Rmetrics - timeDate Package

by Yohan Chalabi, Martin Mächler, and Diethelm Würtz

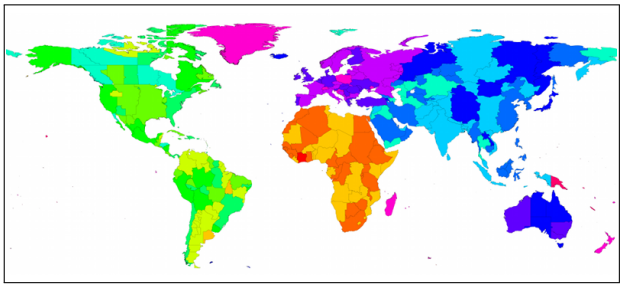

Figure 1: World map with major time zones. [1]

Abstract The management of time and holidays can prove crucial in applications that rely on historical data. A typical example is the aggregation of a data set recorded in different time zones and under different daylight saving time rules. Besides the time zone conversion function, which is well supported by default classes in R, one might need functions to handle special days or holidays. In this respect, the package **timeDate** enhances default date-time classes in R and brings new functionalities to time zone management and the creation of holiday calendars.

Chronological data sets recorded in different time zones play an important role in industrial applications. For example, in financial applications, it is common to aggregate time series that have been recorded in financial centers with different daylight saving time (DST) rules and time zones.

R includes different classes to represent dates and time. The class that holds particular interest for us is the `"POSIXct"` one. It internally records timestamps as the number of seconds from "1970-01-01 UTC", where UTC stands for universal time coordinated. Moreover, it supports the DST and time zone functions by using the rules provided by the operating system (OS). However, at the time **timeDate** (Würtz et al. (2011))—formerly known as **fCalendar**—was first released, the implementation of the DST function was not consistent across OSs. Back then, the main purpose of the package was to have DST rules directly available to bring consistency over OSs. Today, DST support by OSs is not a problematic question as it used to be. As we will show later, the `"timeDate"` class is based on the `"POSIXct"` one. Both classes hence share common functionalities. However, the **timeDate** package has some additional functionalities, which we will emphasize in this note. For related date-time classes in R, see Ripley & Hornik (2001) and Grothendieck & Petzoldt (2004).

Another problem commonly faced in managing time and dates is the midnight standard. Indeed, the problem can be handled in different ways depending on the standard in use. For example, the standard C library does not allow for the "midnight" standard in the format "24:00:00" (Bateman (2000)). However, the **timeDate** package supports this format.

Moreover, **timeDate** includes functions for calendar manipulations, business days, weekends, and public and ecclesiastical holidays. One can handle day count conventions and rolling business conventions. Such periods can pertain to, for example, the last working day of the month. The below examples illustrate this point.

In the remaining part of this note, we first present the structure of the `"timeDate"` class. Then, we explain the creation of `"timeDate"` objects. The use of financial centers with DST rules is described next. Thereafter, we explain the management of holidays. Finally, operations on `"timeDate"` objects such as mathematical operations, rounding, subsetting, and coercions are discussed. Throughout this note, we provide many examples to illustrate the functionalities of the package.

Structure of the `"timeDate"` class

The `"timeDate"` `S4` class is composed of a `"POSIXct"` object that is always in Greenwich Mean Time (GMT) and of a financial center that keeps track of DST. We use the term *financial center* to denote geographical locations. This terminology stems from the main application of the Rmetrics packages. However, this denomination should not stop programmers from using the package in other fields. By default, the local financial center is set to GMT. The default setting can be changed via `setRmetricsOptions(myFinCenter = ....)`. The formal `S4` class is defined as

```
> showClass("timeDate")
Class "timeDate" [package "timeDate"]

Slots:

Name:       Data    format FinCenter
Class:   POSIXct character character
```

where the slot `Data` contains the timestamps in the `POSIXct` class, `format` is the format typically applied to `Data`, and `FinCenter` is the financial center.

Note: we use the abbreviation GMT equivalently to UTC, the universal time coordinated.

[1] The original data set of the world map with time zones is available at http://efele.net/maps/tz/world/. Full and reduced `rda` versions were kindly contributed by Roger Bivand.

"timeDate" object creation

There are different ways to generate a "timeDate" object. It can be generated using either timeDate(), timeSequence(), or timeCalendar().

The function timeDate() creates a "timeDate" object from scratch. It requires a character vector of timestamps and optional arguments to specify the format of this vector and the financial center. The financial center can be specified, as mentioned, via setRmetricsOptions() or (more cleanly) with the argument FinCenter. By default, it is set to GMT[2].

In the following, we illustrate the creation of "timeDate" objects as well as the method used to convert timestamps from different time zones.

We first create character vectors of two timestamps with the default financial center (GMT):

```
> Dates <- c("2009-09-28","2010-01-15")
> Times <- c(  "23:12:55",  "10:34:02")
> charvec <- paste(Dates, Times)
> getRmetricsOption("myFinCenter")
myFinCenter
      "GMT"
> timeDate(charvec)
GMT
[1] [2009-09-28 23:12:55] [2010-01-15 10:34:02]
```

As a second step, we set the local financial center to Zurich and create a "timeDate" object.

```
> setRmetricsOptions(myFinCenter = "Zurich")
> timeDate(charvec)
Zurich
[1] [2009-09-28 23:12:55] [2010-01-15 10:34:02]
```

The third example shows how the timestamps can be conveniently converted into different time zones; charvec is recorded in Tokyo and subsequently converted to our local center, i.e., Zurich (see above):

```
> timeDate(charvec, zone = "Tokyo")
Zurich
[1] [2009-09-28 16:12:55] [2010-01-15 02:34:02]
```

or converted from Zurich to New York:

```
> timeDate(charvec, zone = "Zurich",
+          FinCenter = "NewYork")
NewYork
[1] [2009-09-28 17:12:55] [2010-01-15 04:34:02]
```

It is also possible to use the function finCenter() to view or modify the local center:

```
> td <- timeDate(charvec, zone = "Zurich",
+          FinCenter = "NewYork")
> finCenter(td)
[1] "NewYork"
> finCenter(td) <- "Zurich"
> td
Zurich
[1] [2009-09-28 23:12:55] [2010-01-15 10:34:02]
```

If the format of charvec is not specified, **timeDate** uses an automated date-time format identifier called whichFormat() that supports common date-time formats.

```
> whichFormat(charvec)
[1] "%Y-%m-%d %H:%M:%S"
```

The function timeSequence() creates a "timeDate" object representing an equidistant sequence of points in time. You can specify the range of dates with the arguments from and to. If from is missing, length.out defines the length of the sequence. In the case of a monthly sequence, you can define specific rules. For example, you can generate the sequence with the last days of the month or with the last or n-th Friday of every month. This can be of particular interest in financial applications.

Let us first reset the financial center to an international environment:

```
> setRmetricsOptions(myFinCenter = "GMT")
> # 'timeDate' is now in the financial center "GMT"
> timeDate(charvec)
GMT
[1] [2009-09-28 23:12:55] [2010-01-15 10:34:02]
```

A sequence of days or months can be created as follows:

```
> # first three days in January 2010,
> timeSequence(from = "2010-01-01",
+              to =   "2010-01-03", by = "day")
GMT
[1] [2010-01-01] [2010-01-02] [2010-01-03]
> # first 3 months in 2010:
> timeSequence(from = "2010-01-01",
+              to = "2010-03-31", by = "month")
GMT
[1] [2010-01-01] [2010-02-01] [2010-03-01]
```

The function timeCalendar() creates "timeDate" objects from calendar atoms. You can specify values or vectors of equal length denoting year, month, day, hour, minute, and seconds as integers. For example, the monthly calendar of the current year or a specific calendar in a given time zone can be created as follows:

```
> timeCalendar()
GMT
 [1] [2011-01-01] [2011-02-01] [2011-03-01]
 [4] [2011-04-01] [2011-05-01] [2011-06-01]
 [7] [2011-07-01] [2011-08-01] [2011-09-01]
[10] [2011-10-01] [2011-11-01] [2011-12-01]
```

[2]GMT can be considered as a "virtual" financial center.

The following represents the first four days of January recorded in Tokyo at local time "16:00" and converted to the financial center Zurich:

```
> timeCalendar(2010, m=1, d=1:4, h=16,
+              zone = "Tokyo", FinCenter = "Zurich")
Zurich
[1] [2010-01-01 08:00:00] [2010-01-02 08:00:00]
[3] [2010-01-03 08:00:00] [2010-01-04 08:00:00]
```

Midnight standard

The `"timeDate"` printing format is designed in accordance with the ISO-8601 (1988) standard. It uses the 24-hour clock format. Dates are expressed in the "%Y-%m-%d" format while time-dates are stated in the "%Y-%m-%d %H:%M:%S" format. A special case in the 24-hour clock system is the representation of midnight. It can be equivalently represented by "00:00" and "24:00". The former is usually used to denote the beginning of the day whereas the latter denotes the end. `timeDate` supports the midnight standard as described by the ISO-8601 (1988) standard as illustrated here:

```
> timeDate(ch <- "2010-01-31 24:00:00")
GMT
[1] [2010-02-01]
```

Note, following the revision of this paper, the R source has been amended in May 2011 to also allow "24:00" time specifications, such that a midnight standard will be part of standard R.

Financial centers – via daylight saving time rules

As mentioned earlier, the global financial center can be set with the function `setRmetricsOptions()` and accessed with the function `getRmetricsOption()`. Its default value is set to "`GMT`":

```
> getRmetricsOption("myFinCenter")
myFinCenter
      "GMT"
> # change to Zurich:
> setRmetricsOptions(myFinCenter = "Zurich")
```

From now on, all dates and times are handled in accordance with the Central European time zone and the DST rule for Zurich. Note that setting the financial center to a continent/city that lies outside the time zone used by your OS does not change any of the time settings or environment variables of your computer.

There are almost 400 financial centers supported thanks to the Olson database. They can be accessed by the function `listFinCenter()`, and partial lists can be extracted through the use of regular expressions.

```
> # first few financial centers:
> head(listFinCenter())
[1] "Africa/Abidjan"     "Africa/Accra"
[3] "Africa/Addis_Ababa" "Africa/Algiers"
[5] "Africa/Asmara"      "Africa/Bamako"
> # European centers starting with A or B:
> listFinCenter("Europe/[AB].*") # -> nine
[1] "Europe/Amsterdam"  "Europe/Andorra"
[3] "Europe/Athens"     "Europe/Belgrade"
[5] "Europe/Berlin"     "Europe/Bratislava"
[7] "Europe/Brussels"   "Europe/Bucharest"
[9] "Europe/Budapest"
```

Each financial center has an associated function that returns its DST rules in the form of a `"data.frame"`. These functions share the name of their financial center, e.g., `Zurich()`.

```
> Zurich()[64:67, ]
               Zurich offSet isdst TimeZone
64 2010-03-28 01:00:00   7200     1     CEST
65 2010-10-31 01:00:00   3600     0      CET
66 2011-03-27 01:00:00   7200     1     CEST
67 2011-10-30 01:00:00   3600     0      CET
      numeric
64 1269738000
65 1288486800
66 1301187600
67 1319936400
```

The returned `"data.frame"` shows when the clock was changed in Zurich, the offset in seconds with respect to GMT, a flag that tells us if DST is in effect or not, the time zone abbreviation, and the number of seconds since "1970-01-01" in GMT. The reader interested in the history of DST is referred to Bartky & Harrison (1979).

Note new centers can be easily added as long as their associated functions return a `"data.frame"` with the same structure as described above.

Holidays and calendars

Holidays are usually grouped by their origins. The first category, as the etymology suggests, is based on religious origin. For example, the ecclesiastical calendars of Christian churches are based on cycles of movable and immovable feasts. Christmas, December 25, is the principal immovable feast, whereas Easter is the principal movable feast. Most of the other dates are movable feasts that are determined with respect to Easter, Montes (1996). A second category of holidays is secular holidays, which denotes days that are celebrated internationally and in different cultures, such as Labor Day. Another category of holidays includes days that are relative to natural events. For example, the dates can be related to astronomical events such as cycles of the moon or the equinox. Moreover, there are also country-specific national holidays.

The calculation of holidays might prove tedious in some circumstances. Indeed, the estimation of the Easter date is a complex procedure with different algorithms involved in its computation. The algorithm implemented in the package is the one of Oudin (1940) as quoted in Seidelmann (1992). This approach is valid for any Gregorian calendar year. Further details about holiday calculation can be found in Tøndering (2008).

The dates of Easter for the next five years can be calculated with

```
> thisYear <- getRmetricsOption("currentYear")
> Easter(thisYear:(thisYear+5))
Zurich
[1] [2011-04-24] [2012-04-08] [2013-03-31]
[4] [2014-04-20] [2015-04-05] [2016-03-27]
```

The **timeDate** package includes functions for bank holidays in Canada, France, Germany, Great Britain, Italy, Japan[3], Switzerland, and the US. These holidays can be grouped in calendars. At the moment, the package provides functions for the New York stock exchange, `holidayNYSE()`; for the North American Reliability Council, `holidayNERC()`[4]; for the Toronto stock exchange `holidayTSX()`, and for Zurich, `holidayZURICH()`. Other calendars can be easily implemented given that the package already provides many holidays functions. A list of all holidays is provided in the appendix.

Logical test

It is possible to construct tests for weekdays, weekends, business days, and holidays with the functions `isWeekday()`, `isWeekend()`, `isBizday()` and `isHoliday()`, respectively.

Let us take a sequence of dates around Easter:

```
> Easter(2010)
Zurich
[1] [2010-04-04]
> (tS <- timeSequence(Easter(2010, -2),
+                     Easter(2010, +3)))
Zurich
[1] [2010-04-02] [2010-04-03] [2010-04-04]
[4] [2010-04-05] [2010-04-06] [2010-04-07]
```

We can now extract the weekdays or business days according to a holiday calendar.

```
> (tS1 <- tS[isWeekday(tS)])
Zurich
[1] [2010-04-02] [2010-04-05] [2010-04-06]
[4] [2010-04-07]
> (tS2 <- tS[isBizday(tS, holidayZURICH(2010))])
Zurich
[1] [2010-04-06] [2010-04-07]
> dayOfWeek(tS2)
2010-04-06 2010-04-07
     "Tue"      "Wed"
```

Thank to the comments of one of the referees, we have added a new argument, `wday`, in the functions `isWeekend()`, `isWeekday()`, `isBizday()` and `isHoliday()` that can be used to specify which days should be considered as business days. This is important when using calendars in Islamic countries or in Israel. By default, `wday` specifies the weekdays as Monday to Friday.

Special dates

As mentioned earlier, holidays often refer to a specific date or event. It is therefore crucial to have functions to compute: the first day in a given month, the last day in a given month and year, a given day before or after a date, the n-th occurrences of a day in a specified year/month, or a given last day for a specified year/month. We have summarized these functions in a table in the appendix.

In the following, we demonstrate how to retrieve the last day in each quarter or the second Sunday of each month. Note that days are numbered from 0 to 6 where 0 corresponds to Sunday and 6 to Saturday.

```
> charvec <- c("2011-03-01", "2011-04-01")
> # Last day in quarter
> timeLastDayInQuarter(charvec)
Zurich
[1] [2011-03-31] [2011-06-30]
> # Second Sunday of each month:
> timeNthNdayInMonth(charvec, nday = 0, nth = 2)
Zurich
[1] [2011-03-13] [2011-04-10]
> # Closest Friday that occurred before:
> timeNdayOnOrBefore(charvec, nday = 5)
Zurich
[1] [2011-02-25] [2011-04-01]
```

Operations on `"timeDate"` objects

Just like the other date-time classes in R, the `"timeDate"` class supports common operations. It allows for mathematical operations such as addition, subtraction, and comparisons to be performed. Moreover, methods for the generic functions to concatenate, replicate, sort, re-sample, unify, revert, or lag are available as the well known calls `c()`, `rep()`, `sort()`, `sample()`, `unique()`, `rev()`, and `diff()`, respectively. We spare the reader superfluous examples of functions that are common to other date-time classes. In the rest of the section, we emphasize methods that are not available for other classes or are not strictly identical. The reader is referred to the ebook "Chronological Objects with Rmetrics" (Würtz, Chalabi & Ellis, 2010) for more examples.

[3] The Japanese holidays were contributed by Parlamis Franklin.
[4] `holidayNERC()` was contributed by Joe W. Byers.

Subsetting methods

The **timeDate** package has different functions to subset a `timeDate` object. The usual function '`[`' extracts or replaces subsets of `"timeDate"` objects as expected. However, the package provides some additional functions. For example, the function `window()` extracts a sequence from a starting and ending point. The functions `start()` and `end()` extract the first and last timestamps, respectively.

Coercion methods

The package provides both `S3` and `S4` methods to coerce to and from `"timeDate"` objects. Below, we list the `S4` coercion methods available. The equivalent `S3` methods `as.*` are also provided, although the mixing of `S4` classes and `S3` methods is discouraged. Note that information can be lost in the coercion process if the destination class does not support the same functionality.

```
> showMethods("coerce", class = "timeDate")
Function: coerce (package methods)
from="ANY", to="timeDate"
from="Date", to="timeDate"
from="POSIXt", to="timeDate"
from="timeDate", to="Date"
from="timeDate", to="POSIXct"
from="timeDate", to="POSIXlt"
from="timeDate", to="character"
from="timeDate", to="data.frame"
from="timeDate", to="list"
from="timeDate", to="numeric"
```

We would like to point out a particular difference between the `as.numeric` methods of `"timeDate"` and `"POSIXct"` classes. Indeed, the `as.numeric-timeDate` method returns the time in minutes, in contrast to `as.numeric.POSIXct`, which returns the results in seconds. However, the `as.numeric-timeDate` method has an additional argument, `unit`, to select other units. These include seconds, hours, days, and weeks.

Concatenation method

The concatenation `c()` method for `"timeDate"` objects takes care of the different financial centers of the object to be concatenated. In such cases, all timestamps are transformed to the financial center of the first `"timeDate"` object. This feature is now also supported by R's `"POSIXct"` class. However, it was not available in previous versions of the class.

```
> ZH <- timeDate("2010-01-01 16:00", zone = "GMT",
+                FinCenter = "Zurich")
> NY <- timeDate("2010-01-01 18:00", zone = "GMT",
+                FinCenter = "NewYork")
> c(ZH, NY)
Zurich
[1] [2010-01-01 17:00:00] [2010-01-01 19:00:00]
> c(NY, ZH)
NewYork
[1] [2010-01-01 13:00:00] [2010-01-01 11:00:00]
```

Rounding and truncation methods

The rounding and truncation methods have similar functionalities to those of their counterparts in other date-time classes. However, the default `unit` argument is not the same.

Summary

The **timeDate** package offers functions for calendar manipulations for business days, weekends, and public and ecclesiastical holidays that are of interest in financial applications as well as in other fields. Moreover, the *Financial Center* concept facilitates the mixing of data collected in different time zones and the manipulation of data recorded in the same time zone but with different DST rules.

Acknowledgments

We thank the anonymous referees and the editor for their valuable comments as well as all R users and developers who have helped us to improve the **timeDate** package.

Bibliography

R.I. Bartky and E. Harrison. Standard and Daylight Saving Time. *Scientific American*, 240:46–53, 1979.

R. Bateman. Time Functionality in the Standard C Library, Novell AppNotes, September Issue, 73–85, 2000.

N. Dershowitz and E.M. Reingold. Calendrical Calculations. *Software - Practice and Experience*, 20:899–928, 1990.

N. Dershowitz and E.M. Reingold. *Calendrical Calculations*. Cambridge University Press, 1997.

G. Grothendieck and T. Petzoldt. Date and Time Classes in R. *R News*, 4(1):29-32, 2004.

ISO-8601. *Data Elements and Interchange Formats – Information Interchange, Representation of Dates and Time*. International Organization for Standardization, Reference Number ISO 8601, 1988.

M.J. Montes. Butcher's Algorithm for Calculating the Date of Easter in the Gregorian Calendar, 1996.

B.D. Ripley and K. Hornik. Date-Time Classes. *R-News*, 1(2):8–12, 2001.

P.K. Seidelmann (Editor). *Explanatory Supplement to the Astronomical Almanac*. University Science Books, Herndon, 1992.

C. Tøndering. Frequently Asked Questions about Calendars, 2008. URL http://www.tondering.dk/claus/calendar.html.

D. Würtz and Y. Chalabi with contribution from M. Maechler, J.W. Byers, and others. timeDate: Rmetrics - Chronological and Calendarical Objects, 2011. URL http://cran.r-project.org/web/packages/timeDate/.

D. Würtz, Y. Chalabi and A. Ellis Chronological Objects with Rmetrics, 2010. URL http://www.rmetrics.org/ebooks.

Yohan Chalabi
Finance Online GmbH, Zurich &
Institute of Theoretical Physics, ETH Zurich, Switzerland
chalabi@phys.ethz.ch

Martin Mächler
Seminar für Statistik, ETH Zurich, Switzerland
maechler@stat.math.ethz.ch

Diethelm Würtz
Institute of Theoretical Physics, ETH Zurich, Switzerland
wuertz@phys.ethz.ch

Appendix

Session Info

```
> toLatex(sessionInfo())
```

- R version 2.13.0 (2011-04-13), x86_64-apple-darwin10.7.0
- Locale: C ...
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: timeDate 2130.93

Tables

1. Special dates

timeFirstDayInMonth	First day in a given month
timeLastDayInMonth	Last day in a given month
timeFirstDayInQuarter	First day in a given quarter
timeLastDayInQuarter	Last day in a given quarter
timeNdayOnOrAfter	Day "on-or-after" n-days
timeNdayOnOrBefore	Day "on-or-before" n-days
timeNthNdayInMonth	N-th occurrence of a n-day in month
timeLastNdayInMonth	Last n-day in month

2. List of holidays

Advent1st	JPBunkaNoHi
Advent2nd	JPChildrensDay
Advent3rd	JPComingOfAgeDay
Advent4th	JPConstitutionDay
AllSaints	JPEmperorsBirthday
AllSouls	JPGantan
Annunciation	JPGreeneryDay
Ascension	JPHealthandSportsDay
AshWednesday	JPKeirouNOhi
AssumptionOfMary	JPKenkokuKinenNoHi
BirthOfVirginMary	JPKenpouKinenBi
BoxingDay	JPKinrouKanshaNoHi
CACanadaDay	JPKodomoNoHi
CACivicProvincialHoliday	JPKokuminNoKyujitu
CALabourDay	JPMarineDay
CAThanksgivingDay	JPMidoriNoHi
CAVictoriaDay	JPNatFoundationDay
CHAscension	JPNationHoliday
CHBerchtoldsDay	JPNationalCultureDay
CHConfederationDay	JPNewYearsDay
CHKnabenschiessen	JPRespectForTheAgedDay
CHSechselaeuten	JPSeijinNoHi
CaRemembranceDay	JPShuubunNoHi
CelebrationOfHolyCross	JPTaiikuNoHi
ChristTheKing	JPTennouTanjyouBi
ChristmasDay	JPThanksgivingDay
ChristmasEve	JPUmiNoHi
CorpusChristi	LaborDay
DEAscension	MassOfArchangels
DEChristmasEve	NewYearsDay
DECorpusChristi	PalmSunday
DEGermanUnity	Pentecost
DENewYearsEve	PentecostMonday
Easter	PresentationOfLord
EasterMonday	Quinquagesima
EasterSunday	RogationSunday
Epiphany	Septuagesima
FRAllSaints	SolemnityOfMary
FRArmisticeDay	TransfigurationOfLord
FRAscension	TrinitySunday
FRAssumptionVirginMary	USCPulaskisBirthday
FRBastilleDay	USChristmasDay
FRFetDeLaVictoire1945	USColumbusDay
GBBankHoliday	USDecorationMemorialDay
GBMayDay	USElectionDay
GBMilleniumDay	USGoodFriday
GBSummerBankHoliday	USInaugurationDay
GoodFriday	USIndependenceDay
ITAllSaints	USLaborDay
ITAssumptionOfVirginMary	USLincolnsBirthday
ITEpiphany	USMLKingsBirthday
ITImmaculateConception	USMemorialDay
ITLiberationDay	USNewYearsDay
ITStAmrose	USPresidentsDay
JPAutumnalEquinox	USThanksgivingDay
JPBankHolidayDec31	USVeteransDay
JPBankHolidayJan2	USWashingtonsBirthday
JPBankHolidayJan3	