anytime: Easier Date and Time Conversion

Dirk Eddelbuettel[1]

[1]Department of Statistics, University of Illinois, Urbana-Champaign, IL, USA

This version was compiled on July 20, 2019

The anytime package provides functions which convert from both a number of different input variable types (integer, numeric, character, factor) and different input formats which are tried heuristically offering a powerful and versatile date and time converter that (generally) requires no user input and operates autonomously.

Motivation

R excels at computing with dates, and times. Using a *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. Users must have told `as.POSIXct()` about a million times that the origin is (of course) the epoch. Do we really have to say it a million more times? Similarly, when parsing dates that are *some variant* of the common YYYYMMDD format, do we really have to manually convert from `integer` or `numeric` or `factor` or `ordered` to character? Having one of several common separators and/or date formats (YYYY-MM-DD, YYYY/MM/DD, YYYYMMDD, YYYY-mon-DD and so on, with or without times), do we really need a format string? Or could a smart converter function do this for us?

`anytime()` aims to be that *general purpose* converter returning a proper `POSIXct` (or `Date`) object no matter the input (provided it was parseable), relying on Boost Date_Time for the (efficient, performant) conversion. `anydate()` is an additional wrapper returning a `Date` object instead.

Examples

We first set up the R environment and display for the examples that follow.

```
library(anytime)                  # our library
options(width=50,                 # column width
        digits.secs=6)            # fractional secs
Sys.setenv(TZ=anytime:::getTZ()) # TZ helper
```

From Integer or Numeric or Factor or Ordered. For the numeric date formats in the range of the (numeric) *yyyymmdd* format, we use the `anydate()` function.

```
## integer
anydate(20160101L + 0:2)
#  [1] "2016-01-01" "2016-01-02" "2016-01-03"

## numeric
anydate(20160101 + 0:2)
#  [1] "2016-01-01" "2016-01-02" "2016-01-03"
```

Numeric input also works for datetimes, but the range has to correspond to the range of `as.numeric()` value of `POSIXct` variables:

```
## integer
anytime(1451628000L + 0:2)
#  [1] "2016-01-01 00:00:00 CST"
#  [2] "2016-01-01 00:00:01 CST"
#  [3] "2016-01-01 00:00:02 CST"

## numeric
anytime(1451628000 + 0:2)
#  [1] "2016-01-01 00:00:00 CST"
#  [2] "2016-01-01 00:00:01 CST"
#  [3] "2016-01-01 00:00:02 CST"
```

This is a change from version 0.3.0; the old behaviour (which was not fully consistent in how it treated numeric input values, but convenient for input in the ranges shown here) can be enabled via either an argument to the function or a global options, see `help(anytime)` for details:

```
## integer
anytime(20160101L + 0:2, oldHeuristic=TRUE)
#  [1] "2016-01-01 CST" "2016-01-02 CST"
#  [3] "2016-01-03 CST"

## numeric
anytime(20160101 + 0:2, oldHeuristic=TRUE)
#  [1] "2016-01-01 CST" "2016-01-02 CST"
#  [3] "2016-01-03 CST"
```

Factor or Ordered. Factor variables and their order variant are also supported directly.

```
## factor
anytime(as.factor(20160101 + 0:2))
#  [1] "2016-01-01 CST" "2016-01-02 CST"
#  [3] "2016-01-03 CST"

## ordered
anytime(as.ordered(20160101 + 0:2))
#  [1] "2016-01-01 CST" "2016-01-02 CST"
#  [3] "2016-01-03 CST"
```

Character: Simple. Character input is supported in a variety of formats. We first show simple formats.

```
## Dates: Character
anytime(as.character(20160101 + 0:2))
#  [1] "2016-01-01 CST" "2016-01-02 CST"
#  [3] "2016-01-03 CST"

## Dates: alternate formats
anytime(c("20160101", "2016/01/02", "2016-01-03"))
#  [1] "2016-01-01 CST" "2016-01-02 CST"
#  [3] "2016-01-03 CST"
```

Character: ISO. ISO8661 date (and datetime) formats are supported too, with both 'T' and a space as separator of date and time.

```
## Datetime: ISO with/without fractional seconds
anytime(c("2016-01-01 10:11:12",
          "2016-01-01T10:11:12.345678"))
#  [1] "2016-01-01 10:11:12.000000 CST"
#  [2] "2016-01-01 10:11:12.345678 CST"
```

Character: Textual month formats. Date formats with month abbreviations are supported in a number of common orderings.

```
## ISO style
anytime(c("2016-Sep-01 10:11:12",
          "Sep/01/2016 10:11:12",
          "Sep-01-2016 10:11:12"))
#  [1] "2016-09-01 10:11:12 CDT"
#  [2] "2016-09-01 10:11:12 CDT"
#  [3] "2016-09-01 10:11:12 CDT"

## Datetime: Mixed format
## (cf http://stackoverflow.com/questions/39259184)
anytime(c("Thu Sep 01 10:11:12 2016",
          "Thu Sep 01 10:11:12.345678 2016"))
#  [1] "2016-09-01 10:11:12.000000 CDT"
#  [2] "2016-09-01 10:11:12.345678 CDT"
```

Character: Dealing with DST. This shows an important aspect. When not working in localtime (by overriding to UTC) the *change in difference* to UTC is correctly covered (which the underlying Boost Date_Time library does not do by itself).

```
## Datetime: pre/post DST
anytime(c("2016-01-31 12:13:14",
          "2016-08-31 12:13:14"))
#  [1] "2016-01-31 12:13:14 CST"
#  [2] "2016-08-31 12:13:14 CDT"
## important: catches change
anytime(c("2016-01-31 12:13:14",
          "2016-08-31 12:13:14"), tz="UTC")
#  [1] "2016-01-31 18:13:14 UTC"
#  [2] "2016-08-31 17:13:14 UTC"
```

Technical Details

The actual parsing and conversion is done by two different Boost libraries. First, Boost lexical_cast is used to convert from *anything* to a string representation. This textual representation is then parsed by Boost Date_Time to create the corresponding date, or datetime, type. (There are also a number of special cases where numeric values are directly converted; see below.) We use the **BH** package (Eddelbuettel *et al.*, 2019a) to access these Boost libraries, and rely on **Rcpp** (Eddelbuettel and François, 2011; Eddelbuettel, 2013) for a seamless C++ interface to and from R.

The Boost Date_Time library is addressing the needs for parsing date and datetimes from text. It permits us to loop over a suitably large number of *candidate formats* with considerable ease. The formats are generally variants of the ISO 8601 date format, *i.e.*, of the YYYY-MM-DD ordering. In general, we also allow for textual representation of months, *e.g.*, 'Jan' for January.

```
anytime(c("2016-1-31", "2016-2-1"))
#  [1] "2016-01-31 CST" NA
anytime(c("2016-1-31", "2016-2-1"), useR=TRUE)
#  [1] "2016-01-31 CST" "2016-02-01 CST"
```

The list of current formats can be retrieved by the `getFormats()` function. Users can also add to this list at runtime by calling `addFormats()`, as well as removing formats. User-provided formats are tried before the formats supplied by the package.

```
getFormats()
#   [1] "%Y-%m-%d %H:%M:%S%f"
#   [2] "%Y-%m-%d %H%M%S%f"
#   [3] "%Y/%m/%d %H:%M:%S%f"
#   [4] "%Y%m%d %H%M%S%f"
#   [5] "%Y%m%d %H:%M:%S%f"
#   [6] "%m/%d/%Y %H:%M:%S%f"
#   [7] "%m-%d-%Y %H:%M:%S%f"
#   [8] "%Y-%b-%d %H:%M:%S%f"
#   [9] "%Y/%b/%d %H:%M:%S%f"
#  [10] "%Y%b%d %H%M%S%F"
#  [11] "%Y%b%d %H:%M:%S%F"
#  [12] "%b/%d/%Y %H:%M:%S%f"
#  [13] "%b-%d-%Y %H:%M:%S%f"
#  [14] "%d.%b.%Y %H:%M:%S%f"
#  [15] "%d%b%Y %H%M%S%f"
#  [16] "%d%b%Y %H:%M:%S%f"
#  [17] "%d-%b-%Y %H%M%S%f"
#  [18] "%d-%b-%Y %H:%M:%S%f"
#  [19] "%Y-%B-%d %H:%M:%S%f"
#  [20] "%Y/%B/%d %H:%M:%S%f"
#  [21] "%Y%B%d %H%M%S%f"
#  [22] "%Y%B%d %H:%M:%S%f"
#  [23] "%B/%d/%Y %H:%M:%S%f"
#  [24] "%B-%d-%Y %H:%M:%S%f"
#  [25] "%d.%B.%Y %H:%M:%S%f"
#  [26] "%a %b %d %H:%M:%S%F %Y"
#  [27] "%a %d %b %Y %H:%M:%S%F"
#  [28] "%Y-%m-%d %H:%M:%S%Z"
#  [29] "%a %b %d %H:%M:%S%F xxx %Y"
#  [30] "%Y-%m-%d"
#  [31] "%Y%m%d"
#  [32] "%m/%d/%Y"
#  [33] "%m-%d-%Y"
#  [34] "%Y-%b-%d"
#  [35] "%Y%b%d"
#  [36] "%b/%d/%Y"
#  [37] "%b-%d-%Y"
#  [38] "%d%b%Y%d-%b-%Y%Y-%B-%d"
#  [39] "%Y%B%d"
#  [40] "%B/%d/%Y"
#  [41] "%B-%d-%Y"
```

One minor shortcoming is that single digits dates and months are sometimes encountered: 2001-2-3 for the third of February of 2001. Boost Date_Time will only work with 2001-02-03 which is somewhat more restrictive.

As an alternative, and to be as close to parsing by the R language and system, we also support the parser from R itself. As R does not provide access via its API, we use the **Rcpp** package (Eddelbuettel,

2013; Eddelbuettel *et al.*, 2019b). This is shown in the second invocation of `anytime()` in the preceding example.

Output Formats

A related topic is faithful and easy to read *representation* of datetime objects in output, *i.e.,* formats for printing object.

In the spirit of *no configuration* used on the parsing side, formating support is provided via several functions. These all follow different known standards and are accessible by the name of the standard, or, in one case, the non-standard convention. In all cases, a character representation is returned.

```
pt <- anytime("2016-01-31 12:13:14.123456")
iso8601(pt)
#  [1] "2016-01-31T12:13:14"
rfc2822(pt)
#  [1] "Sun, 31 Jan 2016 12:13:14.123456 -0600"
rfc3339(pt)
#  [1] "2016-01-31T12:13:14.123456-0600"
yyyymmdd(pt)
#  [1] "20160131"
```

Ambiguities

The **anytime** package is designed to operate heuristically on a number of *plausible* and *sane* formats. This cannot possibly cover all conceivable cases.

North America versus the world. In general, **anytime** tries to gently nudge users towards ISO 8601 order of *year* followed by *month* and *day*. But for example in the United States, another prevalent form insists on month-day-year ordering. As many users are likely to encounter such input format, **anytime** accomodates this use provided a separator is used: either a slash (`/`) or a hyphen (`-`) is accepted.

One versus two digits. As mentioned earlier, displaying only one digit for day and/or month is possible. An example would be "2010-2-4" for the fourth of February in 2010. However, the Boost parser does not recognise this and would require "2010-02-04". However, setting the option `useR=TRUE` switches to the R parser which does recognise this input.

Asserts

The **anytime** package also contains two helper functions that can assist in defensive programming by validating input arguments. The `assertTime()` and `assertDate()` functions validate if the given input can be parsed, respectively, as `Datetime` or `Date` objects. In case one of the inputs cannot be parsed, an error is triggered. Otherwise the parsed input is returned invisibly.

Summary

We describe the **anytime** package which offers fast, convenient and reliable date and datetime conversion for R users along with helper functions for formatting and assertions.

References

Eddelbuettel D (2013). *Seamless R and C++ Integration with Rcpp*. Springer, New York. doi:10.1007/978-1-4614-6868-4. ISBN 978-1-4614-6867-7.

Eddelbuettel D, Emerson JW, Kane MJ (2019a). *BH: Boost C++ Header Files*. R package version 1.69.0-1, URL https://CRAN.R-project.org/package=BH.

Eddelbuettel D, François R (2011). "Rcpp: Seamless R and C++ Integration." *Journal of Statistical Software*, **40**(8), 1–18. doi:10.18637/jss.v040.i08. URL http://www.jstatsoft.org/v40/i08/.

Eddelbuettel D, François R, Allaire J, Ushey K, Kou Q, Russel N, Chambers J, Bates D (2019b). *Rcpp: Seamless R and C++ Integration*. R package version 1.0.1, URL package=Rcpp.