Package: anytime (via r-universe)

October 30, 2026

Type Package

Title Anything to 'POSIXct' or 'Date' Converter

Version 0.3.13.1

Date 2026-05-06

Description Convert input in any one of character, integer, numeric, factor, or ordered type into 'POSIXct' (or 'Date') objects, using one of a number of predefined formats, and relying on Boost facilities for date and time parsing.

URL https://eddelbuettel.github.io/anytime, https://github.com/eddelbuettel/anytime, https://dirk.eddelbuettel.com/code/anytime.html

BugReports https://github.com/eddelbuettel/anytime/issues

License GPL (>= 2)

Encoding UTF-8

Depends R (>= 3.2.0)

Imports Rcpp (>= 1.1.1)

LinkingTo Rcpp (>= 1.1.1), BH

Suggests tinytest (>= 1.0.0), gettz

RoxygenNote 6.0.1

VignetteBuilder Rcpp

Repository https://eddelbuettel.r-universe.dev

Date/Publication 2026-07-27 16:52:24 UTC

RemoteUrl https://github.com/eddelbuettel/anytime

RemoteRef HEAD

RemoteSha 95837ec58bd70dcd8c108b9d9b42a3b5de750334

Contents

anytime-package 2
anytime 3
assertDate 7
getFormats 8
iso8601 9

Index **11**

`anytime-package` *Anything to 'POSIXct' or 'Date' Converter*

Description

Convert input in any one of character, integer, numeric, factor, or ordered type into 'POSIXct' (or 'Date') objects, using one of a number of predefined formats, and relying on Boost facilities for date and time parsing.

Details

R excels at computing with dates, and times. Using *typed* representation for your data is highly recommended not only because of the functionality offered but also because of the added safety stemming from proper representation.

But there is a small nuisance cost in interactive work as well as in programming. How often have we told `as.POSIXct()` that the origin is (of course) the epoch. Do we really have to say it again? Similarly, when parsing dates that are *somewhat* in YYYYMMDD format, do we really need to bother converting from integer or numeric or character or factor or ordered with one of dozen separators and/or month forms: YYYY-MM-DD, YYYY/MM/DD, YYYYMMDD, YYYY-mon-DD and so on?

So there may have been a need for a *general purpose* converter returning a proper `POSIXct` (or `Date`) object no matter the input (provided it was somewhat parseable). `anytime()` tries to be that function.

The actual conversion is done by a combination of Boost lexical_cast to go from *(almost) anything* to string representation which is then parsed by Boost Date_Time. An alternate method using the corresponding R functions is also available as a fallback.

Conversion is done by looping over a fixed set of formats until a matching one is found, or returning an error if none is found. The current set of conversion formulae is accessible in the source code, and can now also be accessed in R via `getFormats()`. Formats can be added and removed via the `addFormats()` and `removeFormats{}` functions.

Details on the Boost date format symbols are provided by the Boost date_time documentation and similar (but not identical) to what `strftime` uses.

Author(s)

Dirk Eddelbuettel [aut, cre] (ORCID: <https://orcid.org/0000-0001-6419-907X>)

References

Boost date_time: https://www.boost.org/doc/libs/1_70_0/doc/html/date_time.html

Formats used: https://github.com/eddelbuettel/anytime/blob/master/src/anytime.cpp#L43-L106

Boost format documentation: https://www.boost.org/doc/libs/1_61_0/doc/html/date_time/date_time_io.html#date_time.format_flags

Examples

```
Sys.setenv(TZ=anytime:::getTZ())      ## helper function to try to get TZ
options(digits.secs=6)                ## for fractional seconds below

library(anytime)                      ## load package, caches TZ information

## integer
anydate(20160101L + 0:2)

## numeric
anydate(20160101 + 0:2)

## factor
anydate(as.factor(20160101 + 0:2))

## ordered
anydate(as.ordered(20160101 + 0:2))

## Dates: Character
anydate(as.character(20160101 + 0:2))

## Dates: alternate formats
anydate(c("20160101", "2016/01/02", "2016-01-03"))

## Datetime: ISO with/without fractional seconds
anytime(c("2016-01-01 10:11:12", "2016-01-01 10:11:12.345678"))

## Datetime: ISO alternate (?) with 'T' separator
anytime(c("20160101T101112", "20160101T101112.345678"))

## Short month '%b' (and full month is supported too)
anytime(c("2016-Sep-01 10:11:12", "Sep/01/2016 10:11:12", "Sep-01-2016 10:11:12"))

## Datetime: Mixed format (cf https://stackoverflow.com/questions/39259184)
anytime(c("Thu Sep 01 10:11:12 2016", "Thu Sep 01 10:11:12.345678 2016"))
```

---

anytime *Parse POSIXct or Date objects from input data*

Description

These function use the Boost Date_Time library to parse datetimes (and dates) from strings, integers, factors or even numeric values (which are cast to strings internally). They return a vector of `POSIXct` objects (or `Date` objects in the case of `anydate`). `POSIXct` objects represent dates and time as (possibly fractional) seconds since the 'epoch' of January 1, 1970. A timezone can be set, if none is supplied 'UTC' is set.

Usage

```
anytime(x, tz = getTZ(), asUTC = FALSE,
  useR = getOption("anytimeUseRConversions", FALSE),
  oldHeuristic = getOption("anytimeOldHeuristic", FALSE),
  calcUnique = FALSE)

anydate(x, tz = getTZ(), asUTC = FALSE,
  useR = getOption("anytimeUseRConversions", FALSE), calcUnique = FALSE)

utctime(x, tz = getTZ(), useR = getOption("anytimeUseRConversions", FALSE),
  oldHeuristic = getOption("anytimeOldHeuristic", FALSE),
  calcUnique = FALSE)

utcdate(x, tz = getTZ(), useR = getOption("anytimeUseRConversions", FALSE),
  calcUnique = FALSE)
```

Arguments

`x`	A vector of type character, integer or numeric with date(time) expressions to be parsed and converted.
`tz`	A string with the timezone, defaults to the result of the (internal) `getTZ` function if unset. The `getTZ` function returns the timezone values stored in local package environment, and set at package load time. Also note that this argument applies to the *output*: the returned object will have this timezone set. The timezone is *not* used for the parsing which will always be to localtime, or to UTC is the `asUTC` variable is set (as it is in the related functions `utctime` and `utcdate`). So one can think of the argument as 'shift parsed time object to this timezone'. This is similar to what `format()` in base R does, but our return value is still a `POSIXt` object instead of a character value.
`asUTC`	A logical value indicating if parsing should be to UTC; default is false implying localtime.
`useR`	A logical value indicating if conversion should be done via code from R (via `Rcpp::Function`) instead of the default Boost routines. The default value is the value of the option `anytimeUseRConversions` with a fallback of `FALSE` if the option is unset. In other words, this will be false by default but can be set to true via an option.
`oldHeuristic`	A logical value to enable behaviour as in version 0.2.2 or earlier: interpret a numeric or integer value that could be seen as a YYYYMMDD as a date. If the default value `FALSE` is seen, then numeric values are used as offsets dates (in

anydate or utcdate), and as second offsets for datetimes otherwise. A default value can also be set via the anytimeOldHeuristic option.

calcUnique — A logical value with a default value of FALSE that tells the function to perform the anytime() or anydate() calculation only once for each unique value in the x vector. It results in no difference in inputs or outputs, but can result in a significant speed increases for long vectors where each timestamp appears more than once. However, it will result in a slight slow down for input vectors where each timestamp appears only once.

Details

A number of fixed formats are tried in succession. These include the standard ISO format 'YYYY-MM-DD HH:MM:SS' as well as different local variants including several forms popular in the United States. Two-digit years and clearly ambiguous formats such as '03/04/05' are ignored. In the case of parsing failure a NA value is returned.

Fractional seconds are supported as well. As R itself only supports microseconds, the Boost compile-time option for nano-second resolution has not been enabled.

Value

A vector of POSIXct elements, or, in the case of anydate, a vector of Date objects.

Notes

By default, the (internal) conversion to (fractional) seconds since the epoch is relative to the local time of this system, and therefore not completely independent of the settings of the local system. This is to strike a balance between ease of use and functionality. A more-full featured conversion could be possibly be added with support for arbitrary reference times, but this is (at least) currently outside the scope of this package. See the **RcppCCTZ** package which offers some timezone-shifting and differencing functionality. As of version 0.0.5 one can also parse relative to UTC avoiding the local time issue.

Times and timezones can be tricky. This package offers a heuristic approach, it is likely that some input formats may not be parsed, or worse, be parsed incorrectly. This is not quite a Bobby Tables situation but care must always be taken with user-supplied input.

The Boost Date_Time library cannot parse single digit months or days. So while '2016/09/02' works (as expected), '2016/9/2' will not. Other non-standard formats may also fail.

There is a known issue (discussed at length in issue ticket 5) where Australian times are off by an hour. This seems to affect only Windows, not Linux.

When given a vector, R will coerce it to the type of the first element. Should that be NA, surprising things can happen: c(NA, Sys.Date()) forces both values to numeric and the date will not be parsed correctly (as its integer value becomes numeric before our code sees it). On the other hand, c(Sys.Date(), NA) works as expected parsing as type Date with one missing value. See issue ticket 11 for more.

Another known issue concerns conversion when the timezone is set to 'Europe/London', see GitHub issue tickets 36. 51. 59. and 86. As pointed out in the comment in that last one, the Sys.timezone manual page suggests several alternatives to using 'Europe/London' such as 'GB'.

Yet another known issue arises on Windows due to designs in the Boost library. While we can set the TZ library variable, Boost actually does *not* consult it but rather relies only on the (Windows) tool tzutil. This means that default behaviour should be as expected: dates and/or times are parsed to the local settings. But testing different TZ values (or more precisely, changes via the (unexported) helper function setTZ function as we cache TZ) will only influence the behaviour on Unix or Unix-alike operating systems and not on Windows. See the discussion at issue ticket 96 for more. In short, the recommendation for Windows user is to also set useR=TRUE when setting a timezone argument.

Operating System Impact

On Windows systems, accessing the isdst flag on dates or times before January 1, 1970, can lead to a crash. Therefore, the lookup of this value has been disabled for those dates and times, which could therefore be off by an hour (the common value that needs to be corrected). It should not affect dates, but may affect datetime objects.

Old Heuristic

Up until version 0.2.2, numeric input smaller than an internal cutoff value was interpreted as a date, even if anytime() was called. While convenient, it is also inconsistent as we otherwise take numeric values to be offsets to the epoch. Newer version are consistent: for anydate, a value is taken as *date offset* relative to the epoch (of January 1, 1970). For anytime, it is taken as *seconds offset*. So anytime(60) is one minute past the epoch, and anydate(60) is sixty days past it. The old behaviour can be enabled by setting the oldHeuristic argument to anytime (and utctime) to TRUE. Additionally, the default value can be set via getOption("anytimeOldHeuristic") which can be set to TRUE in startup file. Note that all other inputs such character, factor or ordered are not affected.

Warnings

As of version 0.3.10, a conversion from character resulting in a NA will lead to a warning being emitted. At most one warning per call is given: should numerous unparseable values be present on input, only one warning will be show. R offers mechanism to either suppress warnings, or convert them to errors as described in the help page for options() under the entry for warn.

Author(s)

Dirk Eddelbuettel

References

This StackOverflow answer provided the initial idea: https://stackoverflow.com/a/3787188/143305.

See Also

anytime-package, getFormats

Examples

```
## See the source code for a full list of formats, and the
## or the reference in help('anytime-package') for details
times <- c("2004-03-21 12:45:33.123456",
           "2004/03/21 12:45:33.123456",
           "20040321 124533.123456",
           "03/21/2004 12:45:33.123456",
           "03-21-2004 12:45:33.123456",
           "2004-03-21",
           "20040321",
           "03/21/2004",
           "03-21-2004",
           "20010101")
anytime(times)
anydate(times)
utctime(times)
utcdate(times)

## show effect of tz argument
anytime("2001-02-03 04:05:06")
## adjust parsed time to given TZ argument
anytime("2001-02-03 04:05:06", tz="America/Los_Angeles")
## somewhat equivalent base R functionality
format(anytime("2001-02-03 04:05:06"), tz="America/Los_Angeles")
```

`assertDate` *Convert to Date (or POSIXct) and assert successful conversion*

Description

Converts its input to type `Date` (or `POSIXct`), and asserts that the content is in fact of suitable type by checking for remaining `NA`

Usage

```
assertDate(x)

assertTime(x)
```

Arguments

`x`	An input object suitable for `anydate` or `anytime`

Details

Note that these functions *just check for* `NA` and cannot check for semantic correctness.

Value

A vector of `Date` or `POSIXct` objects. As a side effect, an error will be thrown in any of the input was not convertible.

Author(s)

Dirk Eddelbuettel

Examples

```
assertDate(c("2001/02/03", "2001-02-03", "20010203"))
assertTime(c("2001/02/03 04:05:06", "2001-02-03  04:05:06", "20010203 040506"))
```

`getFormats` *Functions to retrieve, set or remove formats used for parsing dates.*

Description

The time and date parsing and conversion relies on trying a (given and fixed) number of timeformats. The format used is the one employed by the underlying implementation of the Boost date_time library.

Usage

```
getFormats()

addFormats(fmt)

removeFormats(fmt)
```

Arguments

`fmt`	A vector of character values in the form understood by Boost date_time

Value

Nothing in the case of `addFormats`; a character vector of formats in the case of `getFormats`

Author(s)

Dirk Eddelbuettel

See Also

`anytime-package` and references therein

Examples

```
getFormats()
addFormats(c("%d %b %y",      # two-digit date [not recommended], textual month
             "%a %b %d %Y"))  # weekday weeknumber four-digit year
removeFormats("%d %b %y")     # remove first
```

`iso8601` *Format a Datetime object: ISO 8601, RFC 2822 or RFC 3339*

Description

ISO 8601, RFC 2822 and RFC 3339 are a standards for date and time representation covering the formatting of date and time (with or without possible fractional seconds) and timezone information.

Usage

```
iso8601(pt)

rfc2822(pt)

rfc3339(pt)

yyyymmdd(pt)
```

Arguments

`pt`	A `POSIXt` Datetime or a `Date` object

Value

A character object formatted according to ISO 8601, RFC 2822 or RFC 3339

ISO 8601

ISO 8601 is described in some detail in https://en.wikipedia.org/wiki/ISO_8601 and covers multiple date and time formats.

Here, we interpret it more narrowly focussing on a single format each for datetimes and dates. We return datetime object formatted as '2016-09-01T10:11:12' and date object as '2016-09-01'.

If the option `anytimeOldISO8601format` is set to `TRUE`, then the previous format (with a space instead of 'T' to separate date and time) is used.

RFC 2822

RFC 2822 is described in some detail in https://www.ietf.org/rfc/rfc2822.txt and https://en.wikipedia.org/wiki/Email#Internet_Message_Format. The Date and Time formating cover only a subset of the specification in that RFC.

Here, we use it to provide a single format each for datetimes and dates. We return datetime object formatted as 'Thu, 01 Sep 2016 10:11:12.123456 -0500' and date object as 'Thu, 01 Sep 2016'.

RFC 3339

RFC 3339 is described in some detail in https://www.rfc-editor.org/rfc/rfc3339 It refines both earlier standards.

Here, we use it to format datetimes and dates as single and compact strings. We return datetime object formatted as '2016-09-01T10:11:12.123456-0500' and date object as '2016-09-01'.

YYYYMMDD

This is a truly terrible format which needs to die, but refuses to do so. If you are unfortunate enough to be forced to interoperate with code expecting it, you can use this function. But it would be better to take a moment to rewrite such code.

Author(s)

Dirk Eddelbuettel

References

https://en.wikipedia.org/wiki/ISO_8601, https://www.ietf.org/rfc/rfc2822.txt, https://en.wikipedia.org/wiki/Email#Internet_Message_Format, https://www.rfc-editor.org/rfc/rfc3339

Examples

```
iso8601(anytime("2016-09-01 10:11:12.123456"))
iso8601(anydate("2016-Sep-01"))

rfc2822(anytime("2016-09-01 10:11:12.123456"))
rfc2822(anydate("2016-Sep-01"))

rfc3339(anytime("2016-09-01 10:11:12.123456"))
rfc3339(anydate("2016-Sep-01"))

yyyymmdd(anytime("2016-09-01 10:11:12.123456"))
yyyymmdd(anydate("2016-Sep-01"))
```

Index

∗ **package**
anytime-package, 2

addFormats *(*getFormats*)*, 8
anydate *(*anytime*)*, 3
anytime, 3
anytime-package, 2
assertDate, 7
assertTime *(*assertDate*)*, 7

getFormats, *6*, 8

iso8601, 9

removeFormats *(*getFormats*)*, 8
rfc2822 *(*iso8601*)*, 9
rfc3339 *(*iso8601*)*, 9

strftime, *2*
Sys.timezone, *5*

utcdate, *4*
utcdate *(*anytime*)*, 3
utctime, *4*
utctime *(*anytime*)*, 3

yyyymmdd *(*iso8601*)*, 9