- Luke Tierney on namespaces and byte compilation
- Kurt Hornik on packaging, documentation, and testing
- Friedrich Leisch on S4 classes and methods
- Peter Dalgaard on language interfaces, using `.Call` and `.External`
- Douglas Bates on multilevel models in R
- Brian D. Ripley on datamining and related topics

Slides for the keynote addresses, along with abstracts of other papers presented at userR! 2004, are available at the conference web site, `http://www.ci.tuwien.ac.at/Conferences/useR-2004/program.html`.

An innovation of useR! 2004 was the deployment of “island” sessions, in place of the more traditional poster sessions, in which presenters with common interests made brief presentations followed by general discussion and demonstration. Several island sessions were conducted in parallel in the late afternoon on May 20 and 21, and conference participants were able to circulate among these sessions.

The conference sessions were augmented by a lively social program, including a pre-conference reception on the evening of May 19, and a conference trip, in the afternoon of May 22, to the picturesque Wachau valley on the Danube river. The trip was punctuated by a stop at the historic baroque Melk Abbey, and culminated in a memorable dinner at a traditional ‘Heuriger’ restaurant. Of course, much lively discussion took place at informal lunches, dinners, and pub sessions, and some of us took advantage of spring in Vienna to transform ourselves into tourists for a few days before and after the conference.

Everyone with whom I spoke, both at and after useR! 2004, was very enthusiastic about the conference. We look forward to an even larger useR! in 2006.

R Help Desk

Date and Time Classes in R

Gabor Grothendieck and Thomas Petzoldt

Introduction

R-1.9.0 and its contributed packages have a number of datetime (i.e. date or date/time) classes. In particular, the following three classes are discussed in this article:

- *Date*. This is the newest R date class, just introduced into R-1.9.0. It supports dates without times. Eliminating times simplifies dates substantially since not only are times, themselves, eliminated but the potential complications of time zones and daylight savings time vs. standard time need not be considered either. Date has an interface similar to the POSIX classes discussed below making it easy to move between them. It is part of the R base package. Dates in the Date class are represented internally as days since January 1, 1970. More information on Date can be found in `?Dates`. The percent codes used in character conversion with Date class objects can be found in `strptime` (although `strptime` itself is not a Date method).

- *chron*. Package **chron** provides dates and times. There are no time zones or notion of daylight vs. standard time in chron which makes it simpler to use for most purposes than date/time packages that do employ time zones. It is a contributed package available on CRAN so it must be installed and loaded prior to use. Datetimes in the **chron** package are represented internally as days since January 1, 1970 with times represented by fractions of days. Thus 1.5 would be the internal representation of noon on January 2nd, 1970. The **chron** package was developed at Bell Labs and is discussed in James and Pregibon (1993).

- *POSIX classes*. POSIX classes refer to the two classes POSIXct, POSIXlt and their common super class POSIXt. These support times and dates including time zones and standard vs. daylight savings time. They are primarily useful for time stamps on operating system files, data bases and other applications that involve external communication with systems that also support dates, times, time zones and daylight/standard times. They are also useful for certain applications such as world currency markets where time zones are important. We shall refer to these classes collectively as the POSIX classes. POSIXct datetimes are represented as seconds since January 1, 1970 GMT while POSIXlt datetimes are represented by a list of 9 components plus an optional tzone attribute. POSIXt is a common superclass of the two which is normally not accessed directly by the user. We shall focus mostly on POSIXct here. More information on the POSIXt classes are found in `?DateTimeClasses` and in `?strptime`. Additional insight can be obtained by inspecting the internals of POSIX datetime objects using `unclass(x)` where `x` is of any POSIX datetime class. (This works for Date and chron too). The POSIX classes are discussed in Ripley and Hornik (2001).

When considering which class to use, always choose the least complex class that will support the application. That is, *use Date if possible, otherwise use chron and otherwise use the POSIX classes*. Such a strategy will greatly reduce the potential for error and increase the reliability of your application.

A full page table is provided at the end of this article which illustrates and compares idioms written in each of the three datetime systems above.

Aside from the three primary date classes discussed so far there is also the **date** package, which represents dates as days since January 1, 1960. There are some date functions in the **pastecs** package that represent datetimes as years plus fraction of a year. The **date** and **pastecs** packages are not discussed in this article.

Other Applications

Spreadsheets like Microsoft Excel on a Windows PC or OpenOffice.org represent datetimes as days and fraction of days since December 30, 1899 (usually). If `x` is a vector of such numbers then `as.Date("1899-12-30") + floor(x)` will give a vector of Date class dates with respect to Date's origin. Similarly `chron("12/30/1899") + x` will give chron dates relative to chron's origin. Excel on a Mac usually represents dates as days and fraction of days since January 1, 1904 so `as.Date("1904-01-01") + floor(x)` and `chron("01/01/1904") + x` convert vectors of numbers representing such dates to Date and chron respectively. Its possible to set Excel to use either origin which is why the word *usually* was employed above.

SPSS uses October 14, 1582 as the origin thereby representing datetimes as seconds since the beginning of the Gregorian calendar. SAS uses seconds since January 1, 1960. `spss.get` and `sas.get` in package **Hmisc** can handle such datetimes automatically (Alzola and Harrell, 2002).

Time Series

A common use of dates and datetimes are in time series. The ts class represents regular time series (i.e. equally spaced points) and is mostly used if the frequency is monthly, quarterly or yearly. (Other series are labelled numerically rather than using dates.) Irregular time series can be handled by classes irts (in package **tseries**), its (in package **its**) and zoo (in package **zoo**). ts uses a scheme of its own for dates. irts and its use POSIXct dates to represent datetimes. zoo can use any date or timedate package to represent datetimes.

Input

By default, `read.table` will read in numeric data, such as `20040115`, as numbers and will read non-numeric data, such as `12/15/04` or `2004-12-15`, as factors. In either case, one should convert the data to character class using `as.character`. In the second (non-numeric) case one could alternately use the `as.is=` argument to `read.table` to prevent the conversion from character to factor within `read.table`[1].

```
# date col in all numeric format yyyymmdd
df <- read.table("laketemp.txt", header = TRUE)
as.Date(as.character(df$date), "%Y-%m-%d")

# first two cols in format mm/dd/yy hh:mm:ss
# Note as.is= in read.table to force character
library("chron")
df <- read.table("oxygen.txt", header = TRUE,
                 as.is = 1:2)
chron(df$date, df$time)
```

Avoiding Errors

The easiest way to avoid errors is to use the least complex class consistent with your application, as discussed in the Introduction.

With chron, in order to avoid conflicts between multiple applications, it is recommended that the user does not change the default settings of the four chron options:

```
# default origin
options(chron.origin=c(month=1,day=1,year=1970))
# if TRUE abbreviates year to 2 digits
options(chron.year.abb = TRUE)
# function to map 2 digit year to 4 digits
options(chron.year.expand = year.expand)
# if TRUE then year not displayed
options(chron.simplify = FALSE)
```

For the same reason, not only should the global chron origin not be changed but the per-variable origin should not be changed either. If one has a numeric vector of data x representing days since chron date `orig` then `orig+x` represents that data as chron dates relative to the default origin. With such a simple conversion there is really no reason to have to resort to non-standard origins. For example,

[1]This example uses data sets found at `http://www.tu-dresden.de/fghhihb/petzoldt/modlim/data/`.

```
orig <- chron("01/01/60")
x <- 0:9    # days since 01/01/60
# chron dates with default origin
orig + x
```

Regarding POSIX classes, the user should be aware of the following:

- *time zone*. Know which time zone each function that is being passed POSIX dates is assuming. Note that the time of day, day of the week, the day of the month, the month of the year, the year and other date quantities can potentially all differ depending on the time zone and whether it is daylight or standard time. The user *must* keep track of which time zone each function that is being used is assuming. For example, consider:

  ```
  dp <- seq(Sys.time(), len=10, by="day")
  plot(dp, 1:10)
  ```

  This does not use the current wall clock time for plotting today and the next 9 days since `plot` treats the datetimes as relative to GMT. The x values that result will be off from the wall clock time by a number of hours equal to the difference between the current time zone and GMT. See the plot example in table of this article for an illustration of how to handle this. Some functions accept a `tz=` argument, allowing the user to specify the time zone explicitly, but there are cases where the `tz=` argument is ignored. Always check the function with `tz=""` and `tz="GMT"` to be sure that `tz=` is not being ignored. If there is no `tz=` argument check carefully which time zone the function is assuming.

- *OS*. Some of the date calculations done by POSIX functions are passed off to the operating system and so may not work if the operating system has bugs with datetimes. In some cases the R code has compensated for OS bugs but in general *caveat emptor*. Another consequence is that different operating systems will accept different time zones. Normally the current time zone `""` and Greenwich Mean Time `"GMT"` can be assumed to be available but other time zones cannot be assumed to be available across platforms.

- *POSIXlt*. The `tzone` attribute on POSIXlt times are ignored so it is safer to use POSIXct than POSIXlt when performing arithmetic or other manipulations that may depend on time zones. Also, POSIXlt datetimes have an `isdst` component that indicates whether it is daylight savings time (`isdst=1`) or not (`isdst=0`). When converting from another class `isdst` may not be set as expected. Converting to character first and then to POSIXlt is safer than a direct conversion. Datetimes which are one hour off are the typical symptom of this last problem. Note that POSIXlt should not be used in data frames–POSIXct is preferred.

- *conversion*. Conversion between different datetime classes and POSIX classes may not use the time zone expected. Convert to character first to be sure. The conversion recipes given in the accompanying table should work as shown.

Comparison Table

Table 1 provides idioms written in each of the three classes discussed. The reader can use this table to quickly access the commands for those phrases in each of the three classes and to translate commands among the classes.

Bibliography

Alzola, C. F. and Harrell, F. E. (2002): *An Introduction to S and the Hmisc and Design Libraries*. URL `http://biostat.mc.vanderbilt.edu/twiki/pub/Main/RS/sintro.pdf`. 30

James, D. A. and Pregibon, D. (1993): Chronological objects for data analysis. In: *Proceedings of the 25th Symposium of the Interface*. San Diego. URL `http://cm.bell-labs.com/cm/ms/departments/sia/dj/papers/chron.pdf`. 29

Ripley, B. D. and Hornik, K. (2001): Date-time classes. *R News*, 1 (2), 8–11. URL `http://CRAN.R-project.org/doc/Rnews/`. 30

Gabor Grothendieck
`ggrothendieck@myway.com`

Thomas Petzoldt
`petzoldt@rcs.urz.tu-dresden.de`

	Date	chron	POSIXct
now	`Sys.Date()`	`as.chron(as.POSIXct(format(Sys.time()),tz="GMT"))`	`Sys.time()`
origin	`structure(0,class="Date")`	`chron(0)`	`structure(0,class=c("POSIXt","POSIXct"))`
x days since origin	`structure(floor(x),class="Date")`	`chron(x)`	`structure(x*24*60*60,class=c("POSIXt","POSIXct"))` # GMT days
diff in days	`dt1-dt2`	`dc1-dc2`	`difftime(dp1,dp2,unit="day")`[1]
time zone difference			`dp-as.POSIXct(format(dp,tz="GMT"))`
compare	`dt1 > dt2`	`dc1 > dc2`	`dp1 > dp2`
next day	`dt+1`	`dc+1`	`seq(dp0,length=2,by="DSTday")[2]`[2]
previous day	`dt-1`	`dc-1`	`seq(dp0,length=2,by="-1 DSTday")[2]`[3]
x days since date	`dt0+floor(x)`	dc0+x	`seq(dp0,length=2,by=paste(x,"DSTday")[2]`[4]
sequence	`dts <- seq(dt0,length=10,by="day")`	`dcs <- seq(dc0,length=10)`	`dps <- seq(dp0,length=10,by="DSTday")`
plot	`plot(dts,rnorm(10))`	`plot(dcs,rnorm(10))`	`plot(as.POSIXct(format(dps),tz="GMT"),rnorm(10))`[5]
every 2nd week	`seq(dt0,length=3,by="2 week")`	`dc0+seq(0,length=3,by=14)`	`seq(dp0,length=3,by="2 week")`
first day of month	`as.Date(format(dt,"%Y-%m-01"))`	`chron(dc)-month.day.year(dc)$day+1`	`as.POSIXct(format(dp,"%Y-%m-01"))`
month which sorts	`as.numeric(format(dt,"%m"))`	`months(dc)`	as.numeric(format(dp,"%m"))
day of week (Sun=0)	`as.numeric(format(dt,"%w"))`	`as.numeric(dates(dc)-3)%%7`	`as.numeric(format(dp,"%w"))`
day of year	`as.numeric(format(dt,"%j"))`	`as.numeric(format(as.Date(dc),"%j"))`	`as.numeric(format(dp,"%j"))`
mean each month/year	`tapply(x,format(dt,"%Y-%m"),mean)`	`tapply(x,format(as.Date(dc),"%Y-%m"),mean)`	`tapply(x,format(dp,"%Y-%m"),mean)`
12 monthly means	`tapply(x,format(dt,"%m"),mean)`	`tapply(x,months(dc),mean)`	`tapply(x,format(dp,"%m"),mean)`
Output			
yyyy-mm-dd	`format(dt)`	`format(as.Date(dates(dc)))`	`format(dp,"%Y-%m-%d")`
mm/dd/yy	`format(dt,"%m/%d/%y")`	`format(dates(dc))`	`format(dp,"%m/%d/%y")`
Sat Jul 1, 1970	`format(dt,"%a %b %d, %Y")`	`format(as.Date(dates(dc)),"%a %b %d, %Y")`	`format(dp,"%a %b %d, %Y")`
Input			
"1970-10-15"	`as.Date(z)`	`chron(z, format="y-m-d")`	`as.POSIXct(z)`
"10/15/1970"	`as.Date(z,"%m/%d/%Y")`	`chron(z)`	`as.POSIXct(strptime(z,"%m/%d/%Y"))`
Conversion (tz="")			
to Date		`as.Date(dates(dc))`	`as.Date(format(dp))`
to chron	`chron(unclass(dt))`		`chron(format(dp,"%m/%d/%Y"),format(dp,"%H:%M:%S"))`[6]
to POSIXct	`as.POSIXct(format(dt))`	`as.POSIXct(paste(as.Date(dates(dc)),times(dc)%%1))`	
Conversion (tz="GMT")			
to Date		`as.Date(dates(dc))`	`as.Date(dp)`
to chron	`chron(unclass(dt))`		`as.chron(dp)`
to POSIXct	`as.POSIXct(format(dt),tz="GMT")`	`as.POSIXct(dc)`	

Notes: z is a vector of characters. x is a vector of numbers. Variables beginning with dt are Date variables, Variables beginning with dc are chron variables and variables beginning with dp are POSIXct variables Variables ending in 0 are scalars; others are vectors. Expressions involving chron dates assume all chron options are set at default values. See strptime for more % codes

[1]Can be reduced to `dp1-dp2` if its acceptable that datetimes closer than one day are returned in hours or other units rather than days.
[2]Can be reduced to `dp+24*60*60` if one hour deviation at time changes between standard and daylight savings time are acceptable.
[3]Can be reduced to `dp-24*60*60` if one hour deviation at time changes between standard and daylight savings time are acceptable.
[4]Can be reduced to `dp+24*60*60*x` if one hour deviation acceptable when straddling odd number of standard/daylight savings time changes.
[5]Can be reduced to `plot(dps,rnorm(10))` if plotting points at GMT datetimes rather than current datetime suffices.
[6]An equivalent alternative is `as.chron(as.POSIXct(format(dp),tz="GMT"))`

Table 1: Comparison Table