Dates and Times in R

R provides several options for dealing with date and date/time data. The builtin `as.Date` function handles dates (without times); the contributed library `chron` handles dates and times, but does not control for time zones; and the `POSIXct` and `POSIXlt` classes allow for dates and times with control for time zones. The general rule for date/time data in R is to use the simplest technique possible. Thus, for date only data, `as.Date` will usually be the best choice. If you need to handle dates and times, without timezone information, the `chron` library is a good choice; the POSIX classes are especially useful when timezone manipulation is important. Also, don't overlook the various "as." functions (see Section) for converting among the different date types when necessary.

Except for the `POSIXlt` class, dates are stored internally as the number of days or seconds from some reference date. Thus dates in R will generally have a numeric mode, and the `class` function can be used to find the way they are actually being stored. The `POSIXlt` class stores date/time values as a list of components (`hour`, `min`, `sec`, `mon`, etc.) making it easy to extract these parts.

To get the current date, the Sys.Date function will return a `Date` object which can be converted to a different class if necessary.

The following subsections will describe the different types of date values in more detail.

The `as.Date` function allows a variety of input formats through the `format=` argument. The default format is a four digit year, followed by a month, then a day, separated by either dashes or slashes. The following example shows some examples of dates which `as.Date` will accept by default:

```
> as.Date('1915-6-16')
[1] "1915-06-16"
> as.Date('1990/02/17')
[1] "1990-02-17"
```

Code	Value
%d	Day of the month (decimal number)
%m	Month (decimal number)
%b	Month (abbreviated)
%B	Month (full name)
%y	Year (2 digit)
%Y	Year (4 digit)

If your input dates are not in the standard format, a format string can be composed using the elements shown in Table . The following examples show some ways that this can be used:

```
> as.Date('1/15/2001',format='%m/%d/%Y')
[1] "2001-01-15"
> as.Date('April 26, 2001',format='%B %d, %Y')
[1] "2001-04-26"
> as.Date('22JUN01',format='%d%b%y')   # %y is system-specific; use with caution
[1] "2001-06-22"
```

Internally, `Date` objects are stored as the number of days since January 1, 1970, using negative numbers for earlier dates. The `as.numeric` function can be used to convert a `Date` object to its internal form.

To extract the components of the dates, the `weekdays`, `months`, `days` or `quarters` functions can be used. For example, to find the day of the week on which some famous statisticians were born, we can look at the result of the `weekdays` function:

```
> bdays = c(tukey=as.Date('1915-06-16'),fisher=as.Date('1890-02-17'),
+            cramer=as.Date('1893-09-25'), kendall=as.Date('1907-09-06'))
> weekdays(bdays)
      tukey      fisher      cramer     kendall
"Wednesday"    "Monday"    "Monday"    "Friday"
```

For an alternative way of extracting pieces of a day, and for information on possible output formats for `Date` objects, see Section .

The `chron` function converts dates and times to `chron` objects. The dates and times are provided to the `chron` function as separate values, so some preprocessing may be necessary to prepare input date/times for the `chron` function. When using character values, the default format for dates is the decimal month value followed by the decimal day value followed by the year, using the slash as a separator. Alternative formats can be provided by using the codes shown in Table

Format codes for dates	
Code	Value
m	Month (decimal number)
d	Day of the month (decimal number)
y	Year (4 digit)
mon	Month (abbreviated)
month	Month (full name)
Format codes for times	
Code	Value
h	Hour
m	Minute
s	Second

Alternatively, dates can be specified by a numeric value, representing the number of days since January 1, 1970. To input dates stored as the day of the year, the `origin=` argument can be used to interpret numeric dates relative to a different date.

The default format for times consists of the hour, minutes and seconds, separated by colons. Alternative formats can use the codes in Table .

Often the first task when using the `chron` library is to break apart the date and times if they are stored together. In the following example, the `strsplit` function is used to break apart the string.

```
> dtimes = c("2002-06-09 12:45:40","2003-01-29 09:30:40",
+             "2002-09-04 16:45:40","2002-11-13 20:00:40",
+             "2002-07-07 17:30:40")
> dtparts = t(as.data.frame(strsplit(dtimes,' ')))
> row.names(dtparts) = NULL
> thetimes = chron(dates=dtparts[,1],times=dtparts[,2],
+                  format=c('y-m-d','h:m:s'))
> thetimes
[1] (02-06-09 12:45:40) (03-01-29 09:30:40) (02-09-04 16:45:40)
[4] (02-11-13 20:00:40) (02-07-07 17:30:40)
```

Chron values are stored internally as the fractional number of days from January 1, 1970. The `as.numeric` function can be used to access the internal values.

For information on formatting `chron` objects for output, see Section

POSIX represents a portable operating system interface, primarily for UNIX systems, but available on other operating systems as well. Dates stored in the POSIX format are date/time values (like dates with the `chron` library), but also allow modification of time zones. Unlike the `chron` library, which stores times as fractions of days, the POSIX date classes store times to the nearest second, so they provide a more accurate representation of times.

There are two POSIX date/time classes, which differ in the way that the values are stored internally. The `POSIXct` class stores date/time values as the number of seconds since January 1, 1970, while the `POSIXlt` class stores them as a list with elements for second, minute, hour, day, month, and year, among others. Unless you need the list nature of the `POSIXlt` class, the `POSIXct` class is the usual choice for storing dates in R.

The default input format for POSIX dates consists of the year, followed by the month and day, separated by slashes or dashes; for date/time values, the date may be followed by white space and a time in the form hour:minutes:seconds or hour:minutes; thus, the following are examples of valid POSIX date or date/time inputs:

```
1915/6/16
2005-06-24 11:25
1990/2/17 12:20:05
```

If the input times correspond to one of these formats, as.POSIXct can be called directly:

```
> dts = c("2005-10-21 18:47:22","2005-12-24 16:39:58",
+         "2005-10-28 07:30:05 PDT")
> as.POSIXlt(dts)
[1] "2005-10-21 18:47:22" "2005-12-24 16:39:58"
[3] "2005-10-28 07:30:05"
```

If your input date/times are stored as the number of seconds from January 1, 1970, you can create POSIX date values by assigning the appropriate class directly to those values. Since most date manipulation functions refer to the POSIXt psuedo-class, be sure to include it as the first member of the class attribute.

```
> dts = c(1127056501,1104295502,1129233601,1113547501,
+         1119826801,1132519502,1125298801,1113289201)
> mydates = dts
> class(mydates) = c('POSIXt','POSIXct')
> mydates
[1] "2005-09-18 08:15:01 PDT" "2004-12-28 20:45:02 PST"
[3] "2005-10-13 13:00:01 PDT" "2005-04-14 23:45:01 PDT"
[5] "2005-06-26 16:00:01 PDT" "2005-11-20 12:45:02 PST"
[7] "2005-08-29 00:00:01 PDT" "2005-04-12 00:00:01 PDT"
```

Conversions like this can be done more succinctly using the `structure` function:

```
> mydates = structure(dts,class=c('POSIXt','POSIXct'))
```

Code	Meaning	Code	Meaning
%a	Abbreviated weekday	%A	Full weekday
%b	Abbreviated month	%B	Full month

%c	Locale-specific date and time	%d	Decimal date
%H	Decimal hours (24 hour)	%I	Decimal hours (12 hour)
%j	Decimal day of the year	%m	Decimal month
%M	Decimal minute	%p	Locale-specific AM/PM
%S	Decimal second	%U	Decimal week of the year (starting on Sunday)
%w	Decimal Weekday (0=Sunday)	%W	Decimal week of the year (starting on Monday)
%x	Locale-specific Date	%X	Locale-specific Time
%y	2-digit year	%Y	4-digit year
%z	Offset from GMT	%Z	Time zone (character)

The POSIX date/time classes take advantage of the POSIX date/time implementation of your operating system, allowing dates and times in R to be manipulated in the same way they would in, for example a C program. The two most important functions in this regard are strptime, for inputting dates, and strftime, for formatting dates for output. Both of these functions use a variety of formatting codes, some of which are listed in Table , to specify the way dates are read or printed. For example, dates in many logfiles are printed in a format like "16/Oct/2005:07:51:00". To create a POSIXct date from a date in this format, the following call to strptime could be used:

```
> mydate = strptime('16/Oct/2005:07:51:00',format='%d/%b/%Y:%H:%M:%S')
[1] "2005-10-16 07:51:00"
```

Note that non-format characters (like the slashes) are interpreted literally.

When using strptime, an optional time zone can be specified with the tz= option.

Another way to create POSIX dates is to pass the individual components of the time to the ISOdate function. Thus, the first date/time value in the previous example could also be created with a call to ISOdate;

```
> ISOdate(2005,10,21,18,47,22,tz="PDT")
[1] "2005-10-21 18:47:22 PDT"
```

For formatting dates for output, the format function will recognize the type of your input date, and perform any necessary conversions before calling strftime, so strftime rarely needs to be called directly. For example, to print a date/time value in an extended format, we could use:

```
> thedate = ISOdate(2005,10,21,18,47,22,tz="PDT")
> format(thedate,'%A, %B %d, %Y %H:%M:%S')
[1] "Friday, October 21, 2005 18:47:22"
```

When using POSIX dates, the optional usetz=TRUE argument to the format function can be specified to indicate that the time zone should be displayed.

Additionally, as.POSIXlt and as.POSIXct can also accept Date or chron objects, so they can be input as described in the previous sections and converted as needed. Conversion between the two POSIX forms is also possible.

The individual components of a POSIX date/time object can be extracted by first converting to POSIXlt if necessary, and then accessing the components directly:

```
> mydate = as.POSIXlt('2005-4-19 7:01:00')
> names(mydate)
[1] "sec"   "min"   "hour"  "mday"  "mon"    "year"
```

```
[7] "wday"   "yday"   "isdst"
> mydate$mday
[1] 19
```

Many of the statistical summary functions, like mean, min, max, etc are able to transparently handle date
objects. For example, consider the release dates of various versions or R from 1.0 to 2.0:

```
> rdates = scan(what="")
1: 1.0 29Feb2000
3: 1.1 15Jun2000
5: 1.2 15Dec2000
7: 1.3 22Jun2001
9: 1.4 19Dec2001
11: 1.5 29Apr2002
13: 1.6 1Oct2002
15: 1.7 16Apr2003
17: 1.8 8Oct2003
19: 1.9 12Apr2004
21: 2.0 4Oct2004
23:
Read 22 items
> rdates = as.data.frame(matrix(rdates,ncol=2,byrow=TRUE))
> rdates[,2] = as.Date(rdates[,2],format='%d%b%Y')
> names(rdates) = c("Release","Date")
> rdates
   Release       Date
1      1.0 2000-02-29
2      1.1 2000-06-15
3      1.2 2000-12-15
4      1.3 2001-06-22
5      1.4 2001-12-19
6      1.5 2002-04-29
7      1.6 2002-10-01
8      1.7 2003-04-16
9      1.8 2003-10-08
10     1.9 2004-04-12
11     2.0 2004-10-04
```

Once the dates are properly read into R, a variety of calculations can be performed:

```
> mean(rdates$Date)
[1] "2002-05-19"
> range(rdates$Date)
[1] "2000-02-29" "2004-10-04"
> rdates$Date[11] - rdates$Date[1]
Time difference of 1679 days
```

If two times (using any of the date or date/time classes) are subtracted, R will return the results in the form of
a time difference, which represents a difftime object. For example, New York City experienced a major
blackout on July 13, 1997, and another on August 14, 2003. To calculate the time interval between the two
blackouts, we can simply subtract the two dates, using any of the classes that have been introduced:

```
> b1 = ISOdate(1977,7,13)
> b2 = ISOdate(2003,8,14)
> b2 - b1
Time difference of 9528 days
```

If an alternative unit of time was desired, the `difftime` function could be called, using the optional `units=` argument can be used with any of the following values: `"auto"`, `"secs"`, `"mins"`, `"hours"`, `"days"`, or `"weeks"`. So to see the difference between blackouts in terms of weeks, we can use:

```
> difftime(b2,b1,units='weeks')
Time difference of 1361.143 weeks
```

Although `difftime` values are displayed with their units, they can be manipulated like ordinary numeric variables; arithmetic performed with these values will retain the original units.

The `by=` argument to the `seq` function can be specified either as a `difftime` value, or in any units of time that the `difftime` function accepts, making it very easy to generate sequences of dates. For example, to generate a vector of ten dates, starting on July 4, 1976 with an interval of one day between them, we could use:

```
> seq(as.Date('1976-7-4'),by='days',length=10)
 [1] "1976-07-04" "1976-07-05" "1976-07-06" "1976-07-07" "1976-07-08"
 [6] "1976-07-09" "1976-07-10" "1976-07-11" "1976-07-12" "1976-07-13"
```

All the date classes except for `chron` will accept an integer before the interval provided as a `by=` argument. We could create a sequence of dates separated by two weeks from June 1, 2000 to August 1, 2000 as follows:

```
> seq(as.Date('2000-6-1'),to=as.Date('2000-8-1'),by='2 weeks')
[1] "2000-06-01" "2000-06-15" "2000-06-29" "2000-07-13" "2000-07-27"
```

The `cut` function also understands units of `days`, `weeks`, `months`, and `years`, making it very easy to create factors grouped by these units. See Section for details.

Format codes can also be used to extract parts of dates, similar to the `weekdays` and other functions described in Section . We could look at the distribution of weekdays for the R release dates as follows:

```
> table(format(rdates$Date,'%A'))

   Friday     Monday  Thursday    Tuesday Wednesday
        2          3         1          2         3
```

This same technique can be used to convert dates to factors. For example, to create a factor based on the release dates broken down by years we could use:

```
> fdate = factor(format(rdates$Date,'%Y'))
> fdate
 [1] 2000 2000 2000 2001 2001 2002 2002 2003 2003 2004 2004
Levels: 2000 2001 2002 2003 2004
```

```
> cut(thetimes,"year")
[1] 02 03 02 02 02
Levels: 02 < 03
```

File translated from TEX by [TTH](#), version 3.67.

On 3 Feb 2006, 17:06.