It is possible at a later time to download the archive 'recommended.sit' that contains the additional recommended packages not included in the base-only package. As usual the other contributed packages can be found in a separate folder.

The 'bin/macosx' folder contains a MacOS X specific build that will run on a X11 server and it is based on the Darwin kernel, i.e., it is a Unix build that runs on MacOS X. This is provided by Jan de Leeuw (deleeuw@stat.ucla.edu). It comes in three versions:

'R-1.3.0-OSX-base.tar.gz' has the R base distribution. It has Tcl/Tk support, but no support for GNOME.

'R-1.3.0-OSX-recommended.tar.gz' has the R base distribution plus the recommended packages. It has Tcl/Tk support, but no support for GNOME.

'R-1.3.0-OSX-full.tar.gz' has the R base distribution plus 134 compiled packages. It is compiled with both GNOME and Tcl/Tk support.

The 'bin/macosx' folder contains two folders, one containing some additional dynamic libraries upon on which this port is based upon, and another giving replacements parts complied with the ATLAS optimized BLAS.

'ReadMe.txt' files are provided for both versions.

Other changes

GNU a2ps is a fairly versatile any-text-to-postscript processor, useful for typesetting source code from a wide variety of programming languages. 's.ssh', 'rd.ssh' and 'st.ssh' are a2ps style sheets for S code, Rd documentation format, and S transscripts, respectively. These will be included in the next a2ps release and are currently available from the "Other Software" page on CRAN.

Kurt Hornik
Wirtschaftsuniversität Wien, Austria
Technische Universität Wien, Austria
Kurt.Hornik@R-project.org

Friedrich Leisch
Technische Universität Wien, Austria
Friedrich.Leisch@ci.tuwien.ac.at

Date-Time Classes

by Brian D. Ripley and Kurt Hornik

Data in the form of date and/or times are common in some fields, for example times of diagnosis and death in survival analysis, trading days and times in financial time series, and dates of files. We had been considering for some time how best to handle such data in R, and it was the last of these examples that forced us to the decision to include classes for dates and times in R version 1.2.0, as part of the **base** package.

We were adding the function `file.info`. Finding information about files looks easy: Unix users take for granted listings like the following (abbreviated to fit the column width):

```
auk% ls -l
total 1189
...       948 Mar 20 14:12 AUTHORS
...      9737 Apr 24 06:44 BUGS
...     17992 Oct  7  1999 COPYING
...     26532 Feb  2 18:38 COPYING.LIB
...      4092 Feb  4 16:00 COPYRIGHTS
```

but there are a number of subtle issues that hopefully the operating system has taken care of. (The example was generated in the UK in April 2001 on a machine set to the C locale.)

- The format. Two formats are used in the extract above, one for files less than 6 months' old and one for older files. Date formats have an international standard (ISO 8601), and this is not it! In the ISO standard the first date is 2001-03-20 14.12. However, the format is not even that commonly used in the UK, which would be 20 Mar 2001 14:12. The month names indicate that this output was designed for an anglophone reader. In short, the format should depend on the *locale*.

- Time zones. Hopefully the times are in the time zone of the computer reading the files, and take daylight saving time into account, so the first time is in GMT and the second in BST. Somewhat more hopefully, this will be the case even if the files have been mounted from a machine on another continent.

  Note that this can be an issue even if one is only interested in survival times in days. Suppose a patient is diagnosed in New Zealand and dies during surgery in California?

We looked at existing solutions, the R packages **chron** and **date**. These seem designed for dates of the accuracy of a day (although **chron** allows partial days), are US-centric and do not take account of time zones. It was clear we had to look elsewhere for a comprehensive solution.

The most obvious solution was the operating system itself: after all it knew enough about dates to process the file dates in the example above. This was the route we decided to take.

Another idea we had was to look at what the database community uses. The SQL99 ISO standard (see Kline & Kline, 2001) has data types `date`, `time`, `time with time zone`, `timestamp`, `timestamp with time zone` and `interval`. The type `timestamp with time zone` looks to be what we wanted. Unfortunately, what is implemented in the common databases is quite different, and for example the MySQL data type `timestamp` is for dates after 1970-01-01 (according to Kline & Kline).

S-PLUS 5.x and 6.x have S4 classes `"timeDate"` and `"timeSpan"` for date-times and for time intervals. These store the data in whole days (since some origin for `"timeDate"`) and whole milliseconds past midnight, together with a timezone and a preferred printing format.

POSIX standards

We were aware that portability would be a problem with using OS facilities, but not aware how much of a headache it was going to be. The 1989 ISO C standard has some limited support for times, which are extended and made more precise in the ISO C99 standard. But there are few (if any) C99-conformant compilers. The one set of standards we did have a chance with was the POSIX set. Documentation on POSIX standards is hard to come by, Lewine (1991) being very useful if now rather old. Vendors do tend to comply with POSIX (at least superficially) as it is mandated for government computer purchases.

The basic POSIX measure of time, *calendar time*, is the number of seconds since the beginning of 1970, in the UTC timezone (GMT as described by the French). Even that needs a little more precision. There have been 22 *leap seconds* inserted since 1970 (see the R object `.leap.seconds`), and these should be discarded. Most machines would store the number as a signed 32-bit integer, which allows times from the early years of the 20th century up to 2037. We decided this was restrictive, and stored the number of seconds as a C double. In principle this allows the storage of times to sub-second accuracy, but that was an opportunity we overlooked. Note that there is little point in allowing a much wider range of times: timezones are a 19th century introduction, and countries changed to the Gregorian calendar at different dates (Britain and its colonies in September 1752). The corresponding R class we called `POSIXct`.

The raw measure of time is not easily digestible, and POSIX also provides a *'broken-down time'* in a `struct tm` structure. This gives the year, month, day of the month, hour, minute and second, all as integers. Those members completely define the clock time once the time zone is known [1]. The other members, the day of the week, the day of the year (0–365) and a DST flag, can in principle be calculated from the first six. Note that year has a baseline of 1900, so years before 1970 are clearly intended to be used. The corresponding R class we called `POSIXlt` (where the '`lt`' stands for "local time"), which is a list with components as integer vectors, and so can represent a vector of broken-down times. We wanted to keep track of timezones, so where known the timezone is given by an attribute `"tzone"`, the name of the timezone.

Conversions

A high-level language such as R should handle the conversion between classes automatically. For times within the range handled by the operating system we can use the POSIX functions `mktime` to go from broken-down time to calendar time, and `localtime` and `gmtime` to go from calendar time to broken-down time, in the local timezone and UTC respectively. The only way to do the conversion in an arbitrary timezone is to set the timezone *pro tem*[2]. That proved difficult!

We also want to be able to print out and scan in date-times. POSIX provides a function `strftime` to print a date-time in a wide range of formats. The reverse function, `strptime`, to convert a character string to a broken-down time, is not in POSIX but is widely implemented.

Let us look again at the file dates, now using R:

```
> file.info(dir())[, "mtime", drop=FALSE]
                          mtime
AUTHORS      2001-03-20 14:12:22
BUGS         2001-04-24 06:44:10
COPYING      1999-10-07 19:09:39
COPYING.LIB  2001-02-02 18:38:32
COPYRIGHTS   2001-02-04 16:00:49
...
```

This gives the dates in the default (ISO standard) format, and has taken proper account of the timezone change. (Note that this has been applied to a column of a data frame.) When printing just a date-time object the timezone is given, if known. We can easily use other formats as we like.

```
> zz <- file.info(dir())[1:5, "mtime"]
> zz
[1] "2001-03-20 14:12:22 GMT"
[2] "2001-04-24 06:44:10 BST"
[3] "1999-10-07 19:09:39 BST"
[4] "2001-02-02 18:38:32 GMT"
[5] "2001-02-04 16:00:49 GMT"
> format(zz, format="%x %X")
```

[1]ISO C99 adds members `tm_zone` and `tm_leapseconds` in `struct tmx`. It represents both the time zone and the DST by an offset in minutes, information that is not readily available in some of the platforms we looked at.

[2]C99 has functions `zonetime` and `mkxtime` which would avoid this.

```
# locale specific: see also %c or %C.
[1] "03/20/01 14:12:22" "04/24/01 06:44:10"
[3] "10/07/99 19:09:39" "02/02/01 18:38:32"
[5] "02/04/01 16:00:49"
> Sys.setlocale(locale = "en_UK")
[1] "en_UK"
> format(zz, format="%x %X")
[1] "20/03/01 02:12:22 PM" "24/04/01 06:44:10 AM"
[3] "07/10/99 07:09:39 PM" "02/02/01 06:38:32 PM"
[5] "04/02/01 04:00:49 PM"
> format(zz, format="%b %d %Y")
[1] "Mar 20 2001" "Apr 24 2001" "Oct 07 1999"
[4] "Feb 02 2001" "Feb 04 2001"
> format(zz, format="%a %d %b %Y %H:%M")
[1] "Tue 20 Mar 2001 14:12"
[2] "Tue 24 Apr 2001 06:44"
[3] "Thu 07 Oct 1999 19:09"
[4] "Fri 02 Feb 2001 18:38"
[5] "Sun 04 Feb 2001 16:00"
```

It was easy to add conversions from the `chron` and `dates` classes.

The implementation

The original implementation was written under Solaris, and went very smoothly. It was the only OS for which this was the case! Our idea was to use OS facilities where are these available, so we added simple versions of `mktime` and `gmtime` to convert times far into the past or the future ignoring timezones, and then worked out the adjustment on the same day in 2000 in the current timezone.

One advantage of an Open Source project is the ability to borrow from other projects, and we made use of `glibc`'s version of `strptime` to provide one for platforms which lacked it.

Coping with the vagaries of other platforms proved to take far longer. According to POSIX, `mktime` is supposed to return `-1` (which is a valid time) for out-of-range times, but on Windows it crashed for times before 1970-01-01. Such times were admittedly pre-Microsoft! Linux systems do not normally have a `TZ` environment variable set, and this causes crashes in `strftime` when asked to print the timezone, and also complications in temporarily setting timezones (there is no way portably to unset an environment variable from C). Some platforms were confused if the DST flag was set to `-1` ('unknown'). SGI's `strptime` only works after 1970. And so on .... The code became more and more complicated as workarounds were added.

We provided a configure test of whether leap seconds were ignored, and code to work around it if they are not. We never found such a platform, but we have since had a bug report which shows they do exist and we did not get the code quite right first time around.

Describing all the problems we found would make a very long article. We did consider providing all our own code based on `glibc`. In retrospect that would have saved a lot of problems, but created others. Managing a timezone database is really tedious, and we would have had to find out for each OS how to read the local timezone in terms that `glibc` would understand.

Much later we found out that classic MacOS does not really understand timezones, and so workarounds had to be added for that port of R.

Extensions

The implementation we put in version 1.2.0 was not fully complete. One issue which arose was the need to form time differences (the SQL99 `interval` data type). Subtraction of two `POSIXct` or two `POSIXlt` times gave a number of seconds, but subtracting a `POSIXct` time from a `POSIXlt` time failed.

Version 1.3.0 provides general facilities for handling time differences, via a class `"difftime"` with generator function `difftime`. This allows time units of days or hours or minutes or seconds, and aims to make a sensible choice automatically. To allow subtraction to work within R's method dispatch system we needed to introduce a super-class `"POSIXt"`, and a method function `-.POSIXt`. Thus from 1.3.0, calendar-time objects have class `c("POSIXt", "POSIXct")`, and broken-down-time objects have class `c("POSIXt", "POSIXlt")`. Appending the new class rather than prepending would not work, for reasons we leave as an exercise for the reader.

Here is an example of the time intervals between R releases:

```
> ISOdate(2001, 2, 26) - ISOdate(2001, 1, 15)
Time difference of 42 days
> ISOdate(2001, 4, 26) - ISOdate(2001, 2, 26)
Time difference of 59 days
```

The result is of class `"difftime"` and so printed as a number of days: it is stored as a number of seconds.

One has to be slightly careful: compare the second example with

```
> as.POSIXct("2001-04-26") -
    as.POSIXct("2001-02-26")
Time difference of 58.95833 days
> c(as.POSIXct("2001-04-26"),
    as.POSIXct("2001-02-26"))
[1] "2001-04-26 BST" "2001-02-26 GMT"
> c(ISOdate(2001, 4, 26), ISOdate(2001, 2, 26))
[1] "2001-04-26 13:00:00 BST"
[2] "2001-02-26 12:00:00 GMT"
```

The difference is that `ISOdate` chooses midday GMT as the unspecified time of day, and `as.POSIXct` is using midnight in the timezone. As the UK changed to DST between the releases, had the releases occurred at the same time of day the interval would not have been an exact multiple of a 24-hour day. The `round` method can be useful here.

There are many more things one would like to do with date-time objects. We want to know the current time (`Sys.time`) and timezone (`Sys.timezone`).

Methods for `format` provide very flexible ways to convert them to character strings. We have an `axis` method to use them to label graphs. Lots of methods are needed, for `all.equal`, `as.character`, `c`, `cut`, `mean`, `round`, `seq`, `str`, ... And these need to check for appropriateness, so for example sums of dates are not well-defined, whereas means are.

We have also provided convenience functions like `weekdays`, `months` and `quarters`, which either extract information from the `POSIXlt` list or convert using an appropriate `format` argument in a call to the `format` method. The `POSIXt` method for the (new) generic function `julian` converts to Julian dates (the number of days since some origin, often 1970-01-01).

The future

We believe that the date-time classes in base R now provide sufficient flexibility and facilities to cover almost all applications and hence that they should now be used in preference to earlier and more limited systems. Perhaps most important is that these classes be used in inter-system applications such as database connectivity.

References

International Organization for Standardization (1988, 1997, ...) ISO 8601. Data elements and interchange formats – Information interchange – Representation of dates and times. The 1997 version is available on-line at ftp://ftp.qsl.net/pub/g1smd/8601v03.pdf.

Kline, K. and Kline, D. (2001) *SQL in a Nutshell.* O'Reilly.

Lewine, D. (1991) *POSIX Programmer's Guide. Writing Portable UNIX Programs.* O'Reilly & Associates.

Brian D. Ripley
University of Oxford, UK
ripley@stats.ox.ac.uk

Kurt Hornik
Wirtschaftsuniversität Wien, Austria
Technische Universität Wien, Austria
Kurt.Hornik@R-project.org

Installing R under Windows

by Brian D. Ripley

Very few Windows users will have ever experienced compiling a large system, as binary installations of Windows software are universal. Further, users are used to installing software by a point-and-click interface with a minimum of reading of instructions, most often none. The expectation is

> Insert the CD.
>
> If it doesn't auto-run, double-click on a file called `Setup.exe` in the top directory.
>
> Go through a few 'Wizard' pages, then watch a progress bar as files are installed, then click on Finish.

Contrast this with 'untar the sources, run `./configure`, `make` then `make install`'. Each in its own way is simple, but it is really horses for courses.

Every since Guido Masarotto put out a version of R for Windows as a set of `zip` files we have been looking for a way to install R in a style that experienced Windows users will find natural. At last we believe we have found one.

When I first looked at this a couple of years ago most packages (even Open Source ones) used a commercial installer such as InstallShield or Wise. Although I had a copy of InstallShield, I was put off by its size, complexity and the experiences I gleaned, notably from Fabrice Popineau with his `fptex` installation.

Shortly afterwards, MicroSoft introduced their own installer for their Office 2000 suite. This works in almost the same way, except that one double-clicks on a file with extension `.msi`. There is a development kit for this installer and I had expected it to become the installer of choice, but it seems rarely used. (The `Perl` and now `Tcl` ports to Windows do use it.) That makes one of its disadvantages serious: unless you have a current version of Windows (ME or 2000) you need to download the installer `InstMsi.exe`, which is around 1.5Mb and whose installation needs privileges an ordinary user may not have.

In May 1999 I decided to write a simple installer, `rwinst.exe`, using the GraphApp toolkit that Guido had used to write the R for Windows GUI, and this has been in use since. But it was not ideal, for

- It did not use a completely standard 'look-and-feel'.
- It was hard to maintain, and mistakes in an installer are 'mission-critical'.
- Users found it hard to cope with needing several files to do the installation.

The prospect of *recommended* packages at version 1.3.0 was going to require twice as many files, and forced a re-think in early 2001. I had earlier looked at Inno Setup by Jordan Russell (www.jrsoftware.org)