Rmetrics - timeDate Package

by Yohan Chalabi, ETH Zurich, Finance Online Zurich
Martin Mächler, Seminar für Statistik, ETH Zurich
Diethelm Würtz, Institute of Theoretical Physics, ETH Zurich

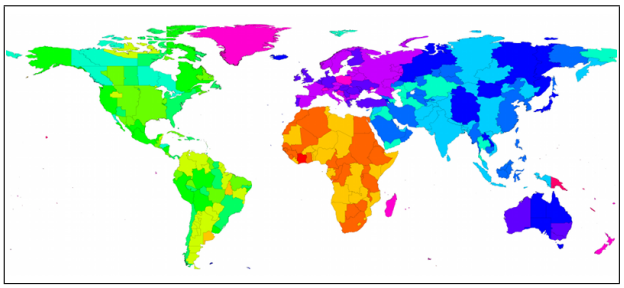

Figure 1: World map with major time zones. [1]

We describe the concepts and methods behind the S4 class `timeDate` used in **Rmetrics** for financial data with time and holiday's management.

The `timeDate` class represents calendar dates and times as follows:

```
> library(timeDate)
> showClass("timeDate")

Class "timeDate" [package "timeDate"]

Slots:

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

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

The `timeDate` class can be summarised as a combination of

- `POSIXct` timestamp objects always in the GMT time zone
- in the human readable ISO-8601 format standard which expresses dates as "%Y-%m-%d" and dates/times as "%Y-%m-%d %H:%M:%S"
- with a financial center used for the time zone and daylight saving rules (DST).

The definition of `timeDate` objects fulfils the conventions of the ANSI C and POSIX standard and the ISO 8601 standard. Additionally, **Rmetrics** has added the concept of a "Financial Center". This allows mixing data collected in different time zones or handling historical data recorded in the same time zone but with different DST rules. The local financial center can be defined via `setRmetricsOptions(myFinCenter = ....)` and refers, by default, to Greenwich Mean Time (GMT).

Moreover, **timeDate** offers sophisticated calendar manipulations for business days, weekends, public and ecclesiastical holidays. This allows handling day count conventions and rolling business conventions according to the rules of ICMA[2], ISDA[3], and SIFMA[4].

In the following sections, we present functions of the **timeDate** package. First, the creation of `timeDate` objects is explained. Second, operations on `timeDate` objects such as mathematical operations, rounding, subsetting and coercions are discussed. Third, the use of a financial center with DST rules is described. Finally, the management of holidays is explained.

How to Create a `timeDate` Object

There are different ways to generate a `timeDate` object. One can either use `timeDate()`, `timeSequence()` or `timeCalendar()`.

The function `timeDate()` creates a `timeDate` object from scratch. It requires a character vector of timestamps with its financial center. The financial center (FC) can be specified, as mentioned, via `setRmetricsOptions()` or (more cleanly) with the argument `FinCenter`. By default, it is set to GMT[5].

```
> # Character Vectors of Dates and Times:
> Dates <- c("2008-09-28","2009-01-15")
> Times <- c(  "23:12:55",  "10:34:02")
> charvec <- paste(Dates, Times)
> # Show my local financial centre
> # by default this is "GMT"
> getRmetricsOptions("myFinCenter")

myFinCenter
      "GMT"

> # Create a 'timeDate' object
> timeDate(charvec)

GMT
[1] [2008-09-28 23:12:55] [2009-01-15 10:34:02]

> # Create a 'timeDate' with my financial center
> # set to Zurich
> setRmetricsOptions(myFinCenter = "Zurich")
> timeDate(charvec)

Zurich
[1] [2008-09-28 23:12:55] [2009-01-15 10:34:02]

> # If 'timeDate' was recorded in a different
> # financial center, it will be automatically
> # converted to your local center, i.e. "Zurich".
> timeDate(charvec, zone = "Tokyo")

Zurich
[1] [2008-09-28 16:12:55] [2009-01-15 02:34:02]
```

[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.
[2] International Capital Market Association
[3] International Swaps and Derivatives Association
[4] Securities Industry and Financial Markets Association
[5] GMT can be considered as a "virtual" financial center

```
> # You can also convert a recorded 'timeDate'
> # from your financial center "Zurich" to
> # another one, for example "NewYork".
> timeDate(charvec, zone = "Zurich",
+          FinCenter = "NewYork")

NewYork
[1] [2008-09-28 17:12:55] [2009-01-15 04:34:02]
```

Note: timeDate uses an automated date/time format identifier for many common date/time formats. Indeed, `whichFormat()` tries to recognise the format of a timestamp character vector.

```
> whichFormat(charvec)

[1] "%Y-%m-%d %H:%M:%S"
```

Note: timeDate always uses the midnight standard, i.e. `00:00:00` marks the beginning of each day.

```
> midnightStandard("2009-01-31 24:00:00")

[1] "2009-02-01 00:00:00"
```

The function `timeSequence()` creates a `timeDate` representing an arithmetic 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's work in an international environment:
> setRmetricsOptions(myFinCenter = "GMT")
> # 'timeDate' is now in the financial center "GMT"
> timeDate(charvec)

GMT
[1] [2008-09-28 23:12:55] [2009-01-15 10:34:02]

> # Daily January 2009 Sequence:
> timeSequence(from = "2009-01-01",
+              to = "2009-01-31", by = "day")

GMT
 [1] [2009-01-01] [2009-01-02] [2009-01-03] [2009-01-04]
 [5] [2009-01-05] [2009-01-06] [2009-01-07] [2009-01-08]
 [9] [2009-01-09] [2009-01-10] [2009-01-11] [2009-01-12]
[13] [2009-01-13] [2009-01-14] [2009-01-15] [2009-01-16]
[17] [2009-01-17] [2009-01-18] [2009-01-19] [2009-01-20]
[21] [2009-01-21] [2009-01-22] [2009-01-23] [2009-01-24]
[25] [2009-01-25] [2009-01-26] [2009-01-27] [2009-01-28]
[29] [2009-01-29] [2009-01-30] [2009-01-31]

> # Monthly 2009 Sequence:
> tS <- timeSequence(from = "2009-01-01",
+                    to = "2009-12-31", by = "month")
> tS

GMT
 [1] [2009-01-01] [2009-02-01] [2009-03-01] [2009-04-01]
 [5] [2009-05-01] [2009-06-01] [2009-07-01] [2009-08-01]
 [9] [2009-09-01] [2009-10-01] [2009-11-01] [2009-12-01]

> # Do you want the last day or the last Friday of
> # the  month Data
> timeLastDayInMonth(tS)

GMT
 [1] [2009-01-31] [2009-02-28] [2009-03-31] [2009-04-30]
 [5] [2009-05-31] [2009-06-30] [2009-07-31] [2009-08-31]
 [9] [2009-09-30] [2009-10-31] [2009-11-30] [2009-12-31]

> timeLastNdayInMonth(tS, nday = 5)

GMT
 [1] [2009-01-30] [2009-02-27] [2009-03-27] [2009-04-24]
 [5] [2009-05-29] [2009-06-26] [2009-07-31] [2009-08-28]
 [9] [2009-09-25] [2009-10-30] [2009-11-27] [2009-12-25]
```

The function `timeCalendar()` creates `timeDate` objects from calendar atoms. You can specify values or vectors of equal length of integers denoting year, month, day, hour, minute and seconds.

```
> # Monthly calendar for Current Year
> timeCalendar()

GMT
 [1] [2009-01-01] [2009-02-01] [2009-03-01] [2009-04-01]
 [5] [2009-05-01] [2009-06-01] [2009-07-01] [2009-08-01]
 [9] [2009-09-01] [2009-10-01] [2009-11-01] [2009-12-01]

> # Daily 'timeDate' for January data
> # from Tokyo local time 16:00
> timeCalendar(2009, m=1, d=1:4, h=16,
+              zone = "Tokyo", FinCenter = "Zurich")

Zurich
[1] [2009-01-01 08:00:00] [2009-01-02 08:00:00]
[3] [2009-01-03 08:00:00] [2009-01-04 08:00:00]

> # Or add 16 hours in seconds ...
> timeCalendar(2009, m=1, d=1:4, zone="Tokyo",
+              FinCenter="Zurich") + 16*3600

Zurich
[1] [2009-01-01 08:00:00] [2009-01-02 08:00:00]
[3] [2009-01-03 08:00:00] [2009-01-04 08:00:00]
```

Operations on `timeDate` Objects

As exemplified in the above code, '`... + 16*3600`', many operations can be performed on `timeDate` objects. For example you can add and subtract, round and truncate, subset, coerce or transform them to other objects. These are only a few of the many available options.

Math Operations

Mathematical operations such as addition, subtraction and comparisons can be performed on `timeDate` objects.

```
> # Date and time "now" :
> now <- Sys.timeDate(); now

GMT
[1] [2009-05-20 21:48:30]

> # and an hour later:
> now + 3600

GMT
[1] [2009-05-20 22:48:30]

> # Which date/time is earlier or later ?
> tC <- timeCalendar(m = 1:6)
> tR <- tC + round(3600*rnorm(6))
> tR > tC

[1] FALSE  TRUE FALSE  TRUE FALSE  TRUE
```

Lagging

The `timeDate` method for `diff()` returns suitably lagged and iterated differences.

```
> # Monthly Dates 2009 and January 2009:
> tC <- c(timeCalendar(2009), timeCalendar(2010, m=1))
> diff(tC)# = number of days of months

Time differences in days
 [1] 31 28 31 30 31 30 31 31 30 31 30 31
attr(,"tzone")
[1] "GMT"

> # Number of days in  total 2009:
> sum(as.integer(diff(tC)))

[1] 365
```

Rounding and Truncating

Dates and times can be rounded or truncated. This can be useful in the case of a frequency lower than seconds, for example hours. To do this you can use the functions `round()` and `trunc()`.

```
> # Round the random time stamps to the nearest hour:
> tC <- timeCalendar(m = 1:4)
> tR <- tC + round(3600*rnorm(4))
> tR

GMT
[1] [2009-01-01 00:50:16] [2009-02-01 00:43:58]
[3] [2009-02-28 23:55:45] [2009-03-31 23:38:14]

> round(tR, "h")

GMT
[1] [2009-01-01 01:00:00] [2009-02-01 01:00:00]
[3] [2009-03-01 00:00:00] [2009-04-01 00:00:00]

> # Truncate by hour or to the next full hour:
> trunc(tR, "h")

GMT
[1] [2009-01-01 00:00:00] [2009-02-01 00:00:00]
[3] [2009-02-28 23:00:00] [2009-03-31 23:00:00]

> trunc(tR + 3600, "h")

GMT
[1] [2009-01-01 01:00:00] [2009-02-01 01:00:00]
[3] [2009-03-01 00:00:00] [2009-04-01 00:00:00]
```

Subsetting

Subsetting is a very important issue in the management of dates and times.

The **timeDate** package has different functions to subset a `timeDate` object. `'['` extracts or replaces subsets of `timeDate` objects, `window()` extracts a piece from a `timeDate` object, and the functions `start()` and `end()` extract the first and last timestamps, respectively.

```
> # Create Monthly Calendar for next year
> cY <- getRmetricsOptions("currentYear")
> tC <- timeCalendar(cY + 1)
> tC

GMT
 [1] [2010-01-01] [2010-02-01] [2010-03-01] [2010-04-01]
 [5] [2010-05-01] [2010-06-01] [2010-07-01] [2010-08-01]
 [9] [2010-09-01] [2010-10-01] [2010-11-01] [2010-12-01]

> # Start, end and length of 'timeDate' objects
> start(tC)

GMT
[1] [2010-01-01]

> end(tC)

GMT
[1] [2010-12-01]

> length(tC)

[1] 12

> # The first Quarter - Several Alternative Solutions:
> tC[1:3]

GMT
[1] [2010-01-01] [2010-02-01] [2010-03-01]

> tC[-(4:length(tC))]

GMT
[1] [2010-01-01] [2010-02-01] [2010-03-01]

> window(tC, start = tC[1], end = tC[3])

GMT
[1] [2010-01-01] [2010-02-01] [2010-03-01]

> cut(tC, from = tC[1], to = tC[3])

GMT
[1] [2010-01-01] [2010-02-01] [2010-03-01]

> tC[tC < tC[4]]

GMT
[1] [2010-01-01] [2010-02-01] [2010-03-01]

> # The Quarterly Series:
> tC[seq(3, 12, by = 3)]

GMT
[1] [2010-03-01] [2010-06-01] [2010-09-01] [2010-12-01]
```

Logical Test

Weekdays, weekends, business days and holidays can be tested with the functions: `isWeekday()`, `isWeekend()`, `isBizday()` and `isHoliday()`.

```
> # A 'timeDate' Sequence around Easter 2009
> Easter(2009)

GMT
[1] [2009-04-12]

> tS <- timeSequence(Easter(2009, -6),
+                    Easter(2009, +7))
> tS

GMT
 [1] [2009-04-06] [2009-04-07] [2009-04-08] [2009-04-09]
 [5] [2009-04-10] [2009-04-11] [2009-04-12] [2009-04-13]
 [9] [2009-04-14] [2009-04-15] [2009-04-16] [2009-04-17]
[13] [2009-04-18] [2009-04-19]

> # Subset weekdays and business days:
> (tW <- tS[isWeekday(tS)])

GMT
 [1] [2009-04-06] [2009-04-07] [2009-04-08] [2009-04-09]
 [5] [2009-04-10] [2009-04-13] [2009-04-14] [2009-04-15]
 [9] [2009-04-16] [2009-04-17]

> dayOfWeek(tW)

2009-04-06 2009-04-07 2009-04-08 2009-04-09 2009-04-10
     "Mon"      "Tue"      "Wed"      "Thu"      "Fri"
2009-04-13 2009-04-14 2009-04-15 2009-04-16 2009-04-17
     "Mon"      "Tue"      "Wed"      "Thu"      "Fri"

> (tB <- tS[isBizday(tS, holidayZURICH())])

GMT
[1] [2009-04-06] [2009-04-07] [2009-04-08] [2009-04-09]
[5] [2009-04-14] [2009-04-15] [2009-04-16] [2009-04-17]

> dayOfWeek(tB)

2009-04-06 2009-04-07 2009-04-08 2009-04-09 2009-04-14
     "Mon"      "Tue"      "Wed"      "Thu"      "Tue"
2009-04-15 2009-04-16 2009-04-17
     "Wed"      "Thu"      "Fri"
```

Coercions and Transformations

`timeDate` objects do not exist in an isolated world. Coercions and transformations allow `timeDate` objects to communicate with other formatted timestamps. But one should bear in mind that information can be lost if the other date/time class does not support the same functionality. The S4 methods to coerce to and from `timeDate` objects are

```
> showMethods("coerce", class = "timeDate")

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

Concatenation and Reordering

It might be sometimes useful to concatenate or reorder `timeDate` objects. The generic functions to concatenate, replicate, sort, re-sample, unify and revert `timeDate` objects are the well known functions `c()`, `rep()`, `sort()`, `sample()`, `unique()` and `rev()`.
Note: The `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 :

```
> # Concatenate the local time stamps to Zurich time ...
> ZH <- timeDate("2009-01-01 16:00:00",
+                zone = "GMT", FinCenter = "Zurich")
> NY <- timeDate("2009-01-01 18:00:00",
+                zone = "GMT", FinCenter = "NewYork")
> c(ZH, NY)

Zurich
[1] [2009-01-01 17:00:00] [2009-01-01 19:00:00]

> c(NY, ZH)

NewYork
[1] [2009-01-01 13:00:00] [2009-01-01 11:00:00]

> # Reorderings:
> tC <- timeCalendar(m = 1:4); tC

GMT
[1] [2009-01-01] [2009-02-01] [2009-03-01] [2009-04-01]

> tS <- sample(tC); tS

GMT
[1] [2009-01-01] [2009-04-01] [2009-02-01] [2009-03-01]

> tO <- sort(tS); tO

GMT
[1] [2009-01-01] [2009-02-01] [2009-03-01] [2009-04-01]

> tV <- rev(tO); tV

GMT
[1] [2009-04-01] [2009-03-01] [2009-02-01] [2009-01-01]

> tU <- unique(c(tS, tS)); tU

GMT
[1] [2009-01-01] [2009-04-01] [2009-02-01] [2009-03-01]
```

Financial Centers

Setting Financial Centers

The financial center can be set or changed by `setRmetricsOptions()` and accessed with the function `getRmetricsOptions()`. Its default value is `"GMT"`:

```
> # What is my current Financial Center ?
> getRmetricsOptions("myFinCenter")

myFinCenter
      "GMT"

> # Change to Zurich:
> setRmetricsOptions(myFinCenter = "Zurich")
```

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

List of Financial Centers

There are almost 400 financial centers supported by Rmetrics thanks to the Olson data base. They can be accessed by the function `listFinCenter()`. Partial lists can also be extracted and displayed using regular expressions:

```
> # All supported Financial Centers Worldwide:
> FCs <- listFinCenter()
> head(FCs) # the first few

[1] "Africa/Abidjan"     "Africa/Accra"
[3] "Africa/Addis_Ababa" "Africa/Algiers"
[5] "Africa/Asmara"      "Africa/Bamako"

> str(FCs)  # the "overview"

 chr [1:397] "Africa/Abidjan" "Africa/Accra" ...

> # European Financial 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"
```

DST Rules

Each financial center has an associated function which returns its daylight saving time rules (DST). Theses functions are named as their financial center name, e.g. `Zurich()`, and return a `data.frame`.

```
> # Show the (first 8) DST Rules for Zurich:
> head(Zurich(), 8)

               Zurich offSet isdst TimeZone     numeric
1 1901-12-14 20:45:52   3600     0      CET -2147397248
2 1941-05-05 00:00:00   7200     1     CEST  -904435200
3 1941-10-06 00:00:00   3600     0      CET  -891129600
4 1942-05-04 00:00:00   7200     1     CEST  -872985600
5 1942-10-05 00:00:00   3600     0      CET  -859680000
6 1981-03-29 01:00:00   7200     1     CEST   354675600
7 1981-09-27 01:00:00   3600     0      CET   370400400
8 1982-03-28 01:00:00   7200     1     CEST   386125200
```

This `data.frame` shows when the clock was changed in Zurich, the offset in seconds with respect to GMT, a flag which tells us if DST is in effect or not, the time zone abbreviation, and the number of seconds since 1970-01-01 in GMT.

Holidays Management

It is non-trivial to implement functions for business days, weekends and holidays. It is not difficult in an algorithmic sense, but it can become tedious to implement the calendar rules.

Special Dates

The **timeDate** package offers various functions to compute:

- the first day in a given month and year,
- the last day in a given month and year,
- the n-days before or after a given date,
- the n-th occurrences of the n-days for a specified year/month,
- or the last n-days for a specified year/month.

Note: n-days are numbered from 0 to 6 where 0 corresponds to Sunday and 6 to Saturday.

- `timeFirstDayInMonth()` computes the first day in a given month and year.
- `timeLastDayInMonth()` computes the last day in a given month and year and.
- `timeFirstDayInQuarter()` computes the first day in a given quarter and year.
- `timeLastDayInQuarter()` computes the last day in a given quarter and year.
- `timeNdayOnOrAfter()` computes dates that are "on-or-after" n-days.
- `timeNthNdayInMonth()` computes n-th occurrence of a n-day in year/month and `timeLastNdayInMonth()` computes the last n-day in year/month.

Holidays

Holidays may have two origins: ecclesiastical or public/federal. The ecclesiastical calendars of Christian churches are based on cycles of movable and immovable feasts. For example Christmas is the principal immovable festival and Easter is the principal movable festival. Most of the other movable festivals are related to Easter.

```
> # List of holidays available in Rmetrics
> listHolidays()
```

```
  [1] "Advent1st"                "Advent2nd"
  [3] "Advent3rd"                "Advent4th"
  [5] "AllSaints"                "AllSouls"
  [7] "Annunciation"             "Ascension"
  [9] "AshWednesday"             "AssumptionOfMary"
 [11] "BirthOfVirginMary"        "BoxingDay"
 [13] "CACanadaDay"              "CACivicProvincialHoliday"
 [15] "CALabourDay"              "CaRemembranceDay"
 [17] "CAThanksgivingDay"        "CAVictoriaDay"
 [19] "CelebrationOfHolyCross"   "CHAscension"
 [21] "CHBerchtoldsDay"          "CHConfederationDay"
 [23] "CHKnabenschiessen"        "ChristmasDay"
 [25] "ChristmasEve"             "ChristTheKing"
 [27] "CHSechselaeuten"          "CorpusChristi"
 [29] "DEAscension"              "DEChristmasEve"
 [31] "DECorpusChristi"          "DEGermanUnity"
 [33] "DENewYearsEve"            "Easter"
 [35] "EasterMonday"             "EasterSunday"
 [37] "Epiphany"                 "FRAllSaints"
 [39] "FRArmisticeDay"           "FRAscension"
 [41] "FRAssumptionVirginMary"   "FRBastilleDay"
 [43] "FRFetDeLaVictoire1945"    "GBBankHoliday"
 [45] "GBMayDay"                 "GBMilleniumDay"
 [47] "GBSummerBankHoliday"      "GoodFriday"
 [49] "ITAllSaints"              "ITAssumptionOfVirginMary"
 [51] "ITEpiphany"               "ITImmaculateConception"
 [53] "ITLiberationDay"          "ITStAmrose"
 [55] "JPAutumnalEquinox"        "JPBankHolidayDec31"
 [57] "JPBankHolidayJan2"        "JPBankHolidayJan3"
 [59] "JPBunkaNoHi"              "JPChildrensDay"
 [61] "JPComingOfAgeDay"         "JPConstitutionDay"
 [63] "JPEmperorsBirthday"       "JPGantan"
 [65] "JPGreeneryDay"            "JPHealthandSportsDay"
 [67] "JPKeirouNOhi"             "JPKenkokuKinenNoHi"
 [69] "JPKenpouKinenBi"          "JPKinrouKanshaNoHi"
 [71] "JPKodomoNoHi"             "JPKokuminNoKyujitu"
 [73] "JPMarineDay"              "JPMidoriNoHi"
 [75] "JPNatFoundationDay"       "JPNationalCultureDay"
 [77] "JPNationHoliday"          "JPNewYearsDay"
 [79] "JPRespectForTheAgedDay"   "JPSeijinNoHi"
 [81] "JPShuubunNoHi"            "JPTaiikuNoHi"
 [83] "JPTennouTanjyouBi"        "JPThanksgivingDay"
 [85] "JPUmiNoHi"                "LaborDay"
 [87] "MassOfArchangels"         "NewYearsDay"
 [89] "PalmSunday"               "Pentecost"
 [91] "PentecostMonday"          "PresentationOfLord"
 [93] "Quinquagesima"            "RogationSunday"
 [95] "Septuagesima"             "SolemnityOfMary"
 [97] "TransfigurationOfLord"    "TrinitySunday"
 [99] "USChristmasDay"           "USColumbusDay"
[101] "USCPulaskisBirthday"      "USDecorationMemorialDay"
[103] "USElectionDay"            "USGoodFriday"
[105] "USInaugurationDay"        "USIndependenceDay"
[107] "USLaborDay"               "USLincolnsBirthday"
[109] "USMemorialDay"            "USMLKingsBirthday"
[111] "USNewYearsDay"            "USPresidentsDay"
[113] "USThanksgivingDay"        "USVeteransDay"
[115] "USWashingtonsBirthday"
```

```
> # The date of Easter for the next 5 years:
> thisYr <- getRmetricsOptions("currentYear")
> Easter(thisYr:(thisYr+5))
```

```
Zurich
[1] [2009-04-12] [2010-04-04] [2011-04-24] [2012-04-08]
[5] [2013-03-31] [2014-04-20]
```

Holiday Calendars

One can easily build new holiday calendars. Three examples are provided in **timeDate** to help programmers to do so: `holidayZURICH()`, the Zurich holiday calendar, `holidayNYSE()`, the NYSE stock exchange

holiday calendar and `holidayTSX()`, the TSX holiday calendar.

Summary

The **timeDate** package offers calendar manipulations for business days, weekends, public and ecclesiastical holidays which might be 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.

Session Info

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

- R version 2.9.0 (2009-04-17), `i686-pc-linux-gnu`
- Locale: `LC_CTYPE=en_US.UTF-8;LC_NUMERI ...`
- Base packages: base, datasets, graphics, grDevices, methods, stats, utils
- Other packages: timeDate 290.85

Bibliography

Bartky R.I., Harrison E. (1979), Standard and Daylight Saving Time, Scientific American 240, pp. 46–53.

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

Becker R.A., Chambers J.M., Wilks A.R. (1988), The New S Language, Wadsworth and Brooks/Cole.

Deggett L.E. (1993), Explanatory Supplement to the Astronomical Almanac, Editor P. Kenneth Seidelmann, University Science Books, Herndon.

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

Dershowitz N., Reingold E.M. (1997), Calendrical Calculations, Cambridge University Press.

ICMA, International Capital Market Association, *www.icma-group.org*.

ISDA, International Swaps and Derivatives Association, Inc., *www.isda.org*.

ISO-8601, (1988), Data Elements and Interchange Formats – Information Interchange, Representation of Dates and Time, International Organization for Standardization, Reference Number ISO 8601, 14 pages.

James D.A., Pregibon D. (1992), Chronological Objects for Data Analysis, Reprint.

Montes M.J (1996), Butcher's Algorithm for Calculating the Date of Easter in the Gregorian Calendar, *no.spam.smart.net*.

R Development Core Team (2000), The 'chron' Package, *www.r-project.org*.

R Development Core Team (2000), The 'date' Package, *www.r-project.org*.

Ripley B.D., Hornik K. (2001); Date-Time Classes, R-News, Vol. 1/2 June 2001, 8–12.

SIFMA, Securities Industry and Financial Management Association, *www.sifma.org*.

Tondering C., (2005), Frequently Asked Questions about Calendars, *www.tondering.dk/claus/calendar.html*.

Therneau T. (1991), S-Plus Date Routines, *www.statlib.org*.

Wikipedia, Day Count Convention, *en.wikipedia.org/wiki/Day_count_convention*.