
```

Arguments

when	POSIXct or character representing a time with a valid format (see POSIXct). It gives the instant of each row.
timezone	character representing a valid timezone (see timezone).
data	a data.frame with as much rows as needed for the created object. Can be NULL (hence the data.frame has zero column and as much rows as needed).
from	**as.TimeInstantDataFrame** object to convert to a TimeInstantDataFrame **RegularTimeInstantDataFrame** POSIXct or character representing a time with a valid format (see POSIXct). It represents the start of the object.
cursor	To convert TimeIntervalDataFrame, it indicates where the TimeInstant must be taken. If 0, start of each intervals is taken as instant ; if 1 end of each intervals is taken as instant. Any other value will determine a weigthed instant between start and end (actually, value higher than 1 or lower than 0 will give instant outside this range).
to	POSIXct or character representing a time with a valid format (see POSIXct). It represents the end of the object. If missing, its value is deduced from 'from', 'by' and 'data'.
by	**RegularTimeInstantDataFrame** a POSIXctp object indicating the increment to use between instants of the object. **merge** specifications of the columns used for merging.
x	TimeInstantDataFrame object (to modify, to extract or to test)
i	indices specifying elements to extract or replace. (See Extract)
j	indices specifying elements to extract or replace. (See Extract)
name	A litteral character string or a name. (See Extract)
drop	Used for compatibility with data.frame methods.
value	New value for the object designated (data.frame, names, row.names, etc.).
y	TimeInstantDataFrame to merge with x
all	logical; see merge
tz	character representing a valid timezone (see timezone).
sort	logical; if TRUE the resulting built/merged TimeInstantDataFrame is ordered according to 'when' values.
f	a 'factor' in the sense that 'as.factor(f)' defines the grouping, or a list of such factors in which case their interaction is used for the grouping. See split.
X	a TimeInstantDataFrame on which the FUN must be applied.
FUN	function to apply over each columns of X.
object	TimeInstantDataFrame object (to modify, to extract or to test)
type	plotting argument, see plot.default
lty	plotting argument, see plot.default
lwd	plotting argument, see plot.default
pch	plotting argument, see plot.default

col	plotting argument, see plot.default
xlim	plotting argument, see plot.default
ylim	plotting argument, see plot.default
log	plotting argument, see plot.default
main	plotting argument, see plot.default
sub	plotting argument, see plot.default
xlab	plotting argument, see plot.default
ylab	plotting argument, see plot.default
ann	plotting argument, see plot.default
axes	plotting argument, see plot.default
asp	plotting argument, see plot.default
height	plotting argument, see barplot
format	plotting argument, see barplot
...	More arguments.

Objects from the Class

Formally, the class consists of a data.frame and, for each row, a POSIXct. This class is provided to deal with punctual time data. Many of such classes are defined in other packages. This one is defined mainly to provide a 'punctual' class compatible with TimeIntervalDataFrame and SubtimeDataFrame.

The construction of the class allows to manipulate objects as if they were data.frame (see 'Access to data' and 'Access to data properties').

Slots

instant: Object of class "POSIXct" corresponding to the instant of each row of the data.frame.

timezone: Object of class "character" indicating the timezone of data both for representation and calculation.

data: Object of class "data.frame" data contained by the object.

TimeInstantDataFrame constructors

Objects can be created by calls of the form

- new("TimeInstantDataFrame", ...) ...argument must be replaced by named arguments corresponding to slots of a TimeInstantDataFrame (see below). See also new.
- TimeInstantDataFrame (when, timezone='UTC', data=NULL, ...) Arguments of the function correspond to object slots.
- RegularTimeInstantDataFrame (from, to, by, timezone='UTC', data=NULL), Wrapper to construct TimeInstantDataFrame with specific properties. Instants of the TimeInstantDataFrame go from 'from' to 'to' regularly spaced by 'by', which is a POSIXctp or an object which can be coerced to.
- as.TimeInstantDataFrame (from, ...) Converting object to TimeInstantDataFrame.

Math

Every functions defined in the Ops group (see `Ops`) can be used width a TimeInstantDataFrame and numeric :

- `tidf * 2`
- `2 * tidf`
- `2:10 == tidf`
- `2^tidf`
- `tidf^2`

Accessing to and manipulating data

The TimeInstantDataFrame class is defined to works like the data.frame class with the difference that a time instant (`POSIXct`) is attached to each rows of the data.frame. Thus to access and manipulate data of a TimeInstantDataFrame the following methods are defined : '$', '$<-', '[', '[<-', '[[', '[[<-'. See `Extract` for details.

With '[' operator, a selection by dates is also available. If 'i' and or 'j' are POSIXt or strings that can be converted to POSIXct (see below), they are considered as the minimal and maximal time limits : all data between those are selected. A string that can be converted to a POSIXct is (in this case only) a string composed of 3 parts separated by white space : 'YYYY-MM-DD HH:MM:SS tz'. The second and third parts are options, thus accepted format are :

- 'YYYY-MM-DD'
- 'YYYY-MM-DD tz'
- 'YYYY-MM-DD HH:MM:SS'
- 'YYYY-MM-DD HH:MM:SS tz'

. If timezone is not given, it is assumed to be the same as the one of the object on which the selection is done.

Other methods have been defined to allow some operations over TimeInstantDataFrame :

- rbind and merge to join two (or more) TimeInstantDataFrame (see `rbind` and `merge`),
- a TimeInstantDataFrame can be splitted exactly the same way that a data.frame can (see `split` in the base package),
- a function can be applied over each column of a TimeInstantDataFrame via the lapply function. If the function returns one value, the resulting value is a `TimeIntervalDataFrame` beginning at the first instant of the object and ending at the latest one ; else if the function returns as much values as the number of rows of the object, the TimeInstantDataFrame given in argument is returned with the new calculated values ; on others cases, a non-TimeInstantDataFrame object is returned.

Because a TimeInstantDataFrame works more or less like a data.frame, the following methods work on a TimeInstantDataFrame : `dim`, `length`, `names`, `names<-`, `ncol`, `nrow`, `row.names`, `row.names<-`.

Access to time properties

A TimeInstantDataFrame can be tested for a few time properties :

regular TRUE if all time instants are equally spaced ;

timezone gives or sets the timezone of the TimeInstantDataFrame ;

when returns a POSIXct object with the time instant of the TimeInstantDataFrame.

Graphic representation

To plot a `TimeInstantDataFrame` available functions are `plot`, `lines`, `points` and `barplot`.

These functions works more or less like their generic definition.

Text representation

To represent a `TimeInstantDataFrame` available functions are `print`, `summary`, `head`, `tail` and `show`.

See Also

`TimeIntervalDataFrame`, `SubtimeDataFrame`

Examples

```
showClass("TimeInstantDataFrame")
```

TimeIntervalDataFrame *Class* "TimeIntervalDataFrame"

Description

Class to hold time data that are NOT 'punctual'.

Usage

```
  TimeIntervalDataFrame(start, end = NULL,
    timezone = "UTC", data = NULL, period = NULL, sort=FALSE, ...)

  as.TimeIntervalDataFrame(from, ...)
  ## S3 method for class 'TimeInstantDataFrame'
as.TimeIntervalDataFrame(from, period, ...)

 RegularTimeIntervalDataFrame(from, to, by, period, timezone = "UTC", data = NULL)

  ## S4 method for signature 'TimeIntervalDataFrame'
x$name
  ## S4 replacement method for signature 'TimeIntervalDataFrame'
x$name <- value
```

```
  ## S3 method for class 'TimeIntervalDataFrame'
x[i, j, drop=FALSE]
  ## S3 replacement method for class 'TimeIntervalDataFrame'
x[i, j] <- value
  ## S4 method for signature 'TimeIntervalDataFrame'
x [[i, j, ...]]
  ## S3 replacement method for class 'TimeIntervalDataFrame'
x[[i, j]] <- value

  ## S3 method for class 'TimeIntervalDataFrame'
rbind(...)
  ## S3 method for class 'TimeIntervalDataFrame'
merge(x, y, by, all=TRUE, tz='UTC', sort=TRUE, ...)
  ## S3 method for class 'TimeIntervalDataFrame'
split(x, f, drop=FALSE, ...)
  ## S4 method for signature 'TimeIntervalDataFrame'
lapply(X, FUN, ...)

  ## S4 method for signature 'TimeIntervalDataFrame'
tapply(X, INDEX, FUN, ...,
       min.coverage=1, weights.arg=NULL, merge.X=TRUE, split.X=FALSE,
       keep.INDEX=TRUE, simplify=TRUE)
  ## S4 method for signature 'TimeIntervalDataFrame'
changeSupport(from, to,
       min.coverage, FUN=NULL, weights.arg=NULL,
       split.from=FALSE, merge.from=TRUE, ...)

  ## S4 method for signature 'TimeIntervalDataFrame'
continuous(x, ...)
  ## S4 replacement method for signature 'TimeIntervalDataFrame'
continuous(x) <- value
  ## S4 method for signature 'TimeIntervalDataFrame'
homogeneous(x, ...)
  ## S4 method for signature 'TimeIntervalDataFrame'
period(x, ...)
  ## S4 method for signature 'TimeIntervalDataFrame'
overlapping(x, ...)
  ## S4 method for signature 'TimeIntervalDataFrame'
regular(x, ...)
  ## S4 method for signature 'TimeIntervalDataFrame'
timezone(object)
  ## S4 replacement method for signature 'TimeIntervalDataFrame'
timezone(object) <- value
  ## S3 method for class 'TimeIntervalDataFrame'
start(x, ...)
  ## S3 method for class 'TimeIntervalDataFrame'
end(x, ...)
  ## S4 method for signature 'TimeIntervalDataFrame'
```

```
when(x, ...)
  ## S4 method for signature 'TimeIntervalDataFrame'
interval(x, ...)

  ## S4 method for signature 'TimeIntervalDataFrame'
dim(x)
  ## S4 method for signature 'TimeIntervalDataFrame'
length(x)
  ## S4 method for signature 'TimeIntervalDataFrame'
names(x)
  ## S4 replacement method for signature 'TimeIntervalDataFrame'
names(x) <- value
  ## S4 method for signature 'TimeIntervalDataFrame'
ncol(x)
  ## S4 method for signature 'TimeIntervalDataFrame'
nrow(x)
  ## S3 method for class 'TimeIntervalDataFrame'
row.names(x)
  ## S3 replacement method for class 'TimeIntervalDataFrame'
row.names(x) <- value

  ## S3 method for class 'TimeIntervalDataFrame'
print(x, tz=NULL, ...)
  ## S3 method for class 'TimeIntervalDataFrame'
summary(object, ...)
  ## S3 method for class 'TimeIntervalDataFrame'
head(x, tz, ...)
  ## S3 method for class 'TimeIntervalDataFrame'
tail(x, tz, ...)
  ## S4 method for signature 'TimeIntervalDataFrame'
show(object)

  ## S3 method for class 'TimeIntervalDataFrame'
plot(x, y=NULL, cursor=NULL,
    type='p', lty=1:6, lwd=1, pch=1:25, col=NULL,
    xlim=NULL, ylim=NULL, log='', main='', sub='', xlab='', ylab='',
    ann=par('ann'), axes=TRUE, asp=NA, ...)
  ## S3 method for class 'TimeIntervalDataFrame'
points(x, y=NULL, cursor=NULL, type='p',
    lty=1:6, lwd=1, pch=1:25, col=NULL, ...)
  ## S3 method for class 'TimeIntervalDataFrame'
lines(x, y=NULL, cursor=NULL, type='l',
    lty=1:6, lwd=1, pch=1:25, col=NULL, ...)
  ## S3 method for class 'TimeIntervalDataFrame'
barplot(height, format='', ...)
```

Arguments

start	POSIXct or character representing a time with a valid format (see POSIXct). It gives the beginning of each interval.
end	POSIXct or character representing a time with a valid format (see POSIXct). It gives the end of each interval. If NULL, see 'Details'.
timezone	character representing a valid timezone (see timezone).
data	a data.frame with as much rows as the length of 'start' and end, or with one row less than the length of 'start' if 'end' is NULL. Can be NULL (hence the data.frame has zero column and as much rows as needed).
period	**TimeIntervalDataFrame** if not NULL, a POSIXctp or a character that can be converted to a POSIXctp (see argument 'unit' of POSIXctp function). See Details to know how to use this argument. **as.TimeIntervalDataFrame** POSIXctp object indicating the period to add to 'when' slot of from to determine the end of the new period (the 'when' is used for the start of period) **RegularTimeIntervalDataFrame** a POSIXctp object indicating the period of each interval. If missing, it is given the value of by.
from	**as.TimeIntervalDataFrame** object to convert to a TimeIntervalDataFrame **RegularTimeIntervalDataFrame** POSIXct or character representing a time with a valid format (see POSIXct). It represents the start of the object. **changeSupport** see changeSupport
to	**RegularTimeIntervalDataFrame** POSIXct or character representing a time with a valid format (see POSIXct). It represents the end of the object. If missing, its value is deduced from 'from', 'by' and 'data'. **changeSupport** see changeSupport
by	**RegularTimeIntervalDataFrame** a POSIXctp object indicating the increment to use between the start of each interval. **merge** specifications of the columns used for merging.
x	TimeIntervalDataFrame object (to modify, to extract or to test)
i	indices specifying elements to extract or replace. (See Extract)
j	indices specifying elements to extract or replace. (See Extract)
name	A litteral character string or a name. (See Extract)
drop	Used for compatibility with data.frame methods.
value	New value for the object designated (data.frame, names, row.names, etc.).
y	TimeIntervalDataFrame to merge with x
all	logical; see merge
tz	character representing a valid timezone (see timezone).
sort	logical; if TRUE the resulting built/merged TimeIntervalDataFrame is ordered according to 'when' values.
f	a 'factor' in the sense that 'as.factor(f)' defines the grouping, or a list of such factors in which case their interaction is used for the grouping. See split.

X	**lapply** a TimeIntervalDataFrame on which the FUN must be applied. **tapply** see tapply for details
FUN	**lapply** function to apply over each columns of X. **tapply,changeSupport** see tapply for details
INDEX, min.coverage, weights.arg, merge.X, split.X, keep.INDEX, simplify, split.from, merge.from	see tapply and/or changeSupport for details.
object	TimeIntervalDataFrame object (to modify, to extract or to test)
type	plotting argument, see plot.default
lty	plotting argument, see plot.default
lwd	plotting argument, see plot.default
pch	plotting argument, see plot.default
col	plotting argument, see plot.default
xlim	plotting argument, see plot.default
ylim	plotting argument, see plot.default
log	plotting argument, see plot.default
main	plotting argument, see plot.default
sub	plotting argument, see plot.default
xlab	plotting argument, see plot.default
ylab	plotting argument, see plot.default
ann	plotting argument, see plot.default
axes	plotting argument, see plot.default
asp	plotting argument, see plot.default
cursor	To convert TimeIntervalDataFrame to a TimeInstantDataFrame before plotting (see TimeInstantDataFrame, it indicates where the TimeInstant must be taken. If 0, start of each intervals is taken as instant ; if 1 end of each intervals is taken as instant. Any other value will determine a weigthed instant between start and end (actually, value higher than 1 or lower than 0 will give instant outside this range).
height	plotting argument, see barplot
format	plotting argument, see barplot
...	More arguments.

Objects from the Class

Formally, the class consists of a data.frame and, for each row, two POSIXct that can be summarize as time interval with the POSIXcti class. This allows to manipulate at once time data without any restriction on time representation : data can occur at different time, data can be discontinuous, data can be heterogeneous (not lasting for a unique period), data can overlay each other, etc. There are several methods to test/deal/ensure that these properties are respected or not, see below.

The construction of the class allows to manipulate objects as if they were data.frame (see 'Access to data' and 'Access to data properties'). Several functions are also available to access to time properties (see 'Access to time properties').

Methods are also available to facilitate the representations of instances of that class : see 'graphic representation' and 'text representation'.

Finally, some specific methods allow to easily deal with aggregation of data over time properties (day, hour, week, special or specific time intervals).

Slots

`start`: Object of class `"POSIXct"` corresponding to the start of each row of the `data.frame`.

`end`: Object of class `"POSIXct"` corresponding to the end of each row of the `data.frame`.

`timezone`: Object of class `"character"` indicating the timezone of data both for representation and calculation.

`data`: Object of class `"data.frame"` data contained by the object.

TimeIntervalDataFrame constructors

Objects can be created by calls of the form

- `new("TimeIntervalDataFrame", ...)` ...argument must be replaced by named arguments corresponding to slots of a TimeIntervalDataFrame (see below). See also `new`.
- `TimeIntervalDataFrame(start, end=NULL, timezone='UTC', data=NULL, ...)` Arguments of the function correspond to object slots. If both `start` and `end` are given, they must have the same length. They are used to define the intervals of the object. If `data` is also given, it must have a number of rows identical to the length of `start` and `end`.

  If only `start` is given, a continuous (see `continuous`) TimeIntervalDataFrame is built. The first element of `start` is the start of the first interval, the second element is the end of the first interval and the start of the second interval. The last element of `start` is only the end of the last interval. This is why `data`, if given, must be one row shorter than `start`.

  If period is given it must be a `POSIXctp` object (or a valid character) and 'start' and 'end' must have length equal to 1. In that case, a TimeIntervalDataFrame will be created with start date equal to start 'floored' by the unit of 'period', end date 'ceiled' by the unit of 'period' and with enough intervals of 'period' length to fit. If 'data' given, it must have a number of rows equal to the number of intervals calculated.
- `RegularTimeIntervalDataFrame(from, to, by, period, timezone='UTC', data=NULL)` Wrapper to construct TimeIntervalDataFrame with specific properties (see details of each argument).
- `as.TimeIntervalDataFrame(from, ...)` Converting object to TimeIntervalDataFrame.

Math

Every functions defined in the Ops group (see `Ops`) can be used width a TimeIntervalDataFrame and numeric :

- `tidf * 2`
- `2 * tidf`

- `2:10 == tidf`
- `2^tidf`
- `tidf^2`

Accessing to and manipulating data

The TimeIntervalDataFrame class is defined to works like the data.frame class with the difference that a time interval (`POSIXcti`) is attached to each rows of the data.frame. Thus to access and manipulate data of a TimeIntervalDataFrame the following methods are defined : '$', '$<-', '[', '[<-', '[[', '[[<-'. See `Extract` for details.

With '[' operator, a selection by dates is also available. If 'i' and or 'j' are POSIXt or strings that can be converted to POSIXct (see below), they are considered as the minimal and maximal time limits : all data between those are selected. A string that can be converted to a POSIXct is (in this case only) a string composed of 3 parts separated by white space : 'YYYY-MM-DD HH:MM:SS tz'. The second and third parts are options, thus accepted format are :

- 'YYYY-MM-DD'
- 'YYYY-MM-DD tz'
- 'YYYY-MM-DD HH:MM:SS'
- 'YYYY-MM-DD HH:MM:SS tz'

. If timezone is not given, it is assumed to be the same as the one of the object on which the selection is done.

Other methods have been defined to allow some operations on TimeIntervalDataFrame :

- rbind and merge to join two (or more) TimeIntervalDataFrame (see `rbind` and `merge`),
- a TimeIntervalDataFrame can be splitted exactly the same way that a data.frame can (see `split` in the base package) and some more possibilities have been defined (see `split` in the timetools package),
- a function can be applied over each column of a TimeIntervalDataFrame via the lapply function. If the function returns one value, the resulting value is a `TimeIntervalDataFrame` beginning at the first instant of the object and ending at the latest one ; else if the function returns as much values as the number of rows of the object, the TimeIntervalDataFrame given in argument is returned with the new calculated values ; on others cases, a non-TimeInterevalDataFrame object is returned.
- 'tapply' can split a TimeIntervalDataFrame and then apply a function over each group (see `tapply`),
- 'changeSupport' act like the 'tapply' function but with a different default behaviour (see `changeSupport`).

Because a TimeIntervalDataFrame works more or less like a data.frame, the following methods work on a TimeIntervalDataFrame : `dim`, `length`, `names`, `names<-`, `ncol`, `nrow`, `row.names`, `row.names<-`.

Access/modify to time properties

A TimeIntervamDataFrame can be tested for a few time properties :

continuous see `continuous` ;

homogeneous see `homogeneous` ;

period see `period` ;

overlapping see `overlapping` ;

regular TRUE if all time intervals are equally spaced ;

timezone gives or sets the timezone of the TimeIntervalDataFrame ;

start returns a POSIXct object with the start time of each intervals ;

end returns a POSIXct object with the end time of each intervals ;

when returns a POSIXcti, i.e. the intervals of the object ;

interval returns a POSIXcti, i.e. the intervals of the object.

Graphic representation

To plot a `TimeIntervalDataFrame` available functions are `plot`, `lines`, `points` and `barplot`.

These functions works more or less like their generic definition.

Text representation

To represent a `TimeIntervalDataFrame` available functions are `print`, `summary`, `head`, `tail` and `show`.

See Also

`TimeInstantDataFrame`, `SubtimeDataFrame`, `POSIXcti`, `POSIXctp`

Examples

```
showClass("TimeIntervalDataFrame")
```

`timezone` *Get or set timezone property*

Description

get or set the timezone of the time object (see `timezone` in the base package).

Usage

```
timezone(object)
timezone(object) <- value
```

Arguments

`object`	object to get or set timezone property.
`value`	specify the new value for timezone. See timezone in the base package.

Details

Changing the timezone of an object consist in reprojecting time coordinates from a system of reference to another. That is to say that not only the 'timezone' attribute is changed : for instance '2012-02-01 14:OO UTC' will be changed in '2012-02-01 15:00 CET' if 'timezone' is set to 'CET' (French local time).

`unit` *define valid units for time objects/retrieve-set time unit of a time object*

Description

The timetools package use a set of valid time units which are roughly : year, month, week, day, hour, minute, second. They can be combined in `subtime objects`. For instance : month of year, minute of day, minute of week, etc.

Usage

```
POSIXt.units(x = NULL, ...)

unit(x, ...)
unit(object) <- value
of(x, ...)
```

Arguments

`x`	a character string representing the needed units for `POSIXt.units`. The object from which the time unit is to retrieve.
`object`	POSIXctp to which the unit is to be changed
`value`	a character or a `POSIXt.units` indicating the new units of object.
`...`	arguments to or from other methods

POSIXt.units(x = NULL, ...)

With no argument, the function return a factor containing the valid time units. With an argument, it returns the units asked for.

unit(x, ...)

Return the time unit of the object. In case 'x' is a `POSIXst`, the unit is the 'left' part of its unit : if 'x' is a 'minute of day', 'unit' will return 'minute'.

of(x, ...)

For POSIXst only, it return the 'right' part of the unit of 'x' : if 'x' is a 'minute of day', 'of' will return 'day'.

`when` *Retrieve the 'timestamp' of a Time*DataFrame*

Description

For Time objects.

Usage

```
when(x, ...)
```

Arguments

`x`	object from which get the timestamp
`...`	arguments to or from other methods

Value

If TimeInstantDataFrame, return the instants of the object ;

if TimeIntervalDataFrame, return the intervals of the object.

if SubtimeDataFrame, return the POSIXst of the object.

See Also

TimeIntervalDataFrame, POSIXcti, TimeInstantDataFrame, POSIXct, SubtimeDataFrame, POSIXst

---

`%included%` *test inclusion of 2 'POSIXcti' objects*

Description

This function test if the first 'POSIXcti' object is included in the second.

Usage

```
i1 %included% i2
```

Arguments

i1	is this object included in the second object ?
i2	is this object include the first one ?

Value

boolean

Examples

```
# to see all existing methods :
methods ('\%included\%')
```

%intersect% *intersects 2 'POSIXcti' objects*

Description

This function allows to find the intersection of two objects of the same class.

Usage

```
i1 %intersect% i2
```

Arguments

i1	first object to intersect
i2	second object to intersect

Value

object of the same class of parameters

Examples

```
# to see all existing methods :
methods ('\%intersect\%')
```

Index