R Installation and Administration

Version 3.3.1 (2016-06-21)

R Core Team

This manual is for R, version 3.3.1 (2016-06-21).

Copyright © 2001–2016 R Core Team

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that this permission notice may be stated in a translation approved by the R Core Team.

Table of Contents

1 Obtaining R 1
- 1.1 Getting and unpacking the sources 1
- 1.2 Getting patched and development versions 1
  - 1.2.1 Using Subversion and rsync 1

2 Installing R under Unix-alikes 3
- 2.1 Simple compilation 3
- 2.2 Help options 4
- 2.3 Making the manuals 4
- 2.4 Installation 6
- 2.5 Uninstallation 8
- 2.6 Sub-architectures 8
  - 2.6.1 Multilib 9
- 2.7 Other Options 9
  - 2.7.1 OpenMP Support 10
- 2.8 Testing an Installation 10

3 Installing R under Windows 12
- 3.1 Building from source 12
  - 3.1.1 Getting the tools 12
  - 3.1.2 Getting the source files 12
  - 3.1.3 Building the core files 13
  - 3.1.4 Building the cairo devices 14
  - 3.1.5 Using ICU for collation 14
  - 3.1.6 Support for libcurl 14
  - 3.1.7 Checking the build 14
  - 3.1.8 Building the manuals 15
  - 3.1.9 Building the Inno Setup installer 15
  - 3.1.10 Building the MSI installer 16
  - 3.1.11 64-bit Windows builds 17
- 3.2 Testing an Installation 17

4 Installing R under OS X 18
- 4.1 Running R under OS X 18
- 4.2 Uninstalling under OS X 19
- 4.3 Multiple versions 19

5 Running R 21

6 Add-on packages 22
- 6.1 Default packages 22
- 6.2 Managing libraries 22
- 6.3 Installing packages 22
  - 6.3.1 Windows 23
  - 6.3.2 OS X 24
  - 6.3.3 Customizing package compilation 25

6.3.4 Multiple sub-architectures . . . 26
6.3.5 Byte-compilation . . . 26
6.3.6 External software . . . 27
6.4 Updating packages . . . 27
6.5 Removing packages . . . 28
6.6 Setting up a package repository . . . 28
6.7 Checking installed source packages . . . 29

7 Internationalization and Localization . . . 30
7.1 Locales . . . 30
7.1.1 Locales under Unix-alikes . . . 30
7.1.2 Locales under Windows . . . 30
7.1.3 Locales under OS X . . . 31
7.2 Localization of messages . . . 31

8 Choosing between 32- and 64-bit builds . . . 33

9 The standalone Rmath library . . . 34
9.1 Unix-alikes . . . 34
9.2 Windows . . . 35

Appendix A Essential and useful other programs under a Unix-alike . . . 37
A.1 Essential programs and libraries . . . 37
A.2 Useful libraries and programs . . . 39
A.2.1 Tcl/Tk . . . 40
A.2.2 Java support . . . 40
A.2.3 Other compiled languages . . . 41
A.3 Linear algebra . . . 41
A.3.1 BLAS . . . 41
A.3.1.1 ATLAS . . . 42
A.3.1.2 ACML . . . 43
A.3.1.3 Goto and OpenBLAS . . . 43
A.3.1.4 Intel MKL . . . 43
A.3.1.5 Shared BLAS . . . 44
A.3.2 LAPACK . . . 44
A.3.3 Caveats . . . 45

Appendix B Configuration on a Unix-alike . . . 46
B.1 Configuration options . . . 46
B.2 Internationalization support . . . 47
B.3 Configuration variables . . . 47
B.3.1 Setting paper size . . . 47
B.3.2 Setting the browsers . . . 47
B.3.3 Compilation flags . . . 48
B.3.4 Making manuals . . . 48
B.4 Setting the shell . . . 48
B.5 Using make . . . 48
B.6 Using FORTRAN . . . 48
B.7 Compile and load flags . . . 49
B.8 Maintainer mode . . . 50

Appendix C Platform notes ... 52
C.1 X11 issues ... 52
C.2 Linux ... 53
C.2.1 Clang ... 54
C.2.2 Intel compilers ... 55
C.2.3 Oracle Solaris Studio compilers ... 55
C.3 OS X ... 56
C.3.1 El Capitan ... 58
C.3.2 Tcl/Tk headers and libraries ... 58
C.3.3 Java ... 58
C.3.4 Frameworks ... 59
C.3.5 Building R.app ... 59
C.4 Solaris ... 60
C.4.1 Using gcc ... 62
C.5 AIX ... 63
C.6 FreeBSD ... 64
C.7 OpenBSD ... 65
C.8 Cygwin ... 65
C.9 New platforms ... 65

Appendix D The Windows toolset ... 67
D.1 LaTeX ... 68
D.2 The Inno Setup installer ... 68
D.3 The command line tools ... 68
D.4 The MinGW-w64 toolchain ... 68
D.5 Useful additional programs ... 69

Function and variable index ... 70

Concept index ... 71

Environment variable index ... 72

1 Obtaining R

Sources, binaries and documentation for R can be obtained via CRAN, the "Comprehensive R Archive Network" whose current members are listed at `https://CRAN.R-project.org/mirrors.html`.

1.1 Getting and unpacking the sources

The simplest way is to download the most recent `R-`*x*`.`*y*`.`*z*`.tar.gz` file, and unpack it with

```
tar -xf R-x.y.z.tar.gz
```

on systems that have a suitable[1] `tar` installed. On other systems you need to have the `gzip` program installed, when you can use

```
gzip -dc R-x.y.z.tar.gz | tar -xf -
```

The pathname of the directory into which the sources are unpacked should not contain spaces, as most `make` programs (and specifically GNU `make`) do not expect spaces.

If you want the build to be usable by a group of users, set `umask` before unpacking so that the files will be readable by the target group (e.g., `umask 022` to be usable by all users). Keep this setting of `umask` whilst building and installing.

If you use a recent GNU version of `tar` and do this as a root account (which on Windows includes accounts with administrator privileges) you may see many warnings about changing ownership. In which case you can use

```
tar --no-same-owner -xf R-x.y.z.tar.gz
```

and perhaps also include the option `--no-same-permissions`. (These options can also be set in the `TAR_OPTIONS` environment variable: if more than one option is included they should be separated by spaces.)

1.2 Getting patched and development versions

A patched version of the current release, '`r-patched`', and the current development version, '`r-devel`', are available as daily tarballs and via access to the R Subversion repository. (For the two weeks prior to the release of a minor (3.x.0) version, '`r-patched`' tarballs may refer to beta/release candidates of the upcoming release, the patched version of the current release being available via Subversion.)

The tarballs are available from `https://stat.ethz.ch/R/daily`. Download `R-patched.tar.gz` or `R-devel.tar.gz` (or the `.tar.bz2` versions) and unpack as described in the previous section. They are built in exactly the same way as distributions of R releases.

1.2.1 Using Subversion and rsync

Sources are also available via `https://svn.R-project.org/R/`, the R Subversion repository. If you have a Subversion client (see `https://subversion.apache.org/`), you can check out and update the current '`r-devel`' from `https://svn.r-project.org/R/trunk/` and the current '`r-patched`' from '`https://svn.r-project.org/R/branches/R-`*x*`-`*y*`-branch/`' (where *x* and *y* are the major and minor number of the current released version of R). E.g., use

```
svn checkout https://svn.r-project.org/R/trunk/ path
```

to check out '`r-devel`' into directory *path* (which will be created if necessary). The alpha, beta and RC versions of an upcoming *x.y.0* release are available from '`https://svn.r-project.org/R/branches/R-`*x*`-`*y*`-branch/`' in the four-week period prior to the release.

[1] e.g. GNU `tar` version 1.15 or later, or that from the '`libarchive`' (as used on OS X versions 10.6 and later) or 'Heirloom Toolchest' distributions.

Note that '`https:`' is required[2], and that the SSL certificate for the Subversion server of the R project should be recognized as from a trusted source.

Note that retrieving the sources by e.g. `wget -r` or `svn export` from that URL will not work (and will give a error early in the `make` process): the Subversion information is needed to build R.

The Subversion repository does not contain the current sources for the recommended packages, which can be obtained by `rsync` or downloaded from CRAN. To use `rsync` to install the appropriate sources for the recommended packages, run `./tools/rsync-recommended` from the top-level directory of the R sources.

If downloading manually from CRAN, do ensure that you have the correct versions of the recommended packages: if the number in the file `VERSION` is '*x.y.z*' you need to download the contents of '`https://CRAN.R-project.org/src/contrib/`*dir*', where *dir* is '*x.y.z*`/Recommended`' for r-devel or *x.y*`-patched/Recommended` for r-patched, respectively, to directory `src/library/Recommended` in the sources you have unpacked. After downloading manually you need to execute `tools/link-recommended` from the top level of the sources to make the requisite links in `src/library/Recommended`. A suitable incantation from the top level of the R sources using `wget` might be (for the correct value of *dir*)

```
wget -r -l1 --no-parent -A\*.gz -nd -P src/library/Recommended \
  https://CRAN.R-project.org/src/contrib/dir
./tools/link-recommended
```

[2] for some Subversion clients '`http:`' may appear to work, but requires continual redirection.

2 Installing R under Unix-alikes

R will configure and build under most common Unix and Unix-alike platforms including '*cpu*`-*-linux-gnu`' for the '`alpha`', '`arm`', '`hppa`', '`ix86`', '`m68k`', '`mips`', '`mipsel`', '`powerpc`', '`s390`', '`sparc`', and '`x86_64`' CPUs, '`x86_64-apple-darwin`', '`i386-sun-solaris`' and '`sparc-sun-solaris`' as well as perhaps (it is tested less frequently on these platforms) '`i386-apple-darwin`', '`i386-*-freebsd`', '`x86_64-*-freebsd`', '`i386-*-netbsd`', '`x86_64/*-openbsd`' and '`powerpc-ibm-aix6*`'

In addition, binary distributions are available for some common Linux distributions and for OS X (formerly Mac OS). See the FAQ for current details. These are installed in platform-specific ways, so for the rest of this chapter we consider only building from the sources.

Cross-building is not possible: installing R builds a minimal version of R and then runs many R scripts to complete the build.

2.1 Simple compilation

First review the essential and useful tools and libraries in Appendix A [Essential and useful other programs under a Unix-alike], page 37, and install those you want or need. Ensure that the environment variable `TMPDIR` is either unset (and `/tmp` exists and can be written in and scripts can be executed from) or points to the absolute path to a valid temporary directory (one from which execution of scripts is allowed) which does not contain spaces.[1]

Choose a directory to install the R tree (R is not just a binary, but has additional data sets, help files, font metrics etc). Let us call this place *R_HOME*. Untar the source code. This should create directories `src`, `doc`, and several more under a top-level directory: change to that top-level directory (At this point North American readers should consult Section B.3.1 [Setting paper size], page 47.) Issue the following commands:

```
./configure
make
```

(See Section B.5 [Using make], page 48, if your make is not called '`make`'.) Users of Debian-based 64-bit systems[2] may need

```
./configure LIBnn=lib
make
```

Then check the built system works correctly by

```
make check
```

Failures are not necessarily problems as they might be caused by missing functionality, but you should look carefully at any reported discrepancies. (Some non-fatal errors are expected in locales that do not support Latin-1, in particular in true `C` locales and non-UTF-8 non-Western-European locales.) A failure in `tests/ok-errors.R` may indicate inadequate resource limits (see Chapter 5 [Running R], page 21).

More comprehensive testing can be done by

```
make check-devel
```

or

```
make check-all
```

see file `tests/README` and Section 2.8 [Testing a Unix-alike Installation], page 10, for the possibilities of doing this in parallel. Note that these checks are only run completely if the recommended packages are installed.

[1] Most aspects will work with paths containing spaces, but external software used by R, e.g. `texi2dvi` version `4.8`, may not.

[2] which use `lib` rather than `lib64` for their primary 64-bit library directories.

If the command `configure` and `make` commands execute successfully, a shell-script front-end called `R` will be created and copied to *`R_HOME`*`/bin`. You can link or copy this script to a place where users can invoke it, for example to `/usr/local/bin/R`. You could also copy the man page `R.1` to a place where your `man` reader finds it, such as `/usr/local/man/man1`. If you want to install the complete R tree to, e.g., `/usr/local/lib/R`, see Section 2.4 [Installation], page 6. Note: you do not *need* to install R: you can run it from where it was built.

You do not necessarily have to build R in the top-level source directory (say, *`TOP_SRCDIR`*). To build in *`BUILDDIR`*, run

```
cd BUILDDIR
TOP_SRCDIR/configure
make
```

and so on, as described further below. This has the advantage of always keeping your source tree clean and is particularly recommended when you work with a version of R from Subversion. (You may need GNU `make` to allow this, and you will need no spaces in the path to the build directory. It is unlikely to work if the source directory has previously been used for a build.)

Now `rehash` if necessary, type *`R`*, and read the R manuals and the R FAQ (files `FAQ` or `doc/manual/R-FAQ.html`, or `https://CRAN.R-project.org/doc/FAQ/R-FAQ.html` which always has the version for the latest release of R).

Note: if you already have R installed, check that where you installed R replaces or comes earlier in your path than the previous installation. Some systems are set up to have `/usr/bin` (the standard place for a system installation) ahead of `/usr/local/bin` (the default place for installation of R) in their default path, and some do not have `/usr/local/bin` on the default path.

2.2 Help options

By default HTML help pages are created when needed rather than being built at install time.

If you need to disable the server and want HTML help, there is the option to build HTML pages when packages are installed (including those installed with R). This is enabled by the `configure` option `--enable-prebuilt-html`. Whether `R CMD INSTALL` (and hence `install.packages`) pre-builds HTML pages is determined by looking at the R installation and is reported by `R CMD INSTALL --help`: it can be overridden by specifying one of the `INSTALL` options `--html` or `--no-html`.

The server is disabled by setting the environment variable `R_DISABLE_HTTPD` to a non-empty value, either before R is started or within the R session before HTML help (including `help.start`) is used. It is also possible that system security measures will prevent the server from being started, for example if the loopback interface has been disabled. See `?tools::startDynamicHelp` for more details.

2.3 Making the manuals

There is a set of manuals that can be built from the sources,

'`fullrefman`'
: Printed versions of all the help pages for base and recommended packages (around 3500 pages).

'`refman`' Printed versions of the help pages for selected base packages (around 2000 pages)

'`R-FAQ`' R FAQ

'`R-intro`' "An Introduction to R".

'`R-data`' "R Data Import/Export".

'`R-admin`' "R Installation and Administration", this manual.

'`R-exts`' "Writing R Extensions".

'`R-lang`' "The R Language Definition".

To make these (with '`fullrefman`' rather than '`refman`'), use

`make pdf`	to create PDF versions
`make info`	to create info files (not '`refman`' nor '`fullrefman`').

You will not be able to build any of these unless you have `texi2any` version 5.1 or later installed, and for PDF you must have `texi2dvi` and `texinfo.tex` installed (which are part of the GNU **texinfo** distribution but are, especially `texinfo.tex`, often made part of the TEX package in re-distributions). For historical reasons, the path to `texi2any` can be set by macro '`MAKEINFO`' in `config.site` (`makeinfo` is nowadays a link to `texi2any`).

The PDF versions can be viewed using any recent PDF viewer: they have hyperlinks that can be followed. The info files are suitable for reading online with Emacs or the standalone GNU `info` program. The PDF versions will be created using the paper size selected at configuration (default ISO a4): this can be overridden by setting `R_PAPERSIZE` on the `make` command line, or setting `R_PAPERSIZE` in the environment and using `make -e`. (If re-making the manuals for a different paper size, you should first delete the file `doc/manual/version.texi`. The usual value for North America would be '`letter`'.)

There are some issues with making the PDF reference manual, `fullrefman.pdf` or `refman.pdf`. The help files contain both ISO Latin1 characters (e.g. in `text.Rd`) and upright quotes, neither of which are contained in the standard LATEX Computer Modern fonts. We have provided four alternatives:

`times` (The default.) Using standard PostScript fonts, Times Roman, Helvetica and Courier. This works well both for on-screen viewing and for printing. One disadvantage is that the Usage and Examples sections may come out rather wide: this can be overcome by using *in addition* either of the options `inconsolata` (on a Unix-alike only if found by `configure`) or `beramono`, which replace the Courier monospaced font by Inconsolata or Bera Sans mono respectively. (You will need a recent version of the appropriate LATEX package **inconsolata**[3] or **bera** installed.)

Note that in most LATEX installations this will not actually use the standard fonts for PDF, but rather embed the URW clones NimbusRom, NimbusSans and (for Courier, if used) NimbusMon.

This needs LATEX packages **times**, **helvetic** and (if used) **courier** installed.

`lm` Using the *Latin Modern* fonts. These are not often installed as part of a TEX distribution, but can obtained from `https://www.ctan.org/tex-archive/fonts/ps-type1/lm/` and mirrors. This uses fonts rather similar to Computer Modern, but is not so good on-screen as `times`.

`cm-super` Using type-1 versions of the Computer Modern fonts by Vladimir Volovich. This is a large installation, obtainable from `https://www.ctan.org/tex-archive/fonts/ps-type1/cm-super/` and its mirrors. These type-1 fonts have poor hinting and so are nowhere near as readable on-screen as the other three options.

`ae` A package to use composites of Computer Modern fonts. This works well most of the time, and its PDF is more readable on-screen than the previous two options. There are three fonts for which it will need to use bitmapped fonts, `tctt0900.600pk`,

[3] Instructions on how to install the latest version are at `https://www.ctan.org/tex-archive/fonts/inconsolata/`.

`tctt1000.600pk` and `tcrm1000.600pk`. Unfortunately, if those files are not available, Acrobat Reader will substitute completely incorrect glyphs so you need to examine the logs carefully.

The default can be overridden by setting the environment variable `R_RD4PDF`. (On Unix-alikes, this will be picked up at install time and stored in `etc/Renviron`, but can still be overridden when the manuals are built, using `make -e`.) The usual[4] default value for `R_RD4PDF` is '`times,inconsolata,hyper`': omit '`hyper`' if you do not want hyperlinks (e.g. for printing the manual) or do not have LaTeX package **hyperref**, and omit '`inconsolata`' if you do not have LaTeX package **inconsolata** installed.

Further options, e.g for **hyperref**, can be included in a file `Rd.cfg` somewhere on your LaTeX search path. For example, if you prefer the text and not the page number in the table of contents to be hyperlinked use

```
\ifthenelse{\boolean{Rd@use@hyper}}{\hypersetup{linktoc=section}}{}
```

or

```
\ifthenelse{\boolean{Rd@use@hyper}}{\hypersetup{linktoc=all}}{}
```

to hyperlink both text and page number.

Ebook versions of most of the manuals in one or both of `.epub` and `.mobi` formats can be made by running in `doc/manual` one of

```
make ebooks
make epub
make mobi
```

This requires `ebook-convert` from `Calibre` (`http://calibre-ebook.com/download`), or from most Linux distributions). If necessary the path to `ebook-convert` can be set as make macro `EBOOK` to by editing `doc/manual/Makefile` (which contains a commented value suitable for OS X).

2.4 Installation

To ensure that the installed tree is usable by the right group of users, set `umask` appropriately (perhaps to '`022`') before unpacking the sources and throughout the build process.

After

```
./configure
make
make check
```

(or, when building outside the source, *`TOP_SRCDIR`*`/configure`, etc) have been completed successfully, you can install the complete R tree to your system by typing

```
make install
```

A parallel make can be used (but run `make` before `make install`). Those using GNU `make` 4.0 or later may want to use `make -j` *`n`* `-O` to avoid interleaving of output.

This will install to the following directories:

`prefix``/bin` or *`bindir`*
: the front-end shell script and other scripts and executables

`prefix``/man/man1` or *`mandir`*`/man1`
: the man page

[4] on a Unix-alike, '`inconsolata`' is omitted if not found by `configure`.

prefix`/LIBnn/R` or *libdir*`/R`
: all the rest (libraries, on-line help system, ...). Here *LIBnn* is usually '`lib`', but may be '`lib64`' on some 64-bit Linux systems. This is known as the R home directory.

where *prefix* is determined during configuration (typically `/usr/local`) and can be set by running `configure` with the option `--prefix`, as in

```
./configure --prefix=/where/you/want/R/to/go
```

where the value should be an absolute path. This causes `make install` to install the R script to `/where/you/want/R/to/go/bin`, and so on. The prefix of the installation directories can be seen in the status message that is displayed at the end of `configure`. The installation may need to be done by the owner of *prefix*, often a root account.

You can install into another directory tree by using

```
make prefix=/path/to/here install
```

at least with GNU or Solaris `make` (but not some older Unix makes).

More precise control is available at configure time via options: see `configure --help` for details. (However, most of the 'Fine tuning of the installation directories' options are not used by R.)

Configure options `--bindir` and `--mandir` are supported and govern where a copy of the `R` script and the `man` page are installed.

The configure option `--libdir` controls where the main R files are installed: the default is '*eprefix*`/LIBnn`', where *eprefix* is the prefix used for installing architecture-dependent files, defaults to *prefix*, and can be set via the configure option `--exec-prefix`.

Each of `bindir`, `mandir` and `libdir` can also be specified on the `make install` command line (at least for GNU `make`).

The `configure` or `make` variables `rdocdir` and `rsharedir` can be used to install the system-independent `doc` and `share` directories to somewhere other than `libdir`. The C header files can be installed to the value of `rincludedir`: note that as the headers are not installed into a subdirectory you probably want something like `rincludedir=/usr/local/include/R-3.3.1`.

If you want the R home to be something other than *libdir*`/R`, use `rhome`: for example

```
make install rhome=/usr/local/lib64/R-3.3.1
```

will use a version-specific R home on a non-Debian Linux 64-bit system.

If you have made R as a shared/static library you can install it in your system's library directory by

```
make prefix=/path/to/here install-libR
```

where `prefix` is optional, and `libdir` will give more precise control.[5] However, you should not install to a directory mentioned in `LDPATHS` (e.g. `/usr/local/lib64`) if you intend to work with multiple versions of R, since that directory may be given precedence over the `lib` directory of other R installations.

```
make install-strip
```

will install stripped executables, and on platforms where this is supported, stripped libraries in directories `lib` and `modules` and in the standard packages.

Note that installing R into a directory whose path contains spaces is not supported, and some aspects (such as installing source packages) will not work.

To install info and PDF versions of the manuals, use one or both of

```
make install-info
make install-pdf
```

[5] This will be needed if more than one sub-architecture is to be installed.

Once again, it is optional to specify `prefix`, `libdir` or `rhome` (the PDF manuals are installed under the R home directory). (`make install-info` needs `Perl` installed if there is no command `install-info` on the system.)

More precise control is possible. For info, the setting used is that of `infodir` (default *`prefix`*`/info`, set by configure option `--infodir`). The PDF files are installed into the R `doc` tree, set by the `make` variable `rdocdir`.

A staged installation is possible, that it is installing R into a temporary directory in order to move the installed tree to its final destination. In this case `prefix` (and so on) should reflect the final destination, and `DESTDIR` should be used: see `https://www.gnu.org/prep/standards/html_node/DESTDIR.html`.

You can optionally install the run-time tests that are part of `make check-all` by

```
make install-tests
```

which populates a `tests` directory in the installation.

2.5 Uninstallation

You can uninstall R by

```
make uninstall
```

optionally specifying `prefix` etc in the same way as specified for installation.

This will also uninstall any installed manuals. There are specific targets to uninstall info and PDF manuals in file `doc/manual/Makefile`.

Target `uninstall-tests` will uninstall any installed tests, as well as removing the directory `tests` containing the test results.

An installed shared/static `libR` can be uninstalled by

```
make prefix=/path/to/here uninstall-libR
```

2.6 Sub-architectures

Some platforms can support closely related builds of R which can share all but the executables and dynamic objects. Examples include builds under Linux and Solaris for different CPUs or 32- and 64-bit builds.

R supports the idea of architecture-specific builds, specified by adding '`r_arch=`*`name`*' to the `configure` line. Here *name* can be anything non-empty, and is used to name subdirectories of `lib`, `etc`, `include` and the package `libs` subdirectories. Example names from other software are the use of `sparcv9` on Sparc Solaris and `32` by `gcc` on '`x86_64`' Linux.

If you have two or more such builds you can install them over each other (and for 32/64-bit builds on one architecture, one build can be done without '`r_arch`'). The space savings can be considerable: on '`x86_64`' Linux a basic install (without debugging symbols) took 74Mb, and adding a 32-bit build added 6Mb. If you have installed multiple builds you can select which build to run by

```
R --arch=name
```

and just running '`R`' will run the last build that was installed.

`R CMD INSTALL` will detect if more than one build is installed and try to install packages with the appropriate library objects for each. This will not be done if the package has an executable `configure` script or a `src/Makefile` file. In such cases you can install for extra builds by

```
R --arch=name CMD INSTALL --libs-only pkg1 pkg2 ...
```

If you want to mix sub-architectures compiled on different platforms (for example '`x86_64`' Linux and '`i686`' Linux), it is wise to use explicit names for each, and you may also need to set `libdir` to ensure that they install into the same place.

When sub-architectures are used the version of `Rscript` in e.g. `/usr/bin` will be the last installed, but architecture-specific versions will be available in e.g. `/usr/lib64/R/bin/exec${R_ARCH}`. Normally all installed architectures will run on the platform so the architecture of `Rscript` itself does not matter. The executable `Rscript` will run the `R` script, and at that time the setting of the `R_ARCH` environment variable determines the architecture which is run.

When running post-install tests with sub-architectures, use

```
R --arch=name CMD make check[-devel|all]
```

to select a sub-architecture to check.

Sub-architectures are also used on Windows, but by selecting executables within the appropriate `bin` directory, *R_HOME*`/bin/i386` or *R_HOME*`/bin/x64`. For backwards compatibility there are executables *R_HOME*`/bin/R.exe` and *R_HOME*`/bin/Rscript.exe`: these will run an executable from one of the subdirectories, which one being taken first from the `R_ARCH` environment variable, then from the `--arch` command-line option[6] and finally from the installation default (which is 32-bit for a combined 32/64 bit R installation).

2.6.1 Multilib

For some Linux distributions[7], there is an alternative mechanism for mixing 32-bit and 64-bit libraries known as *multilib*. If the Linux distribution supports multilib, then parallel builds of R may be installed in the sub-directories `lib` (32-bit) and `lib64` (64-bit). The build to be run may then be selected using the `setarch` command. For example, a 32-bit build may be run by

```
setarch i686 R
```

The `setarch` command is only operational if both 32-bit and 64-bit builds are installed. If there is only one installation of R, then this will always be run regardless of the architecture specified by the `setarch` command.

There can be problems with installing packages on the non-native architecture. It is a good idea to run e.g. `setarch i686 R` for sessions in which packages are to be installed, even if that is the only version of R installed (since this tells the package installation code the architecture needed).

There is a potential problem with packages using Java, as the post-install for a '`i686`' RPM on '`x86_64`' Linux reconfigures Java and will find the '`x86_64`' Java. If you know where a 32-bit Java is installed you may be able to run (as root)

```
export JAVA_HOME=<path to jre directory of 32-bit Java>
setarch i686 R CMD javareconf
```

to get a suitable setting.

When this mechanism is used, the version of `Rscript` in e.g. `/usr/bin` will be the last installed, but an architecture-specific version will be available in e.g. `/usr/lib64/R/bin`. Normally all installed architectures will run on the platform so the architecture of `Rscript` does not matter.

2.7 Other Options

There are many other installation options, most of which are listed by `configure --help`. Almost all of those not listed elsewhere in this manual are either standard `autoconf` options not relevant to R or intended for specialist uses by the R developers.

One that may be useful when working on R itself is the option `--disable-byte-compiled-packages`, which ensures that the base and recommended packages are not byte-compiled. (Alternatively the (make or environment) variable `R_NO_BASE_COMPILE` can be set to a non-empty value for the duration of the build.)

[6] with possible values '`i386`', '`x64`', '`32`' and '`64`'.

[7] mainly on RedHat and Fedora, whose layout is described here.

Option `--with-internal-tzcode` makes use of R's own code and copy of the Olson database for managing timezones. This will be preferred where there are issues with the system implementation, usually involving times after 2037 or before 1916. An alternative time-zone directory[8] can be used, pointed to by environment variable `TZDIR`: this should contain files such as `Europe/London`. On all tested OSes the system timezone was deduced correctly, but if necessary it can be set as the value of environment variable `TZ`.

2.7.1 OpenMP Support

By default `configure` searches for suitable options[9] for OpenMP support for the C, C++98, FORTRAN 77 and Fortran compilers.

Only the C result is currently used for R itself, and only if `MAIN_LD`/`DYLIB_LD` were not specified. This can be overridden by specifying

```
R_OPENMP_CFLAGS
```

Use for packages has similar restrictions (involving `SHLIB_LD` and similar: note that as FORTRAN 77 code is normally linked by the C compiler, both need to support OpenMP) and can be overridden by specifying some of

```
SHLIB_OPENMP_CFLAGS
SHLIB_OPENMP_CXXFLAGS
SHLIB_OPENMP_FCFLAGS
SHLIB_OPENMP_FFLAGS
```

Setting to an empty value will disable OpenMP for that compiler (and configuring with `--disable-openmp` will disable all detection of OpenMP). The `configure` detection test is to compile and link a standalone OpenMP program, which is not the same as compiling a shared object and loading it into the C program of R's executable. Note that overridden values are not tested.

2.8 Testing an Installation

Full post-installation testing is possible only if the test files have been installed with

```
make install-tests
```

which populates a `tests` directory in the installation.

If this has been done, two testing routes are available. The first is to move to the home directory of the R installation (as given by `R.home()`) and run

```
cd tests
## followed by one of
../bin/R CMD make check
../bin/R CMD make check-devel
../bin/R CMD make check-all
```

and other useful targets are `test-BasePackages` and `test-Recommended` to the run tests of the standard and recommended packages (if installed) respectively.

This re-runs all the tests relevant to the installed R (including for example code in the package vignettes), but not for example the ones checking the example code in the manuals nor making the standalone Rmath library. This can occasionally be useful when the operating environment has been changed, for example by OS updates or by substituting the BLAS (see Section A.3.1.5 [Shared BLAS], page 44).

Parallel checking of packages may be possible: set the environment variable `TEST_MC_CORES` to the maximum number of processes to be run in parallel. This affects both checking the

[8] How to prepare such a directory is described in file `src/extra/tzone/Notes` in the R sources.

[9] for example, `-fopenmp`, `-xopenmp` or `-qopenmp`. This includes for `clang` 3.7.x and the Intel C compiler.

package examples (part of `make check`) and package sources (part of `make check-devel` and `make check-recommended`). It does require a `make` command which supports the `make -j n` option: most do but on Solaris you need to select GNU `make` or `dmake`. Where parallel checking of package sources is done, a log file *`pngname`*`.log` is left in the `tests` directory for inspection.

Alternatively, the installed R can be run, preferably with `--vanilla`. Then

```
Sys.setenv(LC_COLLATE = "C", LC_TIME = "C", LANGUAGE = "en")
library("tools")
testInstalledBasic("both")
testInstalledPackages(scope = "base")
testInstalledPackages(scope = "recommended")
```

runs the basic tests and then all the tests on the standard and recommended packages. These tests can be run from anywhere: the basic tests write their results in the `tests` folder of the R home directory and run fewer tests than the first approach: in particular they do not test things which need Internet access—that can be tested by

```
testInstalledBasic("internet")
```

These tests work best if `diff` (in `Rtools*.exe` for Windows users) is in the path.

It is possible to test the installed packages (but not their package-specific tests) by `testInstalledPackages` even if `make install-tests` was not run.

Note that the results may depend on the language set for times and messages: for maximal similarity to reference results you may want to try setting (before starting the R session)

```
LANGUAGE=en
```

and use a UTF-8 or Latin-1 locale.

3 Installing R under Windows

The `bin/windows` directory of a CRAN site contains binaries for a base distribution and a large number of add-on packages from CRAN to run on 32- or 64-bit Windows (XP or later) on '`ix86`' and '`x86_64`' CPUs.

Your file system must allow long file names (as is likely except perhaps for some network-mounted systems). If it doesn't also support conversion to short name equivalents (a.k.a. DOS 8.3 names), then R *must* be installed in a path that does not contain spaces.

Installation is *via* the installer `R-3.3.1-win.exe`. Just double-click on the icon and follow the instructions. When installing on a 64-bit version of Windows the options will include 32- or 64-bit versions of R (and the default is to install both). You can uninstall R from the Control Panel.

Note that you will be asked to choose a language for installation, and that choice applies to both installation and un-installation but not to running R itself.

See the R Windows FAQ (`https://CRAN.R-project.org/bin/windows/base/rw-FAQ.html`) for more details on the binary installer.

3.1 Building from source

R can be built as either a 32-bit or 64-bit application on Windows: to build the 64-bit application you need a 64-bit edition of Windows: such an OS can also be used to build 32-bit R.

The standard installer combines 32-bit and 64-bit builds into a single executable which can then be installed into the same location and share all the files except the `.exe` and `.dll` files and some configuration files in the `etc` directory.

Building is only tested in a 8-bit locale: using a multi-byte locale (as used for CJK languages) is unsupported and may not work (the scripts do try to select a '`C`' locale; Windows may not honour this).

NB: The build process is currently being changed to require external binary distributions of third-party software. Their location is set using macro `EXT_LIBS` with default setting `$(LOCAL_SOFT)`; the $(LOCAL_SOFT) macro defaults to `$(R_HOME)/extsoft`. This directory can be populated using `make rsync-extsoft`. The location can be overridden by setting `EXT_LIBS` to a different path in `src/gnuwin32/MkRules.local`. A suitable collection of files can also be obtained from `https://CRAN.R-project.org/bin/windows/extsoft` or `https://www.stats.ox.ac.uk/pub/Rtools/libs.html`.

3.1.1 Getting the tools

If you want to build R from the sources, you will first need to collect, install and test an extensive set of tools. See Appendix D [The Windows toolset], page 67, (and perhaps updates in `https://CRAN.R-project.org/bin/windows/Rtools/`) for details.

The `Rtools*.exe` executable installer described in Appendix D [The Windows toolset], page 67, also includes some source files in addition to the R source as noted below. You should run it first, to obtain a working `tar` and other necessities. Choose a "Full installation", and install the extra files into your intended R source directory, e.g. `C:/R`. The directory name *should not contain spaces*. We will call this directory *`R_HOME`* below.

3.1.2 Getting the source files

You need to collect the following sets of files:

- Get the R source code tarball `R-3.3.1.tar.gz` from CRAN. Open a command window (or another shell) at directory *R_HOME*, and run

```
tar -xf R-3.3.1.tar.gz
```

to create the source tree in *R_HOME*. **Beware**: do use `tar` to extract the sources rather than tools such as WinZip. If you are using an account with administrative privileges you may get a lot of messages which can be suppressed by

```
tar --no-same-owner -xf R-3.3.1.tar.gz
```

or perhaps better, set the environment variable `TAR_OPTIONS` to the value '`--no-same-owner --no-same-permissions`'.

It is also possible to obtain the source code using Subversion; see Chapter 1 [Obtaining R], page 1, for details.

- If you are not using a tarball you need to obtain copies of the recommended packages from CRAN. Put the `.tar.gz` files in *R_HOME*`/src/library/Recommended` and run `make link-recommended`. If you have an Internet connection, you can do this automatically by running in *R_HOME*`/src/gnuwin32`

  ```
  make rsync-recommended
  ```

- The binary distributions of external software. Download

  ```
  https://www.stats.ox.ac.uk/pub/Rtools/goodies/multilib/local323.zip
  ```

  (or a more recent version if appropriate), create an empty directory, say `c:/R/extsoft`, and unpack it in that directory by e.g.

  ```
  unzip local323.zip -d c:/R/extsoft
  ```

- Make a local copy of the configuration rules by

  ```
  cd R_HOME/src/gnuwin32
  cp MkRules.dist MkRules.local
  ```

  and edit `MkRules.local`, uncommenting `EXT_LIBS` and setting it to the appropriate path (in our example `c:/R/extsoft`).

  Look through the file `MkRules.local` and make any other changes needed: in particular, this is where a 64-bit build is selected and the locations are set of external software for ICU collation and the cairo-based devices.

The following additional item is normally installed by `Rtools*.exe`. If instead you choose to do a completely manual build you will also need

- The Tcl/Tk support files are contained in `Rtools*.exe` and available as `.zip` files from `https://www.stats.ox.ac.uk/pub/Rtools`. Please make sure you install the right version: there is a 32-bit version and a 64-bit version. They should be installed to *R_HOME*, creating directory `Tcl` there.

3.1.3 Building the core files

Set the environment variable `TMPDIR` to the absolute path to a writable directory, with a path specified with forward slashes and no spaces. (The default is `/tmp`, which may not be useful on Windows.)

You may need to compile under a case-honouring file system: we found that a `samba`-mounted file system (which maps all file names to lower case) did not work.

Open a command window at *R_HOME*`/src/gnuwin32`, then run

```
make all recommended vignettes
```

and sit back and wait while the basic compile takes place.

Notes:

- We have had reports that earlier versions of anti-virus software locking up the machine, but not for several years. However, aggressive anti-virus checking such as the on-access scanning of Sophos can slow the build down several-fold.

- You can run a parallel make by e.g.

```
make -j4 all
make -j4 recommended
make vignettes
```

  but this is only likely to be worthwhile on a multi-core machine with ample memory, and is not 100% reliable.
- It is possible (mainly for those working on R itself) to set the (make or environment) variable `R_NO_BASE_COMPILE` to a non-empty value, which inhibits the byte-compilation of the base and recommended packages.

3.1.4 Building the cairo devices

The devices based on cairographics (`svg`, `cairo_pdf`, `cairo_ps` and the `type = "cairo"` versions of `png`, `jpeg`, `tiff` and `bmp`) are implemented in a separate DLL `winCairo.dll` which is loaded when one of these devices is first used. It is not built by default, and needs to be built (after `make all`) by `make cairodevices`.

To enable the building of these devices you need to install the static cairographics libraries built by Simon Urbanek at `https://www.rforge.net/Cairo/files/cairo-current-win.tar.gz`. Set the macro '`CAIRO_HOME`' in `MkRules.local`. (Note that this tarball unpacks with a top-level directory `src/`: '`CAIRO_HOME`' needs to include that directory in its path.)

3.1.5 Using ICU for collation

It is recommended to build R to support ICU (International Components for Unicode, `http://site.icu-project.org/`) for collation, as is commonly done on Unix-alikes.

Two settings are needed in `MkRules.local`,

```
# set to use ICU
# USE_ICU = YES
# path to parent of ICU headers
ICU_PATH = /path/to/ICU
```

The first should be uncommented and the second set to the top-level directory of a suitably packaged binary build of ICU, for example that at `https://www.stats.ox.ac.uk/pub/Rtools/goodies/ICU_531.zip`. Depending on the build, it may be necessary to edit the macro `ICU_LIBS`.

Unlike on a Unix-alike, it is normally necessary to call `icuSetCollate` to set a locale before ICU is actually used for collation, or set the environment variable `R_ICU_LOCALE`.

3.1.6 Support for libcurl

`libcurl` version 7.28.0 or later can be used to support `curlGetHeaders` and the `"libcurl"` methods of `download.file` and `url`.

A suitable distribution can be found *via* `https://www.stats.ox.ac.uk/pub/Rtools/libs.html` and its unpacked location should be specified in file `MkRules.local`.

For secure use of e.g. '`https://`' URLs Windows users may need to specify the path to up-to-date *CA root certificates*: see `?download.file`.

3.1.7 Checking the build

You can test a build by running

```
make check
```

The recommended packages can be checked by

```
make check-recommended
```

Other levels of checking are

```
make check-devel
```

for a more thorough check of the R functionality, and

```
make check-all
```

for both `check-devel` and `check-recommended`.

If a test fails, there will almost always be a `.Rout.fail` file in the directory being checked (often `tests/Examples` or `tests`): examine the file to help pinpoint the problem.

Parallel checking of package sources (part of `make check-devel` and `make check-recommended`) is possible: see the environment variable `TEST_MC_CORES` to the maximum number of processes to be run in parallel.

3.1.8 Building the manuals

The PDF manuals require **texinfo** 5.1 or later, and can be made by

```
make manuals
```

If you want to make the info versions (not including the Reference Manual), use

```
cd ../../doc/manual
make -f Makefile.win info
```

(all assuming you have `pdftex`/`pdflatex` installed and in your path).

See the Section 2.3 [Making the manuals], page 4, section in the Unix-alike section for setting options such as the paper size and the fonts used.

By default it is assumed that **texinfo** is not installed, and the manuals will not be built. The comments in file `MkRules.dist` describe settings to build them. (Copy that file to `MkRules.local` and edit it.) The **texinfo** 5.x package for use on Windows is available at `https://www.stats.ox.ac.uk/pub/Rtools/`: you will also need to install `Perl`[1]

3.1.9 Building the Inno Setup installer

You need to have the files for a complete R build, including bitmap and Tcl/Tk support and the manuals (which requires **texinfo** installed), as well as the recommended packages and Inno Setup (see Section D.2 [The Inno Setup installer], page 68).

Once everything is set up

```
make distribution
make check-all
```

will make all the pieces and the installer and put them in the `gnuwin32/cran` subdirectory, then check the build. This works by building all the parts in the sequence:

`rbuild` (the executables, the FAQ docs etc.)
`rpackages` (the base packages)
`htmldocs` (the HTML documentation)
`cairodevices` (the cairo-based graphics devices)
`recommended` (the recommended packages)
`vignettes` (the vignettes in base packages:
only needed if building from an `svn` checkout)
`manuals` (the PDF manuals)
`rinstaller` (the install program)
`crandir` (the CRAN distribution directory, only for 64-bit builds)

The parts can be made individually if a full build is not needed, but earlier parts must be built before later ones. (The `Makefile` doesn't enforce this dependency—some build targets

[1] Suitable distributions include Strawberry Perl, `http://strawberryperl.com/` and ActivePerl, `https://www.activestate.com/activeperl`.

force a lot of computation even if all files are up to date.) The first four targets are the default build if just `make` (or `make all`) is run.

Parallel make is not supported and likely to fail.

If you want to customize the installation by adding extra packages, replace `make rinstaller` by something like

```
make rinstaller EXTRA_PKGS='pkg1 pkg2 pkg3'
```

An alternative way to customize the installer starting with a binary distribution is to first make an installation of R from the standard installer, then add packages and make other customizations to that installation. Then (after having customized file `MkRules`, possibly *via* `MkRules.local`, and having made R in the source tree) in `src/gnuwin32/installer` run

```
make myR IMAGEDIR=rootdir
```

where `rootdir` is the path to the root of the customized installation (in double quotes if it contains spaces or backslashes).

Both methods create an executable with a standard name such as `R-3.3.1-win.exe`, so please rename it to indicate that it is customized. If you intend to *distribute* a customized installer please do check that license requirements are met – note that the installer will state that the contents are distributed under GPL and this has a requirement for *you* to supply the complete sources (including the R sources even if you started with a binary distribution of R, and also the sources of any extra packages (including their external software) which are included).

The defaults for the startup parameters may also be customized. For example

```
make myR IMAGEDIR=rootdir MDISDI=1
```

will create an installer that defaults to installing R to run in SDI mode. See `src/gnuwin32/installer/Makefile` for the names and values that can be set.

The standard CRAN distribution of a 32/64-bit installer is made by first building 32-bit R (just

```
make 32-bit
```

is needed), and then (in a separate directory) building 64-bit R with the macro `HOME32` set in file `MkRules.local` to the top-level directory of the 32-bit build. Then the `make rinstaller` step copies the files that differ between architectures from the 32-bit build as it builds the installer image.

3.1.10 Building the MSI installer

It is also possible to build an installer for use with Microsoft Installer. This is intended for use by sysadmins doing automated installs, and is not recommended for casual use.

It makes use of the Windows Installer XML (WiX) toolkit *version 3.5* (or perhaps later, untested) available from `http://wixtoolset.org/`. Once WiX is installed, set the path to its home directory in `MkRules.local`.

You need to have the files for a complete R build, including bitmap and Tcl/Tk support and the manuals, as well as the recommended packages. There is no option in the installer to customize startup options, so edit `etc/Rconsole` and `etc/Rprofile.site` to set these as required. Then

```
cd installer
make msi
```

which will result in a file with a name like `R-3.3.1-win32.msi`. This can be double-clicked to be installed, but those who need it will know what to do with it (usually by running `msiexec /i` with additional options). Properties that users might want to set from the `msiexec` command line include '`ALLUSERS`', '`INSTALLDIR`' (something like `c:\Program Files\R\R-3.3.1`) and '`RMENU`'

(the path to the 'R' folder on the start menu) and 'STARTDIR' (the starting directory for R shortcuts, defaulting to something like c:\Users\name\Documents\R).

The MSI installer can be built both from a 32-bit build of R (R-3.3.1-win32.msi) and from a 64-bit build of R (R-3.3.1-win64.msi, optionally including 32-bit files by setting the macro HOME32, when the name is R-3.3.1-win.msi). Unlike the main installer, a 64-bit MSI installer can only be run on 64-bit Windows.

Thanks to David del Campo (Dept of Statistics, University of Oxford) for suggesting WiX and building a prototype installer.

3.1.11 64-bit Windows builds

To build a 64-bit version of R you need a 64-bit toolchain: the only one discussed here is based on the work of the MinGW-w64 project (http://sourceforge.net/projects/mingw-w64/, but commercial compilers such as those from Intel and PGI could be used (and have been by R redistributors).

Support for MinGW-w64 was developed in the R sources over the period 2008–10 and was first released as part of R 2.11.0. The assistance of Yu Gong at a crucial step in porting R to MinGW-w64 is gratefully acknowledged, as well as help from Kai Tietz, the lead developer of the MinGW-w64 project.

Windows 64-bit is now completely integrated into the R and package build systems: a 64-bit build is selected in file MkRules.local.

3.2 Testing an Installation

The Windows installer contains a set of test files used when building R.

The Rtools are not needed to run these tests. but more comprehensive analysis of errors will be given if diff is in the path (and errorsAreFatal = FALSE is then not needed below).

Launch either Rgui or Rterm, preferably with --vanilla. Then run

```
Sys.setenv(LC_COLLATE = "C", LANGUAGE = "en")
library("tools")
testInstalledBasic("both")
testInstalledPackages(scope = "base", errorsAreFatal = FALSE)
testInstalledPackages(scope = "recommended", errorsAreFatal = FALSE)
```

runs the basic tests and then all the tests on the standard and recommended packages. These tests can be run from anywhere: they write some of their results in the tests folder of the R home directory (as given by R.home()), and hence may need to be run under the account used to install R.

The results of example(md5sums) when testing **tools** will differ from the reference output as some files are installed with Windows' CRLF line endings.

4 Installing R under OS X

The front page of a CRAN site has a link 'Download R for OS X'. Click on that, then download the file `R-3.3.1.pkg` and install it. This runs on OS X 10.9 and later (Mavericks, Yosemite, El Capitan, ...).

Installers for R-patched and R-devel are usually available from `https://r.research.att.com`.

For some older versions of the OS you can in principle (it is little tested) install R from the sources.

It is important that if you use a binary installer package that your OS is fully updated: look at 'Updates' from the 'App Store' to be sure. (If using XQuartz, check that is current.)

To install, just double-click on the icon of the file you downloaded. At the 'Installation Type' stage, note the option to 'Customize'. This currently shows four components: everyone will need the 'R Framework' component: the remaining components are optional. (The 'Tcl/Tk' component is needed to use package **tcltk**. The 'Texinfo' component is only needed by those installing source packages.)

This is an Apple Installer package. If you encounter any problem during the installation, please check the Installer log by clicking on the "Window" menu and item "Installer Log". The full output (select "Show All Log") is useful for tracking down problems. Note the the installer is clever enough to try to upgrade the last-installed version of the application where you installed it (which may not be where you want this time ...).

Various parts of the build require XQuartz to be installed: : see `https://xquartz.macosforge.org/`. These include the **tcltk** package and the `X11` device: attempting to use these without XQuartz will remind you.

If you update your OS X version, you should re-install R (and perhaps XQuartz): the installer tailors the installation to the current version of the OS.

For building R from source, see Section C.3 [OS X], page 56.

4.1 Running R under OS X

There are two ways to run R on OS X from a CRAN binary distribution.

There is a GUI console normally installed with the R icon in `/Applications` which you can run by double-clicking (e.g. from Launchpad or Finder). (If you cannot find it there it was possibly installed elsewhere so try searching for it in Spotlight.) This is usually referred to as R.app to distinguish it from command-line R: its user manual is currently part of the OS X FAQ at `https://cran.r-project.org/bin/macosx/RMacOSX-FAQ.html` and can be viewed from R.app's 'Help' menu.

You can run command-line R and `Rscript` from a Terminal[1] so these can be typed as commands like any other Unix-alike: see the next chapter of this manual. There are some small differences which may surprise users of R on other platforms, notably the default location of the personal library directory (under `~/Library/R`, e.g. `~/Library/R/3.3/library`), and that warnings, messages and other output to `stderr` are highlighted in bold.

It has been reported that running R.app under Yosemite may fail if no preferences are stored, so if it fails when launched for the very first time, try it again (the first attempt will store some preferences).

[1] The installer as puts links to `R` and `Rscript` in `/usr/bin` (Mavericks, Yosemite) or `/usr/local/bin` (El Capitan and later). If these are missing, you can run directly the versions in `/Library/Frameworks/R.framework/Resources/`.

Users of R.app need to be aware of the 'App Nap' feature (`https://developer.apple.com/library/mac/releasenotes/MacOSX/WhatsNewInOSX/Articles/MacOSX10_9.html`) which can cause R tasks to appear to run very slowly when not producing output in the console. Here are ways to avoid it:

- Ensure that the console is completely visible (or at least the activity indicator at the top right corner is visible).
- In a Terminal, run

  ```
  defaults write org.R-project.R NSAppSleepDisabled -bool YES
  ```

  (see `https://developer.apple.com/library/mac/releasenotes/MacOSX/WhatsNewInOSX/Articles/MacOSX10_9.html`).

Using the `X11` device or the X11-based versions of `View()` and `edit()` for data frames and matrices (the latter are the default for command-line R but not R.app) requires an X sub-system to be installed: see Section C.3 [OS X], page 56. (As do the **tcltk** package and some third-party packages.)

4.2 Uninstalling under OS X

R for OS X consists of two parts: the GUI (R.app) and the R framework. The un-installation is as simple as removing those folders (e.g. by dragging them into the Trash). The typical installation will install the GUI into the `/Applications/R.app` folder and the R framework into the `/Library/Frameworks/R.framework` folder. The links to `R` and `Rscript` in `/usr/bin` or `/usr/local/bin` should also be removed.

If you want to get rid of R more completely using a Terminal, simply run (use `/usr/local/bin` as from El Capitan):

```
sudo rm -rf /Library/Frameworks/R.framework /Applications/R.app \
   /usr/bin/R /usr/bin/Rscript
```

The installation consists of four Apple packages: `org.r-project.R.x86_64.fw.pkg`, `org.r-project.R.x86_64.GUI.pkg`, `org.r-project.x86_64.tcltk.x11` and `org.r-project.x86_64.texinfo` (not all of which need be installed). You can use `pkgutil --forget` if you want the Apple Installer to forget about the package without deleting its files (useful for the R framework when installing multiple R versions in parallel), or after you have deleted the files.

Uninstalling the Tcl/Tk or Texinfo components (which are installed under `/usr/local`) is not as simple. You can list the files they installed in a Terminal by

```
pkgutil --files org.r-project.x86_64.tcltk.x11
pkgutil --files org.r-project.x86_64.texinfo
```

These are paths relative to `/`, the root of the file system.

4.3 Multiple versions

The installer will remove any previous version of the R framework which it finds installed. This can be avoided by using `pkgutil --forget` (see the previous section). However, note that different versions are installed under `/Library/Frameworks/R.framework/Versions` as `3.2`, `3.3` and so on, so it is not possible to have different '`3.x.y`' versions installed for the same '`x`'.

A version of R can be run directly from the command-line as e.g.

```
/Library/Frameworks/R.framework/Versions/3.3/Resources/bin/R
```

However, R.app will always run the 'current' version, that is the last installed version. A small utility, `Rswitch.app` (available at `https://r.research.att.com/#other`), can be used to change the 'current' version. This is of limited use as R.app is compiled against a particular

version of R and will likely crash if switched to an earlier version. This may allow you to install a development version of R (de-selecting R.APP) and then switch back to the release version.

5 Running R

How to start R and what command-line options are available is discussed in Section "Invoking R" in *An Introduction to R*.

You should ensure that the shell has set adequate resource limits: R expects a stack size of at least 8MB and to be able to open at least 256 file descriptors. (Any modern OS should have default limits at least as large as these, but apparently NetBSD may not. Use the shell command `ulimit` (`sh`/`bash`) or `limit` (`csh`/`tcsh`) to check.)

R makes use of a number of environment variables, the default values of many of which are set in file *`R_HOME`*`/etc/Renviron` (there are none set by default on Windows and hence no such file). These are set at `configure` time, and you would not normally want to change them – a possible exception is `R_PAPERSIZE` (see Section B.3.1 [Setting paper size], page 47). The paper size will be deduced from the '`LC_PAPER`' locale category if it exists and `R_PAPERSIZE` is unset, and this will normally produce the right choice from '`a4`' and '`letter`' on modern Unix-alikes (but can always be overridden by setting `R_PAPERSIZE`).

Various environment variables can be set to determine where R creates its per-session temporary directory. The environment variables `TMPDIR`, `TMP` and `TEMP` are searched in turn and the first one which is set and points to a writable area is used. If none do, the final default is `/tmp` on Unix-alikes and the value of `R_USER` on Windows. The path should be an absolute path not containing spaces (and it is best to avoid non-alphanumeric characters such as `+`).

Some Unix-alike systems are set up to remove files and directories periodically from `/tmp`, for example by a `cron` job running `tmpwatch`. Set `TMPDIR` to another directory before starting long-running jobs on such a system.

Note that `TMPDIR` will be used to execute `configure` scripts when installing packages, so if `/tmp` has been mounted as '`noexec`', `TMPDIR` needs to be set to a directory from which execution is allowed.

6 Add-on packages

It is helpful to use the correct terminology. A *package* is loaded from a *library* by the function `library()`. Thus a library is a directory containing installed packages; the main library is *R_HOME*`/library`, but others can be used, for example by setting the environment variable `R_LIBS` or using the R function `.libPaths()`.

6.1 Default packages

The set of packages loaded on startup is by default

```
> getOption("defaultPackages")
[1] "datasets"  "utils"     "grDevices" "graphics"  "stats"     "methods"
```

(plus, of course, **base**) and this can be changed by setting the option in startup code (e.g. in `~/.Rprofile`). It is initially set to the value of the environment variable `R_DEFAULT_PACKAGES` if set (as a comma-separated list). Setting `R_DEFAULT_PACKAGES=NULL` ensures that only package **base** is loaded.

Changing the set of default packages is normally used to reduce the set for speed when scripting: in particular not using **methods** will reduce the start-up time by a factor of up to two (and this is done by `Rscript`). But it can also be used to customize R, e.g. for class use.

6.2 Managing libraries

R packages are installed into *libraries*, which are directories in the file system containing a subdirectory for each package installed there.

R comes with a single library, *R_HOME*`/library` which is the value of the R object '`.Library`' containing the standard and recommended[1] packages. Both sites and users can create others and make use of them (or not) in an R session. At the lowest level '`.libPaths()`' can be used to add paths to the collection of libraries or to report the current collection.

R will automatically make use of a site-specific library *R_HOME*`/site-library` if this exists (it does not in a vanilla R installation). This location can be overridden by setting[2] '`.Library.site`' in *R_HOME*`/etc/Rprofile.site`, or (not recommended) by setting the environment variable `R_LIBS_SITE`. Like '`.Library`', the site libraries are always included by '`.libPaths()`'.

Users can have one or more libraries, normally specified by the environment variable `R_LIBS_USER`. This has a default value (to see it, use '`Sys.getenv("R_LIBS_USER")`' within an R session), but that is only used if the corresponding directory actually exists (which by default it will not).

Both `R_LIBS_USER` and `R_LIBS_SITE` can specify multiple library paths, separated by colons (semicolons on Windows).

6.3 Installing packages

Packages may be distributed in source form or compiled binary form. Installing source packages which contain C/C++/Fortran code requires that compilers and related tools be installed. Binary packages are platform-specific and generally need no special tools to install, but see the documentation for your platform for details.

Note that you may need to specify implicitly or explicitly the library to which the package is to be installed. This is only an issue if you have more than one library, of course.

[1] unless they were excluded in the build.

[2] its binding is locked once the startup files have been read, so users cannot easily change it.

Ensure that the environment variable `TMPDIR` is either unset (and `/tmp` exists and can be written in and executed from) or is the absolute path to a valid temporary directory, not containing spaces.

For most users it suffices to call '`install.packages(`*`pkgname`*`)`' or its GUI equivalent if the intention is to install a CRAN package and internet access is available.[3] On most systems '`install.packages()`' will allow packages to be selected from a list box (typically with several thousand items).

To install packages from source on a Unix-alike use in a terminal

```
R CMD INSTALL -l /path/to/library pkg1 pkg2 ...
```

The part '`-l /path/to/library`' can be omitted, in which case the first library of a normal R session is used (that shown by `.libPaths()[1]`).

There are a number of options available: use `R CMD INSTALL --help` to see the current list.

Alternatively, packages can be downloaded and installed from within R. First choose your nearest CRAN mirror using `chooseCRANmirror()`. Then download and install packages **pkg1** and **pkg2** by

```
> install.packages(c("pkg1", "pkg2"))
```

The essential dependencies of the specified packages will also be fetched. Unless the library is specified (argument `lib`) the first library in the library search path is used: if this is not writable, R will ask the user (in an interactive session) if the default personal library should be created, and if allowed to will install the packages there.

If you want to fetch a package and all those it depends on (in any way) that are not already installed, use e.g.

```
> install.packages("Rcmdr", dependencies = TRUE)
```

`install.packages` can install a source package from a local `.tar.gz` file (or a URL to such a file) by setting argument `repos` to `NULL`: this will be selected automatically if the name given is a single `.tar.gz` file.

`install.packages` can look in several repositories, specified as a character vector by the argument `repos`: these can include a CRAN mirror, Bioconductor, R-forge, rforge.net, local archives, local files, ...). Function `setRepositories()` can select amongst those repositories that the R installation is aware of.

Naive users sometimes forget that as well as installing a package, they have to use `library` to make its functionality available.

6.3.1 Windows

What `install.packages` does by default is different on Unix-alikes (except OS X) and Windows. On Unix-alikes it consults the list of available *source* packages on CRAN (or other repository/ies), downloads the latest version of the package sources, and installs them (via `R CMD INSTALL`). On Windows it looks (by default) first at the list of *binary* versions of packages available for your version of R and downloads the latest versions (if any). If no binary version is available or the source version is newer, it will install the source versions of packages without compiled C/C++/Fortran code, and offer to do so for those with, if `make` is available (and this can be tuned by option `"install.packages.compile.from.source"`).

On Windows `install.packages` can also install a binary package from a local `zip` file (or the URL of such a file) by setting argument `repos` to `NULL`. `Rgui.exe` has a menu `Packages` with a GUI interface to `install.packages`, `update.packages` and `library`.

Windows binary packages for R are distributed as a single binary containing either or both architectures (32- and 64-bit).

[3] If a proxy needs to be set, see `?download.file`.

A few of the binary packages need other software to be installed on your system: see for example `https://CRAN.R-project.org/bin/windows/contrib/3.2/@ReadMe`. Packages using Gtk+ (**Cairo** (`https://CRAN.R-project.org/package=Cairo`), **RGtk2** (`https://CRAN.R-project.org/package=RGtk2`), **cairoDevice** (`https://CRAN.R-project.org/package=cairoDevice`) and those that depend on them) need the `bin` directory of a bundled distribution of Gtk2 from `http://ftp.gnome.org/pub/gnome/binaries/win32/gtk+` or `http://ftp.gnome.org/pub/gnome/binaries/win64/gtk+` in the path: it should work to have both 32- and 64-bit Gtk+ `bin` directories in the path on a 64-bit version of R.

`R CMD INSTALL` works in Windows to install source packages. No additional tools are needed if the package does not contain compiled code, and `install.packages(type="source")` will work for such packages (and for those with compiled code if the tools (see Appendix D [The Windows toolset], page 67) are on the path, and the variables `BINPREF` and `BINPREF64` are set properly; see the discussion below). We have seen occasional permission problems after unpacking source packages on some systems: these have been circumvented by setting the environment variable `R_INSTALL_TAR` to '`tar.exe`'.

If you have only a source package that is known to work with current R and just want a binary Windows build of it, you could make use of the building service offered at `http://win-builder.r-project.org/`.

For almost all packages `R CMD INSTALL` will attempt to install both 32- and 64-bit builds of a package if run from a 32/64-bit install of R. It will report success if the installation of the architecture of the running `R` succeeded, whether or not the other architecture was successfully installed. The exceptions are packages with a non-empty `configure.win` script or which make use of `src/Makefile.win`. If `configure.win` does something appropriate to both architectures use[4] option `--force-biarch`: otherwise `R CMD INSTALL --merge-multiarch` can be applied to a source tarball to merge separate 32- and 64-bit installs. (This can only be applied to a tarball, and will only succeed if both installs succeed.)

If you have a package without compiled code and no Windows-specific help, you can zip up an installation on another OS and install from that zip file on Windows. However, such a package can be installed from the sources on Windows without any additional tools.

Packages with compiled code may need to have paths to the compilers set explicitly, and there is provision to make use of a system-wide library of installed external software. The compiler paths are set using the `make` variables `BINPREF` (and in some cases `BINPREF64`). The library location is set using `make` variable `LOCAL_SOFT`, to give an equivalent of `/usr/local` on a Unix-alike. All of these can be set in `src/gnuwin32/MkRules.local` when R is built from sources (see the comments in `src/gnuwin32/MkRules.dist`), or in file[5] `etc/i386/Makeconf` or `etc/x64/Makeconf` for an installed version of R. In the latter case only `BINPREF` is used, with the 64 bit path used in `etc/x64/Makeconf`. The version used by CRAN can be installed as described in Section 3.1 [Building from source], page 12.

6.3.2 OS X

On OS X `install.packages` works as it does on other Unix-alike systems, but there are additional types starting with `mac.binary` (available for the CRAN distribution but not when compiling from source: `mac.binary.mavericks` for a 'Mavericks' build with `"default"` a synonym for the appropriate variant) which can be passed to `install.packages` in order to download and install binary packages from a suitable repository. These OS X binary package files have the extension '`.tgz`'. The R.app GUI provides menus for installation of either binary or source packages, from CRAN or local files.

[4] for a small number of CRAN packages where this is known to be safe and is needed by the autobuilder this is the default. Look at the source of `tools:::.install_packages` for the list. It can also be specified in the package's `DESCRIPTION` file.

[5] or by adding it in a file such as `etc/i386/Makevars.site`, which does not exist by default.

On R builds using binary packages, the default is type `both`: this looks first at the list of binary packages available for your version of R and installs the latest versions (if any). If no binary version is available or the source version is newer, it will install the source versions of packages without compiled C/C++/Fortran code and offer to do so for those with, if `make` is available.

Note that most binary packages including compiled code are tied to a particular series (e.g. R 3.2.x or 3.3.x) of R.

Installing source packages which do not contain compiled code should work with no additional tools. For others you will need the 'Command Line Tools' for `Xcode` and compilers which match those used to build R: see Section C.3 [OS X], page 56.

Package **rJava** (`https://CRAN.R-project.org/package=rJava`) and those which depend on it need a Java runtime installed and several packages need X11 installed, including those using Tk. See Section C.3 [OS X], page 56, and Section C.3.3 [Java (OS X)], page 58.

Tcl/Tk extensions `BWidget` and `Tktable` are part of the Tcl/Tk contained in the R installer. These are required by a number of CRAN and Bioconductor packages.

A few of the binary packages need other software to be installed on your system. In particular packages using Gtk+ (**RGtk2** (`https://CRAN.R-project.org/package=RGtk2`), **cairoDevice** (`https://CRAN.R-project.org/package=cairoDevice`) and those that depend on them) need the GTK framework installed from `https://r.research.att.com/libs/`: the appropriate version at the time of writing was `https://r.research.att.com/libs/GTK_2.24.17-X11.pkg`

The default compilers specified in `/Library/Frameworks/R.framework/Resources/etc/Makeconf` depend on the version of OS X under which R was installed, and are appropriate for the latest version of the 'Command Line Tools' for that version of OS X and the recommended version of Fortran (see Section C.3 [OS X], page 56). The settings can be changed, either by editing that file or in a file such as `~/.R/Makevars` (see the next section). Entries which may need to be changed include '`CC`', '`CXX`', '`FC`', '`F77`', '`FLIBS`' and the corresponding flags, and perhaps '`CXXCPP`', '`DYLIB_LD`', '`MAIN_LD`', '`SHLIB_CXXLD`', '`SHLIB_FCLD`' and '`SHLIB_LD`'.

So for example you could select `clang` for both C and C++ with extensive checking by having in `~/.R/Makevars`

```
CC=clang
CXX=clang++
CFLAGS=-mtune=native -g -O2 -Wall -pedantic -Wconversion
CXXFLAGS=-mtune=native -g -O2 -Wall -pedantic -Wconversion
```

6.3.3 Customizing package compilation

The R system and package-specific compilation flags can be overridden or added to by setting the appropriate Make variables in the personal file *HOME*`/.R/Makevars-`*R_PLATFORM* (but *HOME*`/.R/Makevars.win` or *HOME*`/.R/Makevars.win64` on Windows), or if that does not exist, *HOME*`/.R/Makevars`, where '`R_PLATFORM`' is the platform for which R was built, as available in the `platform` component of the R variable `R.version`. An alternative personal file can be specified *via* the environment variable `R_MAKEVARS_USER`.

Package developers are encouraged to use this mechanism to enable a reasonable amount of diagnostic messaging ("warnings") when compiling, such as e.g. `-Wall -pedantic` for tools from GCC, the Gnu Compiler Collection.

Note that this mechanism can also be used when it necessary to change the optimization level for a particular package. For example

```
## for C code
CFLAGS=-g -O -mtune=native
## for C++ code
```

```
CXXFLAGS=-g -O -mtune=native
## for Fortran code
FFLAGS=-g -O -mtune=native
## for Fortran 9x code
FCFLAGS=-g -O -mtune=native
```

Another use is to override the settings in a binary installation of R. For example, to use a different Fortran compiler on OS X

```
F77 = /usr/local/gfortran/bin/gfortran
FC = /usr/local/gfortran/bin/gfortran
FLIBS = -L/usr/local/gfortran/lib/gcc/x86_64-apple-darwin14/5.2.0
  -L/usr/local/gfortran/lib -lgfortran -lquadmath -lm
```

(line split for legibility here).

There is also provision for a site-wide `Makevars.site` file under *`R_HOME`*`/etc` (in a sub-architecture-specific directory if appropriate). This is read immediately after `Makeconf`, and an alternative file can be specified by environment variable `R_MAKEVARS_SITE`.

Note that these mechanisms do not work with packages which fail to pass settings down to sub-makes, perhaps reading `etc/Makeconf` in makefiles in subdirectories. Fortunately such packages are unusual.

6.3.4 Multiple sub-architectures

When installing packages from their sources, there are some extra considerations on installations which use sub-architectures. These are commonly used on Windows but can in principle be used on other platforms.

When a source package is installed by a build of R which supports multiple sub-architectures, the normal installation process installs the packages for all sub-architectures. The exceptions are

Unix-alikes
: where there is an `configure` script, or a file `src/Makefile`.

Windows
: where there is a non-empty `configure.win` script, or a file `src/Makefile.win` (with some exceptions where the package is known to have an architecture-independent `configure.win`, or if `--force-biarch` or field '`Biarch`' in the `DESCRIPTION` file is used to assert so).

In those cases only the current architecture is installed. Further sub-architectures can be installed by

```
R CMD INSTALL --libs-only pkg
```

using the path to `R` or `R --arch` to select the additional sub-architecture. There is also `R CMD INSTALL --merge-multiarch` to build and merge the two architectures, starting with a source tarball.

6.3.5 Byte-compilation

The base and recommended packages are byte-compiled by default. Other packages can be byte-compiled on installation by using `R CMD INSTALL` with option `--byte-compile` or by `install.packages(type = "source", INSTALL_opts = "--byte-compile")`.

Not all contributed packages work correctly when byte-compiled. For most packages (especially those which make extensive use of compiled code) the speed-up is small. Unless a package is used frequently the time spent in byte-compilation can outweigh the time saved in execution: also byte-compilation can add substantially to the installed size of the package.

Byte-compilation can be controlled on a per-package basis by the 'ByteCompile' field in the DESCRIPTION file.

6.3.6 External software

Some R packages contain compiled code which links to external software libraries. Unless the external library is statically linked (which is done as much as possible for binary packages on Windows and OS X), the libraries have to be found when the package is loaded and not just when it is installed. How this should be done depends on the OS (and in some cases the version).

For Unix-alikes except OS X the primary mechanism is the ld.so cache controlled by ldconfig: external dynamic libraries recorded in that cache will be found. Standard library locations will be covered by the cache, and well-designed software will add its locations (as for example **openmpi** does on Fedora). The secondary mechanism is to consult the environment variable LD_LIBRARY_PATH. Now the R script controls that variable, and sets it to the concatenation of R_LD_LIBRARY_PATH, R_JAVA_LD_LIBRARY_PATH and the environment value of LD_LIBRARY_PATH. The first two have defaults which are normally set when R is installed (but can be overridden in the environment) so LD_LIBRARY_PATH is the best choice for a user to set.

On OS X the primary mechanism is to embed the absolute path to dependent dynamic libraries into an object when it is compiled. Few R packages arrange to do so, but it can be edited[6] *via* install_name_tool — that only deals with direct dependencies and those would also need to be compiled to include the absolute paths of their dependencies. If the choice of absolute path is to be deferred to load time, how they are resolved is described in man dyld: the role of LD_LIBRARY_PATH is replaced on OS X by DYLD_LIBRARY_PATH and latterly DYLD_FALLBACK_LIBRARY_PATH. Running R CMD otool -L on the package shared object will show where (if anywhere) its dependencies are resolved. DYLD_FALLBACK_LIBRARY_PATH is preferred (and it is that which is manipulated by the R script), but as from 10.11 ('El Capitan') the default behaviour had been changed for security reasons to discard these environment variables when invoking a shell script (and R is a shell script). That makes the only portable option to set R_LD_LIBRARY_PATH in the environment, something like

```
export R_LD_LIBRARY_PATH="`R RHOME`/lib:/opt/local/lib"
```

The precise rules for where Windows looks for DLLs are complex and depend on the version of Windows. But for present purposes the main solution is to put the directories containing the DLLs the package links to (and any those DLLs link to) on the PATH. 64-bit versions of Windows will ignore 32-bit DLLs from 64-bit R and *vice versa*.

The danger with any of the methods which involve setting environment variables is of inadvertently masking a system library. This is less for DYLD_FALLBACK_LIBRARY_PATH and for *appending* to PATH on Windows (as it should already contain the system library paths).

6.4 Updating packages

The command update.packages() is the simplest way to ensure that all the packages on your system are up to date. It downloads the list of available packages and their current versions, compares it with those installed and offers to fetch and install any that have later versions on the repositories.

An alternative interface to keeping packages up-to-date is provided by the command packageStatus(), which returns an object with information on all installed packages and packages available at multiple repositories. The print and summary methods give an overview of installed and available packages, the upgrade method offers to fetch and install the latest versions of outdated packages.

[6] They need to have been created using -headerpad_max_install_names, which is the default for an R package.

One sometimes-useful additional piece of information that `packageStatus()` returns is the status of a package, as `"ok"`, `"upgrade"` or `"unavailable"` (in the currently selected repositories). For example

```
> inst <- packageStatus()$inst
> inst[inst$Status != "ok", c("Package", "Version", "Status")]
                  Package Version      Status
Biobase           Biobase   2.8.0 unavailable
RCurl               RCurl   1.4-2     upgrade
Rgraphviz       Rgraphviz  1.26.0 unavailable
rgdal               rgdal  0.6-27     upgrade
```

6.5 Removing packages

Packages can be removed in a number of ways. From a command prompt they can be removed by

```
R CMD REMOVE -l /path/to/library pkg1 pkg2 ...
```

From a running R process they can be removed by

```
> remove.packages(c("pkg1", "pkg2"),
                  lib = file.path("path", "to", "library"))
```

Finally, one can just remove the package directory from the library.

6.6 Setting up a package repository

Utilities such as `install.packages` can be pointed at any CRAN-style repository, and R users may want to set up their own. The 'base' of a repository is a URL such as `http://www.stats.ox.ac.uk/pub/RWin`: this must be an URL scheme that `download.packages` supports (which also includes '`ftp://`' and '`file://`' and (from R 3.3.0 and perhaps earlier) '`https://`'). Under that base URL there should be directory trees for one or more of the following types of package distributions:

- `"source"`: located at `src/contrib` and containing `.tar.gz` files. Other forms of compression can be used, e.g. `.tar.bz2` or `.tar.xz` files. Complete repositories contain the sources corresponding to any binary packages, and in any case it is wise to have a `src/contrib` area with a possibly empty `PACKAGES` file.
- `"win.binary"`: located at `bin/windows/contrib/`*x.y* for R versions *x.y.z* and containing `.zip` files for Windows.
- `"mac.binary.mavericks"`: located at `bin/macosx/mavericks/contrib/`*3.y* for the CRAN build for 'Mavericks' (and later) for R versions *3.y.z*, containing `.tgz` files.

Each terminal directory must also contain a `PACKAGES` file. This can be a concatenation of the `DESCRIPTION` files of the packages separated by blank lines, but only a few of the fields are needed. The simplest way to set up such a file is to use function `write_PACKAGES` in the **tools** package, and its help explains which fields are needed. Optionally there can also be a `PACKAGES.gz` file, a `gzip`-compressed version of `PACKAGES`—as this will be downloaded in preference to `PACKAGES` it should be included for large repositories. (If you have a mis-configured server that does not report correctly non-existent files you may need `PACKAGES.gz`.)

To add your repository to the list offered by `setRepositories()`, see the help file for that function.

Incomplete repositories are better specified *via* a `contriburl` argument than *via* being set as a repository.

A repository can contain subdirectories, when the descriptions in the `PACKAGES` file of packages in subdirectories must include a line of the form

```
Path: path/to/subdirectory
```

—once again `write_PACKAGES` is the simplest way to set this up.

6.7 Checking installed source packages

It can be convenient to run `R CMD check` on an installed package, particularly on a platform which uses sub-architectures. The outline of how to do this is, with the source package in directory *pkg* (or a tarball filename):

```
R CMD INSTALL -l libdir pkg > pkg.log 2>&1
R CMD check -l libdir --install=check:pkg.log pkg
```

Where sub-architectures are in use the `R CMD check` line can be repeated with additional architectures by

```
R --arch arch CMD check -l libdir --extra-arch --install=check:pkg.log pkg
```

where `--extra-arch` selects only those checks which depend on the installed code and not those which analyse the sources. (If multiple sub-architectures fail only because they need different settings, e.g. environment variables, `--no-multiarch` may need to be added to the `INSTALL` lines.) On Unix-alikes the architecture to run is selected by `--arch`: this can also be used on Windows with *R_HOME*`/bin/R.exe`, but it is more usual to select the path to the `Rcmd.exe` of the desired architecture.

So on Windows to install, check and package for distribution a source package from a tarball which has been tested on another platform one might use

```
.../bin/i386/Rcmd INSTALL -l libdir tarball --build > pkg.log 2>&1
.../bin/i386/Rcmd check -l libdir --extra-arch --install=check:pkg.log pkg
.../bin/x64/Rcmd check -l libdir --extra-arch --install=check:pkg.log pkg
```

where one might want to run the second and third lines in a different shell with different settings for environment variables and the path (to find external software, notably for Gtk+).

`R CMD INSTALL` can do a `i386` install and then add the `x64` DLL from a single command by

```
R CMD INSTALL --merge-multiarch -l libdir tarball
```

and `--build` can be added to zip up the installation.

7 Internationalization and Localization

Internationalization refers to the process of enabling support for many human languages, and *localization* to adapting to a specific country and language.

Current builds of R support all the character sets that the underlying OS can handle. These are interpreted according to the current `locale`, a sufficiently complicated topic to merit a separate section. Note though that R has no built-in support for right-to-left languages and bidirectional output, relying on the OS services. For example, how character vectors in UTF-8 containing both English digits and Hebrew characters are printed is OS-dependent (and perhaps locale-dependent).

The other aspect of the internationalization is support for the translation of messages. This is enabled in almost all builds of R.

7.1 Locales

A *locale* is a description of the local environment of the user, including the preferred language, the encoding of characters, the currency used and its conventions, and so on. Aspects of the locale are accessed by the R functions `Sys.getlocale` and `Sys.localeconv`.

The system of naming locales is OS-specific. There is quite wide agreement on schemes, but not on the details of their implementation. A locale needs to specify

- A human language. These are generally specified by a lower-case two-character abbreviation following ISO 639 (see e.g. `https://en.wikipedia.org/wiki/ISO_639-1`).
- A 'territory', used mainly to specify the currency. These are generally specified by an upper-case two-character abbreviation following ISO 3166 (see e.g. `https://en.wikipedia.org/wiki/ISO_3166`).
- A charset encoding, which determines both how a byte stream should be divided into characters, and which characters the subsequences of bytes represent. Sometimes the combination of language and territory is used to specify the encoding, for example to distinguish between traditional and simplified Chinese.
- Optionally, a modifier, for example to indicate that Austria is to be considered pre- or post-Euro. The modifier is also used to indicate the script (`@latin`, `@cyrillic` for Serbian, `@iqtelif`) or language dialect (e.g. `@saaho`, a dialect of Afar, and `@bokmal` and `@nynorsk`, dialects of Norwegian regarded by some OSes as separate languages, `no` and `nn`).

R is principally concerned with the first (for translations) and third. Note that the charset may be deducible from the language, as some OSes offer only one charset per language.

7.1.1 Locales under Unix-alikes

Modern Linux uses the XPG[1] locale specifications which have the form '`en_GB`', '`en_GB.UTF-8`', '`aa_ER.UTF-8@saaho`', '`de_AT.iso885915@euro`', the components being in the order listed above. (See `man locale` and `locale -a` for more details.) Similar schemes are used by most Unix-alikes: some (including some distributions of Linux) use '`.utf8`' rather than '`.UTF-8`'.

Note that whereas UTF-8 locales are nowadays almost universally used, locales such as '`en_GB`' use 8-bit encodings for backwards compatibility.

7.1.2 Locales under Windows

Windows also uses locales, but specified in a rather less concise way. Most users will encounter locales only via drop-down menus, but more information and lists can be found at `https://`

[1] 'X/Open Portability Guide', which has had several versions.

msdn.microsoft.com/en-us/library/hzz3tw78(v=vs.80) (or if Microsoft moves it yet again, search for 'Windows language country strings').

It offers only one encoding per language.

Some care is needed with Windows' locale names. For example, chinese is Traditional Chinese and not Simplified Chinese as used in most of the Chinese-speaking world.

7.1.3 Locales under OS X

OS X supports locales in its own particular way, but the R GUI tries to make this easier for users. See https://developer.apple.com/documentation/MacOSX/Conceptual/BPInternational/ for how users can set their locales. As with Windows, end users will generally only see lists of languages/territories. Users of R in a terminal may need to set the locale to something like 'en_GB.UTF-8' if it defaults to 'C' (as it sometimes does when logging in remotely and for batch jobs: note whether Terminal sets the LANG environment variable is an (advanced) preference, but does so by default).

Internally OS X uses a form similar to Linux: the main difference from other Unix-alikes is that where a character set is not specified it is assumed to be UTF-8.

7.2 Localization of messages

The preferred language for messages is by default taken from the locale. This can be overridden first by the setting of the environment variable LANGUAGE and then[2] by the environment variables LC_ALL, LC_MESSAGES and LANG. (The last three are normally used to set the locale and so should not be needed, but the first is only used to select the language for messages.) The code tries hard to map locales to languages, but on some systems (notably Windows) the locale names needed for the environment variable LC_ALL do not all correspond to XPG language names and so LANGUAGE may need to be set. (One example is 'LC_ALL=es' on Windows which sets the locale to Estonian and the language to Spanish.)

It is usually possible to change the language once R is running *via* (not Windows) Sys.setlocale("LC_MESSAGES", "new_locale"), or by setting an environment variable such as LANGUAGE, *provided*[3] the language you are changing to can be output in the current character set. But this is OS-specific, and has been known to stop working on an OS upgrade.

Messages are divided into *domains*, and translations may be available for some or all messages in a domain. R makes use of the following domains.

- Domain R for the C-level error and warning messages from the R interpreter.
- Domain R-*pkg* for the R stop, warning and message messages in each package, including R-base for the **base** package.
- Domain *pkg* for the C-level messages in each package.
- Domain RGui for the menus etc of the R for Windows GUI front-end.

Dividing up the messages in this way allows R to be extensible: as packages are loaded, their message translation catalogues can be loaded too.

R can be built without support for translations, but it is enabled by default.

R-level and C-level domains are subtly different, for example in the way strings are canonicalized before being passed for translation.

Translations are looked for by domain according to the currently specified language, as specifically as possible, so for example an Austrian ('de_AT') translation catalogue will be used in preference to a generic German one ('de') for an Austrian user. However, if a specific translation

[2] On some systems setting LC_ALL or LC_MESSAGES to 'C' disables LANGUAGE.

[3] If you try changing from French to Russian except in a UTF-8 locale, you will most likely find messages change to English.

catalogue exists but does not contain a translation, the less specific catalogues are consulted. For example, R has catalogues for '`en_GB`' that translate the Americanisms (e.g., '`gray`') in the standard messages into English.[4] Two other examples: there are catalogues for '`es`', which is Spanish as written in Spain and these will by default also be used in Spanish-speaking Latin American countries, and also for '`pt_BR`', which are used for Brazilian locales but not for locales specifying Portugal.

Translations in the right language but the wrong charset are made use of by on-the-fly re-encoding. The `LANGUAGE` variable (only) can be a colon-separated list, for example '`se:de`', giving a set of languages in decreasing order of preference. One special value is '`en@quot`', which can be used in a UTF-8 locale to have American error messages with pairs of single quotes translated to Unicode directional quotes.

If no suitable translation catalogue is found or a particular message is not translated in any suitable catalogue, 'English'[5] is used.

See `https://developer.r-project.org/Translations30.html` for how to prepare and install translation catalogues.

[4] the language written in England: some people living in the USA appropriate this name for their language.

[5] with Americanisms.

8 Choosing between 32- and 64-bit builds

Almost all current CPUs have both 32- and 64-bit sets of instructions. Most OSes running on such CPUs offer the choice of building a 32-bit or a 64-bit version of R (and details are given below under specific OSes). For most a 32-bit version is the default, but for some (e.g., '`x86_64`' Linux and OS X >= 10.6) 64-bit is.

All current versions of R use 32-bit integers and ISO/IEC 60559[1] double-precision reals, and so compute to the same precision[2] and with the same limits on the sizes of numerical quantities. The principal difference is in the size of the pointers.

64-bit builds have both advantages and disadvantages:

- The total virtual memory space made available to a 32-bit process is limited by the pointer size to 4GB, and on most OSes to 3GB (or even 2GB). The limits for 64-bit processes are much larger (e.g. 8–128TB).

  R allocates memory for large objects as needed, and removes any unused ones at garbage collection. When the sizes of objects become an appreciable fraction of the address limit, fragmentation of the address space becomes an issue and there may be no hole available that is the size requested. This can cause more frequent garbage collection or the inability to allocate large objects. As a guide, this will become an issue for 32-bit builds with objects more than 10% of the size of the address space (around 300Mb) or when the total size of objects in use is around one third (around 1Gb).
- Only 64-bit builds support 'long vectors', those with 2^{31} or more elements (which needs at least 16GB of storage for each numeric vector).
- Most 32-bit OSes by default limit file sizes to 2GB (and this may also apply to 32-bit builds on 64-bit OSes). This can often be worked around: and `configure` selects suitable defines if this is possible. (We have also largely worked around that limit on 32-bit Windows.) 64-bit builds have much larger limits.
- Because the pointers are larger, R's basic structures are larger. This means that R objects take more space and (usually) more time to manipulate. So 64-bit builds of R will, all other things being equal, run slower than 32-bit builds. (On Sparc Solaris the difference was 15-20%.)
- However, 'other things' may not be equal. In the specific case of '`x86_64`' *vs* '`ix86`', the 64-bit CPU has features (such as SSE2 instructions) which are guaranteed to be present but are optional on the 32-bit CPU, and also has more general-purpose registers. This means that on chips like a desktop Intel i7 the vanilla 64-bit version of R has been around 10% faster on both Linux and OS X. (Laptop CPUs are usually relatively slower in 64-bit mode.)

So, for speed you may want to use a 32-bit build (especially on a laptop), but to handle large datasets (and perhaps large files) a 64-bit build. You can often build both and install them in the same place: See Section 2.6 [Sub-architectures], page 8. (This is done for the Windows binary distributions.)

Even on 64-bit builds of R there are limits on the size of R objects (see `help("Memory-limits")`, some of which stem from the use of 32-bit integers (especially in FORTRAN code). For example, the dimensions of an array are limited to $2^{31} - 1$.

[1] also known as IEEE 754

[2] at least when storing quantities: the on-FPU precision is allowed to vary

9 The standalone Rmath library

The routines supporting the distribution and special[1] functions in R and a few others are declared in C header file `Rmath.h`. These can be compiled into a standalone library for linking to other applications. (Note that they are not a separate library when R is built, and the standalone version differs in several ways.)

The makefiles and other sources needed are in directory `src/nmath/standalone`, so the following instructions assume that is the current working directory (in the build directory tree on a Unix-alike if that is separate from the sources).

`Rmath.h` contains '`R_VERSION_STRING`', which is a character string containing the current R version, for example `"3.3.0"`.

There is full access to R's handling of `NaN`, `Inf` and `-Inf` via special versions of the macros and functions

```
ISNAN, R_FINITE, R_log, R_pow and R_pow_di
```

and (extern) constants `R_PosInf`, `R_NegInf` and `NA_REAL`.

There is no support for R's notion of missing values, in particular not for `NA_INTEGER` nor the distinction between `NA` and `NaN` for doubles.

A little care is needed to use the random-number routines. You will need to supply the uniform random number generator

```
double unif_rand(void)
```

or use the one supplied (and with a shared library or DLL you may have to use the one supplied, which is the Marsaglia-multicarry with an entry point

```
set_seed(unsigned int, unsigned int)
```

to set its seeds).

The facilities to change the normal random number generator are available through the constant N01_kind. This takes values from the enumeration type

```
typedef enum {
    BUGGY_KINDERMAN_RAMAGE,
    AHRENS_DIETER,
    BOX_MULLER,
    USER_NORM,
    INVERSION,
    KINDERMAN_RAMAGE
} N01type;
```

(and '`USER_NORM`' is not available).

9.1 Unix-alikes

If R has not already been made in the directory tree, `configure` must be run as described in the main build instructions.

Then (in `src/nmath/standalone`)

```
make
```

will make standalone libraries `libRmath.a` and `libRmath.so` (`libRmath.dylib` on OS X): '`make static`' and '`make shared`' will create just one of them.

To use the routines in your own C or C++ programs, include

```
#define MATHLIB_STANDALONE
```

[1] e.g. Bessel, beta and gamma functions

```
#include <Rmath.h>
```

and link against '`-lRmath`' (and '`-lm`' if needed on your OS). The example file `test.c` does nothing useful, but is provided to test the process (via `make test`). Note that you will probably not be able to run it unless you add the directory containing `libRmath.so` to the `LD_LIBRARY_PATH` environment variable (`libRmath.dylib`, `DYLD_FALLBACK_LIBRARY_PATH` on OS X).

The targets

```
make install
make uninstall
```

will (un)install the header `Rmath.h` and shared and static libraries (if built). Both `prefix=` and `DESTDIR` are supported, together with more precise control as described for the main build.

'`make install`' installs a file for `pkg-config` to use by e.g.

```
$(CC) `pkg-config --cflags libRmath` -c test.c
$(CC) `pkg-config --libs libRmath` test.o -o test
```

On some systems '`make install-strip`' will install a stripped shared library.

9.2 Windows

You need to set up[2] almost all the tools to make R and then run (in a Unix-like shell)

```
(cd ../../gnuwin32; make MkRules)
(cd ../../include; make -f Makefile.win config.h Rconfig.h Rmath.h)
make -f Makefile.win
```

Alternatively, in a `cmd.exe` shell use

```
cd ../../include
make -f Makefile.win config.h Rconfig.h Rmath.h
cd ../nmath/standalone
make -f Makefile.win
```

This creates a static library `libRmath.a` and a DLL `Rmath.dll`. If you want an import library `libRmath.dll.a` (you don't need one), use

```
make -f Makefile.win shared implib
```

To use the routines in your own C or C++ programs using MinGW-w64, include

```
#define MATHLIB_STANDALONE
#include <Rmath.h>
```

and link against '`-lRmath`'. This will use the first found of `libRmath.dll.a`, `libRmath.a` and `Rmath.dll` in that order, so the result depends on which files are present. You should be able to force static or dynamic linking *via*

```
-Wl,-Bstatic -lRmath -Wl,Bdynamic
-Wl,-Bdynamic -lRmath
```

or by linking to explicit files (as in the '`test`' target in `Makefile.win`: this makes two executables, `test.exe` which is dynamically linked, and `test-static.exe`, which is statically linked).

It is possible to link to `Rmath.dll` using other compilers, either directly or via an import library: if you make a MinGW-w64 import library as above, you will create a file `Rmath.def` which can be used (possibly after editing) to create an import library for other systems such as Visual C++.

If you make use of dynamic linking you should use

```
#define MATHLIB_STANDALONE
#define RMATH_DLL
```

[2] including copying `MkRules.dist` to `MkRule.local` and selecting the architecture.

```
#include <Rmath.h>
```

to ensure that the constants like `NA_REAL` are linked correctly. (Auto-import will probably work with MinGW-w64, but it is better to be sure. This is likely to also work with VC++, Borland and similar compilers.)

Appendix A Essential and useful other programs under a Unix-alike

This appendix gives details of programs you will need to build R on Unix-like platforms, or which will be used by R if found by `configure`.

Remember that some package management systems (such as RPM and Debian/Ubuntu's) make a distinction between the user version of a package and the development version. The latter usually has the same name but with the extension '`-devel`' or '`-dev`': you need both versions installed.

A.1 Essential programs and libraries

You need a means of compiling C and FORTRAN 90 (see Section B.6 [Using FORTRAN], page 48). Your C compiler should be ISO/IEC 60059[1], POSIX 1003.1 and C99-compliant.[2] R tries to choose suitable flags for the C compilers it knows about, but you may have to set `CC` or `CFLAGS` suitably. For many versions of `gcc` with `glibc` this means including `-std=gnu99`[3]. (Note that options essential to run the compiler even for linking, such as those to set the architecture, should be specified as part of `CC` rather than in `CFLAGS`.)

Unless you do not want to view graphs on-screen (or use OS X) you need '`X11`' installed, including its headers and client libraries. For recent Fedora/RedHat distributions it means (at least) RPMs '`libX11`', '`libX11-devel`', '`libXt`' and '`libXt-devel`'. On Debian/Ubuntu we recommend the meta-package '`xorg-dev`'. If you really do not want these you will need to explicitly configure R without X11, using `--with-x=no`.

The command-line editing (and command completion) depends on the GNU `readline` library (including its headers): version 4.2 or later is needed for all the features to be enabled. Otherwise you will need to configure with `--with-readline=no` (or equivalent).

A suitably comprehensive `iconv` function is essential. The R usage requires `iconv` to be able to translate between `"latin1"` and `"UTF-8"`, to recognize `""` (as the current encoding) and `"ASCII"`, and to translate to and from the Unicode wide-character formats `"UCS-[24][BL]E"` — this is true by default for `glibc`[4] but not of most commercial Unixes. However, you can make use of GNU `libiconv` (as used on OS X: see `https://www.gnu.org/software/libiconv/`).

The OS needs to have enough support[5] for wide-character types: this is checked at configuration. A small number of POSIX functions[6] are essential, and others[7] will be used if available.

Installations of `zlib` (version 1.2.5 or later), `libbz2` (version 1.0.6 or later: called **bzip2-libs/bzip2-devel** or **libbz2-1.0/libbz2-dev** by some Linux distributions), `liblzma`[8] version 5.0.3 or later are required.

PCRE[9] (version 8.32 or later, although versions 8.10–8.31 will be accepted with a deprecation warning) is required (or just its library and headers if packaged separately). PCRE must be

[1] also known as IEEE 754

[2] Note that C11 compilers need not be C99-compliant: R requires support for `double complex` and variable-length arrays which are optional in C11 but is mandatory in C99.

[3] `-std=c99` excludes POSIX functionality, but `config.h` will turn on all GNU extensions to include the POSIX functionality. The default mode for GCC 5.1 and later is `-std=gnu11`.

[4] However, it is possible to break the default behaviour of `glibc` by re-specifying the `gconv` modules to be loaded.

[5] specifically, the C99 functionality of headers `wchar.h` and `wctype.h`, types `wctans_t` and `mbstate_t` and functions `mbrtowc`, `mbstowcs`, `wcrtomb`, `wcscoll`, `wcstombs`, `wctrans`, `wctype`, and `iswctype`.

[6] including the `opendir`, `readdir`, `closedir`, `popen`, `stat`, `glob`, `access`, `getcwd` and `chdir` system calls, and either `putenv` or `setenv`.

[7] such as `realpath`, `symlink`.

[8] most often distributed as part of `xz`: possible names in Linux distributions include `xz-devel`/`xz-libs` and `liblzma-dev`.

[9] sometimes known as PCRE1, and not PCRE2 which started at version 10.0.

built with UTF-8 support (not the default, and checked by `configure`) and support for Unicode properties is assumed by some R packages. JIT support is desirable for the best performance: support for this and Unicode properties can be checked at run-time by calling `pcre_config()`. If building PCRE for use with R a suitable `configure` command might be

```
/configure --enable-utf --enable-unicode-properties --enable-jit --disable-cpp
```

The `--enable-jit` flag is supported for most common CPUs.

Library `libcurl` (version 7.28.0 or later) is required. Information on `libcurl` is found from the `curl-config` script: if that is missing or needs to be overridden[10] there are macros to do so described in file `config.site`.

A `tar` program is needed to unpack the sources and packages (including the recommended packages). A version[11] that can automagically detect compressed archives is preferred for use with `untar()`: the configure script looks for `gtar` and `gnutar` before `tar` – use environment variable `TAR` to override this.

There need to be suitable versions of the tools `grep` and `sed`: the problems are usually with old AT&T and BSD variants. `configure` will try to find suitable versions (including looking in `/usr/xpg4/bin` which is used on some commercial Unixes).

You will not be able to build most of the manuals unless you have `texi2any` version 5.1 or later installed, and if not most of the HTML manuals will be linked to a version on CRAN. To make PDF versions of the manuals you will also need file `texinfo.tex` installed (which is part of the GNU **texinfo** distribution but is often made part of the TEX package in re-distributions) as well as `texi2dvi`.[12] Further, the versions of `texi2dvi` and `texinfo.tex` need to be compatible: we have seen problems with older TEX distributions.

If you want to build from the R Subversion repository then `texi2any` is highly recommended as it is used to create files which are in the tarball but not stored in the Subversion repository.

The PDF documentation (including `doc/NEWS.pdf`) and building vignettes needs `pdftex` and `pdflatex`. We require LATEX version `2005/12/01` or later (for UTF-8 support). Building PDF package manuals (including the R reference manual) and vignettes is sensitive to the version of the LATEX package **hyperref** and we recommend that the TEX distribution used is kept up-to-date. A number of standard LATEX packages are required (including **url** and some of the font packages such as **times**, **helvetic**, **ec** and **cm-super**) and others such as **hyperref** and **inconsolata** are desirable (and without them you may need to change R's defaults: see Section 2.3 [Making the manuals], page 4). Note that package **hyperref** (currently) requires packages **kvoptions**, **ltxcmds** and **refcount**. For distributions based on TeX Live the simplest approach may be to install collections **collection-latex**, **collection-fontsrecommended**, **collection-latexrecommended**, **collection-fontsextra** and **collection-latexextra** (assuming they are not installed by default): Fedora uses names like **texlive-collection-fontsextra** and Debian/Ubuntu like **texlive-fonts-extra**.

The essential programs should be in your `PATH` at the time `configure` is run: this will capture the full paths.

---

[10] for example to specify static linking with a build which has both shared and static libraries.

[11] Such as GNU `tar` 1.15 or later, `bsdtar` (from `https://github.com/libarchive/libarchive/`, as used by FreeBSD and OS X 10.6 and later) or `tar` from the Heirloom Toolchest (`http://heirloom.sourceforge.net/tools.html`).

[12] `texi2dvi` is normally a shell script. Some versions (including that from **texinfo** 5.2 and 6.0) need to be run under `bash` rather than a Bourne shell, especially on Solaris. Some of the issues which have been observed with broken versions of `texi2dvi` can be circumvented by setting the environment variable `R_TEXI2DVICMD` to the value `emulation`.

A.2 Useful libraries and programs

The ability to use translated messages makes use of `gettext` and most likely needs GNU `gettext`: you do need this to work with new translations, but otherwise the version contained in the R sources will be used if no suitable external `gettext` is found.

The 'modern' version of the `X11()`, `jpeg()`, `png()` and `tiff()` graphics devices uses the `cairo` and (optionally) `Pango` libraries. Cairo version 1.2.0 or later is required. Pango needs to be at least version 1.10, and 1.12 is the earliest version we have tested. (For Fedora users we believe the `pango-devel` RPM and its dependencies suffice.) R checks for `pkg-config`, and uses that to check first that the '`pangocairo`' package is installed (and if not, '`cairo`') and if additional flags are needed for the '`cairo-xlib`' package, then if suitable code can be compiled. These tests will fail if `pkg-config` is not installed[13], and are likely to fail if `cairo` was built statically (unusual). Most systems with `Gtk+` 2.8 or later installed will have suitable libraries

For the best font experience with these devices you need suitable fonts installed: Linux users will want the `urw-fonts` package. On platforms which have it available, the `msttcorefonts` package[14] provides TrueType versions of Monotype fonts such as Arial and Times New Roman. Another useful set of fonts is the 'liberation' TrueType fonts available at `https://fedorahosted.org/liberation-fonts/`,[15] which cover the Latin, Greek and Cyrillic alphabets plus a fair range of signs. These share metrics with Arial, Times New Roman and Courier New, and contain fonts rather similar to the first two (`https://en.wikipedia.org/wiki/Liberation_fonts`). Then there is the 'Free UCS Outline Fonts' project (`https://www.gnu.org/software/freefont/`) which are OpenType/TrueType fonts based on the URW fonts but with extended Unicode coverage. See the R help on `X11` on selecting such fonts.

The bitmapped graphics devices `jpeg()`, `png()` and `tiff()` need the appropriate headers and libraries installed: `jpeg` (version 6b or later, or `libjpeg-turbo`) or `libpng` (version 1.2.7 or later) and `zlib` or `libtiff` (any recent version – 3.9.[4567] and 4.0.[23] have been tested) respectively. They also need support for either `X11` or `cairo` (see above). Should support for these devices **not** be required or broken system libraries need to be avoided there are `configure` options `--without-libpng`, `--without-jpeglib` and `--without-libtiff`. For most system installations the TIFF libraries will require JPEG libraries to be present and perhaps linked explicitly, so `--without-jpeglib` may also disable the `tiff()` device. The `tiff()` devices only require a basic build of `libtiff` (not even JPEG support is needed). Recent versions allow several other libraries to be linked into `libtiff` such as `lzma`, `jbig` and `jpeg12`, and these may need also to be present.

Option `--with-system-tre` is also available: it needs a recent version of TRE. (The current sources are in the `git` repository at `https://github.com/laurikari/tre/`, but at the time of writing the resulting build will not pass its checks.).

Library `liblzma` from `xz-utils` version 5.0.3 or later (including 5.2.x) will be used if installed: the version in the R sources can be selected instead by configuring with `--without-system-xz`. Systems differ in what they call the package including this: e.g. on Fedora the library is in '`xz-libs`' and the headers in '`xz-devel`'.

An implementation of XDR is required, and the R sources contain one which is likely to suffice (although a system version may have higher performance). XDR is part of RPC and historically has been part of `libc` on a Unix-alike. However some builds of `glibc` hide it with the intention that the TI-RPC library be used instead, in which case `libtirpc` (and its

[13] If necessary the path to `pkg-config` can be specified by setting `PKGCONF` in `config.site`, on the `configure` command line or in the environment.

[14] also known as `ttf-mscorefonts-installer` in the Debian/Ubuntu world: see also `https://en.wikipedia.org/wiki/Core_fonts_for_the_Web`.

[15] `ttf-liberation` in Debian/Ubuntu.

development version) needs to be installed, and its headers need to be on the C include path or in `/usr/include/tirpc`.

Use of the X11 clipboard selection requires the `Xmu` headers and libraries. These are normally part of an X11 installation (e.g. the Debian meta-package '`xorg-dev`'), but some distributions have split this into smaller parts, so for example recent versions of Fedora require the '`libXmu`' and '`libXmu-devel`' RPMs.

Some systems (notably OS X and at least some FreeBSD systems) have inadequate support for collation in multibyte locales. It is possible to replace the OS's collation support by that from ICU (International Components for Unicode, `http://site.icu-project.org/`), and this provides much more precise control over collation on all systems. ICU is available as sources and as binary distributions for (at least) most Linux distributions, Solaris, FreeBSD and AIX, usually as `libicu` or `icu4c`. It will be used by default where available: should a very old or broken version of ICU be found this can be suppressed by `--without-ICU`.

The `bitmap` and `dev2bitmap` devices and function `embedFonts()` use ghostscript (`http://www.ghostscript.com/`). This should either be in your path when the command is run, or its full path specified by the environment variable `R_GSCMD` at that time.

A.2.1 Tcl/Tk

The **tcltk** package needs Tcl/Tk >= 8.4 installed: the sources are available at `https://www.tcl.tk/`. To specify the locations of the Tcl/Tk files you may need the configuration options

`--with-tcltk`
: use Tcl/Tk, or specify its library directory

`--with-tcl-config=`*`TCL_CONFIG`*
: specify location of `tclConfig.sh`

`--with-tk-config=`*`TK_CONFIG`*
: specify location of `tkConfig.sh`

or use the configure variables `TCLTK_LIBS` and `TCLTK_CPPFLAGS` to specify the flags needed for linking against the Tcl and Tk libraries and for finding the `tcl.h` and `tk.h` headers, respectively. If you have both 32- and 64-bit versions of Tcl/Tk installed, specifying the paths to the correct config files may be necessary to avoid confusion between them.

Versions of Tcl/Tk up to 8.5.19 and 8.6.4 have been tested (including most versions of 8.4.x, but not recently).

Note that the `tk.h` header includes[16] X11 headers, so you will need X11 and its development files installed.

A.2.2 Java support

The build process looks for Java support on the host system, and if it finds it sets some settings which are useful for Java-using packages (such as **rJava** (`https://CRAN.R-project.org/package=rJava`) and **JavaGD** (`https://CRAN.R-project.org/package=JavaGD`)). This check can be suppressed by configure option `--disable-java`. Configure variable `JAVA_HOME` can be set to point to a specific JRE/JDK, on the `configure` command line or in the environment.

Principal amongst these settings are some library paths to the Java libraries and JVM, which are stored in environment variable `R_JAVA_LD_LIBRARY_PATH` in file *`R_HOME`*`/etc/ldpaths` (or a sub-architecture-specific version). A typical setting for '`x86_64`' Linux is

```
JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.71-1.b15.fc22.x86_64/jre
R_JAVA_LD_LIBRARY_PATH=${JAVA_HOME}/lib/amd64/server
```

[16] This is true even for the 'Aqua' version of Tk on OS X, but distributions of that include a copy of the X11 files needed.

Unfortunately this depends on the exact version of the JRE/JDK installed, and so may need updating if the Java installation is updated. This can be done by running `R CMD javareconf` which updates settings in both *R_HOME*`/etc/Makeconf` and *R_HOME*`/etc/ldpaths`. See `R CMD javareconf --help` for details: note that this needs to be done by the account owning the R installation.

Another way of overriding those settings is to set the environment variable `R_JAVA_LD_LIBRARY_PATH` (before R is started, hence not in `~/.Renviron`), which suffices to run already-installed Java-using packages. For example

```
R_JAVA_LD_LIBRARY_PATH=/usr/lib/jvm/java-1.8.0/jre/lib/amd64/server
```

It may be possible to avoid this by specifying an invariant link as the path when configuring. For example, on that system any of

```
JAVA_HOME=/usr/lib/jvm/java
JAVA_HOME=/usr/lib/jvm/java-1.8.0
JAVA_HOME=/usr/lib/jvm/java-1.8.0/jre
```

worked.

A.2.3 Other compiled languages

Some add-on packages need a C++ compiler. This is specified by the configure variables `CXX`, `CXXFLAGS` and similar. `configure` will normally find a suitable compiler. However, in most cases this will be a C++98 compiler, and as from R 3.1.0 it is possible to specify an alternative compiler for use with C++11 by the configure variables `CXX1X`, `CXX1XSTD`, `CXX1XFLAGS` and similar. Again, `configure` will normally find a suitable value for `CXX1XSTD` if the compiler given by `CXX` is capable of compiling C++11 code, but it is possible that a completely different compiler will be needed.

Other packages need full Fortran 90 (or later) support. For source files with extension `.f90` or `.f95`, the compiler defined by the macro `FC` is used by `R CMD INSTALL`. This is found when R is configured and is often the same as `F77`: note that it is detected by the name of the command without a test that it can actually compile Fortran 90 code. Set the configure variable `FC` to override this if necessary: variables `FCFLAGS`, `FCPICFLAGS`, `FCLIBS`, `SHLIB_FCLD` and `SHLIB_FCLDFLAGS` might also need to be set.

See file `config.site` in the R source for more details about these variables.

A.3 Linear algebra

A.3.1 BLAS

The linear algebra routines in R can make use of enhanced BLAS (Basic Linear Algebra Subprograms, `http://www.netlib.org/blas/faq.html`) routines. However, these have to be explicitly requested at configure time: R provides an internal BLAS which is well-tested and will be adequate for most uses of R.

You can specify a particular BLAS library *via* a value for the configuration option `--with-blas` and not to use an external BLAS library by `--without-blas` (the default). If `--with-blas` is given with no `=`, its value is taken from the environment variable `BLAS_LIBS`, set for example in `config.site`. If neither the option nor the environment variable supply a value, a search is made for a suitable BLAS. If the value is not obviously a linker command (starting with a dash or giving the path to a library), it is prefixed by '`-l`', so

```
--with-blas="foo"
```

is an instruction to link against '`-lfoo`' to find an external BLAS (which needs to be found both at link time and run time).

The configure code checks that the external BLAS is complete (it must include all double precision and double complex routines, as well as `LSAME`), and appears to be usable. However,

an external BLAS has to be usable from a shared object (so must contain position-independent code), and that is not checked.

Some enhanced BLASes are compiler-system-specific (`sunperf` on Solaris[17], `libessl` on IBM, `Accelerate` on OS X). The correct incantation for these is often found *via* `--with-blas` with no value on the appropriate platforms.

Some of the external BLASes are multi-threaded. One issue is that R profiling (which uses the `SIGPROF` signal) may cause problems, and you may want to disable profiling if you use a multi-threaded BLAS. Note that using a multi-threaded BLAS can result in taking more CPU time and even more elapsed time (occasionally dramatically so) than using a similar single-threaded BLAS. On a machine running other tasks, there can be contention for CPU caches that reduces the effectiveness of the optimization of cache use by a BLAS implementation.

Note that under Unix (but not under Windows) if R is compiled against a non-default BLAS and `--enable-BLAS-shlib` is **not** used, then all BLAS-using packages must also be. So if R is re-built to use an enhanced BLAS then packages such as **quantreg** (`https://CRAN.R-project.org/package=quantreg`) will need to be re-installed.

R relies on ISO/IEC 60559 compliance of an external BLAS. This can be broken if for example the code assumes that terms with a zero factor are always zero and do not need to be computed—whereas `x*0` can be `NaN`. This is checked in the test suite.

External BLAS implementations often make less use of extended-precision floating-point registers and will almost certainly re-order computations. This can result in less accuracy than using the internal BLAS, and may result in different solutions, e.g. different signs in SVD and eigendecompositions.

The URIs for several of these BLAS are subject to frequent gratuitous changes, so you will need to search for their current locations.

A.3.1.1 ATLAS

ATLAS (`http://math-atlas.sourceforge.net/`) is a "tuned" BLAS that runs on a wide range of Unix-alike platforms. Unfortunately it is built by default as a static library that on some platforms cannot be used with shared objects such as are used in R packages. Be careful when using pre-built versions of ATLAS (they seem to work on '`ix86`' platforms, but not always on '`x86_64`' ones).

The usual way to specify ATLAS will be via

```
--with-blas="-lf77blas -latlas"
```

if the libraries are in the library path, otherwise by

```
--with-blas="-L/path/to/ATLAS/libs -lf77blas -latlas"
```

For example, '`x86_64`' Fedora needs

```
--with-blas="-L/usr/lib64/atlas -lf77blas -latlas"
```

For systems with multiple CPU cores it is possible to use a multi-threaded version of ATLAS, by specifying

```
--with-blas="-lptf77blas -lpthread -latlas"
```

Consult its installation guide for how to build ATLAS with position-independent code, and as a shared library.

[17] Using the Oracle Solaris Studio `cc` and `f95` compilers

A.3.1.2 ACML

For 'x86_64' processors[18] under Linux there is the AMD Core Math Library (ACML). For the gcc version we could use

```
--with-blas="-lacml"
```

if the appropriate library directory (such as /opt/acml5.1.0/gfortran64/lib) is in the LD_LIBRARY_PATH. For other compilers, see the ACML documentation. There is a multithreaded Linux version of ACML available for recent versions of gfortran. To make use of this you will need something like

```
--with-blas="-L/opt/acml5.1.0/gfortran64_mp/lib -lacml_mp"
```

(and you may need to arrange for the directory to be in ld.so cache).

See see Section A.3.1.5 [Shared BLAS], page 44, for an alternative (and in many ways preferable) way to use ACML.

The version last tested (5.1.0) failed the reg-BLAS.R test in its handling of NAs.

A.3.1.3 Goto and OpenBLAS

Dr Kazushige Goto wrote a tuned BLAS for several processors and OSes, which was frozen in mid-2010. The final version is known as GotoBLAS2, and was re-released under a much less restrictive licence. Once it is built and installed, it can be used by configuring R with

```
--with-blas="-lgoto2"
```

See see Section A.3.1.5 [Shared BLAS], page 44, for an alternative (and in many ways preferable) way to use it.

OpenBLAS (http://www.openblas.net/) is a descendant project with support for some later CPUs (e.g. Intel Sandy Bridge). Once installed it can be used by something like

```
--with-blas="-lopenblas"
```

or as a shared BLAS.

A.3.1.4 Intel MKL

For Intel processors (and perhaps others) and some distributions of Linux, there is Intel's Math Kernel Library. You are strongly encouraged to read the MKL User's Guide, which is installed with the library, before attempting to link to MKL. This includes a 'link line advisor' which will suggest appropriate incantations: its use is recommended. Or see https://software.intel.com/en-us/articles/intel-mkl-link-line-advisor.

There are also versions of MKL for OS X and Windows, but at the time these were tried they did not work with the standard compilers used for R on those platforms.

The MKL interface has changed several times and may change again: the following examples have been used with versions 10.3 to 11.3, for GCC compilers on 'x86_64'.

To a sequential version of MKL we used

```
MKL_LIB_PATH=/path/to/intel_mkl/lib/intel64
export LD_LIBRARY_PATH=$MKL_LIB_PATH
MKL="-L${MKL_LIB_PATH} -lmkl_gf_lp64 -lmkl_core -lmkl_sequential"
./configure --with-blas="$MKL" --with-lapack
```

The option --with-lapack is used since MKL contains a tuned copy of LAPACK as well as BLAS (see Section A.3.2 [LAPACK], page 44), although this can be omitted.

Threaded MKL may be used by replacing the line defining the variable MKL by

```
MKL="-L${MKL_LIB_PATH} -lmkl_gf_lp64 -lmkl_core \
```

[18] and 'i686' for earlier versions.

```
-lmkl_gnu_thread -dl -lpthread"
```

The default number of threads will be chosen by the OpenMP software, but can be controlled by setting `OMP_NUM_THREADS` or `MKL_NUM_THREADS`, and in recent versions seems to default to a sensible value for sole use of the machine.

It has been reported that

```
--with-blas='-mkl=parallel' --with-lapack
```

worked with the Intel 2015.3 compilers on Centos 6.

A.3.1.5 Shared BLAS

The BLAS library will be used for many of the add-on packages as well as for R itself. This means that it is better to use a shared/dynamic BLAS library, as most of a static library will be compiled into the R executable and each BLAS-using package.

R offers the option of compiling the BLAS into a dynamic library `libRblas` stored in *`R_HOME`*`/lib` and linking both R itself and all the add-on packages against that library.

This is the default on all platforms except AIX unless an external BLAS is specified and found: for the latter it can be used by specifying the option `--enable-BLAS-shlib`, and it can always be disabled via `--disable-BLAS-shlib`.

This has both advantages and disadvantages.

- It saves space by having only a single copy of the BLAS routines, which is helpful if there is an external static BLAS such as used to be standard for ATLAS.
- There may be performance disadvantages in using a shared BLAS. Probably the most likely is when R's internal BLAS is used and R is *not* built as a shared library, when it is possible to build the BLAS into `R.bin` (and `libR.a`) without using position-independent code. However, experiments showed that in many cases using a shared BLAS was as fast, provided high levels of compiler optimization are used.
- It is easy to change the BLAS without needing to re-install R and all the add-on packages, since all references to the BLAS go through `libRblas`, and that can be replaced. Note though that any dynamic libraries the replacement links to will need to be found by the linker: this may need the library path to be changed in *`R_HOME`*`/etc/ldpaths`.

Another option to change the BLAS in use is to symlink a dynamic BLAS library (such as ACML or Goto's) to *`R_HOME`*`/lib/libRblas.so`. For example, just

```
mv R_HOME/lib/libRblas.so R_HOME/lib/libRblas.so.keep
ln -s /opt/acml5.1.0/gfortran64_mp/lib/libacml_mp.so R_HOME/lib/libRblas.so
```

will change the BLAS in use to multithreaded ACML. A similar link works for some versions of Goto BLAS, OpenBLAS and MKL (provided the appropriate `lib` directory is in the run-time library path or `ld.so` cache).

A.3.2 LAPACK

Provision is made for using an external LAPACK library, principally to cope with BLAS libraries which contain a copy of LAPACK (such as `sunperf` on Solaris, `Accelerate` on OS X and ACML and MKL on '`ix86`'/'`x86_64`' Linux). At least LAPACK version 3.2 is required. This can only be done if `--with-blas` has been used.

However, the likely performance gains are thought to be small (and may be negative), and the default is not to search for a suitable LAPACK library, and this is definitely **not** recommended. You can specify a specific LAPACK library or a search for a generic library by the configuration option `--with-lapack`. The default for `--with-lapack` is to check the BLAS library and then look for an external library '`-llapack`'. Sites searching for the fastest possible linear algebra

may want to build a LAPACK library using the ATLAS-optimized subset of LAPACK. To do so specify something like

```
--with-lapack="-L/path/to/ATLAS/libs -llapack -lcblas"
```

since the ATLAS subset of LAPACK depends on `libcblas`. A value for `--with-lapack` can be set *via* the environment variable `LAPACK_LIBS`, but this will only be used if `--with-lapack` is specified (as the default value is `no`) and the BLAS library does not contain LAPACK.

Since ACML contains a full LAPACK, if selected as the BLAS it can be used as the LAPACK *via* `--with-lapack`.

If you do use `--with-lapack`, be aware of potential problems with bugs in the LAPACK sources (or in the posted corrections to those sources). In particular, bugs in `DGEEV` and `DGESDD` have resulted in error messages such as

```
DGEBRD gave error code -10
```

. Other potential problems are incomplete versions of the libraries, seen several times in Linux distributions over the years.

Please **do** bear in mind that using `--with-lapack` is 'definitely **not** recommended': it is provided **only** because it is necessary on some platforms and because some users want to experiment with claimed performance improvements. Reporting problems where it is used unnecessarily will simply irritate the R helpers.

Note too the comments about ISO/IEC 60559 compliance in the section of external BLAS: these apply equally to an external LAPACK, and for example the Intel MKL documentation says

> LAPACK routines assume that input matrices do not contain IEEE 754 special values such as INF or NaN values. Using these special values may cause LAPACK to return unexpected results or become unstable.

We rely on limited support in LAPACK for matrices with 2^{31} or more elements: it is quite possible that an external LAPACK will not have that support.

If you have a pure FORTRAN 77 compiler which cannot compile LAPACK it may be possible to use CLAPACK from `http://www.netlib.org/clapack/` by something like

```
-with-lapack="-lclapack -lf2c"
```

provided these were built with position-independent code and the calling conventions for double complex function return values match those in the BLAS used, so it may be simpler to use CLAPACK built to use CBLAS and

```
-with-lapack="-lclapack -lcblas -lf2c"
```

A.3.3 Caveats

As with all libraries, you need to ensure that they and R were compiled with compatible compilers and flags. For example, this has meant that on Sun Sparc using the native compilers the flag `-dalign` is needed if `sunperf` is to be used.

On some systems it has been necessary that an external BLAS/LAPACK was built with the same FORTRAN compiler used to build R.

Appendix B Configuration on a Unix-alike

B.1 Configuration options

`configure` has many options: running

```
./configure --help
```

will give a list. Probably the most important ones not covered elsewhere are (defaults in brackets)

`--with-x` use the X Window System [yes]

`--x-includes=`*DIR*
: X include files are in *DIR*

`--x-libraries=`*DIR*
: X library files are in *DIR*

`--with-readline`
: use readline library (if available) [yes]

`--enable-R-profiling`
: attempt to compile support for `Rprof()` [yes]

`--enable-memory-profiling`
: attempt to compile support for `Rprofmem()` and `tracemem()` [no]

`--enable-R-shlib`
: build R as a shared/dynamic library [no]

`--enable-BLAS-shlib`
: build the BLAS as a shared/dynamic library [yes, except on AIX]

You can use `--without-foo` or `--disable-foo` for the negatives.

You will want to use `--disable-R-profiling` if you are building a profiled executable of R (e.g. with '`-pg`)'.

Flag `--enable-R-shlib` causes the make process to build R as a dynamic (shared) library, typically called `libR.so`, and link the main R executable `R.bin` against that library. This can only be done if all the code (including system libraries) can be compiled into a dynamic library, and there may be a performance[1] penalty. So you probably only want this if you will be using an application which embeds R. Note that C code in packages installed on an R system linked with `--enable-R-shlib` is linked against the dynamic library and so such packages cannot be used from an R system built in the default way. Also, because packages are linked against R they are on some OSes also linked against the dynamic libraries R itself is linked against, and this can lead to symbol conflicts.

For maximally effective use of `valgrind`, R should be compiled with valgrind instrumentation. The `configure` option is `--with-valgrind-instrumentation=`*level*, where *level* is 0, 1 or 2. (Level 0 is the default and does not add anything.) The system headers for `valgrind` can be requested by option `--with-system-valgrind-headers`: they will be used if present (on Linux they may be in a separate package such as **valgrind-devel**). Note though that there is no guarantee that the code in R will be compatible with very old[2] or future `valgrind` headers.

If you need to re-configure R with different options you may need to run `make clean` or even `make distclean` before doing so.

The `configure` script has other generic options added by `autoconf` and which are not supported for R: in particular building for one architecture on a different host is not possible.

[1] We have measured 15–20% on '`i686`' Linux and around 10% on '`x86_64`' Linux.

[2] We believe that versions 3.4.0 to 3.10.1 are compatible.

B.2 Internationalization support

Translation of messages is supported via GNU `gettext` unless disabled by the configure option `--disable-nls`. The `configure` report will show `NLS` as one of the 'Additional capabilities' if support has been compiled in, and running in an English locale (but not the `C` locale) will include

```
Natural language support but running in an English locale
```

in the greeting on starting R.

B.3 Configuration variables

If you need or want to set certain configure variables to something other than their default, you can do that by either editing the file `config.site` (which documents many of the variables you might want to set: others can be seen in file `etc/Renviron.in`) or on the command line as

```
./configure VAR=value
```

If you are building in a directory different from the sources, there can be copies of `config.site` in the source and the build directories, and both will be read (in that order). In addition, if there is a file `~/.R/config`, it is read between the `config.site` files in the source and the build directories.

There is also a general `autoconf` mechanism for `config.site` files, which are read before any of those mentioned in the previous paragraph. This looks first at a file specified by the environment variable `CONFIG_SITE`, and if not is set at files such as `/usr/local/share/config.site` and `/usr/local/etc/config.site` in the area (exemplified by `/usr/local`) where R would be installed.

These variables are *precious*, implying that they do not have to be exported to the environment, are kept in the cache even if not specified on the command line, checked for consistency between two configure runs (provided that caching is used), and are kept during automatic reconfiguration as if having been passed as command line arguments, even if no cache is used.

See the variable output section of `configure --help` for a list of all these variables.

If you find you need to alter configure variables, it is worth noting that some settings may be cached in the file `config.cache`, and it is a good idea to remove that file (if it exists) before re-configuring. Note that caching is turned *off* by default: use the command line option `--config-cache` (or `-C`) to enable caching.

B.3.1 Setting paper size

One common variable to change is `R_PAPERSIZE`, which defaults to '`a4`', not '`letter`'. (Valid values are '`a4`', '`letter`', '`legal`' and '`executive`'.)

This is used both when configuring R to set the default, and when running R to override the default. It is also used to set the paper size when making PDF manuals.

The configure default will most often be '`a4`' if `R_PAPERSIZE` is unset. (If the (Debian Linux) program `paperconf` is found or the environment variable `PAPERSIZE` is set, these are used to produce the default.)

B.3.2 Setting the browsers

Another precious variable is `R_BROWSER`, the default HTML browser, which should take a value of an executable in the user's path or specify a full path.

Its counterpart for PDF files is `R_PDFVIEWER`.

B.3.3 Compilation flags

If you have libraries and header files, e.g., for GNU readline, in non-system directories, use the variables `LDFLAGS` (for libraries, using '`-L`' flags to be passed to the linker) and `CPPFLAGS` (for header files, using '`-I`' flags to be passed to the C/C++ preprocessors), respectively, to specify these locations. These default to '`-L/usr/local/lib`' (`LDFLAGS`, '`-L/usr/local/lib64`' on most 64-bit Linux OSes) and '`-I/usr/local/include`' (`CPPFLAGS`) to catch the most common cases. If libraries are still not found, then maybe your compiler/linker does not support re-ordering of `-L` and `-l` flags (this has been reported to be a problem on HP-UX with the native `cc`). In this case, use a different compiler (or a front end shell script which does the re-ordering).

These flags can also be used to build a faster-running version of R. On most platforms using `gcc`, having '`-O3`' in `CFLAGS` and `FFLAGS` produces worthwhile performance gains with `gcc` and `gfortran`, but may result in a less reliable build (both segfaults and incorrect numeric computations have been seen). On systems using the GNU linker (especially those using R as a shared library), it is likely that including '`-Wl,-O1`' in `LDFLAGS` is worthwhile, and ''`-Bdirect,--hash-style=both,-Wl,-O1`'' is recommended at `https://lwn.net/Articles/192624/`. Tuning compilation to a specific CPU family (e.g. '`-mtune=native`' for `gcc`) can give worthwhile performance gains, especially on older architectures such as '`ix86`'.

B.3.4 Making manuals

The default settings for making the manuals are controlled by `R_RD4PDF` and `R_PAPERSIZE`.

B.4 Setting the shell

By default the shell scripts such as `R` will be '`#!/bin/sh`' scripts (or using the `SHELL` chosen by `configure`). This is almost always satisfactory, but on a few systems `/bin/sh` is not a Bourne shell or clone, and the shell to be used can be changed by setting the configure variable `R_SHELL` to a suitable value (a full path to a shell, e.g. `/usr/local/bin/bash`).

B.5 Using make

To compile R, you will most likely find it easiest to use GNU `make`, although the Sun `make` works on Solaris. The native `make` has been reported to fail on SGI Irix 6.5 and Alpha/OSF1 (aka Tru64).

To build in a separate directory you need a `make` that supports the `VPATH` variable, for example GNU `make` and Sun `make`.

`dmake` has also been used. e.g, on Solaris 10.

If you want to use a `make` by another name, for example if your GNU `make` is called '`gmake`', you need to set the variable `MAKE` at configure time, for example

```
./configure MAKE=gmake
```

B.6 Using FORTRAN

To compile R, you need a FORTRAN compiler. The default is to search for `f95`, `fort`, `xlf95`, `ifort`, `ifc`, `efc`, `pgf95` `lf95`, `gfortran`, `ftn`, `g95`, `f90`, `xlf90`, `pghpf`, `pgf90`, `epcf90`, `g77`, `f77`, `xlf`, `frt`, `pgf77`, `cf77`, `fort77`, `fl32`, `af77` (in that order)[3], and use whichever is found first; if none is found, R cannot be compiled. However, if `CC` is `gcc`, the matching FORTRAN compiler (`g77` for `gcc` 3 and `gfortran` for `gcc` 4) is used if available.

The search mechanism can be changed using the configure variable `F77` which specifies the command that runs the FORTRAN 77 compiler. If your FORTRAN compiler is in a non-

[3] On HP-UX `fort77` is the POSIX compliant FORTRAN compiler, and comes after `g77`.

standard location, you should set the environment variable `PATH` accordingly before running `configure`, or use the configure variable `F77` to specify its full path.

If your FORTRAN libraries are in slightly peculiar places, you should also look at `LD_LIBRARY_PATH` or your system's equivalent to make sure that all libraries are on this path.

Note that only FORTRAN compilers which convert identifiers to lower case are supported.

You must set whatever compilation flags (if any) are needed to ensure that FORTRAN `integer` is equivalent to a C `int` pointer and FORTRAN `double precision` is equivalent to a C `double` pointer. This is checked during the configuration process.

Some of the FORTRAN code makes use of `COMPLEX*16` variables, which is a Fortran 90 extension. This is checked for at configure time[4], but you may need to avoid compiler flags asserting FORTRAN 77 compliance.

Compiling the version of LAPACK in the R sources also requires some Fortran 90 extensions, but these are not needed if an external LAPACK is used.

It might be possible to use `f2c`, the FORTRAN-to-C converter (`http://www.netlib.org/f2c`), via a script. (An example script is given in `scripts/f77_f2c`: this can be customized by setting the environment variables `F2C`, `F2CLIBS`, `CC` and `CPP`.) You will need to ensure that the FORTRAN type `integer` is translated to the C type `int`. Normally `f2c.h` contains '`typedef long int integer;`', which will work on a 32-bit platform but needs to be changed to '`typedef int integer;`' on a 64-bit platform. If your compiler is not `gcc` you will need to set `FPICFLAGS` appropriately. Also, the included LAPACK sources contain constructs that `f2c` is unlikely to be able to process, so you would need to use an external LAPACK library (such as CLAPACK from `http://www.netlib.org/clapack/`).

B.7 Compile and load flags

A wide range of flags can be set in the file `config.site` or as configure variables on the command line. We have already mentioned

`CPPFLAGS` header file search directory (`-I`) and any other miscellaneous options for the C and C++ preprocessors and compilers

`LDFLAGS` path (`-L`), stripping (`-s`) and any other miscellaneous options for the linker

and others include

`CFLAGS` debugging and optimization flags, C

`MAIN_CFLAGS`
ditto, for compiling the main program

`SHLIB_CFLAGS`
for shared objects

`FFLAGS` debugging and optimization flags, FORTRAN

`SAFE_FFLAGS`
ditto for source files which need exact floating point behaviour

`MAIN_FFLAGS`
ditto, for compiling the main program

`SHLIB_FFLAGS`
for shared objects

`MAIN_LDFLAGS`
additional flags for the main link

[4] as well as its equivalence to the `Rcomplex` structure defined in `R_ext/Complex.h`.

`SHLIB_LDFLAGS`
: additional flags for linking the shared objects

`LIBnn`
: the primary library directory, `lib` or `lib64`

`CPICFLAGS`
: special flags for compiling C code to be turned into a shared object

`FPICFLAGS`
: special flags for compiling Fortran code to be turned into a shared object

`CXXPICFLAGS`
: special flags for compiling C++ code to be turned into a shared object

`FCPICFLAGS`
: special flags for compiling Fortran 95 code to be turned into a shared object

`DEFS`
: defines to be used when compiling C code in R itself

Library paths specified as `-L/lib/path` in `LDFLAGS` are collected together and prepended to `LD_LIBRARY_PATH` (or your system's equivalent), so there should be no need for `-R` or `-rpath` flags.

Variables such as `CPICFLAGS` are determined where possible by `configure`. Some systems allows two types of PIC flags, for example '`-fpic`' and '`-fPIC`', and if they differ the first allows only a limited number of symbols in a shared object. Since R as a shared library has about 6200 symbols, if in doubt use the larger version.

To compile a profiling version of R, one might for example want to use '`MAIN_CFLAGS=-pg`', '`MAIN_FFLAGS=-pg`', '`MAIN_LDFLAGS=-pg`' on platforms where '`-pg`' cannot be used with position-independent code.

Beware: it may be necessary to set `CFLAGS` and `FFLAGS` in ways compatible with the libraries to be used: one possible issue is the alignment of doubles, another is the way structures are passed.

On some platforms `configure` will select additional flags for `CFLAGS`, `CPPFLAGS`, `FFLAGS`, `CXXFLAGS` and `LIBS` in `R_XTRA_CFLAGS` (and so on). These are for options which are always required, for example to force IEC 60559 compliance.

B.8 Maintainer mode

There are several files that are part of the R sources but can be re-generated from their own sources by configuring with option `--enable-maintainer-mode` and then running `make` in the build directory. This requires other tools to be installed, discussed in the rest of this section.

File `configure` is created from `configure.ac` and the files under `m4` by `autoconf` and `aclocal`. There is a formal version requirement on `autoconf` of 2.62 or later, but it is unlikely that anything other than the most recent versions have been thoroughly tested.

File `src/include/config.h` is created by `autoheader`.

Grammar files `*.y` are converted to C sources by an implementation of `yacc`, usually `bison -y`: these are found in `src/main` and `src/library/tools/src`. It is known that earlier versions of `bison` generate code which reads (and in some cases writes) outside array bounds: `bison` 2.6.1 was found to be satisfactory.

The ultimate sources for package **compiler** are in its `noweb` directory. To re-create the sources from `src/library/compiler/noweb/compiler.nw`, the command `notangle` is required. This is likely to need to be installed from the sources at `https://www.cs.tufts.edu/~nr/noweb/` (and can also be found on CTAN). The package sources are only re-created even in maintainer mode if `src/library/compiler/noweb/compiler.nw` has been updated.

It is likely that in future creating `configure` will need the GNU 'autoconf archive' installed. This can be found at `https://www.gnu.org/software/autoconf-archive/` and as a package (usually called **autoconf-archive**) in most packaged distributions, for example Debian, Fedora, OpenCSW, Homebrew and MacPorts.

Appendix C Platform notes

This section provides some notes on building R on different Unix-alike platforms. These notes are based on tests run on one or two systems in each case with particular sets of compilers and support libraries. Success in building R depends on the proper installation and functioning of support software; your results may differ if you have other versions of compilers and support libraries.

Older versions of this manual (for R < 2.10.0) contain notes on platforms such as HP-UX, IRIX and Alpha/OSF1 for which we have had no recent reports.

C macros to select particular platforms can be tricky to track down (there is a fair amount of misinformation on the Web). The Wiki (currently) at `http://sourceforge.net/p/predef/wiki/Home/` can be helpful. The R sources currently use

```
AIX: _AIX
Cygwin: __CYGWIN__
FreeBSD: __FreeBSD__
HP-UX: __hpux__, __hpux
IRIX: sgi, __sgi
Linux: __linux__
OS X: __APPLE__
NetBSD: __NetBSD__
OpenBSD: __OpenBSD__
Solaris: __sun, sun
Windows: _WIN32, _WIN64
```

C.1 X11 issues

The '`X11()`' graphics device is the one started automatically on Unix-alikes when plotting. As its name implies, it displays on a (local or remote) X server, and relies on the services provided by the X server.

The 'modern' version of the '`X11()`' device is based on '`cairo`' graphics and (in most implementations) uses '`fontconfig`' to pick and render fonts. This is done on the server, and although there can be selection issues, they are more amenable than the issues with '`X11()`' discussed in the rest of this section.

When X11 was designed, most displays were around 75dpi, whereas today they are of the order of 100dpi or more. If you find that X11() is reporting[1] missing font sizes, especially larger ones, it is likely that you are not using scalable fonts and have not installed the 100dpi versions of the X11 fonts. The names and details differ by system, but will likely have something like Fedora's

```
xorg-x11-fonts-75dpi
xorg-x11-fonts-100dpi
xorg-x11-fonts-ISO8859-2-75dpi
xorg-x11-fonts-Type1
xorg-x11-fonts-cyrillic
```

and you need to ensure that the '`-100dpi`' versions are installed and on the X11 font path (check via `xset -q`). The '`X11()`' device does try to set a pointsize and not a pixel size: laptop users may find the default setting of 12 too large (although very frequently laptop screens are set to a fictitious dpi to appear like a scaled-down desktop screen).

More complicated problems can occur in non-Western-European locales, so if you are using one, the first thing to check is that things work in the `C` locale. The likely issues are a failure to

[1] for example, `X11 font at size 14 could not be loaded`.

find any fonts or glyphs being rendered incorrectly (often as a pair of ASCII characters). X11 works by being asked for a font specification and coming up with its idea of a close match. For text (as distinct from the symbols used by plotmath), the specification is the first element of the option `"X11fonts"` which defaults to

```
"-adobe-helvetica-%s-%s-*-*-%d-*-*-*-*-*-*-*"
```

If you are using a single-byte encoding, for example ISO 8859-2 in Eastern Europe or KOI8-R in Russian, use `xlsfonts` to find an appropriate family of fonts in your encoding (the last field in the listing). If you find none, it is likely that you need to install further font packages, such as '`xorg-x11-fonts-ISO8859-2-75dpi`' and '`xorg-x11-fonts-cyrillic`' shown in the listing above.

Multi-byte encodings (most commonly UTF-8) are even more complicated. There are few fonts in '`iso10646-1`', the Unicode encoding, and they only contain a subset of the available glyphs (and are often fixed-width designed for use in terminals). In such locales *fontsets* are used, made up of fonts encoded in other encodings. If the locale you are using has an entry in the '`XLC_LOCALE`' directory (typically `/usr/share/X11/locale`, it is likely that all you need to do is to pick a suitable font specification that has fonts in the encodings specified there. If not, you may have to get hold of a suitable locale entry for X11. This may mean that, for example, Japanese text can be displayed when running in '`ja_JP.UTF-8`' but not when running in '`en_GB.UTF-8`' on the same machine (although on some systems many UTF-8 X11 locales are aliased to '`en_US.UTF-8`' which covers several character sets, e.g. ISO 8859-1 (Western European), JISX0208 (Kanji), KSC5601 (Korean), GB2312 (Chinese Han) and JISX0201 (Kana)).

On some systems scalable fonts are available covering a wide range of glyphs. One source is TrueType/OpenType fonts, and these can provide high coverage. Another is Type 1 fonts: the URW set of Type 1 fonts provides standard typefaces such as Helvetica with a larger coverage of Unicode glyphs than the standard X11 bitmaps, including Cyrillic. These are generally not part of the default install, and the X server may need to be configured to use them. They might be under the X11 `fonts` directory or elsewhere, for example,

```
/usr/share/fonts/default/Type1
/usr/share/fonts/ja/TrueType
```

C.2 Linux

Linux is the main development platform for R, so compilation from the sources is normally straightforward with the standard compilers.

Remember that some package management systems (such as RPM and deb) make a distinction between the user version of a package and the developer version. The latter usually has the same name but with the extension '`-devel`' or '`-dev`': you need both versions installed. So please check the `configure` output to see if the expected features are detected: if for example '`readline`' is missing add the developer package. (On most systems you will also need '`ncurses`' and its developer package, although these should be dependencies of the '`readline`' package(s).) You should expect to see in the `configure` summary

```
Interfaces supported:      X11, tcltk
External libraries:        readline, zlib, bzlib, lzma, PCRE, curl
Additional capabilities:   PNG, JPEG, TIFF, NLS, cairo, ICU
```

When R has been installed from a binary distribution there are sometimes problems with missing components such as the FORTRAN compiler. Searching the '`R-help`' archives will normally reveal what is needed.

It seems that '`ix86`' Linux accepts non-PIC code in shared libraries, but this is not necessarily so on other platforms, in particular on 64-bit CPUs such as '`x86_64`'. So care can be needed with BLAS libraries and when building R as a shared library to ensure that position-independent

code is used in any static libraries (such as the Tcl/Tk libraries, `libpng`, `libjpeg` and `zlib`) which might be linked against. Fortunately these are normally built as shared libraries with the exception of the ATLAS BLAS libraries.

The default optimization settings chosen for `CFLAGS` etc are conservative. It is likely that using `-mtune` will result in significant performance improvements on recent CPUs (especially for '`ix86`'): one possibility is to add `-mtune=native` for the best possible performance on the machine on which R is being installed: if the compilation is for a site-wide installation, it may still be desirable to use something like `-mtume=core2`.[2] It is also possible to increase the optimization levels to `-O3`: however for many versions of the compilers this has caused problems in at least one CRAN package.

For platforms with both 64- and 32-bit support, it is likely that

```
LDFLAGS="-L/usr/local/lib64 -L/usr/local/lib"
```

is appropriate since most (but not all) software installs its 64-bit libraries in `/usr/local/lib64`. To build a 32-bit version of R on '`x86_64`' with Fedora 21 we used

```
CC="gcc -m32"
CXX="g++ -m32"
F77="gfortran -m32"
FC=${F77}
OBJC=${CC}
LDFLAGS="-L/usr/local/lib"
LIBnn=lib
```

Note the use of '`LIBnn`': '`x86_64`' Fedora installs its 64-bit software in `/usr/lib64` and 32-bit software in `/usr/lib`. Linking will skip over inappropriate binaries, but for example the 32-bit Tcl/Tk configure scripts are in `/usr/lib`. It may also be necessary to set the `pkg-config` path, e.g. by

```
export PKG_CONFIG_PATH=/usr/local/lib/pkgconfig:/usr/lib/pkgconfig
```

64-bit versions of Linux are built with support for files > 2Gb, and 32-bit versions will be if possible unless `--disable-largefile` is specified.

To build a 64-bit version of R on '`ppc64`' (also known as '`powerpc64`') with `gcc` 4.1.1, Ei-ji Nakama used

```
CC="gcc -m64"
CXX="gxx -m64"
F77="gfortran -m64"
FC="gfortran -m64"
CFLAGS="-mminimal-toc -fno-optimize-sibling-calls -g -O2"
FFLAGS="-mminimal-toc -fno-optimize-sibling-calls -g -O2"
```

the additional flags being needed to resolve problems linking against `libnmath.a` and when linking R as a shared library.

C.2.1 Clang

R has been built with Linux '`ix86`' and '`x86_64`' C and C++ compilers (`http://clang.llvm.org`) based on the Clang front-ends, invoked by `CC=clang CXX=clang++`, together with `gfortran`. These take very similar options to the corresponding GCC compilers.

This has to be used in conjunction with a Fortran compiler: the `configure` code will remove `-lgcc` from `FLIBS`, which is needed for some versions of `gfortran`.

The current default for `clang++` is to use the C++ runtime from the installed `g++`. Using the runtime from the `libc++` project (`http://libcxx.llvm.org/`) has also been tested: for some R packages only the variant using `libcxxabi` was successful.

[2] or `-mtune=corei7` for Intel Core i3/15/17 with `gcc >= 4.6.0`.

Most builds of `clang` have no OpenMP support. Builds of version 3.7.0 or later may.[3]

C.2.2 Intel compilers

Intel compilers have been used under '`ix86`' and '`x86_64`' Linux. Brian Ripley used version 9.0 of the compilers for '`x86_64`' on Fedora Core 5 with

```
CC=icc
CFLAGS="-g -O3 -wd188 -ip -mp"
F77=ifort
FLAGS="-g -O3 -mp"
CXX=icpc
CXXFLAGS="-g -O3 -mp"
FC=ifort
FCFLAGS="-g -O3 -mp"
ICC_LIBS=/opt/compilers/intel/cce/9.1.039/lib
IFC_LIBS=/opt/compilers/intel/fce/9.1.033/lib
LDFLAGS="-L$ICC_LIBS -L$IFC_LIBS -L/usr/local/lib64"
SHLIB_CXXLD=icpc
```

`configure` will add '`-c99`' to `CC` for C99-compliance. This causes warnings with `icc` 10 and later, so use `CC="icc -std=c99"` there. The flag `-wd188` suppresses a large number of warnings about the enumeration type '`Rboolean`'. Because the Intel C compiler sets '`__GNUC__`' without complete emulation of `gcc`, we suggest adding `CPPFLAGS=-no-gcc`.

To maintain correct IEC 60559 arithmetic you most likely need add flags to `CFLAGS`, `FFLAGS` and `CXXFLAGS` such as `-mp` (shown above) or `-fp-model precise -fp-model source`, depending on the compiler version.

Others have reported success with versions 10.x and 11.x. % https://stat.ethz.ch/pipermail/r-devel/2015-September/071717.html BjÃ¸rn-Helge Mevik reported success with version 2015.3 of the compilers, using (for a SandyBridge CPU on Centos 6.x)

```
fast="-fp-model precise -ip -O3 -opt-mem-layout-trans=3 -xHost -mavx"
CC=icc
CFLAGS="$fast -wd188"
F77=ifort
FFLAGS="$fast"
CXX=icpc
CXXFLAGS="$fast"
FC=$F77
FCFLAGS=$F77FLAGS
```

C.2.3 Oracle Solaris Studio compilers

Brian Ripley tested the Sun Studio 12 compilers, since renamed to Oracle Solaris Studio. On '`x86_64`' Linux with

```
CC=suncc
CFLAGS="-xO5 -xc99 -xlibmil -nofstore"
CPICFLAGS=-Kpic
F77=sunf95
FFLAGS="-O5 -libmil -nofstore"
FPICFLAGS=-Kpic
CXX="sunCC -library=stlport4"
```

[3] This also needs the OpenMP runtime, which is usually distributed separately, e.g. at `http://llvm.org/releases`.

```
CXXFLAGS="-xO5 -xlibmil -nofstore -features=tmplrefstatic"
CXXPICFLAGS=-Kpic
FC=sunf95
FCFLAGS=$FFLAGS
FCPICFLAGS=-Kpic
LDFLAGS=-L/opt/sunstudio12.1/rtlibs/amd64
SHLIB_LDFLAGS=-shared
SHLIB_CXXLDFLAGS=-G
SHLIB_FCLDFLAGS=-G
SAFE_FFLAGS="-O5 -libmil"
```

`-m64` could be added, but was the default. Do not use `-fast`: see the warnings under Solaris. (The C++ options are also explained under Solaris.)

Others have found on at least some versions of '`ix86`' Linux that the configure flag `--disable-largefile` was needed (since `glob.h` on that platform presumed `gcc` was being used).

C.3 OS X

The instructions here are for '`x86_64`' builds on 10.9 (Mavericks) or later. In principle[4] R can be built for 10.6 to 10.8 but these has not been tested recently.

To build R you need Apple's 'Command Line Tools': these can be (re-)installed by `xcode-select --install`. (If you have a fresh OS installation, running e.g. `make` in a terminal will offer the installation of the command-line tools. If you have installed Xcode, this provides the command-line tools. The tools will need to be reinstalled when OS X is upgraded, as upgrading partially removes them.)

You need `readline`[5] and a Fortran compiler. Those and other binary components are available from `https://r.research.att.com/libs`: you are likely to need `pcre` and `xz` (recent OS X provides libraries but not headers for these).

An X sub-system is required unless configuring using `--without-x`: see `https://xquartz.macosforge.org/`.

To use the `quartz()` graphics device you need to configure with `--with-aqua` (which is the default): `quartz()` then becomes the default device when running R at the console and X11 would only be used for the command-line-R data editor/viewer and one version of Tcl/Tk. (This needs an Objective-C compiler[6] which can compile the source code of `quartz()`.)

Use `--without-aqua` if you want a standard Unix-alike build: apart from disabling `quartz()` and the ability to use the build with R.app, it also changes the default location of the personal library (see `?.libPaths`).

Various compilers can be used. The current CRAN distribution of R is built using

```
CC=clang
CXX=clang++
F77=gfortran-4.8
FC=$F77
OBJC=clang
CFLAGS=-Wall -mtune=core2 -g -O2
CXXFLAGS=-Wall -mtune=core2 -g -O2'
OBJCFLAGS=-Wall -mtune=core2 -g -O2
```

[4] It will be necessary to install later versions of software such as `libcurl`.

[5] Apple provides a partial emulation of GNU readline 4.2 based on the NetBSD `editline` library. That is not recommended but for the time being R's installation scripts will make use of it.

[6] These days that is defined by Apple's implementation of `clang`, so it is strongly recommended to use that.

```
F77FLAGS=-Wall -g -O2
FCFLAGS=$F77FLAGS
```

with `clang` and `clang++` from the 'Command Line Tools' and the Fortran compiler from `https://r.research.att.com/libs/gfortran-4.8.2-darwin13.tar.bz2`.[7]

Other builds of `gfortran` are available: see `https://gcc.gnu.org/wiki/GFortranBinaries` and `http://coudert.name/software.html`. To use one of these with a binary distribution of R you will probably need to specify the name or path in a personal or site `Makevars` file (see Section 6.3.3 [Customizing package compilation], page 25).

More recent and complete distributions of `clang` are usually available from `http://llvm.org/releases/`. In particular, these include support for the 'Address Sanitizer' (not included by Apple until Xcode 7) and for OpenMP[8] in version 3.7.0 and later.

Pre-compiled versions of many of the Section A.2 [Useful libraries and programs], page 39, are available from `https://r.research.att.com/libs/`. You will most likely want at least `jpeg` and `tiff`. `pkg-config` is not provided by Apple and used for many packages: it will also be used if present when configuring the `X11()` and bitmap devices.

Support for `cairo` (without `Pango`) can be enabled in two ways: both need `pkg-config` available. XQuartz ships `cairo` and its version will be selected if its `pkg-config` files are first on the configuration path: for example by setting

```
export PKG_CONFIG_PATH=/opt/X11/lib/pkgconfig:/usr/local/lib/pkgconfig:/usr/lib/pkgcon
```

or appending that variable to the `configure` command. Otherwise the binary libraries at `https://r.research.att.com/libs/` can be used: `cairo`, `fontconfig`, `freetype`, `pixman` and `pkgconfig-system-stubs-darwin13.tar.gz` are needed, plus `libpng` for PNG support.

The `Accelerate` library can be used *via* the configuration options

```
--with-blas="-framework Accelerate" --with-lapack
```

to provide potentially higher-performance versions of the BLAS and LAPACK routines.[9]

Looking at the top of `/Library/Frameworks/R.framework/Resources/etc/Makeconf` will show the compilers and configuration options used for the CRAN binary package for R: at the time of writing the non-default options

```
--enable-memory-profiling --enable-R-framework
```

were used.

Configure option `--with-internal-tzcode` is the default on OS X, as the system implementation of time zones does not work correctly for times before 1902 or after 2037 (despite using a 64-bit `time_t`).

The TEX implementation used by the developers is MacTeX (`https://www.tug.org/mactex/`): the full installation is about 4GB, but a smaller version is available at `https://www.tug.org/mactex/morepackages.html`: you will need to add some packages, e.g. for the 2015 version we needed to add **cm-super**, **helvetic**, **inconsolata** and **texinfo** which brought this to about 410MB. '`TeX Live Utility`' (available *via* the MacTeX front page) provides a graphical means to manage TEX packages.

One OS X quirk is that the default path has `/usr/local/bin` after `/usr/bin`, contrary to common practice on Unix-alikes. This means that if you install tools from the sources they will by default be installed under `/usr/local` and not supersede the system versions.

If you upgrade your OS you should re-install the 'Command Line Tools' and may need to re-install XQuartz and Java (this has been needed for some upgrades but not others).

---

[7] This is a tarball which needs to be unpacked in the Terminal by e.g. `sudo tar -zxf gfortran-4.8.2-darwin13.tar.bz2 -C /`. It does not run on Core 2 Duo Macs.

[8] This also needs the OpenMP runtime, which is distributed separately at that site.

[9] It is reported that for some non-Apple toolchains `CPPFLAGS` needed to contain `-D__ACCELERATE__`.

C.3.1 El Capitan

There are problems resulting from the new-to-El-Capitan restriction that only Apple is allowed to install software under `/usr`: this affects *inter alia* `MacTeX` and `XQuartz`. For `MacTeX` it is necessary to include `/Library/TeX/texbin` in your path rather than `/usr/texbin`. Upgrading will move disallowed files to under `/Library/SystemMigration/usr`: this includes `/usr/X11R6`, `/usr/texbin`, `/usr/bin/R`, `/usr/bin/Rscript` but not the link `/usr/X11`.

`configure` can be told to look for `X11` in `XQuartz`'s main location of `/opt/X11`, e.g. by

```
--x-includes=/opt/X11/include --x-libraries=/opt/X11/lib
```

although the linked versions under `/usr/X11` will be found (if the link is present).

C.3.2 Tcl/Tk headers and libraries

If you plan to use the `tcltk` package for R, you need to install a distribution of Tcl/Tk. There are two alternatives. If you use R.app you will want to use X11-based Tcl/Tk (as used on other Unix-alikes), which is installed as part of the CRAN binary for R. This may need

```
-with-tcltk=/usr/local/lib
```

or

```
--with-tcl-config=/usr/local/lib/tclConfig.sh
--with-tk-config=/usr/local/lib/tkConfig.sh
```

Note that this requires a matching XQuartz installation.

There is also a native ('Aqua') version of Tcl/Tk which produces widgets in the native OS X style: this will not work with R.app because of conflicts over the OS X menu, but for those only using command-line R this provides a much more intuitive interface to Tk for experienced Mac users. Most versions of OS X come with Aqua Tcl/Tk libraries, but these are not at all recent versions of Tcl/Tk (8.5.9 in El Capitan, which is not even the latest patched version in that series). It is better to install Tcl/Tk 8.6.x from the sources or a binary distribution from `https://www.activestate.com/activetcl/downloads`. Configure R with

```
--with-tcl-config=/Library/Frameworks/Tcl.framework/tclConfig.sh
--with-tk-config=/Library/Frameworks/Tk.framework/tkConfig.sh
```

(for the versions bundled with OS X, use paths starting with `/System/Library`).

If you need to find out which distribution of Tk is in use at run time, use

```
library(tcltk)
tclvalue(.Tcl("tk windowingsystem"))  # "x11" or "aqua"
```

C.3.3 Java

The situation with Java support on OS X is messy.[10]

OS X no longer comes with an installed Java runtime (JRE), and an OS X upgrade removes one if already installed: it is intended to be installed at first use. Check if a JRE is installed by running `java -version` in a `Terminal` window: if Java is not installed this should prompt you to install it. You can also install directly the latest Java from Oracle (currently from `http://www.oracle.com/technetwork/java/javase/downloads/index.html`).

You may need to install what Apple calls 'legacy Java'[11] to suppress pop-up messages even if you have a current version installed.

To see what compatible versions of Java are currently installed, run `/usr/libexec/java_home -V -a x86_64`. If needed, set the environment variable `JAVA_HOME` to choose between these, both when R is built from the sources and when `R CMD javareconf` is run.

[10] For more details see `http://www.macstrategy.com/article.php?3`.

[11] e.g. `Java For OS X 2015-001` from `https://support.apple.com/kb/DL1572`.

Configuring and building R both looks for a JRE and for support for compiling JNI programs (used by packages **rJava** (`https://CRAN.R-project.org/package=rJava`) and **JavaGD** (`https://CRAN.R-project.org/package=JavaGD`)); the latter requires a JDK (Java SDK) and not just a JRE.

The build process tries to fathom out what JRE/JDK to use, but it may need some help, e.g. by setting `JAVA_HOME`. An Apple JRE can be specified explicitly by something like

```
JAVA_HOME=/System/Library/Frameworks/JavaVM.framework/Home
JAVA_CPPFLAGS="-I/System/Library/Frameworks/JavaVM.framework/Headers"
JAVA_LD_LIBRARY_PATH=
JAVA_LIBS="-framework JavaVM"
```

The Oracle JDK can be specified explicitly by something like

```
JAVA_HOME=/Library/Java/JavaVirtualMachines/jdk1.8.0_72.jdk/Contents/Home
JAVA_CPPFLAGS="-I/${JAVA_HOME}/include -I/${JAVA_HOME}/include/darwin"
JAVA_LD_LIBRARY_PATH="${JAVA_HOME}/jre/lib/server"
JAVA_LIBS="-L/${JAVA_HOME}/jre/lib/server -ljvm"
```

in `config.site`.

Note that it is necessary to set the environment variable `NOAWT` to `1` to install many of the Java-using packages.

C.3.4 Frameworks

The CRAN build of R is installed as a framework, which is selected by the option

```
./configure --enable-R-framework
```

(This is intended to be used with an Apple toolchain: other compilers may not support frameworks correctly. It was the default prior to R 3.3.0.)

It is only needed if you want to build R for use with the R.app console, and implies `--enable-R-shlib` to build R as a dynamic library. This option configures R to be built and installed as a framework called `R.framework`. The default installation path for `R.framework` is `/Library/Frameworks` but this can be changed at configure time by specifying the flag `--enable-R-framework[=`*DIR*`]` (or `--prefix`) or at install time *via*

```
make prefix=/where/you/want/R.framework/to/go install
```

Note that installation as a framework is non-standard (especially to a non-standard location) and Unix utilities may not support it (e.g. the `pkg-config` file `libR.pc` will be put somewhere unknown to `pkg-config`).

C.3.5 Building R.app

Note that building the R.app GUI console is a separate project, using Xcode. Before compiling R.app make sure the current version of R is installed in `/Library/Frameworks/R.framework` and working at the command-line (this can be a binary install).

The current sources can be checked out by

```
svn co https://svn.r-project.org/R-packages/trunk/Mac-GUI
```

and built by loading the `R.xcodeproj` project (select the `R` target and a suitable configuration), or from the command-line by e.g.

```
xcodebuild -target R -configuration Release
```

See also the `INSTALL` file in the checkout or directly at `https://svn.r-project.org/R-packages/trunk/Mac-GUI/INSTALL`.

R.app does not need to be installed in any specific way. Building R.app results in the R.app bundle which appears as one R icon. This application bundle can be run anywhere and it is customary to place it in the `/Applications` folder.

C.4 Solaris

R has been built successfully on Solaris 10 (both Sparc and 'x86') using the (zero cost) Oracle Solaris Studio compilers: there has been some success with gcc 4/gfortran. (Recent Sun machines are AMD Opterons or Intel Xeons ('amd64') rather than 'x86', but 32-bit 'x86' executables are the default.)

There have been few reports on Solaris 11, with no known extra issues. Solaris 9 and earlier are now so old that it is unlikely that R is still used with them, and they will not be considered here.

The Solaris versions of several of the tools needed to build R (e.g. make, ar and ld) are in /usr/ccs/bin, so if using those tools ensure this is in your path. A version of the preferred GNU tar is (if installed) in /usr/sfw/bin. It may be necessary to avoid the tools in /usr/ucb: POSIX-compliant versions of some tools can be found in /usr/xpg4/bin and /usr/xpg6/bin.

A large selection of Open Source software can be installed from https://www.opencsw.org, by default installed under /opt/csw. Solaris 10 ships with bzlib version 1.0.6 (sufficient) but zlib version 1.2.3 (too old): OpenCSW has 1.2.8.

You will need GNU libiconv and readline: the Solaris version of iconv is not sufficiently powerful.

The native make suffices to build R but a small number of packages require GNU make (some without good reason and without declaring it as 'SystemRequirements' in the DESCRIPTION file).

Some people have reported that the Solaris libintl needs to be avoided, for example by using --disable-nls or --with-included-gettext or using libintl from OpenCSW. (On the other hand, there have been many successful installs which automatically detected libintl from OpenCSW or selected the included gettext.)

The support for the C99 long double type on Sparc hardware uses quad-precision arithmetic, and this is usually slow because it is done by software emulation. On such systems the configure option --disable-long-double can be used for faster but less accurate computations.

The Solaris time-zone conversion services seem to be unreliable pre-1916 in Europe (when daylight-savings time was first introduced): most often reporting in the non-existent DST variant. Using configure option --with-internal-tzcode is recommended, and required if you find time-zone abbreviations being given odd values (as has been seen on 64-bit builds without it).

When using the Oracle compilers[12] do *not* specify -fast, as this disables IEEE arithmetic and make check will fail.

It has been reported that some Solaris installations need

```
INTERNET_LIBS="-lsocket -lnsl"
```

on the configure command line or in file config.site; however, there have been many successful installs without this.

A little juggling of paths was needed to ensure GNU libiconv (in /usr/local) was used rather than the Solaris iconv:

```
CC="cc -xc99"
CFLAGS="-O -xlibmieee"
F77=f95
FFLAGS=-O
CXX="CC -library=stlport4"
CXXFLAGS=-O
```

[12] including gcc for Sparc from Oracle.

```
FC=f95
FCFLAGS=$FFLAGS
FCLIBS="-lfai -lfsu"
R_LD_LIBRARY_PATH="/usr/local/lib:/opt/csw/gcc4/lib:/opt/csw/lib"
```

For a 64-bit target add `-m64` to the compiler macros and use something like `LDFLAGS=-L/usr/local/lib/sparcv9` or `LDFLAGS=-L/usr/local/lib/amd64` as appropriate. It will also be necessary to point `pkg-config` at the 64-bit directories, e.g. one of

```
PKG_CONFIG_PATH=/opt/csw/lib/amd64/pkgconfig:/usr/lib/amd64/pkgconfig
PKG_CONFIG_PATH=/opt/csw/lib/sparcv9/pkgconfig:/usr/lib/sparcv9/pkgconfig
```

and to specify a 64-bit Java VM by e.g.

```
JAVA_CPPFLAGS="-I${JAVA_HOME}/../include -I${JAVA_HOME}/../include/solaris"
JAVA_LD_LIBRARY_PATH=${JAVA_HOME}/lib/amd64/server
JAVA_LIBS="-L${JAVA_HOME}/lib/amd64/server \
  -R${JAVA_HOME}/lib/amd64/server -ljvm"
```

With Solaris Studio 12.[23] on Sparc, `FCLIBS` needs to be

```
FCLIBS="-lfai -lfai2 -lfsu"
```

(and possibly other Fortran libraries, but this suffices for the packages currently on CRAN).

Currently '`amd64`' and '`sparcv9`' builds work out-of-the-box with Sun Studio 12u1 but not Solaris Studio 12.2 and 12.3: `libRblas.so` and `lapack.so` are generated with code that causes relocation errors (which is being linked in from the Fortran libraries). This means that building 64-bit R as a shared library may be impossible with Solaris Studio >= 12.2. For a standard build the trick seems to be to manually set `FLIBS` to avoid the troublesome libraries. For example, on '`amd64`' set in `config.site` something like

```
FLIBS_IN_SO="-R/opt/solarisstudio12.3/lib/amd64
  /opt/solarisstudio12.3/lib/amd64/libfui.so
  /opt/solarisstudio12.3/lib/amd64/libfsu.so"
```

For 64-bit Sparc, set in `config.site` something like

```
FLIBS="-R/opt/solarisstudio12.3/prod/lib/sparc/64
 -lifai -lsunimath -lfai -lfai2 -lfsumai -lfprodai -lfminlai -lfmaxlai
 -lfminvai -lfmaxvai -lfui -lsunmath -lmtsk
 /opt/solarisstudio12.3/prod/lib/sparc/64/libfsu.so.1"
```

By default the Solaris Studio compilers do not by default conform to the C99 standard (appendix F 8.9) on the return values of functions such as `log`: use `-xlibmieee` to ensure this.

You can target specific Sparc architectures for (slightly) higher performance: `-xtarget=native` (in `CFLAGS` etc) tunes the compilation to the current machine.

Using `-xlibmil` in `CFLAGS` and `-xlibmil` in `FFLAGS` allows more system mathematical functions to be inlined.

On '`x86`' you will get marginally higher performance *via*

```
CFLAGS="-xO5 -xc99 -xlibmieee -xlibmil -nofstore -xtarget=native"
FFLAGS="-O5 -libmil -nofstore -xtarget=native"
CXXFLAGS="-xO5 -xlibmil -nofstore -xtarget=native"
SAFE_FFLAGS="-libmil -fstore -xtarget=native"
```

but the use of `-nofstore` can be less numerically stable, and some packages (notably **mgcv** (`https://CRAN.R-project.org/package=mgcv`) on '`x86`') failed to compile at higher optimization levels with version 12.3.

The Solaris Studio compilers provide several implementations of the C++98 standard which select both the set of headers and a C++ runtime library. These are selected by the `-library` flag, which as it is needed for both compiling and linking is best specified as part of the compiler.

The examples above use 'stlport4', currently the most modern of the options: the default (but still needed to be specified as it is needed for linking) is 'Cstd': see http://www.oracle.com/technetwork/server-storage/solaris/cmp-stlport-libcstd-142559.html. Note though that most external Solaris C++ libraries will have been built with 'Cstd' and so an R package using such libraries also needs to be. Occasionally the option -library=stlport4,Crun has been needed.

Several CRAN packages using C++ need the more liberal interpretation given by adding

```
CXXFLAGS="-features=tmplrefstatic"
```

The performance library sunperf is available for use with the Solaris Studio compilers. If selected as a BLAS, it must also be selected as LAPACK *via* (for Solaris Studio 12.2 and later)

```
./configure --with-blas='-library=sunperf' --with-lapack
```

This has often given test failures in the past, in several different places. At the time of writing it fails in tests/reg-BLAS.R, and on some builds, including for 'amd64', it fails in example(eigen).

Parsing very complex R expressions needs a lot of stack space when the Oracle compilers are used: several packages require the stack increased to at least 20MB.

C.4.1 Using gcc

If using gcc, ensure that the compiler was compiled for the version of Solaris in use. (This can be ascertained from gcc -v.) gcc makes modified versions of some header files, and several reports of problems were due to using gcc compiled on one version of Solaris on a later version.

The notes here are for gcc set up to use the Solaris linker: it can also be set up to use GNU ld, but that has not been tested.

Compilation for a 32-bit Sparc target with gcc 4.9.2 needed

```
CPPFLAGS=-I/opt/csw/include
LDFLAGS="-L/opt/csw/gcc4/lib -L/opt/csw/lib"
```

and for a 64-bit Sparc target

```
CC="gcc -m64"
F77="gfortran -m64"
CXX="g++ -m64"
FC=$F77
CPPFLAGS=-I/opt/csw/include
LDFLAGS="-L/opt/csw/gcc4/lib/sparcv9 -L/opt/csw/lib/sparcv9"
```

Note that paths such as /opt/csw/gcc4/lib/sparcv9 may need to be in the LD_LIBRARY_PATH during configuration.

The compilation can be tuned to a particular cpu: the CRAN check system uses -mtune=niagara2.

Compilation for an 'x86' target with gcc 4.9.2 needed

```
CC="/opt/csw/gcc4/bin/gcc -m32"
CPPFLAGS="-I/opt/csw/include -I/usr/local/include"
F77="/opt/csw/gcc4/bin/gfortran -m32"
CXX="/opt/csw/gcc4/bin/g++ -m32"
FC="/opt/csw/gcc4/bin/gfortran -m32"
LDFLAGS="-L/opt/csw/gcc4/lib -L/opt/csw/lib -L/usr/local/lib"
```

(-L/opt/csw/lib is needed since TeX Live was built using 32-bit gcc, and we need /opt/csw/lib in R_LD_LIBRARY_PATH.)

For an 'amd64' target with gcc 4.9.2 we used

```
CC="/opt/csw/gcc4/bin/gcc -m64"
CPPFLAGS="-I/opt/csw/include -I/usr/local/include"
```

```
F77="/opt/csw/gcc4/bin/gfortran -m64"
FPICFLAGS=-fPIC
CXX="/opt/csw/gcc4/bin/g++ -m64"
FC=$F77
FCPICFLAGS=$FPICFLAGS
LDFLAGS="-L/opt/csw/gcc4/lib/amd64 -L/opt/csw/lib/amd64"
```

C.5 AIX

We no longer support AIX prior to 4.2, and `configure` will throw an error on such systems.

Ei-ji Nakama was able to build under AIX 5.2 on '`powerpc`' with GCC 4.0.3 in several configurations. 32-bit versions could be configured with `--without-iconv` as well as `--enable-R-shlib`. For 64-bit versions he used

```
OBJECT_MODE=64
CC="gcc -maix64"
CXX="g++ -maix64"
F77="gfortran -maix64"
FC="gfortran -maix64"
```

and was also able to build with the IBM `xlc` and Hitachi `f90` compilers by

```
OBJECT_MODE=64
CC="xlc -q64"
CXX="g++ -maix64"
F77="f90 -cpu=pwr4 -hf77 -parallel=0 -i,L -O3 -64"
FC="f90 -cpu=pwr4 -hf77 -parallel=0 -i,L -O3 -64"
FLIBS="-L/opt/ofort90/lib -lhf90vecmath -lhf90math -lf90"
```

Some systems have `f95` as an IBM compiler that does not by default accept FORTRAN 77. It needs the flag `-qfixed=72`, or to be invoked as `xlf_r`.

The AIX native `iconv` does not support encodings '`latin1`' nor '`""`' and so cannot be used. (As far as we know GNU `libiconv` could be installed.)

Fan Long reported success on AIX 5.3 using

```
OBJECT_MODE=64
LIBICONV=/where/libiconv/installed
CC="xlc_r -q64"
CFLAGS="-O -qstrict"
CXX="xlC_r -q64"
CXXFLAGS="-O -qstrict"
F77="xlf_r -q64"
AR="ar -X64"
CPPFLAGS="-I$LIBICONV/include -I/usr/lpp/X11/include/X11"
LDFLAGS="-L$LIBICONV/lib -L/usr/lib -L/usr/X11R6/lib"
```

On one AIX 6.x system it was necessary to use `R_SHELL` to set the default shell to be Bash rather than Zsh.

Kurt Hornik and Stefan Theussl at WU (Wirtschaftsuniversität Wien) successfully built R on a '`powerpc`' (8-CPU Power6 system) running AIX 6.1, configuring with or without `--enable-R-shlib` (Ei-ji Nakama's support is gratefully acknowledged).

It helps to describe the WU build environment first. A small part of the software needed to build R and/or install packages is available directly from the AIX Installation DVDs, e.g., Java 6 and X11. Additional open source software (OSS) is packaged for AIX in `.rpm` files and available from both IBM's "AIX Toolbox for Linux Applications" (`http://www-03.ibm.com/systems/power/software/aix/linux/`) and `http://www.oss4aix.org/download/`. The

latter website typically offers more recent versions of the available OSS. All tools needed and libraries downloaded from these repositories (e.g., GCC, Make, `libreadline`, etc.) are typically installed to `/opt/freeware`, hence corresponding executables are found in `/opt/freeware/bin` which thus needs to be in `PATH` for using these tools. As on other Unix systems one needs GNU `libiconv` as the AIX version of iconv is not sufficiently powerful. Additionally, for proper Unicode compatibility one should install the corresponding package from the ICU project (`http://www.icu-project.org/download/`), which offers pre-compiled binaries for various platforms which in case of AIX can be installed via unpacking the tarball to the root file system. For full LaTeX support one can install the TeX Live DVD distribution (`https://www.tug.org/texlive/`): it is recommended to update the distribution using the `tlmgr` update manager. For 64-bit R builds supporting Tcl/Tk this needs to installed from the sources as available pre-compiled binaries supply only 32-bit shared objects.

The recent WU testing was done using compilers from both the GNU Compiler Collection (version 4.2.4) which is available from one of the above OSS repositories, and the IBM C/C++ (XL C/C++ 10.01) as well as FORTRAN (XL Fortran 12.01) compilers (`http://www14.software.ibm.com/webapp/download/byproduct.jsp#X`).

To compile for a 64-bit '`powerpc`' (Power6 CPU) target one can use

```
CC ="gcc -maix64 -pthread"
CXX="g++ -maix64 -pthread"
FC="gfortran -maix64 -pthread"
F77="gfortran -maix64 -pthread"
CFLAGS="-O2 -g -mcpu=power6"
FFLAGS="-O2 -g -mcpu=power6"
FCFLAGS="-O2 -g -mcpu=power6"
```

for the GCC and

```
CC=xlc
CXX=xlc++
FC=xlf
F77=xlf
CFLAGS="-qarch=auto -qcache=auto -qtune=auto -O3 -qstrict -ma"
FFLAGS="-qarch=auto -qcache=auto -qtune=auto -O3 -qstrict"
FCFLAGS="-qarch=auto -qcache=auto -qtune=auto -O3 -qstrict"
CXXFLAGS="-qarch=auto -qcache=auto -qtune=auto -O3 -qstrict"
```

for the IBM XL compilers. For the latter, it is important to note that the decision for generating 32-bit or 64-bit code is done by setting the `OBJECT_MODE` environment variable appropriately (recommended) or using an additional compiler flag (`-q32` or `-q64`). By default the IBM XL compilers produce 32 bit code. Thus, to build R with 64-bit support one needs to either export `OBJECT_MODE=64` in the environment or, alternatively, use the `-q64` compiler options.

It is strongly recommended to install Bash and use it as the configure shell, e.g., via setting `CONFIG_SHELL=/usr/bin/bash` in the environment, and to use GNU Make (e.g., via (`MAKE=/opt/freeware/bin/make`).

Further installation instructions to set up a proper R development environment can be found in the "R on AIX" project on R-Forge (`https://R-Forge.R-project.org/projects/aix/`).

C.6 FreeBSD

There have been few recent reports on FreeBSD.

There is a 'port' at `https://www.freebsd.org/ports/math.html`, for R 3.0.2 at the time of writing. Davor Cubranic reported some success on x86_64 FreeBSD 10.2 for R 3.2.2.

Use of ICU for collation and the `configure` option `--with-internal-tzcode` are desirable workarounds.

C.7 OpenBSD

Ingo Feinerer installed R version 3.2.2 on OpenBSD 5.8 arch '`amd64`' (their name for '`x86_64`'). Details of the build (and patches applied) are at `http://cvsweb.openbsd.org/cgi-bin/cvsweb/ports/math/R/`.

C.8 Cygwin

The 32-bit version has never worked well enough to pass R's `make check`, and residual support from earlier experiments was removed in R 3.3.0.

The 64-bit version is completely unsupported.

C.9 New platforms

There are a number of sources of problems when installing R on a new hardware/OS platform. These include

Floating Point Arithmetic: R requires arithmetic compliant with IEC 60559, also known as IEEE 754. This mandates the use of plus and minus infinity and `NaN` (not a number) as well as specific details of rounding. Although almost all current FPUs can support this, selecting such support can be a pain. The problem is that there is no agreement on how to set the signalling behaviour; Sun/Sparc, SGI/IRIX and '`ix86`' Linux require no special action, FreeBSD requires a call to (the macro) `fpsetmask(0)` and OSF1 required that computation be done with a `-ieee_with_inexact` flag etc. On a new platform you must find out the magic recipe and add some code to make it work. This can often be done via the file `config.site` which resides in the top level directory.

Beware of using high levels of optimization, at least initially. On many compilers these reduce the degree of compliance to the IEEE model. For example, using `-fast` on the Solaris Studio compilers has caused R's `NaN` to be set incorrectly, and `gcc`'s `-ffast-math` and `clang`'s `-Ofast` have given incorrect results.

Shared Objects: There seems to be very little agreement across platforms on what needs to be done to build shared objects. there are many different combinations of flags for the compilers and loaders. GNU libtool cannot be used (yet), as it currently does not fully support FORTRAN: one would need a shell wrapper for this). The technique we use is to first interrogate the X window system about what it does (using `xmkmf`), and then override this in situations where we know better (for tools from the GNU Compiler Collection and/or platforms we know about). This typically works, but you may have to manually override the results. Scanning the manual entries for `cc` and `ld` usually reveals the correct incantation. Once you know the recipe you can modify the file `config.site` (following the instructions therein) so that the build will use these options.

It seems that `gcc` 3.4.x and later on '`ix86`' Linux defeat attempts by the LAPACK code to avoid computations entirely in extended-precision registers, so file `src/modules/lapack/dlamc.f` may need to be compiled without optimization. Set the configure variable `SAFE_FFLAGS` to the flags to be used for this file. If configure detects GNU FORTRAN it adds flag `-ffloat-store` to `FFLAGS`. (Other settings are needed when using `icc` on '`ix86`' Linux, for example. Using `-mpc64` is preferable on more recent GCC compilers.)

If you do manage to get R running on a new platform please let us know about it so we can modify the configuration procedures to include that platform.

If you are having trouble getting R to work on your platform please feel free to use the '`R-devel`' mailing list to ask questions. We have had a fair amount of practice at porting R to new platforms ...

Appendix D The Windows toolset

If you want to build R or add-on packages from source in Windows, you will need to collect, install and test an extensive set of tools. See `https://CRAN.R-project.org/bin/windows/Rtools/` for the current locations and other updates to these instructions. (Most Windows users will not need to build add-on packages from source; see Chapter 6 [Add-on packages], page 22, for details.)

We have found that the build process for R is quite sensitive to the choice of tools: please follow our instructions **exactly**, even to the choice of particular versions of the tools.[1] The build process for add-on packages is somewhat more forgiving, but we recommend using the exact toolset at first, and only substituting other tools once you are familiar with the process.

This appendix contains a lot of prescriptive comments. They are here as a result of bitter experience. Please do not report problems to the R mailing lists unless you have followed all the prescriptions.

We have collected most of the necessary tools (unfortunately not all, due to license or size limitations) into an executable installer named `Rtools*.exe`, available from `https://CRAN.R-project.org/bin/windows/Rtools/`. You should download and run it, choosing the default "Package authoring installation" to build add-on packages, or the "full installation" if you intend to build R.

You will need the following items to build R and packages. See the subsections below for detailed descriptions.

- The command line tools (in `Rtools*.exe`)
- The MinGW-w64 32/64-bit toolchain to compile C, Fortran and C++.

For installing simple source packages containing data or R source but no compiled code, none of these are needed.

A complete build of R including PDF manuals, and producing the installer will also need the following:

- LaTeX
- The Inno Setup installer
- (optional) `qpdf`

It is important to set your `PATH` properly. The installer `Rtools*.exe` optionally sets the path to components that it installs.

Your `PATH` may include `.` first, then the `bin` directories of the tools, the compiler toolchain and LaTeX. Do not use filepaths containing spaces: you can always use the short forms (found by `dir /x` at the Windows command line). Network shares (with paths starting `\\`) are not supported.

For example for a 32-bit build, all on one line,

```
PATH=c:\Rtools\bin;c:\Rtools\gcc-4.6.3\bin;c:\MiKTeX\miktex\bin;
     c:\R\R-3.2\bin\i386;c:\windows;c:\windows\system32
```

It is essential that the directory containing the command line tools comes first or second in the path: there are typically like-named tools[2] in other directories, and they will **not** work. The ordering of the other directories is less important, but if in doubt, use the order above.

Our toolset contains copies of Cygwin DLLs that may conflict with other ones on your system if both are in the path at once. The normal recommendation is to delete the older ones; however, at one time we found our tools did not work with a newer version of the Cygwin DLLs, so it may be safest not to have any other version of the Cygwin DLLs in your path.

[1] For example, the Cygwin version of `make 3.81` fails to work correctly.

[2] such as `sort`, `find` and perhaps `make`.

D.1 LaTeX

The 'MiKTeX' (http://www.miktex.org/) distribution of LaTeX includes a suitable port of pdftex. This can be set up to install extra packages 'on the fly', which is the simplest way to use it (and the default). The 'basic' version of 'MiKTeX' almost suffices: when last checked packages

```
epsf inconsolata mptopdf url
```

needed to be added (on the fly or *via* the 'MiKTeX' Package Manager) to install R. In any case ensure that the **inconsolata** package is installed—you can check with the 'MiKTeX' Package Manager.

The Rtools*.exe installer does *not* include any version of LaTeX.

It is also possible to use the TeX Live distribution from https://www.tug.org/texlive/.

Please read Section 2.3 [Making the manuals], page 4, about how to make fullrefman.pdf and set the environment variable R_RD4PDF suitably; ensure you have the required fonts installed or that 'MiKTeX' is set up to install LaTeX packages on first use.

D.2 The Inno Setup installer

To make the installer package (R-3.3.1-win.exe) we currently require the Unicode version of Inno Setup 5.3.7 or later from http://jrsoftware.org/. This is *not* included in Rtools*.exe.

Copy file src/gnuwin32/MkRules.dist to src/gnuwin32/MkRules.local and edit it to set ISDIR to the location where Inno Setup was installed.

D.3 The command line tools

This item is installed by the Rtools*.exe installer.

If you choose to install these yourself, you will need suitable versions of at least basename, cat, cmp, comm, cp, cut, date, diff, du, echo, expr, gzip, ls, make, makeinfo, mkdir, mv, rm, rsync, sed, sh, sort, tar, texindex, touch and uniq; we use those from the Cygwin distribution (https://www.cygwin.com/) or compiled from the sources. You will also need zip and unzip from the Info-ZIP project (http://www.info-zip.org/). All of these tools are in Rtools*.exe.

Beware: 'Native' ports of make are **not** suitable (including those called 'MinGW make' at the MinGW SourceForge site and mingw32-make in some MinGW-w64 distributions). There were also problems with other versions of the Cygwin tools and DLLs. To avoid frustration, please use our tool set, and make sure it is at the front of your path (including before the Windows system directories). If you are using a Windows shell, type PATH at the prompt to find out.

You may need to set the environment variable CYGWIN to a value including 'nodosfilewarning' to suppress messages about Windows-style paths.

D.4 The MinGW-w64 toolchain

Technically you need more than just a compiler so the set of tools is referred to as a 'toolchain'.

The preferred toolchain is part of Rtools*.exe: this uses a beta version of gcc 4.6.3 and version 2.0.1 of the MinGW-w64 project's runtime.

This toolchain uses *multilib*: that is there is a single front-end such as gcc.exe for each of the compilers and 32-bit (the default) and 64-bit compilation are selected by the flags[3] -m32 and -m64 respectively. The tools are all 32-bit Windows executables and should be able to run

[3] these flags apply to the compilers: some of the tools use different flags. 32-bit builds are the default.

on any current version of Windows—however you do need a 64-bit version of Windows to build 64-bit R as the build process runs R.

To select a 32-bit or 64-bit build of R, set the options in `MkRules.local` appropriately (following the comments in the file).

Some external software libraries will need to be re-compiled under the new toolchain: especially those providing a C++ interface. Many of those used by CRAN packages are available from `https://www.stats.ox.ac.uk/pub/Rtools/multilib/`. Users developing packages with **Rcpp** (`https://CRAN.R-project.org/package=Rcpp`) need to ensure that they use a version built with exactly the same toolchain as their package: the recommendation is to build **Rcpp** (`https://CRAN.R-project.org/package=Rcpp`) from its sources yourself.

There is support for OpenMP and pthreads in this toolchain. As the performance of OpenMP on Windows is poor for small tasks, it is not used for R itself.

D.5 Useful additional programs

The process of making the installer will make use of `qpdf` to compact some of the package vignettes, if it is available. Windows binaries of `qpdf` are available from `http://sourceforge.net/projects/qpdf/files/`. Set the path to the `qpdf` installation in file `MkRules.local`.

Developers of packages will find some of the 'goodies' at `https://www.stats.ox.ac.uk/pub/Rtools/goodies` useful.

There is a version of the `file` command that identifies the type of files, and is used by `Rcmd check` if available. The binary distribution is included in `Rtools*.exe`.

The file `xzutils.zip` contains the program `xz` which can be used to (de)compress files with that form of compression.

Function and variable index

C

`configure` 3, 4, 6, 7, 47, 48

I

`install.packages` 23

M

`make` 48

R

`R_HOME` 3
`remove.packages` 28

U

`update.packages` 27

Concept index

A

AIX 63

B

BLAS library 41, 57, 62

F

FORTRAN 48
FreeBSD 64

I

Installation 6
Installing under Unix-alikes 3
Installing under Windows 12
Internationalization 30

L

LAPACK library 44, 57, 62
Libraries 22
Libraries, managing 22
Libraries, site 22
Libraries, user 22
Linux 3, 53
Locale 30
Localization 30

M

Manuals 4
Manuals, installing 7

O

Obtaining R 1
OpenBSD 65
OS X 3, 18, 56

P

Packages 22
Packages, default 22
Packages, installing 22
Packages, removing 28
Packages, updating 27

R

Repositories 28

S

Site libraries 22
Solaris 60
Sources for R 1
Subversion 1, 38

U

User libraries 22

V

Vignettes 38

W

winCairo.dll 14

Environment variable index

B

BINPREF 24
BINPREF64 24
BLAS_LIBS 41

C

CC 49
CONFIG_SITE 47
CPP 49
CYGWIN 68

D

DESTDIR 8, 35

F

F2C 49
F2CLIBS 49
FPICFLAGS 49

J

JAVA_HOME 40

L

LANG 31
LANGUAGE 31, 32
LAPACK_LIBS 45
LC_ALL 31
LC_COLLATE 11
LC_MESSAGES 31
LD_LIBRARY_PATH 35, 43, 49, 50, 62
LOCAL_SOFT 24

O

OBJECT_MODE 64

P

PAPERSIZE 47
PATH 38, 49, 64, 67

R

R_ARCH 9
R_BROWSER 47
R_DEFAULT_PACKAGES 22
R_DISABLE_HTTPD 4
R_GSCMD 40
R_INSTALL_TAR 24
R_JAVA_LD_LIBRARY_PATH 40, 41
R_LIBS 22
R_LIBS_SITE 22
R_LIBS_USER 22
R_PAPERSIZE 5, 21, 47, 48
R_PDFVIEWER 47
R_RD4PDF 6, 48, 68
R_SHELL 63
R_USER 21

T

TAR 38
TAR_OPTIONS 1, 13
TEMP 21
TMP 21
TMPDIR 3, 13, 21, 22