Introduction to the mondate Package

Dan Murphy

November 25, 2015

Abstract:

Dates and the results of date arithmetic are exchanged throughout the world using the international standards "Data elements and interchange formats — Information interchange — Representation of dates and times" (ISO 8601:2004) assembled by the International Organization for Standardization (ISO). When accountants, actuaries and other financial services analysts use the R Statistical Environment ("R") for creating models, they have access to tried-and-true implementations of those standards through R's offering of two "objects" ("classes") to the user: POSIXt and Date. Those two objects satisfy the vast majority of date/time user requirements, an important factor in R's rise in in popularity as the go-to computational tool.

Those base objects are not completely satisfactory, however. There are two key accounting concepts that are somewhat cumbersome to implement using base R alone: *accounting date* and *account aging in month units*.

The purpose of this paper is show a way to implement ISO-8601:2004-consistent solutions to those requirements in an R package.

Background

Base R – R version 3.2.2 (2015-08-14) as of this writing – uses two general classes for representing dates and times, `POSIXt` and `Date`.

The POSIXt class

The `POSIXt` class of objects was modeled after the POSIX standards.[1] R's implementation is in the form of two separate but related classes, `POSIXlt` and `POSIXct`.

The `POSIXlt` class (the "l" stands for "local", but "list" is also a useful mnemonic) is a list of a sufficient number of elements such as "year", "mon", "hour", "sec", etc. so as to completely determine an instant of time "to sub-second accuracy"[2] in the user's time zone. The `POSIXct` class (the "c" stands for "calendar time"[3]) is built around a `numeric` that holds the number of seconds since the beginning of January 1, 1970, not in the user's time zone but in the UTC time zone.

`POSIXct` objects are more "compact" (another mnemonic) and, as such, the more suitable for storage in data.frames. However, it is the `POSIXlt` class of objects that is used "behind the scenes" when communicating with the user via character representations of dates and times. The technical distinctions between the two classes are generally of little concern to the user. However, R's handling of time-zones when communicating with the user can be somewhat confusing. Here is an example.

Let us define the object `a` to be the first instant of 1970, which of course occurs zero seconds "after the beginning of 1970." We will store that instant as a `POSIXct` object without regard to the time zone.

[1] IEEE Std 1003.1, a.k.a. "POSIX.1.", is a standard for computer operating systems that covers a variety of topics. More information for IEE Std 1003.1 can be found online; see for example http://www.opengroup.org/austin/papers/backgrounder.html.

[2] Ripley, B. D. and Hornik, K. (2001) Date-time classes. R News, 1/2, p. 9. http://www.r-project.org/doc/Rnews/Rnews_2001-2.pdf

[3] In R, see the help for any of these classes. R help on a topic can be read by invoking the function `help` at the console with the topic name in quotes, e.g., `help("POSIXt")`. A shorthand version uses the question mark alone – `?POSIXt` – often found in the footnotes herein.

```
a <- as.POSIXct(0, origin="1970-01-01")
print(a)
```

```
## [1] "1969-12-31 16:00:00 PST"
```

Note that when printing the value of `a` at the console, R displays that value in the author's local time zone (Pacific Standard Time), which appears as hour 16 of December 31, 1969. It is true when the year 1970 came into being in the UTC time zone it was still 8pm December 31st on the California coast, but that was not how `a` was defined so that is not how one might expect `a` to appear when printed. The reason `a` displays in local (PST) time is that R uses the POSIXlt class at the intermediary between POSIXct and the character representation to the user, which, as mentioned previously, defaults to the user's local time zone. To confirm for yourself that `a` really does hold the value "zero seconds after the beginning of 1970", simply strip away the accoutrements surrounding `a` with the `unclass` function:

```
unclass(a)
```

```
## [1] 0
## attr(,"tzone")
## [1] ""
```

So "under the hood" `a` is simply the number 0 (zero) with a "tzone" attribute (which R uses to drive its date/time "class" behavior). To force R to represent `a` in the UTC time zone, one option is to use the `format` function with the `tz` argument as follows:[4]

```
format(a, tz="UTC")
```

```
## [1] "1970-01-01"
```

In the remainder of this paper it will be assumed that the timezone is

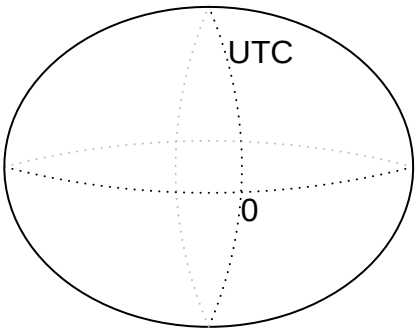

Figure 1: UTC

The Date Class

The `Date` class in R holds the number of days since the beginning of January 1, 1970.

It will be important to visit a few "basic concepts" that people understand intuitively but must be defined precisely if computers are to understand too:

[4]Other options exist. For a good summary, see David Smith's blog "Converting time zones in R: tips, tricks and pitfalls," *Revolutions*, June 02, 2009, http://blog.revolutionanalytics.com/2009/06/converting-time-zones.html. See also the `with_tz` function in the `lubridate` package on CRAN: https://cran.r-project.org/web/packages/lubridate/index.html

- time interval
- duration
- other?

time interval

> **Definition:** time interval
> A time interval is "part of the time axis limited by two instants. [It] comprises all instants between the two limiting instants and, unless otherwise stated, the limiting instants themselves."[5]

Figure 1 below is a diagram of two time intervals A and B. Each includes its respective endpoints as indicated by the square brackets. A and B overlap at time point 1.

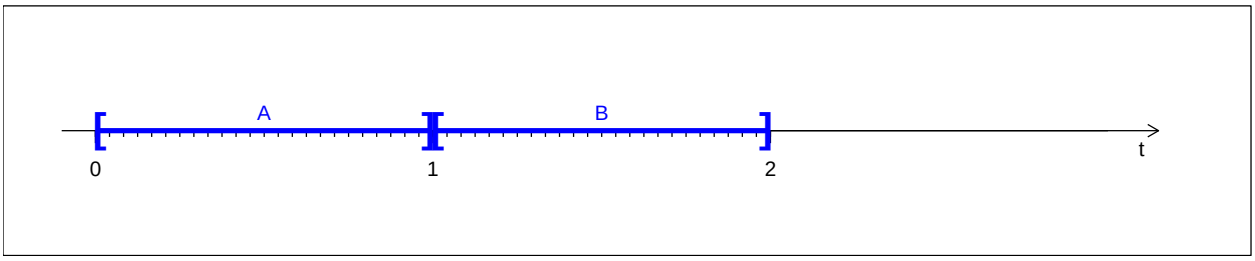

Figure 2: generic time intervals A and B

Time intervals are generally endowed with more evocative names than "A" and "B", however. For example, if Figure 1 is a diagram of the first few days of calendar year 1970, then more descriptive names for calendar days A and B would be "1970-01-01" and "1970-01-02", using ISO 8601's "complete representation" of a calendar date.[6] Those names also apply to all time instants enclosed by the endpoints. This leads to some ambiguity in this example because midnight between January 1st and 2nd is included in both intervals. Such ambiguity can be easily avoided, however, by leaving one of the endpoints "open."

In the case of calendar days, the question is how to represent midnight.

Midnight

Section 4.2.3 of ISO 8601 says that midnight between January 1st and 2nd, 1970 can be represented as either
Hour 00) 1970-01-02 00:00:00 or
Hour 24) 1970-01-01 24:00:00.
"The choice of representation ... will depend upon any association with a date, or a time interval."[7]

In the design of an experiment, it seeem natural to associate the beginning of the experiment as belonging to the first time period of the experiment. That suggests a time interval that is closed on the left and open on the right where the initiation of a time interval is included in the interval. Base R's POSIXt implementation follows the hour 00 approach.

POSIXt

Base R's "POSIXt" classes of objects store time as the number of seconds that have elapsed since the beginning of 1970. Midnight is the first instant of a calendar day. That is easily seen. Here we tell R to store midnight ending January 1, 1970 in the variable `a`. R is smart enough to understand the hour 24 notation.

[5] ISO 8601 2.1.3
[6] ISO 8601:2004 4.1.2.2
[7] ISO 8601:2004 4.2.3

However, when we ask R to print that value back to us at the console, it says that the time point represented in the variable `a` is a member of the second day of January.

```
a <- as.POSIXct("1970-01-01 24:00:00", tz = "UTC")
print(a)
```

```
## [1] "1970-01-02 UTC"
```

Figure 2 below is an illustration of a few of the calendar days surrounding the transition from 1969 to 1970, where the endpoints of the time intervals are represented by POSIXt objects. Since hour 00 begins each day, calendar days are represented by left-closed/right-open time intervals. The endpoints are modeled by POSIXt objects.

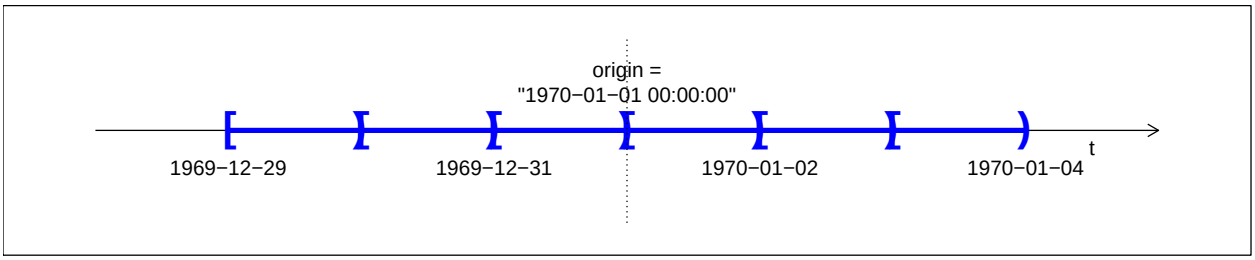

Figure 3: Calendar days surrounding January 1, 1970, time interval borders modeled by POSIXt instance `a`

R's left-closed/right-open paradigm runs consistently throughout its representation of time intervals. In Figure 3 below are shown five different time intervals in increasing magnitude. Calendar day, calendar month, and calendar year are defined terms in ISO 8601:2004. "Calendar second, minute and hour" are represented in the same spirit.

Each interval has a "label" (a "mark" in 8601 terminology, also called a "name" in R) corresponding to the instant beginning the interval. Note that all first intervals in Fugure 4 are labeled "1970-01-01". This is an example of R being "helpful": if the instant corresponds to midnight, R only shows the date.[8] For example, to show the full date and time for the first instant of 1970 try the format specifier "%Y-%m-%d %H:%M:%S".

```
FirstInstant <- as.POSIXlt(0, origin="1970-01-01", tz="UTC")
format(FirstInstant, usetz=FALSE)
```

```
## [1] "1970-01-01"
```

```
format(FirstInstant, format = "%Y-%m-%d %H:%M:%S", usetz=FALSE)
```

```
## [1] "1970-01-01 00:00:00"
```

Although R's choice of hour 00 is understandable and defensible, it is nevertheless arbitrary. The hour 24 choice could just as arbitrarily have been made, in which case the intervals in figure 2 would have been left-open/right-closed. Indeed, if Figure 2 were rotated around the origin, the resulting diagram would illustrate a sequence of left-open/right-closed time intervals.

Labeling an interval

rotate around origin – "dual"

"cut"

leap seconds occur at midnight beginning July 1st's UTC as necessary

[8] In R, see argument `format` under help for `strftime`.

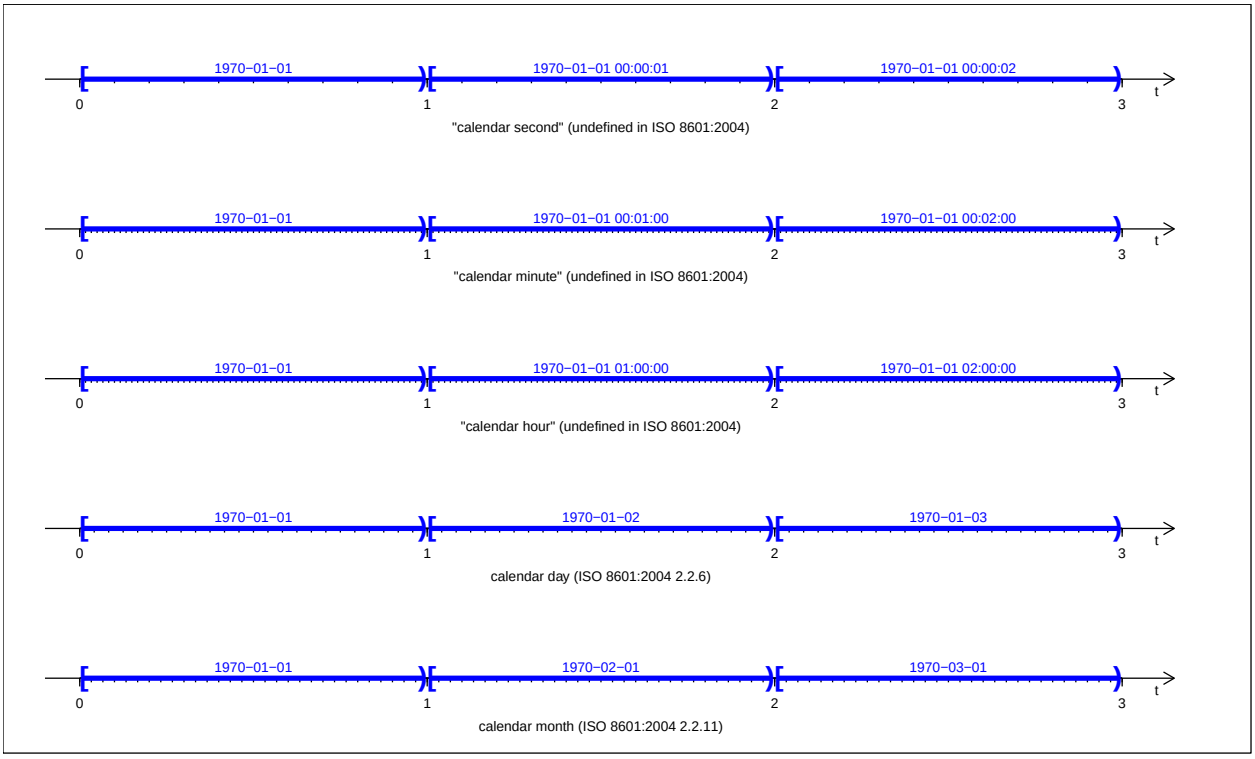

Figure 4: Different types of time intervals in R

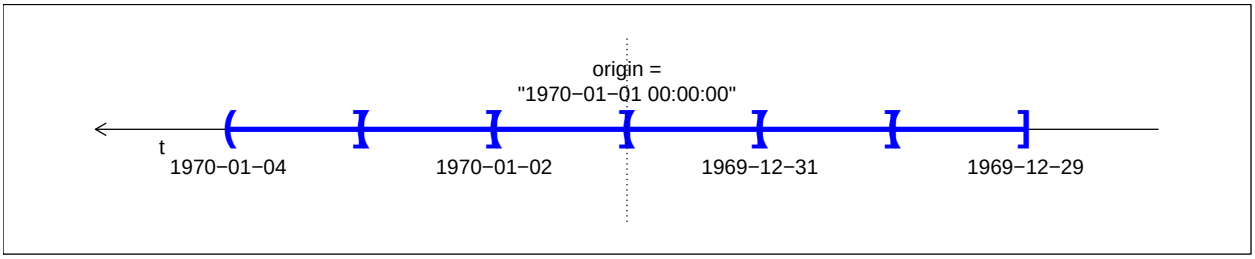

Figure 5: Calendar days surrounding January 1, 1970, from a different perspective

```
format(.leap.seconds, format = "%Y-%m-%d %H:%M:%S", tz="UTC",usetz=TRUE)
```

```
##  [1] "1972-07-01 00:00:00 UTC" "1973-01-01 00:00:00 UTC"
##  [3] "1974-01-01 00:00:00 UTC" "1975-01-01 00:00:00 UTC"
##  [5] "1976-01-01 00:00:00 UTC" "1977-01-01 00:00:00 UTC"
##  [7] "1978-01-01 00:00:00 UTC" "1979-01-01 00:00:00 UTC"
##  [9] "1980-01-01 00:00:00 UTC" "1981-07-01 00:00:00 UTC"
## [11] "1982-07-01 00:00:00 UTC" "1983-07-01 00:00:00 UTC"
## [13] "1985-07-01 00:00:00 UTC" "1988-01-01 00:00:00 UTC"
## [15] "1990-01-01 00:00:00 UTC" "1991-01-01 00:00:00 UTC"
## [17] "1992-07-01 00:00:00 UTC" "1993-07-01 00:00:00 UTC"
## [19] "1994-07-01 00:00:00 UTC" "1996-01-01 00:00:00 UTC"
## [21] "1997-07-01 00:00:00 UTC" "1999-01-01 00:00:00 UTC"
## [23] "2006-01-01 00:00:00 UTC" "2009-01-01 00:00:00 UTC"
## [25] "2012-07-01 00:00:00 UTC" "2015-07-01 00:00:00 UTC"
```

? strptime

Remember that in most time zones some times do not occur and some occur twice because of transitions to/from 'daylight saving' (also known as 'summer') time. strptime does not validate such times (it does not assume a specific time zone), but conversion by as.POSIXct will do so. Conversion by strftime and formatting/printing uses OS facilities and may return nonsensical results for non-existent times at DST transitions.

?DateTimeClasses

Unfortunately, the conversion is complicated by the operation of time zones and leap seconds (26 days have been 86401 seconds long so far, the last at the time of writing being added in 2015: the times of the extra seconds are in the object .leap.seconds). The details of this are entrusted to the OS services where possible. It seems that some rare systems used to use leap seconds, but all known current platforms ignore them (as required by POSIX). This is detected and corrected for at build time, so "POSIXct" times used by R do not include leap seconds on any platform.

A few times have specific issues. First, the leap seconds are ignored, and real times such as "2005-12-31 23:59:60" are (probably) treated as the next second. However, they will never be generated by R, and are unlikely to arise as input. Second, on some OSes there is a problem in the POSIX/C99 standard with "1969-12-31 23:59:59 UTC", which is -1 in calendar time and that value is on those OSes also used as an error code. Thus as.POSIXct("1969-12-31 23:59:59", format = "%Y-%m-%d %H:%M:%S", tz = "UTC") may give NA, and hence as.POSIXct("1969-12-31 23:59:59", tz = "UTC") will give "1969-12-31 23:59:00". Other OSes (including the code used by R on Windows) report errors separately and so are able to handle that time as valid.

Fractional seconds are printed only if options("digits.secs") is set: see strftime.

duration

> A duration is the "non-negative quantity ['magnitude'] attributed to a time interval, the value of which is equal to the difference between the time points of the final instant and the initial instant of the time interval."[9]

Think of a duration as the length of the vector ("magnitude") representing a time interval. In the figure below, the magnitude of the vector is one – second, day, month, etc. – depending on the context.

[9] ISO 8601 2.1.6

```r
par(mar=c(1,2,1,2))
plot(2, 0, xlim=c(0,5), ylim=c(-.5, .5), col="green", pch = 20, axes=F,
     xlab="", ylab="", main="duration")
arrows(x0=2, y0=0, x1=3, col="green", lwd =2)
box()
```

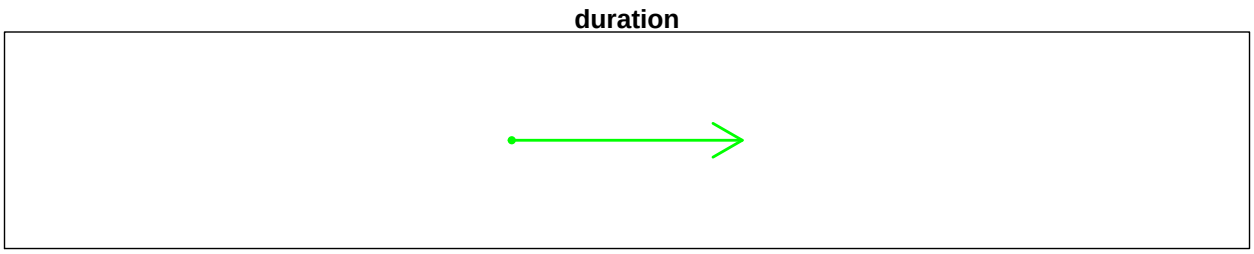

Figure 6:

Under the International System of Units (SI), the base unit of duration is "seconds". Indeed, whenever a duration is expressed in units other than "seconds" that length of time is referred to as a "nominal duration."[10] Examples of nominal durations are calendar day (its length in seconds can vary depending on leap seconds and daylight/standard time shifts), calendar month (its length in seconds varies, in addition to calendar day variability, due to differing numbers of calendar days in a month), and calendar year (in addition to calendar day variability, its length can vary due to the addition of a leap day). We will see examples of this behavior in R below.

seconds `seconds` are the base unit for expressing duration.

As pertains R, that means ages need not rely on classes as granular as POSIXt; the `Date` class, which keeps track of calendar days, should suffice.

When a contract specifies an inception date but no time, e.g., an insurance policy, for the event E corresponding to the financial responsibilities incepting therein, it is customary to consider those responsibilities to start at hour 00. When considering the age of the aggregation of events occurring during a time interval, financial and accounting models will "tag" those events with the same instant of inception. Two general instants can be found, particularly in the actuarial literature:

1. the beginning instant of the interval
2. the midpoint of the interval.

It would seem that the calculation of an age in units of "days" would be straightforward, but there are a couple of complicating factors. First, ISO 8601 defines a "day" to be the unit of time equal to exactly 24 hours. It defines a "calendar day" to be the interval of time between successive midnights. Those calendar day intervals are usually one day in length, but for two exceptions:

1. Leap seconds: Every once in a while a second must be added to or subtracted from Coordinated Universal Time (UTC) to realign UTC solar time (UT1).
2. Daylight time: Frequently the clock is adjusted by local authorities.

TRUE TRUE

30

[10] ISO 8601 2.1.7

30.0416667

30.0416667

2595600

TRUE

TRUE

Accounting Date

> Definition: Accounting Date
> An accounting date is the cutoff date for reflecting events and recording amounts as paid or unpaid in a financial statement or accounting system. The accounting date is sometimes referred to as the "as of" date.[11]

Although financial statements are always stated as of a date, say, December 31, 2015, it would be more thorough to assign a time as well, so as to identify all transactions impacting the financial statements as of that date. Some say "any particular moment on the 31st" could be used.[12] However, payments can potentially occur up to and including midnight December 31, 2015, so if a time designator is to be used, midnight would be the most appropriate.

Another reason for the closing instant to be the end of the day is for the purposes of date arithmetic. For example, the instant closing the month of December is 31 days from the beginning of December.

The International Organization for Standardization (ISO) sets the standards for computer representation of dates and times as a string of characters.[13] Regarding "midnight", the standards recognize that the instant 24:00:00 (hour 24) marking the end of one calendar day coincides with the instant 00:00:00 (hour 00) marking the start of the next calendar day. Furthermore, "the choice of [hour 24 or hour 00] will depend upon any association with a date, or a time interval. [Hour 24] representations are preferred to represent the *end* of a time interval."[14] The concept of the closing of the books at the end of a calendar year, calendar month or calendar day[15] suggests that hour 24 is preferred for representing an accounting date.

ISO 8601's representation of the accounting date referred to as "close of the 2015 calendar year" or "as of year end 2015" is therefore "2015-12-31 24:00:00".

On the other hand, base R's implementation of the 8601 standards[16] stores time as the number of seconds since the beginning of 1970, so midnight is represented as hour 00 of the beginning of each day. Therefore, the representation of "as of year end 2015" in R is "2016-01-01 00:00:00" which is January 1st when shortened to just the date.[17] However, to refer to the accounting date "as of year end 2015" with a label that uses the month "January" does not satisfactorily communicate the "as of December 31, 2015" accounting date concept.

It would be preferable that R and a user be able to communicate that concept using a label with "December" in it and also have that label refer to the instant in time separating calendar years 2015 and 2016 (`as.Date("2016-01-01")` in R).

That is the purpose of the "asof" class in the mondate package.

[11] Casualty Actuarial Society *Statement of Principles Regarding Property and Casualty Unpaid Claims Estimates*

[12] "it would be more accurate to write December 31, 20XX, 11:59:59, or any particular moment on the 31st." http://www.investopedia.com/university/accounting/accounting5.asp

[13] ISO 8601:2004, *Data elements and interchange formats – Information interchange – Representation of dates and times*, (c) ISO 2004

[14] ISO 8601 4.2.3. Emphasis added.

[15] defined as "time intervals" by ISO 8601

[16] The `POSIXt` classes. In R see ?DateTimeClasses. For additional background, see https://en.wikipedia.org/wiki/Unix_time

[17] ISO 8601 2.1.5 recognizes that a "date" can represent an instant in time as well as a span of time ("duration").

Elapsed Month: "emonth"

> Definition: Age
> The age of event E as of accounting date A is the length of time from the occurrence of event E to hour 24 on accounting date A.

Note that units are not mentioned in the definition of age. This is because the appropriate unit will be specific to the use case. In the vast majority of accounting cases, age is measured in units of days, months or years, not seconds.

It turns out that the definitions of units days, months and years in ISO 8601 is not particularly straightforward, with complexity in proportion to the width of the unit. Due to the fundamental monthly, quarterly, and yearly accounting cycles, it is generally not productive to used data defined more granularly than "month", but even that level of granularity has its complexities.

This paper adds to that complexity with another take on "month": "emonth."

There is little ambiguity in defining elapsed time (durations) in units of days. ISO 8601 defines a R's two Date/Time classes facilitate date/time arithmetic in uits of seconds and days.

- Since R's POSIXt classes store time as the number of seconds since the beginning of 1970,

so elapsed time in units of seconds is easily combined with a POSIXt object to yield another POSIXt object unambiguously.

- The Date class stores time as the number of days since the beginning of 1970, so elapsed time in units of days is easily combined with a Date object to yield another Date object unambiguously.

It is more complicated with units of months because month periods are comprised of different numbers of days. But this complication can be overcome with two key observations.

1. It is commonly accepted that the length of time between the beginning and end of a month is one "month".
2. Given any time t in any given month, there is a unique portion p, $0<=p<=1$, that represents the portion of the month completed by time t. Conversely, given any portion p, the point in time of a month completed as of that portion of the month can be easily found.

Those two observations give rise to the definition of the "month-time" of an instant in time:

> Definition: Month Time
> The month time of time t is a real number measuring the number of months since the beginning of 1970 to time t. The fractional part of the real number represents the portion of t's month to have expired by time t.

R has two classes, POSIXlt and POSIXct

Sentence B

Base R follows that standard and represents an instant of time as the number of seconds that have transpired since the beginning of 1970.

It is well known that "months" are comprised of different numbers of days. Although it is widely accepted that the length of time between the beginning and end of a month is one "month", it is also recognized that month durations cannot be defined consistently in units of days. Those observations give rise to a different type of unit: "elapsed month" or "emonth".

Definition: Elapsed Month ("emonth")
The length of time between two instants in time *t1* and *t2* in units of "emonths" is mondate(*t2*) - mondate(*t1*) where *mondate(t)* is a real number representing the number of months between the beginning of 1970 and time *t*.

Example
mondate("1970-02-01") = 1
mondate("1971-01-01") = 12 mondate("1970-02-15 00:00:00") = 1.5 because the beginning of February 15, 1970 is halfway through February 1970. as.Date("1971-01-01") - as.Date("1971-02-01") = 10.5 emonths

Conversely, given any real number t, it is straightforward to find the time ?YYYYMMDD?HH:MM:SS.zzzzz? that is t months away from the beginning of 1970 by first counting whole months, then counting into the next month (if necessary) the number of days and seconds corresponding to that month per the fractional value of t.

Defintion: Elapsed Years
The number of elapsed years between times *t1* and *t2* equals the number of elapsed months divided by 12.

Case Study:

ABC Ins. Co. started on 1/1/2010 to write earthquake insurance in California. As claims are made, ABC defines the occurrence date of a claim claim to be the date of the earthquake and stores that date in an ISO-8601 compliant object (`POSIXt`). ABC has had good luck so far – only 20 claims have been made. Here are their occurrence dates and their know values as of 12/31/2015:

data ?

To complete Sentence B, we need one more definition:

Definition: The age of accident year AY as of AOD is AOD - AY-01-01.

The accident year age of the 20 claims is the vector
mondate(?2015-12-31?) ? year(occurrenceDate)-01-01

http://smallbusiness.chron.com/differences-dates-between-balance-sheet-income-sheet-24881.html

"Balance Sheet Date A balance sheet often states that it is prepared as of a specific date, referred to as the balance sheet date. The balance sheet reports on a company's financial conditions, namely the values of the company's assets, liabilities and shareholders' equity. Values are measured in terms of their monetary amounts at particular points in time rather than over any periods. At the end of an accounting cycle, with the accounting books closed to recording new business transactions, companies can summarize their financial conditions as of the cycle's end."

http://www.casact.org/professionalism/standards/princip/SOP-Regarding-Property-and-Casualty-Unpaid-Claims-Estimates_Final%204-22-2015.pdf

Bibliography

http://www.timeanddate.com/time/gmt-utc-time.html

Greenwich Mean Time (GMT) is often interchanged or confused with Coordinated Universal Time (UTC). But GMT is a time zone and UTC is a time standard.

Although GMT and UTC share the same current time in practice, there is a basic difference between the two:

GMT is a time zone officially used in some European and African countries. The time can be displayed using both the 24-hour format (0 - 24) or the 12-hour format (1 - 12 am/pm). UTC is not a time zone, but a time standard that is the basis for civil time and time zones worldwide. This means that no country or territory officially uses UTC as a local time.

Universal time

https://en.wikipedia.org/wiki/Universal_Time#Versions

Universal Time (UT) is a time standard based on Earth's rotation. It is a modern continuation of Greenwich Mean Time (GMT), i.e., the mean solar time on the Prime Meridian at Greenwich. In fact, the expression "Universal Time" is ambiguous (when accuracy of better than a few seconds is required), as there are several versions of it, the most commonly used being Coordinated Universal Time (UTC) and UT1 (see below).[1] All of these versions of UT, except for UTC, are based on Earth's rotation relative to distant celestial objects (stars and quasars), but with a scaling factor and other adjustments to make them closer to solar time. UTC is based on International Atomic Time, with leap seconds added to keep it within 0.9 second of UT1.

Versions There are several versions of Universal Time:

UT0 is Universal Time determined at an observatory by observing the diurnal motion of stars or extragalactic radio sources, and also from ranging observations of the Moon and artificial Earth satellites. The location of the observatory is considered to have fixed coordinates in a terrestrial reference frame (such as the International Terrestrial Reference Frame) but the position of the rotational axis of the Earth wanders over the surface of the Earth; this is known as polar motion. UT0 does not contain any correction for polar motion. The difference between UT0 and UT1 is on the order of a few tens of milliseconds. The designation UT0 is no longer in common use.[11] UT1 is the principal form of Universal Time. While conceptually it is mean solar time at 0° longitude, precise measurements of the Sun are difficult. Hence, it is computed from observations of distant quasars using long baseline interferometry, laser ranging of the Moon and artificial satellites, as well as the determination of GPS satellite orbits. UT1 is the same everywhere on Earth, and is proportional to the rotation angle of the Earth with respect to distant quasars, specifically, the International Celestial Reference Frame (ICRF), neglecting some small adjustments. The observations allow the determination of a measure of the Earth's angle with respect to the ICRF, called the Earth Rotation Angle (ERA, which serves as a modern replacement for Greenwich Mean Sidereal Time). UT1 is required to follow the relationship ERA $= 2pi(0.7790572732640 + 1.00273781191135448\text{Tu})$ radians where Tu = (Julian UT1 date - 2451545.0)[12]

UTC (Coordinated Universal Time) is an atomic timescale that approximates UT1. It is the international standard on which civil time is based. It ticks SI seconds, in step with TAI. It usually has 86,400 SI seconds per day but is kept within 0.9 seconds of UT1 by the introduction of occasional intercalary leap seconds. As of 2015, these leaps have always been positive (the days which contained a leap second were 86,401 seconds long). Whenever a level of accuracy better than one second is not required, UTC can be used as an approximation of UT1. The difference between UT1 and UTC is known as DUT1.[15]

https://commons.wikimedia.org/wiki/File:World_Time_Zones_Map.png

By TimeZonesBoy (US Central Intelligence Agency) [Public domain], via Wikimedia Commons (Attribution not legally required)

David Smith's blog "Converting time zones in R: tips, tricks and pitfalls," *Revolutions*, June 02, 2009 http://blog.revolutionanalytics.com/2009/06/converting-time-zones.html

Lubridate
Grolemund, Garrett and Hadley Wickham, "Dates and Times Made Easy with lubridate", *Journal of Statistical Software*, April 2011, Volume 40, Issue 3.

Today is November 27, 2015

Ripley, Brian D. and Kurt Hornik, "Date-Time Classes", *R News*, Vol. 1/2, June 2001. [https://www.r-project.org/doc/Rnews/Rnews_2001-2.pdf]

POSIX.1
http://www.opengroup.org/austin/papers/backgrounder.html
http://www.opengroup.org/austin/papers/posix_faq.html

From ?strptime References

The POSIX 1003.1 standard, which is in some respects stricter than ISO 8601.

See LC_TIME in Base Definitions: Detailed ToC

file:///Users/danielmurphy/Downloads/susv4tc1/basedefs/toc.html

See folder susv4tc1 in mondate vignettes folder

file:///Users/danielmurphy/Downloads/susv4tc1/basedefs/V1_chap07.html#tag_07_03_05

Here's a pretty good paper about dates, Lubridate, Chron: http://biostat.mc.vanderbilt.edu/wiki/pub/Main/ColeBeck/datestimes.pdf