Package 'datetimeoffset'

March 24, 2025

Type Package

Title Datetimes with Optional UTC Offsets and/or Heterogeneous Time Zones

Version 1.0.0

Description Supports import/export for a number of datetime string standards and R datetime classes often including lossless re-export of any original reduced precision including 'ISO 8601' <https://en.wikipedia.org/wiki/ISO_8601> and 'pdfmark' <https://opensource.adobe.com/dc-acrobat-sdk-docs/library/pdfmark/> datetime strings. Supports local/global datetimes with optional UTC offsets and/or (possibly heterogeneous) time zones with up to nanosecond precision.

URL https://trevorldavis.com/R/datetimeoffset/, https://github.com/trevorld/r-datetimeoffset

BugReports https://github.com/trevorld/r-datetimeoffset/issues

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.1

Depends R (>= 3.4.0)

Imports clock (>= 0.7.3), methods, purrr (>= 1.0.0), vctrs (>= 0.5.0)

Suggests lubridate (>= 1.9.0), nanotime (>= 0.3.11), knitr, parttime, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr, rmarkdown

Config/testthat/edition 3

NeedsCompilation no

Author Trevor L. Davis [aut, cre] (<https://orcid.org/0000-0001-6341-4639>)

Maintainer Trevor L. Davis <trevor.l.davis@gmail.com>

Repository CRAN

Date/Publication 2025-03-24 16:50:02 UTC

Contents

`as_datetimeoffset` *Coerce to "datetimeoffset" objects*

Description

`as_datetimeoffset()` coerces to `datetimeoffset()` objects.

Usage

```
as_datetimeoffset(x, ...)

## S3 method for class 'datetimeoffset'
as_datetimeoffset(x, ...)

## S3 method for class 'Date'
as_datetimeoffset(x, tz = NA_character_, ...)

## Default S3 method:
as_datetimeoffset(x, ...)

## S3 method for class 'integer'
as_datetimeoffset(x, ...)

## S3 method for class 'numeric'
as_datetimeoffset(x, ...)

## S3 method for class 'POSIXt'
```

```
as_datetimeoffset(x, ...)

## S3 method for class 'character'
as_datetimeoffset(x, tz = NA_character_, ...)

## S3 method for class 'nanotime'
as_datetimeoffset(x, tz = "GMT", ...)

## S3 method for class 'partial_time'
as_datetimeoffset(x, ...)

## S3 method for class 'clock_year_month_day'
as_datetimeoffset(x, ...)

## S3 method for class 'clock_year_month_weekday'
as_datetimeoffset(x, ...)

## S3 method for class 'clock_iso_year_week_day'
as_datetimeoffset(x, ...)

## S3 method for class 'clock_year_quarter_day'
as_datetimeoffset(x, ...)

## S3 method for class 'clock_year_day'
as_datetimeoffset(x, ...)

## S3 method for class 'clock_naive_time'
as_datetimeoffset(x, ...)

## S3 method for class 'clock_sys_time'
as_datetimeoffset(x, ...)

## S3 method for class 'clock_zoned_time'
as_datetimeoffset(x, ...)
```

Arguments

x	An R object that can reasonably be coerced to a `datetimeoffset()` object such as a string in pdfmark date or ISO 8601 datetime formats or something with an `as.POSIXct()` method.
...	Further arguments to certain methods.
tz	Time zone to use for the conversion. Ignored by `as_datetimeoffset.Date()`. Generally need not be a single value.

Value

A `datetimeoffset()` vector

Examples

```
# ISO 8601 examples
as_datetimeoffset("2020-05-15")
as_datetimeoffset("20200515")
as_datetimeoffset("2020-05-15T08:23:16")
as_datetimeoffset("20200515T082316")
as_datetimeoffset("2020-05-15T08:23:16.003Z")
as_datetimeoffset("20200515T082316Z")
as_datetimeoffset("2020-05-15T08:23:16+03:30")
as_datetimeoffset("20200515T082316+0330")

# Misc supported `as.POSIXlt()` `tryFormats` examples
as_datetimeoffset("2020/05/15 08:23:16")

# pdfmark datetime examples
as_datetimeoffset("D:20200515")
as_datetimeoffset("D:20200515082316")
as_datetimeoffset("D:20200515082316+03'30'")

as_datetimeoffset(Sys.time())
```

---

datetimeoffset *Datetime object with optional UTC offsets and/or timezones*

Description

`datetimeoffset()` creates a datetime with possible UTC offset object. It can be used to represent datetimes with possible UTC offsets (without necessarily any knowledge of the time zone).

Usage

```
datetimeoffset(
  year = NA_integer_,
  month = NA_integer_,
  day = NA_integer_,
  hour = NA_integer_,
  minute = NA_integer_,
  second = NA_integer_,
  nanosecond = NA_integer_,
  subsecond_digits = NA_integer_,
  hour_offset = NA_integer_,
  minute_offset = NA_integer_,
  tz = NA_character_
)
```

Arguments

year	Year (integer, optional)
month	Month (integer, optional)
day	Day (integer, optional)
hour	Hour (integer, optional)
minute	Minute (integer, optional)
second	Second (integer, optional)
nanosecond	Nanosecond (integer, optional)
subsecond_digits	Number of digits used by fractional seconds (integer, optional)
hour_offset	UTC offset in hours (integer, optional)
minute_offset	UTC offset in minutes (integer, optional). Will be coerced to a non-negative value.
tz	Time zone (character, optional)

Value

A vctrs record with class datetimeoffset.

Examples

```
datetimeoffset(2020)
datetimeoffset(2020, 5)
datetimeoffset(2020, 5, 15)
datetimeoffset(2020, 5, 15, 8)
datetimeoffset(2020, 5, 15, 8, 23)
datetimeoffset(2020, 5, 15, 8, 23, 16) # local time with unknown timezone
if ("US/Pacific" %in% OlsonNames())
  datetimeoffset(2020, 5, 15, 8, 23, 16, tz = "US/Pacific")
datetimeoffset(2020, 5, 15, 8, 23, 16, tz = "GMT")
datetimeoffset(2020, 5, 15, 8, 23, 16, hour_offset = -7)
datetimeoffset(2020, 5, 15, 8, 23, 16, hour_offset = -7, minute_offset = 30)
```

---

datetimeoffset-invalid

Invalid datetimeoffset datetimes

Description

invalid_detect() detects invalid datetimes. invalid_any() returns TRUE if any datetimes are invalid. invalid_count() returns number of invalid datetimes. invalid_remove() removes invalid datetimes. invalid_resolve() resolves invalid datetimes.

Usage

```
## S3 method for class 'datetimeoffset'
invalid_detect(x)

## S3 method for class 'datetimeoffset'
invalid_resolve(x, ..., invalid = "NA", nonexistent = "NA")

## S3 method for class 'datetimeoffset'
invalid_any(x)

## S3 method for class 'datetimeoffset'
invalid_count(x)

## S3 method for class 'datetimeoffset'
invalid_remove(x)
```

Arguments

x	A `datetimeoffset()` object.
...	Ignored.
invalid	Invalid date resolution strategy. See `clock::invalid_resolve()`.
nonexistent	Nonexistent (because of DST spring forward) time resolution strategy. See `clock::as_zoned_time.clock_naive_time()`.

Details

`datetimeoffset()` datetimes can be considered invalid for three main reasons:

1. An invalid "calendar date" such as `"2020-02-30"` (there are less than 30 days in February).
2. A "nonexistent" datetime due to a Daylight Savings Time "spring forward" such as `"2020-03-08T02:59:59[America/L`
3. Incorrect UTC offsets such as `"2020-03-08T01:59:59-08[America/New_York]"` (that particular Eastern time has a UTC offset of `-05`)

Value

`invalid_detect()`, `invalid_remove()`, and `invalid_resolve()` return `datetimeoffset()` vectors. `invalid_count()` returns an integer and `invalid_any()` returns a logical value.

Examples

```
# invalid date because April only has 30 days
dts <- c("2019-04-30T03:30:00", "2019-04-31T02:30:00")
dts <- as_datetimeoffset(dts)
clock::invalid_detect(dts)
clock::invalid_any(dts)
clock::invalid_count(dts)
clock::invalid_remove(dts)
clock::invalid_resolve(dts)
```

```
clock::invalid_resolve(dts, invalid = "previous")
clock::invalid_resolve(dts, invalid = "previous-day")

# non-existent time because of DST "spring forward"
if ("America/Los_Angeles" %in% OlsonNames()) {
  dt <- as_datetimeoffset("2020-03-08T02:59:59[America/Los_Angeles]")
  print(clock::invalid_detect(dt))
  clock::invalid_resolve(dt, nonexistent = "roll-forward")
}

# incorrect UTC offsets
if ("America/New_York" %in% OlsonNames()) {
  dt <- as_datetimeoffset("2020-03-08T01:59:59-08[America/New_York]")
  print(clock::invalid_detect(dt))
  clock::invalid_resolve(dt)
}
```

`datetimeoffset_utilities`

Various "datetimeoffset" object utilities

Description

`is_datetimeoffset()` tests whether a datetime object is of the "datetimeoffset" class. `NA_datetimeoffset_` provides a "missing" "datetimeoffset" object. `datetimeoffset_now()` returns the current time in the corresponding time zone(s).

Usage

```
is_datetimeoffset(x)

NA_datetimeoffset_

datetimeoffset_now(tz = Sys.timezone())
```

Arguments

`x`	An object to be tested
`tz`	Time zone(s)

Format

An object of class `datetimeoffset` (inherits from `vctrs_rcrd`, `vctrs_vctr`) of length 1.

Value

`is_datetimeoffset()` returns a logical vector. `datetimeoffset_now()` returns a `datetimeoffset()` vector.

Examples

```
is_datetimeoffset(as_datetimeoffset(Sys.time()))
is_datetimeoffset(Sys.time())

is.na(NA_datetimeoffset_)
is.na(as_datetimeoffset(""))

if (all(c("America/Los_Angeles", "America/New_York") %in% OlsonNames()))
  datetimeoffset_now(c("America/Los_Angeles", "America/New_York"))
```

`datetime_at_tz` *Change time zones while preserving UTC time*

Description

`datetime_at_tz()` changes time zones while preserving UTC time (instead of clock time).

Usage

```
datetime_at_tz(x, tz = "", ...)

## S3 method for class 'datetimeoffset'
datetime_at_tz(
  x,
  tz = "",
  ...,
  ambiguous = "error",
  nonexistent = "error",
  fill = NA_character_
)

## S3 method for class 'clock_zoned_time'
datetime_at_tz(x, tz = "", ...)

## S3 method for class 'POSIXt'
datetime_at_tz(x, tz = "", ...)

## Default S3 method:
datetime_at_tz(x, tz = "", ...)
```

Arguments

`x`	A datetime object.
`tz`	The target timezone to change to.
`...`	Ignored

ambiguous	What to do when the "clock time" in the new time zone is ambiguous. See clock::as_zoned_time.clock_naive_time().
nonexistent	What to do when the "clock time" in the new time zone doesn't exist. See clock::as_zoned_time.clock_naive_time().
fill	If timezone and UTC offset info is missing what timezone to assume. See fill_tz().

Value

A datetime object. The UTC time should be the same but with a different time zone.

See Also

set_tz() changes time zones while preserving clock time (instead of UTC time).

Examples

```
if(all(c("America/Los_Angeles", "America/New_York") %in% OlsonNames())) {
  dt0 <- as_datetimeoffset("2020-01-01T01:01[America/Los_Angeles]")
  dt <- datetime_at_tz(dt0, "America/New_York")
  print(dt)
  dt <- datetime_at_tz(as.POSIXct(dt0), "America/New_York")
  print(dt)
  dt <- datetime_at_tz(clock::as_zoned_time(dt0), "America/New_York")
  print(dt)

  # Can also use `lubridate::with_tz()`
  if (requireNamespace("lubridate")) {
    dt <- lubridate::with_tz(dt0, "America/New_York")
    print(dt)
  }
}
```

---

datetime_cast *Widen/narrow datetime precision*

Description

datetime_widen() sets a floor on the minimum "precision" in the datetime vector by setting any missing elements to their minimum possible value. datetime_narrow() sets a cap on the maximum "precision" by setting any more precise elements missing. datetime_cast() sets the precision exactly by calling both datetime_narrow() and datetime_widen().

Usage

```
datetime_narrow(x, precision, ...)

## S3 method for class 'datetimeoffset'
datetime_narrow(x, precision, ...)

## S3 method for class 'clock_calendar'
datetime_narrow(x, precision, ...)

## S3 method for class 'clock_time_point'
datetime_narrow(
  x,
  precision,
  ...,
  method = c("floor", "round", "ceiling", "cast")
)

## S3 method for class 'POSIXt'
datetime_narrow(
  x,
  precision,
  ...,
  method = c("floor", "round", "ceiling"),
  nonexistent = "error",
  ambiguous = x
)

datetime_widen(x, precision, ...)

## S3 method for class 'datetimeoffset'
datetime_widen(
  x,
  precision,
  ...,
  year = 0L,
  month = 1L,
  day = 1L,
  hour = 0L,
  minute = 0L,
  second = 0L,
  nanosecond = 0L,
  na_set = FALSE
)

## S3 method for class 'clock_calendar'
datetime_widen(x, precision, ...)

## S3 method for class 'clock_time_point'
```

```
datetime_widen(x, precision, ...)

## S3 method for class 'POSIXt'
datetime_widen(x, precision, ...)

datetime_cast(x, precision, ...)

## Default S3 method:
datetime_cast(x, precision, ...)
```

Arguments

x	A datetime vector. Either `datetimeoffset()`, a "clock" "calendar", or a "clock" "time point".
precision	Precision to narrow/widen to. Either "missing", "year", "month", "day", "hour", "minute", "second", or "nanosecond".
...	Used by some methods. The default method for `datetime_cast()` will pass this to both `datetime_narrow()` and `datetime_widen()`.
method	Depending on the class either "floor", "ceiling", "round", and/or "cast".
nonexistent	What to do when the "clock time" in the new time zone doesn't exist. See `clock::as_zoned_time.clock_naive_time()`.
ambiguous	What to do when the "clock time" in the new time zone is ambiguous. See `clock::as_zoned_time.clock_naive_time()`.
year	If missing what year to assume
month	If missing what month to assume
day	If missing what day to assume
hour	If missing what hour to assume
minute	If missing what minute to assume
second	If missing what second to assume
nanosecond	If missing what nanosecond to assume
na_set	If `TRUE` widen the "missing" datetimes as well.

Value

A datetime vector.

Examples

```
dts <- as_datetimeoffset(c(NA_character_, "2020", "2020-04-10", "2020-04-10T10:10"))
datetime_precision(dts)
datetime_narrow(dts, "day")
datetime_widen(dts, "day")
datetime_cast(dts, "day")

datetime_widen(datetimeoffset(2020L), "day", month = 6, day = 15)
```

```
  # vectorized "precision" is allowed
  datetime_narrow(as_datetimeoffset(Sys.time()),
                  c("year", "day", "second"))
  datetime_widen(NA_datetimeoffset_, c("year", "day", "second"), na_set = TRUE)

  library("clock")
  ymd <- year_month_day(1918, 11, 11, 11)
  datetime_narrow(ymd, "day")
  datetime_narrow(ymd, "second") # already narrower than "second"
  datetime_widen(ymd, "second")
  datetime_widen(ymd, "day") # already wider than "day"

  ## Not run:
    # comparable {clock} calendar methods throw an error in certain cases
    clock::calendar_narrow(ymd, "second") # already narrower than "second"
    clock::calendar_widen(ymd, "day") # already wider than "day"

## End(Not run)

  nt <- as_naive_time(ymd)
  datetime_narrow(nt, "day")
  datetime_narrow(nt, "second")
  datetime_widen(nt, "second")
  datetime_widen(nt, "day")
  datetime_cast(nt, "day") # same as clock::time_point_floor(nt, "day")
  datetime_cast(nt, "day", method = "cast") # same as clock::time_point_cast(nt, "day")
```

`datetime_precision` *Datetime precision*

Description

`datetime_precision()` returns the "precision" of a datetime vector's datetimes. `precision_to_int()` converts the precision to an integer.

Usage

```
datetime_precision(x, ...)

## S3 method for class 'datetimeoffset'
datetime_precision(x, range = FALSE, unspecified = FALSE, ...)

## S3 method for class 'clock_calendar'
datetime_precision(x, ...)

## S3 method for class 'clock_time_point'
datetime_precision(x, ...)

## S3 method for class 'clock_zoned_time'
```

```
datetime_precision(x, ...)

## S3 method for class 'nanotime'
datetime_precision(x, ...)

precision_to_int(precision)
```

Arguments

x	A datetime vector. Either `datetimeoffset()`, a "clock" "calendar", or a "clock" "time".
...	Used by some S3 methods.
range	If TRUE return just the minimum and maximum "precision".
unspecified	If TRUE use the smallest non-missing component's as the precision even if there is a missing value for a larger component.
precision	A datetime precision (as returned by `datetime_precision()`).

Value

`datetime_precision()` returns a character vector of precisions. Depending on the object either "missing", "year", "quarter", "month", "week", "day", "hour", "minute", "second", "millisecond", "microsecond", or "nanosecond". `precision_to_int()` returns an integer vector.

Examples

```
dts <- as_datetimeoffset(c("2020", "2020-04-10", "2020-04-10T10:10"))
datetime_precision(dts)
datetime_precision(dts, range = TRUE)

dt <- datetimeoffset(2020, NA_integer_, 10)
datetime_precision(dt)
datetime_precision(dt, unspecified = TRUE)

precision_to_int("year") < precision_to_int("day")

library("clock")
datetime_precision(year_month_day(1918, 11, 11))
datetime_precision(sys_time_now())
datetime_precision(zoned_time_now(Sys.timezone()))
```

---

fill_tz *Fill in missing time zones and/or UTC offsets*

Description

`fill_tz()` fills in missing time zones. `fill_utc_offsets()` fills in missing UTC offsets.

Usage

```
fill_tz(x, tz = "")

fill_utc_offsets(x, ambiguous = "NA")
```

Arguments

x	A datetime object
tz	Timezone used to fill in missing time zones
ambiguous	What to do when the "clock time" in the new time zone is ambiguous. See clock::as_zoned_time.clock_naive_time().

Value

A datetime object

Examples

```
dts <- as_datetimeoffset(c("2020-01-01T01:01:01", "2020-01-01T01:01:01Z"))
fill_tz(dts, "UTC")
fill_tz(dts, Sys.timezone())
clock::as_sys_time(dts)
clock::as_sys_time(fill_tz(dts, "UTC"))
clock::as_zoned_time(dts)
clock::as_zoned_time(fill_tz(dts, ""))

if ("America/New_York" %in% OlsonNames()) {
  # non-ambiguous UTC offsets
  dt <- as_datetimeoffset("2020-11-01T12:30:00[America/New_York]")
  cat("unfilled: ", get_utc_offsets(dt), "\n")
  dt <- fill_utc_offsets(dt)
  cat("filled: ", get_utc_offsets(dt), "\n")

  # ambiguous UTC offsets due to DST
  dt0 <- as_datetimeoffset("2020-11-01T01:30:00[America/New_York]")
  dt <- fill_utc_offsets(dt0)
  cat('`ambiguous = "NA"` (default): ', get_utc_offsets(dt), "\n")
  dt <- fill_utc_offsets(dt0, ambiguous = "earliest")
  cat('`ambiguous = "earliest"`: ', get_utc_offsets(dt), "\n")
  dt <- fill_utc_offsets(dt0, ambiguous = "latest")
  cat('`ambiguos = "latest"`: ', get_utc_offsets(dt), "\n")
}
```

---

format *Convert datetime objects to character*

Description

format() returns a datetime string with as much **known** information possible (RFC 3339 with de facto standard time zone extension). format_iso8601() returns an ISO 8601 datetime string. format_pdfmark() returns a pdfmark datetime string with as much **known** information possible. format_strftime() allows base::strftime() style formatting. format_nanotime() allows CCTZ style formatting. format_edtf() returns an Extended Date Time Format (EDTF) string. format_exiftool() returns the date/time string expected by exiftool.

Usage

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

format_iso8601(
  x,
  offsets = TRUE,
  precision = NULL,
  sep = ":",
  mode = c("normal", "toml", "xmp"),
  ...
)

format_pdfmark(x, prefix = "D:")

format_edtf(x, offsets = TRUE, precision = NULL, usetz = FALSE, ...)

format_exiftool(x, mode = c("normal", "xmp", "pdf"), ...)

format_strftime(
  x,
  format = "%Y-%m-%d %H:%M:%S",
  tz = get_tz(x),
  usetz = FALSE,
  fill = mode_tz(x)
)

format_nanotime(
  x,
  format = "%Y-%m-%dT%H:%M:%E9S%Ez",
  tz = get_tz(x),
  fill = ""
)
```

Arguments

x	A datetimeoffset() object.
...	Ignored
offsets	Include the UTC offsets in the formatting

precision	The amount of precision: either "year", "month", "day", "hour", "minute", "second", "decisecond", "centisecond", "millisecond", "hundred microseconds", "ten microseconds", "microsecond", "hundred nanoseconds", "ten nanoseconds", or "nanosecond". If NULL then full precision for the object is shown.
sep	UTC offset separator. Either ":" or "".
mode	If mode = "pdf" only output supported PDF docinfo datetime values. If mode = "toml" only output supported TOML datetime values. If mode = "xmp" only output valid XMP metadata datetime values.
prefix	Prefix to use. Either "D:" (default) or "".
usetz	Include the time zone in the formatting
format	For format_strftime() see base::strftime(). For format_nanotime() see https://github.com/google/cctz/blob/6e09ceb/include/time_zone.h#L197.
tz	A character string specifying the time zone to be used for the conversion. Can be a length greater than one.
fill	If timezone and UTC offset info is missing what timezone to assume. See fill_tz().

Value

A character vector

Examples

```
# ISO 8601 datetimes
format_iso8601(as_datetimeoffset("2020-05"))
format_iso8601(as_datetimeoffset("2020-05-10 20:15"))
format_iso8601(as_datetimeoffset("2020-05-10 20:15:05-07"))
if (requireNamespace("lubridate"))
  lubridate::format_ISO8601(as_datetimeoffset("2020-05-10 20:15:05-07"))

# pdfmark datetimes
format_pdfmark(as_datetimeoffset("2020-05"))
format_pdfmark(as_datetimeoffset("2020-05-10 20:15"))
format_pdfmark(as_datetimeoffset("2020-05-10 20:15:05-07"))

# strftime style formatting
dt <- as_datetimeoffset("2020-05-10 20:15")
format_strftime(dt)
format_strftime(dt, format = "%c")

# CCTZ style formatting
if (requireNamespace("nanotime")) {
  dt <- as_datetimeoffset(Sys.time())
  format_nanotime(dt, format = "%F %H:%M:%E7S %Ez") # SQL Server datetimeoffset
}

# EDTF style formatting
format_edtf(as_datetimeoffset("2020-05"))
```

```
format_edtf(as_datetimeoffset("2020-05-10T20:15:05-07"))
dt <- datetimeoffset(2020, NA_integer_, 10)
format_edtf(dt)

# `exiftool` formatting
format_exiftool(as_datetimeoffset("2020:05:10"))
format_exiftool(as_datetimeoffset("2020:05:10 20:15"))
format_exiftool(as_datetimeoffset("2020:05:10 20:15:05-07:00"))
```

`from_datetimeoffset` *Convert to other datetime objects*

Description

We register S3 methods to convert `datetimeoffset()` objects to other R datetime objects:

Usage

```
## S3 method for class 'datetimeoffset'
as.Date(x, ...)

## S3 method for class 'datetimeoffset'
as_date(x, ...)

## S3 method for class 'datetimeoffset'
as.POSIXct(x, tz = mode_tz(x), ..., fill = "")

## S3 method for class 'datetimeoffset'
as_date_time(x, zone = mode_tz(x), ..., fill = NA_character_)

## S3 method for class 'datetimeoffset'
as.POSIXlt(x, tz = mode_tz(x), ..., fill = "")

## S3 method for class 'datetimeoffset'
as_year_month_day(x, ...)

## S3 method for class 'datetimeoffset'
as_year_month_weekday(x, ...)

## S3 method for class 'datetimeoffset'
as_iso_year_week_day(x, ...)

## S3 method for class 'datetimeoffset'
as_year_quarter_day(x, ..., start = NULL)

## S3 method for class 'datetimeoffset'
as_year_day(x, ...)
```

```
## S3 method for class 'datetimeoffset'
as_naive_time(x, ...)

## S3 method for class 'datetimeoffset'
as_sys_time(
  x,
  ...,
  ambiguous = "error",
  nonexistent = "error",
  fill = NA_character_
)

## S3 method for class 'datetimeoffset'
as_zoned_time(
  x,
  zone = mode_tz(x),
  ...,
  ambiguous = "error",
  nonexistent = "error",
  fill = NA_character_
)

## S3 method for class 'datetimeoffset'
as_weekday(x, ...)
```

Arguments

x	A datetimeoffset() object
...	Ignored
tz, zone	What time zone to assume
fill	If timezone and UTC offset info is missing what timezone to assume. See fill_tz().
start	The month to start the fiscal year in. See clock::as_year_quarter_day().
ambiguous	What to do when the "clock time" in the new time zone is ambiguous. See clock::as_zoned_time.clock_naive_time().
nonexistent	What to do when the "clock time" in the new time zone doesn't exist. See clock::as_zoned_time.clock_naive_time().

Details

We register S3 methods for the following:

- as.Date() and clock::as_date() returns the "local" date as a base::Date() object
- as.POSIXct() and clock::as_date_time() returns the "local" datetime as a base::POSIXct() object
- as.POSIXlt() returns the "local" datetime as a base::POSIXlt() object

- nanotime::as.nanotime() returns the "global" datetime as a nanotime::nanotime() object
- parttime::as.parttime() returns the "local" datetime as a parttime::parttime() object
- clock::as_year_month_day() returns a clock::year_month_day() calendar
- clock::as_year_month_weekday() returns a clock::year_month_weekday() calendar
- clock::as_iso_year_week_day() returns a clock::iso_year_week_day() calendar
- clock::as_year_quarter_day() returns a clock::year_quarter_day() calendar
- clock::as_year_day() returns a clock::year_day() calendar
- clock::as_naive_time() returns a "clock" naive-time
- clock::as_sys_time() returns a "clock" sys-time
- clock::as_zoned_time() returns a "clock" zoned-time
- clock::as_weekday() returns a clock::weekday() object

Value

A datetime object vector

Examples

```
# {base}
today <- as_datetimeoffset(Sys.Date())
now <- as_datetimeoffset(Sys.time())

as.Date(today)
as.Date(now)
as.POSIXct(now)
as.POSIXlt(now)

# {clock}
clock::as_date(today)
clock::as_date_time(now)

clock::as_year_month_day(now)
clock::as_year_month_weekday(now)
clock::as_iso_year_week_day(now)
clock::as_year_quarter_day(now)
clock::as_year_day(now)

clock::as_naive_time(now)
clock::as_sys_time(now)
clock::as_zoned_time(now)

clock::as_weekday(now)

if (requireNamespace("nanotime")) {
  nanotime::as.nanotime(now)
}
```

```
if (requireNamespace("parttime")) {
  parttime::as.parttime(now)
}
```

`getset_utc_offsets` *Get/set UTC offset strings*

Description

`get_utc_offsets()` and `set_utc_offsets()` gets/sets UTC offset strings

Usage

```
get_utc_offsets(x, sep = ":")

set_utc_offsets(x, value)
```

Arguments

`x`	A `datetimeoffset()` object
`sep`	Separator between hour and minute offsets. Either ":" or "".
`value`	Replace UTC offset string

Value

`get_utc_offsets()` returns a character string of UTC offset info. `set_utc_offsets()` returns a datetime (whose UTC offset info has been set).

See Also

`get_hour_offset()`, `set_hour_offset()`, `get_minute_offset()`, and `set_minute_offset()` allow getting/setting the separate individual hour/minute offset components with integers. `fill_utc_offsets()` fills any missing UTC offsets using non-missing time zones.

Examples

```
dt <- as_datetimeoffset("2020-01-01T01:01")
get_utc_offsets(dt)
dt <- set_utc_offsets(dt, "-07:00")
get_utc_offsets(dt)
dt <- set_utc_offsets(dt, "+0800")
get_utc_offsets(dt)
dt <- set_utc_offsets(dt, "+00")
get_utc_offsets(dt)
dt <- set_utc_offsets(dt, NA_character_)
get_utc_offsets(dt)
```

getters *Get datetime components*

Description

Getter methods for `datetimeoffset()` objects.

Usage

```
## S3 method for class 'datetimeoffset'
get_year(x)

## S3 method for class 'datetimeoffset'
get_month(x)

## S3 method for class 'datetimeoffset'
get_day(x)

## S3 method for class 'datetimeoffset'
get_hour(x)

## S3 method for class 'datetimeoffset'
get_minute(x)

## S3 method for class 'datetimeoffset'
get_second(x)

## S3 method for class 'datetimeoffset'
get_nanosecond(x)

get_subsecond_digits(x)

## S3 method for class 'datetimeoffset'
get_subsecond_digits(x)

## Default S3 method:
get_subsecond_digits(x)

get_hour_offset(x)

## S3 method for class 'datetimeoffset'
get_hour_offset(x)

## Default S3 method:
get_hour_offset(x)

## S3 method for class 'POSIXt'
```

```
get_hour_offset(x)

get_minute_offset(x)

## S3 method for class 'datetimeoffset'
get_minute_offset(x)

## Default S3 method:
get_minute_offset(x)

## S3 method for class 'POSIXt'
get_minute_offset(x)

get_tz(x)

## S3 method for class 'datetimeoffset'
get_tz(x)

## S3 method for class 'Date'
get_tz(x)

## S3 method for class 'POSIXt'
get_tz(x)

## S3 method for class 'clock_zoned_time'
get_tz(x)

## Default S3 method:
get_tz(x)
```

Arguments

x	A datetime object.

Details

We implement `datetimeoffset()` support for the following S3 methods from `clock`:

- `get_year()`
- `get_month()`
- `get_day()`
- `get_hour()`
- `get_minute()`
- `get_second()`
- `get_nanosecond()`

We also implemented new S3 getter methods:

- get_subsecond_digits()
- get_hour_offset()
- get_minute_offset()
- get_tz()

We also implement datetimeoffset() support for the following S3 methods from lubridate:

- year()
- month()
- mday()
- hour()
- minute()
- second()
- tz()
- date()

Value

The component

Examples

```
library("clock")
if ("Europe/Paris" %in% OlsonNames()) {
  dt <- as_datetimeoffset("1918-11-11T11:11:11.1234+00:00[Europe/Paris]")
} else {
  dt <- as_datetimeoffset("1918-11-11T11:11:11.1234")
}
get_year(dt)
get_month(dt)
get_day(dt)
get_hour(dt)
get_minute(dt)
get_second(dt)
get_nanosecond(dt)
get_subsecond_digits(dt)
get_hour_offset(dt)
get_minute_offset(dt)
get_tz(dt)
if (require("lubridate")) {
  paste0(year(dt), "-", month(dt), "-", day(dt),
         "T", hour(dt), ":", minute(dt), ":", second(dt),
         "[", tz(dt), "]")
}
```

mode_tz *Get most common time zone*

Description

'mode_tz()' gets the most common time zone in the datetime object. If a tie we use the time zone used first. Intended for use when coercing from a datetime object that supports multiple heterogeneous time zones to a datetime object that only supports one time zone

Usage

```
mode_tz(x, ...)

## S3 method for class 'datetimeoffset'
mode_tz(x, tz = "", ...)

## Default S3 method:
mode_tz(x, ...)
```

Arguments

x	A datetime object.
...	Ignored
tz	A timezone string to use for missing time zones. "" will be treated as equivalent to Sys.timezone().

Value

Timezone string

Examples

```
dt <- as_datetimeoffset(Sys.time())
print(mode_tz(dt))
if (all(c("America/Los_Angeles", "America/New_York") %in% OlsonNames())) {
  dt <- as_datetimeoffset("2020-01-01",
                          tz = c("America/Los_Angeles", "America/New_York"))
  print(mode_tz(dt))

  print(Sys.timezone()) # timezone to be used for missing time zones
  dt <- as_datetimeoffset("2020-01-01",
                          tz = c("America/New_York", NA_character_, NA_character_))
  print(mode_tz(dt))
}
```

setters *Set datetime components*

Description

Setter methods for `datetimeoffset()` objects.

Usage

```
## S3 method for class 'datetimeoffset'
set_year(x, value, ..., na_set = FALSE)

## S3 method for class 'datetimeoffset'
set_month(x, value, ..., na_set = FALSE)

## S3 method for class 'datetimeoffset'
set_day(x, value, ..., na_set = FALSE)

## S3 method for class 'datetimeoffset'
set_hour(x, value, ..., na_set = FALSE)

## S3 method for class 'datetimeoffset'
set_minute(x, value, ..., na_set = FALSE)

## S3 method for class 'datetimeoffset'
set_second(x, value, ..., na_set = FALSE)

## S3 method for class 'datetimeoffset'
set_nanosecond(x, value, ..., na_set = FALSE, digits = NULL)

set_subsecond_digits(x, value, ...)

## S3 method for class 'datetimeoffset'
set_subsecond_digits(x, value, ..., na_set = FALSE)

set_hour_offset(x, value, ...)

## S3 method for class 'datetimeoffset'
set_hour_offset(x, value, ..., na_set = FALSE)

set_minute_offset(x, value, ...)

## S3 method for class 'datetimeoffset'
set_minute_offset(x, value, ..., na_set = FALSE)

set_tz(x, value, ...)
```

```
## S3 method for class 'datetimeoffset'
set_tz(x, value, ..., na_set = FALSE)

## S3 method for class 'clock_zoned_time'
set_tz(x, value, ..., nonexistent = "error", ambiguous = "error")

## Default S3 method:
set_tz(x, value, ...)
```

Arguments

x	A datetime object.
value	The replacement value. For `set_day()` this can also be "last".
...	Currently ignored.
na_set	If `TRUE` set component for `NA` datetimes (making them no longer `NA`)
digits	If `NULL` do not update the `subsecond_digits` field. Otherwise an integer vector (`1L` through `9L` or `NA_integer_`) to update the `subsecond_digits` field with.
nonexistent	What to do when the "clock time" in the new time zone doesn't exist. See `clock::as_zoned_time.clock_naive_time()`.
ambiguous	What to do when the "clock time" in the new time zone is ambiguous. See `clock::as_zoned_time.clock_naive_time()`.

Details

We implement `datetimeoffset()` support for the following S3 methods from `clock`:

- `set_year()`
- `set_month()`
- `set_day()`
- `set_hour()`
- `set_minute()`
- `set_second()`
- `set_nanosecond()`

We also implemented new S3 setter methods:

- `set_hour_offset()`
- `set_minute_offset()`
- `set_tz()` (changes system time but not clock time)

We also implement `datetimeoffset()` support for the following S4 methods from `lubridate`:

- `year<-()`
- `month<-()`
- `day<-()`
- `hour<-()`
- `minute<-()`
- `second<-()`
- `date<-()`

Value

A datetime object.

Examples

```
library("clock")
dt <- NA_datetimeoffset_
dt <- set_year(dt, 1918L, na_set = TRUE)
dt <- set_month(dt, 11L)
dt <- set_day(dt, 11L)
dt <- set_hour(dt, 11L)
dt <- set_minute(dt, 11L)
dt <- set_second(dt, 11L)
dt <- set_nanosecond(dt, 123456789L)
dt <- set_subsecond_digits(dt, 4L)
dt <- set_hour_offset(dt, 0L)
dt <- set_minute_offset(dt, 0L)
dt <- set_tz(dt, "Europe/Paris")
format(dt)

if (require("lubridate")) {
  dt <- datetimeoffset(0L)
  year(dt) <- 1918L
  month(dt) <- 11L
  day(dt) <- 11L
  hour(dt) <- 11L
  minute(dt) <- 11L
  second(dt) <- 11L
  if (packageVersion("lubridate") > '1.8.0' &&
      "Europe/Paris" %in% OlsonNames()) {
    tz(dt) <- "Europe/Paris"
  }
  format(dt)
}
```

---

subsecond *Subsecond helper getter/setter*

Description

Helper getter/setter methods for the subseconds (aka fractional seconds) of `datetimeoffset()` objects.

Usage

```
## S3 method for class 'datetimeoffset'
get_millisecond(x)

## S3 method for class 'datetimeoffset'
```

```
set_millisecond(x, value, ..., na_set = FALSE, digits = 3L)

## S3 method for class 'datetimeoffset'
get_microsecond(x)

## S3 method for class 'datetimeoffset'
set_microsecond(x, value, ..., na_set = FALSE, digits = 6L)

get_subsecond(x, ...)

## S3 method for class 'datetimeoffset'
get_subsecond(x, digits = get_subsecond_digits(x), ...)

set_subsecond(x, value, digits = 1L, ...)

## S3 method for class 'datetimeoffset'
set_subsecond(x, value, digits = 1L, ..., na_set = FALSE)
```

Arguments

x	A datetime object.
value	The replacement value. For `set_day()` this can also be "last".
...	Currently ignored.
na_set	If `TRUE` set component for `NA` datetimes (making them no longer `NA`)
digits	If `NULL` do not update the `subsecond_digits` field. Otherwise an integer vector (`1L` through `9L` or `NA_integer_`) to update the `subsecond_digits` field with.

Details

Internally `datetimeoffset()` objects represent subseconds with two fields:

1. Nanoseconds (as an integer)
2. Number of subsecond digits (as an integer)

One can explicitly get/set these fields with

- `get_nanosecond()` / `set_nanosecond()`
- `get_subsecond_digits()` / `set_subsecond_digits()`

We implement `datetimeoffset()` support for the following S3 methods from `clock`:

- `get_millisecond()`
- `get_microsecond()`
- `set_millisecond()` (note sets any non-zero microsecond/nanosecond elements to zero)
- `set_microsecond()` (note sets any non-zero nanosecond elements to zero)

We implement the following new S3 methods:

- `get_subsecond()`
- `set_subsecond()`

Value

`get_millisecond()`, `get_microsecond()`, and `get_subsecond()` returns an integer vector. `set_millisecond()`, `set_microsecond()`, and `set_subsecond()` returns a datetime vector.

Examples

```
library("clock")
dt <- as_datetimeoffset("2020-01-01T10:10:10.123456789")
format(dt)
get_millisecond(dt)
get_microsecond(dt)
get_subsecond(dt, 1L)
get_subsecond(dt, 7L)

set_microsecond(dt, 123456L)
set_millisecond(dt, 123L)
set_subsecond(dt, 12L, digits = 2L)
set_subsecond(dt, 12L, digits = 3L)
```

---

weekdays.datetimeoffset

Additional datetime extractors

Description

Additional datetime extractors for `datetimeoffset()` objects.

Usage

```
## S3 method for class 'datetimeoffset'
weekdays(x, abbreviate = FALSE)

## S3 method for class 'datetimeoffset'
months(x, abbreviate = FALSE)

## S3 method for class 'datetimeoffset'
quarters(x, ...)

## S3 method for class 'datetimeoffset'
julian(x, origin = as.Date("1970-01-01"), ...)
```

Arguments

`x`	A `datetimeoffset()` datetime
`abbreviate`	Logical vector for whether the names should be abbreviated
`...`	Ignored
`origin`	Length one datetime of origin

Details

We implement `datetimeoffset()` support for the following S3 methods from base:

- `weekdays()`
- `months()`
- `quarters()`
- `julian()`

There is also `datetimeoffset()` support for the following methods from `lubridate`:

- `isoyear()` and `epiyear()`
- `quarter()` and `semester()`
- `week()`, `isoweek()`, and `epiweek()`
- `wday()` and `wday<-()`
- `qday()` and `qday<-()`
- `yday()` and `yday<-()`
- `am()` and `pm()`
- `days_in_month()`
- `dst()`
- `leap_year()`

Value

`weekdays()`, `months()`, `quarters()`, `julian()` return character vectors. See `base::weekdays()` for more information.

Examples

```
dto <- datetimeoffset_now()
print(dto)
weekdays(dto)
months(dto)
quarters(dto)
julian(dto)

if (require("lubridate")) {
  cat("`isoyear(dto)`: ", isoyear(dto), "\n")
  cat("`epiyear(dto)`: ", epiyear(dto), "\n")
  cat("`semester(dto)`: ", semester(dto), "\n")
  cat("`quarter(dto)`: ", quarter(dto), "\n")
  cat("`week(dto)`: ", week(dto), "\n")
  cat("`isoweek(dto)`: ", isoweek(dto), "\n")
  cat("`epiweek(dto)`: ", epiweek(dto), "\n")
  cat("`wday(dto)`: ", wday(dto), "\n")
  cat("`qday(dto)`: ", qday(dto), "\n")
  cat("`yday(dto)`: ", yday(dto), "\n")
  cat("`am(dto)`: ", am(dto), "\n")
```

```
  cat("`pm(dto)`: ", pm(dto), "\n")
  cat("`days_in_month(dto)`: ", days_in_month(dto), "\n")
  cat("`dst(dto)`: ", dst(dto), "\n")
  cat("`leap_year(dto)`: ", leap_year(dto), "\n")
}
```

Index