

Package ‘timechange’

November 4, 2022

Title Efficient Manipulation of Date-Times

Version 0.1.1

Description Efficient routines for manipulation of date-time objects while accounting for time-zones and daylight saving times. The package includes utilities for updating of date-time components (year, month, day etc.), modification of time-zones, rounding of date-times, period addition and subtraction etc. Parts of the 'CCTZ' source code, released under the Apache 2.0 License, are included in this package. See <https://github.com/google/cctz> for more details.

Depends R (>= 3.3)

License GPL-3

Encoding UTF-8

LinkingTo cpp11 (>= 0.2.7)

Suggests testthat, knitr

SystemRequirements A system with zoneinfo data (e.g. /usr/share/zoneinfo) as well as a recent-enough C++11 compiler (such as g++-4.8 or later). On Windows the zoneinfo included with R is used.

BugReports <https://github.com/vspinu/timechange/issues>

URL <https://github.com/vspinu/timechange/>

RoxygenNote 7.2.1

NeedsCompilation yes

Author Vitalie Spinu [aut, cre],
Google Inc. [ctb, cph]

Maintainer Vitalie Spinu <spinuvit@gmail.com>

Repository CRAN

Date/Publication 2022-11-04 15:30:02 UTC

R topics documented:

timechange-package	2
time-zones	2
time_add	4
time_get	7
time_round	8
time_update	12

Index	15
--------------	-----------

timechange-package	<i>Package timechange</i>
--------------------	---------------------------

Description

Utilities for efficient updating of date-times components while accounting for time-zones and day-light saving times. When it makes sense functions provide a refined control of what happens in ambiguous situations through `roll_month` and `roll_dst` arguments.

Author(s)

Vitalie Spinu (<spinuvit@gmail.com>)

See Also

Useful links:

- <https://github.com/vspinu/timechange/>
- Report bugs at <https://github.com/vspinu/timechange/issues>

time-zones	<i>Time-zone manipulation</i>
------------	-------------------------------

Description

`time_at_tz` returns a date-time as it would appear in a different time zone. The actual moment of time measured does not change, just the time zone it is measured in. `time_at_tz` defaults to the Universal Coordinated time zone (UTC) when an unrecognized time zone is supplied.

`time_force_tz` returns the date-time that has the same clock time as input time, but in the new time zone. Although the new date-time has the same clock time (e.g. the same values in the seconds, minutes, hours, etc.) it is a different moment of time than the input date-time. Computation is vectorized over both `time` and `tz` arguments.

`time_clock_at_tz` retrieves day clock time in specified time zones. Computation is vectorized over both `dt` and `tz` arguments, `tz` defaults to the `timezone` of `time`.

Usage

```
time_at_tz(time, tz = "UTC")

time_force_tz(
  time,
  tz = "UTC",
  tzout = tz[[1]],
  roll_dst = c("boundary", "post")
)

time_clock_at_tz(time, tz = NULL, units = "secs")
```

Arguments

time	a date-time object (POSIXct, POSIXlt, Date) or a list of date-time objects. When a list, all contained elements are updated the new list is returned.
tz	a character string containing the time zone to convert to. R must recognize the name contained in the string as a time zone on your system. For <code>time_force_tz</code> and <code>time_clock_at_tzs</code> , <code>tz</code> can be a vector of heterogeneous time-zones, in which case <code>time</code> and <code>tz</code> arguments are paired. If <code>time</code> and <code>tz</code> lengths differ, the smaller one is recycled according with usual R conventions.
tzout	timezone of the output date-time vector. Meaningful only when <code>tz</code> argument is a vector of heterogenous time-zones. This argument is necessary because R date-time vectors cannot hold elements with different time-zones.
roll_dst	same as in <code>time_add</code> which see.
units	passed directly to <code>as.difftime()</code> .

Value

a POSIXct object with the updated time zone

Examples

```
x <- as.POSIXct("2009-08-07 00:00:00", tz = "America/New_York")
time_at_tz(x, "UTC")
time_force_tz(x, "UTC")
time_force_tz(x, "Europe/Amsterdam")

## DST skip:

y <- as.POSIXct("2010-03-14 02:05:05", tz = "UTC")
time_force_tz(y, "America/New_York", roll = "boundary")
time_force_tz(y, "America/New_York", roll = "post")
time_force_tz(y, "America/New_York", roll = "pre")
time_force_tz(y, "America/New_York", roll = "NA")
```

```
## DST skipped and repeated

y <- as.POSIXct(c("2010-03-14 02:05:05 UTC", "2014-11-02 01:35:00"), tz = "UTC")
time_force_tz(y, "America/New_York", roll_dst = c("NA", "pre"))
time_force_tz(y, "America/New_York", roll_dst = c("boundary", "post"))

## Heterogeneous time-zones:

x <- as.POSIXct(c("2009-08-07 00:00:01", "2009-08-07 01:02:03"), tz = "UTC")
time_force_tz(x, tz = c("America/New_York", "Europe/Amsterdam"))
time_force_tz(x, tz = c("America/New_York", "Europe/Amsterdam"), tzout = "America/New_York")

x <- as.POSIXct("2009-08-07 00:00:01", tz = "UTC")
time_force_tz(x, tz = c("America/New_York", "Europe/Amsterdam"))

## Local clock:
x <- as.POSIXct(c("2009-08-07 01:02:03", "2009-08-07 10:20:30"), tz = "UTC")
time_clock_at_tz(x, units = "secs")
time_clock_at_tz(x, units = "hours")
time_clock_at_tz(x, "Europe/Amsterdam")

x <- as.POSIXct("2009-08-07 01:02:03", tz = "UTC")
time_clock_at_tz(x, tz = c("America/New_York", "Europe/Amsterdam", "Asia/Shanghai"), unit = "hours")
```

time_add

Arithmetics with periods

Description

Add periods to date-time objects. Periods track the change in the "clock time" between two civil times. They are measured in common civil time units: years, months, days, hours, minutes, and seconds.

Usage

```
time_add(
  time,
  periods = NULL,
  year = NULL,
  month = NULL,
  week = NULL,
  day = NULL,
  hour = NULL,
  minute = NULL,
  second = NULL,
  roll_month = "preday",
  roll_dst = c("post", "pre"),
  ...
)
```

```

time_subtract(
  time,
  periods = NULL,
  year = NULL,
  month = NULL,
  week = NULL,
  day = NULL,
  hour = NULL,
  minute = NULL,
  second = NULL,
  roll_month = "preday",
  roll_dst = c("pre", "post"),
  ...
)

```

Arguments

time	date-time object
periods	string of units to add/subtract (not yet implemented) or a named list of the form <code>list(year = 1, month = 2, ...)</code> .
year, month, week, day, hour, minute, second	Units to be added to time. Units except for seconds are converted to integer values. Components are replicated according to <code>vctrs</code> semantics, i.e vectors must be either of length 1 or same length as <code>time</code> vector.
roll_month	controls how addition of months and years behaves when standard arithmetic rules exceed limits of the resulting date's month. Possible values are "preday", "boundary", "postday", "full" and "NA". See "Details" or <code>[(timechange::time_add())]</code> for further details.
roll_dst	is a string vector of length one or two. When two values are supplied they specify how to roll date-times when they fall into "skipped" and "repeated" DST transitions respectively. Singleton strings is replicated to the length of two. Possible values are: <ul style="list-style-type: none"> * <code>`pre`</code> - Use the time before the transition boundary. * <code>`boundary`</code> - Use the time exactly at the boundary transition. * <code>`post`</code> - Use the time after the boundary transition. * <code>`NA`</code> - Produce NAs when the resulting time falls inside the problematic interval. <p>For example <code>roll_dst = c("pre", "NA")</code> indicates that for repeated intervals return the time in the earlier interval and for skipped intervals return NA.</p>
...	deprecated

Details

Arithmetic operations with multiple period units (years, months etc) are applied in decreasing size order, from year to second. Thus `time_add(x, month = 1, days = 3)` first adds 1 month to `x`, then adds to the resulting date 3 days.

Generally period arithmetic is undefined due to the irregular nature of civil time and complexities with DST transitions. **'timechange'** allows for a refined control of what happens when an addition of irregular periods (years, months, days) results in "unclear" date.

Let's start with an example. What happens when you add "1 month 3 days" to "2000-01-31 01:02:03"? **'timechange'** operates by applying larger periods first. First months are added 1 + 1 = February which results in non-existent time of 2000-02-31 01:02:03. Here the roll_month adjustment kicks in. After the adjustment, the remaining 3 days are added.

roll_month can be one of the following:

- boundary - if rolling over a month boundary occurred due to setting units smaller than month, the date is adjusted to the beginning of the month (the boundary). For example, 2000-01-31 01:02:03 + 1 month = 2000-02-01 01:02:03.
- preday - roll back to the last valid day of the previous month (pre-boundary day) preserving the H, M, S units. For example, 2000-01-31 01:02:03 + 1 month = 2000-02-28 01:02:03. This is the default.
- postday - roll to the first day post-boundary preserving the H, M, S units. For example, 2000-01-31 01:02:03 + 1 month = 2000-03-01 01:02:03.
- full - full rolling. No adjustment is done to the simple arithmetic operations (the gap is skipped as if it's not there). For example, 2000-01-31 01:02:03 + 1 month + 3 days is equivalent to 2000-01-01 01:02:03 + 1 month + 31 days + 3 days resulting in 2000-03-05 01:02:03.
- NA - if end result was rolled over the month boundary due to addition of units smaller than month (day, hour, minute, second) produce NA.
- NAym - if intermediate date resulting from first adding years and months ends in a non-existing date (e.g. Feb 31) produce NA. This is how period addition in lubridate works for historical reasons.

Examples

```
# Addition

## Month gap
x <- as.POSIXct("2000-01-31 01:02:03", tz = "America/Chicago")
time_add(x, month = 1, roll_month = "postday")
time_add(x, month = 1, roll_month = "preday")
time_add(x, month = 1, roll_month = "boundary")
time_add(x, month = 1, roll_month = "full")
time_add(x, month = 1, roll_month = "NA")
time_add(x, month = 1, day = 3, roll_month = "postday")
time_add(x, month = 1, day = 3, roll_month = "preday")
time_add(x, month = 1, day = 3, roll_month = "boundary")
time_add(x, month = 1, day = 3, roll_month = "full")
time_add(x, month = 1, day = 3, roll_month = "NA")

## DST gap
x <- as.POSIXlt("2010-03-14 01:02:03", tz = "America/Chicago")
time_add(x, hour = 1, minute = 50, roll_dst = "pre")
time_add(x, hour = 1, minute = 50, roll_dst = "boundary")
time_add(x, hour = 1, minute = 50, roll_dst = "post")
```

```
##' time_add(x, hours = 1, minutes = 50, roll_dst = "NA")

# SUBTRACTION

## Month gap
x <- as.POSIXct("2000-03-31 01:02:03", tz = "America/Chicago")
time_subtract(x, month = 1, roll_month = "postday")
time_subtract(x, month = 1, roll_month = "preday")
time_subtract(x, month = 1, roll_month = "boundary")
time_subtract(x, month = 1, roll_month = "full")
time_subtract(x, month = 1, roll_month = "NA")
time_subtract(x, month = 1, day = 0, roll_month = "postday")
time_subtract(x, month = 1, day = 3, roll_month = "postday")
time_subtract(x, month = 1, day = 0, roll_month = "preday")
time_subtract(x, month = 1, day = 3, roll_month = "preday")
time_subtract(x, month = 1, day = 3, roll_month = "boundary")
time_subtract(x, month = 1, day = 3, roll_month = "full")
time_subtract(x, month = 1, day = 3, roll_month = "NA")

## DST gap
y <- as.POSIXlt("2010-03-15 01:02:03", tz = "America/Chicago")
time_subtract(y, hour = 22, minute = 50, roll_dst = "pre")
time_subtract(y, hour = 22, minute = 50, roll_dst = "boundary")
time_subtract(y, hour = 22, minute = 50, roll_dst = "post")
time_subtract(y, hour = 22, minute = 50, roll_dst = "NA")
```

time_get

*Get components of a date-time object***Description**

Get components of a date-time object

Usage

```
time_get(
  time,
  components = c("year", "month", "yday", "mday", "wday", "hour", "minute", "second"),
  week_start = getOption("timechange.week_start", 1)
)
```

Arguments

time	a date-time object
components	a character vector of components to return. Component is one of "year", "month", "yday", "day", "mday", "wday", "hour", "minute", "second" where "day" is the same as "mday".
week_start	week starting day (Default is 1, Monday). Set timechange.week_start option to change this globally.

Value

A data.frame of the requested components

Examples

```
x <- as.POSIXct("2019-02-03")
time_get(x)
```

time_round	<i>Round, floor and ceiling for date-time objects</i>
------------	---

Description

timechange provides rounding to the nearest unit or multiple of a unit with fractional support whenever makes sense. Units can be specified flexibly as strings. All common abbreviations are supported - secs, min, mins, 2 minutes, 3 years, 2s, 1d etc.

Usage

```
time_round(
  time,
  unit = "second",
  week_start = getOption("timechange.week_start", 1)
)

time_floor(
  time,
  unit = "seconds",
  week_start = getOption("timechange.week_start", 1)
)

time_ceiling(
  time,
  unit = "seconds",
  change_on_boundary = inherits(time, "Date"),
  week_start = getOption("timechange.week_start", 1)
)
```

Arguments

time	a date-time vector (Date, POSIXct or POSIXlt)
unit	a character string specifying a time unit or a multiple of a unit. Valid base periods for civil time rounding are second, minute, hour, day, week, month, bimonth, quarter, season, halfyear and year. The only unit for absolute time rounding is asecond. Other absolute units can be achieved with multiples of asecond ("60a", "3600a" etc). See "Details" and examples. Arbitrary unique English abbreviations are allowed. One letter abbreviations follow strptime

formats "y", "m", "d", "M", "H", "S". Multi-unit rounding of weeks is currently not supported.

Rounding for a unit is performed from the parent's unit origin. For example when rounding to seconds origin is start of the minute. When rounding to days, origin is first date of the month. See examples.

With fractional sub-unit ($unit < 1$) rounding with child unit is performed instead. For example $0.5mins == 30secs$, $.2hours == 12min$ etc.

Please note that for fractions which don't match exactly to integer number of the child units only the integer part is used for computation. For example $.7days = 16.8hours$ will use 16 hours during the computation.

`week_start` When unit is weeks, this is the first day of the week. Defaults to 1 (Monday).

`change_on_boundary`

If NULL (the default) don't change instants on the boundary (`time_ceiling(ymd_hms('2000-01-01 00:00:00'))` is `2000-01-01 00:00:00`), but round up Date objects to the next boundary (`time_ceiling(ymd("2000-01-01"), "month")` is `"2000-02-01"`). When TRUE, instants on the boundary are rounded up to the next boundary. When FALSE, date-time on the boundary are never rounded up (this was the default for **lubridate** prior to v1.6.0. See section Rounding Up Date Objects below for more details.

Value

An object of the same class as the input object. When input is a Date object and unit is smaller than day a POSIXct object is returned.

Civil Time vs Absolute Time rounding

Rounding in civil time is done on actual clock time (ymdHMS) and is affected by civil time irregularities like DST. One important characteristic of civil time rounding is that floor (ceiling) does not produce civil times that are bigger (smaller) than the rounded civil time.

Absolute time rounding (with `aseconds`) is done on the absolute time (number of seconds since origin), Thus, rounding of `aseconds` allows for fractional seconds and multi-units larger than 60. See examples of rounding around DST transition where rounding in civil time does not give same result as rounding with the corresponding `X aseconds`.

Please note that absolute rounding to fractions smaller than 1ms will result to large precision errors due to the floating point representation of the POSIXct objects.

Note on `time_round()`

For rounding date-times which is exactly halfway between two consecutive units, the convention is to round up. Note that this is in line with the behavior of R's `base::round.POSIXt()` function but does not follow the convention of the `base::round()` function which "rounds to the even digit" per IEC 60559.

Ceiling of Date objects

By default rounding up Date objects follows 3 steps:

1. Convert to an instant representing lower bound of the Date: 2000-01-01 → 2000-01-01 00:00:00
2. Round up to the **next** closest rounding unit boundary. For example, if the rounding unit is month then next closest boundary of 2000-01-01 is 2000-02-01 00:00:00.
The motivation for this is that the "partial" 2000-01-01 is conceptually an interval (2000-01-01 00:00:00 – 2000-01-02 00:00:00) and the day hasn't started clocking yet at the exact boundary 00:00:00. Thus, it seems wrong to round up a day to its lower boundary.
The behavior on the boundary can be changed by setting `change_on_boundary` to a non-NULL value.
3. If rounding unit is smaller than a day, return the instant from step 2 (POSIXct), otherwise convert to and return a Date object.

See Also

[base::round\(\)](#)

Examples

```
## print fractional seconds
options(digits.secs=6)

x <- as.POSIXct("2009-08-03 12:01:59.23")
time_round(x, ".5 asec")
time_round(x, "sec")
time_round(x, "second")
time_round(x, "asecond")
time_round(x, "minute")
time_round(x, "5 mins")
time_round(x, "5M") # "M" for minute "m" for month
time_round(x, "hour")
time_round(x, "2 hours")
time_round(x, "2H")
time_round(x, "day")
time_round(x, "week")
time_round(x, "month")
time_round(x, "bimonth")
time_round(x, "quarter") == time_round(x, "3 months")
time_round(x, "halfyear")
time_round(x, "year")

x <- as.POSIXct("2009-08-03 12:01:59.23")
time_floor(x, ".1 asec")
time_floor(x, "second")
time_floor(x, "minute")
time_floor(x, "M")
time_floor(x, "hour")
time_floor(x, "day")
time_floor(x, "week")
time_floor(x, "m")
time_floor(x, "month")
time_floor(x, "bimonth")
```

```

time_floor(x, "quarter")
time_floor(x, "season")
time_floor(x, "halfyear")
time_floor(x, "year")

x <- as.POSIXct("2009-08-03 12:01:59.23")
time_ceiling(x, ".1 asec")
time_ceiling(x, "second")
time_ceiling(x, "minute")
time_ceiling(x, "5 mins")
time_ceiling(x, "hour")
time_ceiling(x, "day")
time_ceiling(x, "week")
time_ceiling(x, "month")
time_ceiling(x, "bimonth") == time_ceiling(x, "2 months")
time_ceiling(x, "quarter")
time_ceiling(x, "season")
time_ceiling(x, "halfyear")
time_ceiling(x, "year")

## behavior on the boundary
x <- as.Date("2000-01-01")
time_ceiling(x, "month")
time_ceiling(x, "month", change_on_boundary = FALSE)

## As of R 3.4.2 POSIXct printing of fractional numbers is wrong
as.POSIXct("2009-08-03 12:01:59.3", tz = "UTC") ## -> "2009-08-03 12:01:59.2 UTC"
time_ceiling(x, ".1 asec") ## -> "2009-08-03 12:01:59.2 UTC"

## Civil Time vs Absolute Time Rounding

# "2014-11-02 01:59:59.5 EDT" before 1h backroll at 2AM
x <- .POSIXct(1414907999.5, tz = "America/New_York")
x
time_ceiling(x, "hour") # "2014-11-02 02:00:00 EST"
time_ceiling(x, "minute")
time_ceiling(x, "sec")
difftime(time_ceiling(x, "s"), x)
time_ceiling(x, "1a") # "2014-11-02 01:00:00 EST"
difftime(time_ceiling(x, "a"), x)

# "2014-11-02 01:00:00.5 EST" after 1h backroll at 2AM
x <- .POSIXct(1414908000.5, tz = "America/New_York")
x
time_floor(x, "hour") # "2014-11-02 01:00:00 EST"
difftime(time_floor(x, "hour"), x)
time_floor(x, "3600a") # "2014-11-02 01:00:00 EST" - 25m
difftime(time_floor(x, "a"), x)

## behavior on the boundary when rounding multi-units

x <- as.POSIXct("2009-08-28 22:56:59.23", tz = "UTC")
time_ceiling(x, "3.4 secs") # "2009-08-28 22:57:03.4"

```

```

time_ceiling(x, "50.5 secs") # "2009-08-28 22:57:50.5"
time_ceiling(x, "57 min") # "2009-08-28 22:57:00"
time_ceiling(x, "56 min") # "2009-08-28 23:56:00"
time_ceiling(x, "7h") # "2009-08-29 07:00:00"
time_ceiling(x, "7d") # "2009-08-29 00:00:00"
time_ceiling(x, "8d") # "2009-09-09 00:00:00"
time_ceiling(x, "8m") # "2009-09-01 00:00:00"
time_ceiling(x, "6m") # "2010-01-01 00:00:00"
time_ceiling(x, "7m") # "2010-08-01 00:00:00"

x <- as.POSIXct("2010-11-25 22:56:57", tz = "UTC")
time_ceiling(x, "6sec") # "2010-11-25 22:57:00"
time_ceiling(x, "60sec") # "2010-11-25 22:57:00"
time_ceiling(x, "6min") # "2010-11-25 23:00:00"
time_ceiling(x, "60min") # "2010-11-25 23:00:00"
time_ceiling(x, "4h") # "2010-11-26 00:00:00"
time_ceiling(x, "15d") # "2010-12-01 00:00:00"
time_ceiling(x, "15d") # "2010-12-01 00:00:00"
time_ceiling(x, "6m") # "2011-01-01 00:00:00"

```

time_update

Update components of a date-time object

Description

Update components of a date-time object

Usage

```

time_update(
  time,
  updates = NULL,
  year = NULL,
  month = NULL,
  yday = NULL,
  mday = NULL,
  wday = NULL,
  hour = NULL,
  minute = NULL,
  second = NULL,
  tz = NULL,
  roll_month = "preday",
  roll_dst = c("boundary", "post"),
  week_start = getOption("timechange.week_start", 1),
  exact = FALSE
)

```

Arguments

time	a date-time object
updates	a named list of components
year, month, yday, wday, mday, hour, minute, second	components of the date-time to be updated. All components except second will be converted to integer. Components are replicated according to <code>vctrs</code> semantics, i.e. vectors must be either of length 1 or same length as <code>time</code> vector.
tz	time zone component (a singleton character vector)
roll_month	controls how addition of months and years behaves when standard arithmetic rules exceed limits of the resulting date's month. Possible values are "preday", "boundary", "postday", "full" and "NA". See "Details" or <code>[(timechange::time_add())]</code> for further details.
roll_dst	is a string vector of length one or two. When two values are supplied they specify how to roll date-times when they fall into "skipped" and "repeated" DST transitions respectively. Singleton strings is replicated to the length of two. Possible values are: <ul style="list-style-type: none"> * <code>`pre`</code> - Use the time before the transition boundary. * <code>`boundary`</code> - Use the time exactly at the boundary transition. * <code>`post`</code> - Use the time after the boundary transition. * <code>`NA`</code> - Produce NAs when the resulting time falls inside the problematic interval. For example <code>roll_dst = c("pre", "NA")</code> indicates that for repeated intervals return the time in the earlier interval and for skipped intervals return NA.
week_start	first day of the week (default is 1, Monday). Set <code>timechange.week_start</code> option to change this globally.
exact	logical (TRUE), whether the update should be exact. If set to FALSE no rolling or unit-recycling is allowed and NA is produced whenever the units of the end date-time don't match the provided units. This can occur when an end date falls into a gap (e.g. DST or Feb.29) or when large components (e.g. <code>hour = 25</code>) are supplied and result in crossing boundaries of higher units. When <code>exact = TRUE</code> , <code>roll_month</code> and <code>roll_dst</code> arguments are ignored.

Value

A date-time with the requested elements updated. Retain its original class unless the original class is `Date` and at least one of the hour, minute, second or `tz` is supplied, in which case a `POSIXct` object is returned.

See Also

`[time_add()]`

Examples

```
date <- as.Date("2009-02-10")
time_update(date, year = 2010, month = 1, mday = 1)
```

```
time_update(date, year = 2010, month = 13, mday = 1)
time_update(date, minute = 10, second = 3)
time_update(date, minute = 10, second = 3, tz = "America/New_York")

time <- as.POSIXct("2015-02-03 01:02:03", tz = "America/New_York")
time_update(time, month = 2, mday = 31, roll_month = "preday")
time_update(time, month = 2, mday = 31, roll_month = "boundary")
time_update(time, month = 2, mday = 31, roll_month = "postday")
time_update(time, month = 2, mday = 31, exact = TRUE)
time_update(time, month = 2, mday = 31, exact = FALSE)

## DST skipped
time <- as.POSIXct("2015-02-03 01:02:03", tz = "America/New_York")
time_update(time, year = 2016, yday = 10)
time_update(time, year = 2016, yday = 10, tz = "Europe/Amsterdam")
time_update(time, second = 30, tz = "America/New_York")
```

Index

`as.difftime()`, [3](#)

`base::round()`, [9](#), [10](#)

`base::round.POSIXt()`, [9](#)

`time-zones`, [2](#)

`time_add`, [4](#)

`time_at_tz (time-zones)`, [2](#)

`time_ceiling (time_round)`, [8](#)

`time_clock_at_tz (time-zones)`, [2](#)

`time_floor (time_round)`, [8](#)

`time_force_tz (time-zones)`, [2](#)

`time_get`, [7](#)

`time_round`, [8](#)

`time_subtract (time_add)`, [4](#)

`time_update`, [12](#)

`timechange (timechange-package)`, [2](#)

`timechange-package`, [2](#)