R Installation and Administration

Version 2.7.0 (2008-04-22)

R Development Core Team

ISBN 3-900051-09-7

Table of Contents

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

on systems that have GNU `tar` installed. On other systems you need at least to have the `gzip` program installed. Then you can use

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

The pathname of the directory into which the sources are unpacked should not contain spaces, as `make` (specifically GNU `make` 3.80) does not expect spaces.

If you need to transport the sources on floppy disks, you can download the '`R-x.y.z.tar.gz-split.*`' files and paste them together at the destination with (Unix)

```
cat R-x.y.z-split.* > R-x.y.z.tar.gz
```

and proceed as above. 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.)

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 (2.x.0) version, '`r-patched`' will refer to beta/release candidates of the upcoming release, the patched version of the current release being available only via Subversion.)

The tarballs are available from ftp://ftp.stat.math.ethz.ch/pub/Software/R/. Download either '`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 subversion.tigris.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*. 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.

Note that '`https:`' is required, and that the SSL certificate for the Subversion server of the R project is

```
Certificate information:
 - Hostname: svn.r-project.org
 - Valid: from Jul 16 08:10:01 2004 GMT until Jul 14 08:10:01 2014 GMT
 - Issuer: Department of Mathematics, ETH Zurich, Zurich, Switzerland, CH
 - Fingerprint: c9:5d:eb:f9:f2:56:d1:04:ba:44:61:f8:64:6b:d9:33:3f:93:6e:ad
```

(currently, there is no "trusted certificate"). You can accept this certificate permanently and will not be asked about it anymore.

Note that retrieving the sources by e.g. `wget -r` or `svn export` from that URL will not work: 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 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 '`http://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

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

2 Installing R under Unix-alikes

R will configure and build under a number of common Unix and Unix-alike platforms including '*cpu*`-*-linux-gnu`' for the '`alpha`', '`arm`', '`hppa`', '`ix86`', '`ia64`', '`m68k`', '`mips`', '`mipsel`', '`powerpc`', '`s390`', '`sparc`', and '`x86_64`' CPUs, '`powerpc-apple-darwin`', '`i386-apple-darwin`' and '`sparc-sun-solaris`', as well as probably (it is tested less frequently on these platforms) '`i386-*-freebsd`', '`x86_64-*-freebsd`', '`i386-*-netbsd`', '`i386-*-openbsd`', '`i386-sun-solaris`', '`mips-sgi-irix`' and '`alpha-dec-osf*`'.

In addition, binary distributions are available for some common Linux distributions and for Mac OS X. 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.

2.1 Simple compilation

First review the essential and useful tools and libraries in Appendix A [Essential and useful other programs under Unix], page 26, 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 executed from) or points to a valid temporary directory.

Choose a place 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. (At this point North American readers should consult Section B.3.1 [Setting paper size], page 34.) Issue the following commands:

```
./configure
make
```

(See Section B.4 [Using make], page 35 if your make is not called '`make`'.)

Then check the built system works correctly by

```
make check
```

Failures are not necessarily problems as they might be caused by missing functionality,[1] 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-European locales.)

To re-run the tests including those successfully run you would need

```
make check FORCE=FORCE
```

More comprehensive testing can be done by

```
make check-devel
```

or

```
make check-all
```

see '`tests/README`'.

If the command `configure` and `make` commands execute successfully, the R binary will be copied to '*R_HOME*`/bin/exec/R`'. In addition, a shell-script front-end called '`R`' will be created and copied to the same directory. You can 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 2.3 [Installation], page 5. 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

[1] for example, if you configured R with '`--without-iconv`' or '`--without-recommended`'.

```
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 the pathname of the build directory should not contain spaces.)

Make will also build plain text help pages as well as HTML and LaTeX versions of the R object documentation (the three kinds can also be generated separately using `make help`, `make html` and `make latex`).

For those obtaining R *via* Subversion, one additional step is necessary:

```
make vignettes
```

which makes the **grid** vignettes (which are contained in the tarballs): it takes several minutes.

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

2.2 Making the manuals

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

'`refman`' Printed versions of all the help 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, use

`make dvi`	to create DVI versions
`make pdf`	to create PDF versions
`make info`	to create info files (not '`refman`').

You will not be able to build any of these unless you have `makeinfo` version 4.7 or later installed, and for DVI or 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).

The DVI versions can be previewed and printed using standard programs such as `xdvi` and `dvips`. The PDF versions can be viewed using Acrobat Reader or (fairly recent versions of) `xpdf` and `ghostscript`: they have hyperlinks that can be followed in the first two. The info files are suitable for reading online with Emacs or the standalone GNU Info. The DVI and PDF versions will be created using the papersize 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 papersize, you should first delete the file '`doc/manual/version.texi`'.)

There are some issues with making the reference manual, and in particular with the PDF version '`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 for PDF.) Using standard PostScript fonts. This works well both for on-screen viewing and for printing. The one disadvantage is that the Usage and Examples sections may come out rather wide.

`lm` Using the *Latin Modern* fonts. These are not often installed as part of a TEX distribution, but can obtained from 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 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 so readable on-screen as the other three options.

`ae` (The default for DVI.) 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`', '`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 variables `R_RD4PDF` and `R_RD4DVI`. (On Unix, these will be picked up at install time.) The default value for `R_RD4PDF` is `times,hyper`: omit `hyper` if you do not want hyperlinks, e.g. for printing. The default for `R_RD4DVI` is `ae`.

2.3 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
```

This will install to the following directories:

'*`prefix`*`/bin`' or '*`bindir`*'
: the front-end shell script

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

'*`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
```

This causes `make install` to install the R executable 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`. You can install into another directory tree by using

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

at least with GNU `make` (but not e.g. Solaris 8's `make`).

More precise control is available at configure time via options: see `configure --help` for details. (However, many of them are currently unused.)

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-2.7.0`.

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

```
make install rhome=/usr/local/lib64/R-2.6.0
```

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

If you have made R as a shared/dynamic 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.

```
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.

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

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

Once again, it is optional to specify `prefix`, `libdir` or `rhome` (the DVI and PDF manuals are installed under the R home directory).

More precise control is possible. For info, the setting used is that of `infodir` (default '*`prefix`*`/info`', set by configure option '`--infodir`'). The DVI and 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 http://www.gnu.org/prep/standards/html_node/DESTDIR.html

Parallel makes are supported for making R only, not for installation nor for checking.

2.4 Uninstallation

You can uninstall R by

```
make uninstall
```

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

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

2.5 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 Solaris for different chips (in particular, 32- and 64-bit builds), 64- and 32- bit builds on 'x86_64' Linux and different CPUs ('ppc', 'ppc64', 'i386' and 'x86_64') under Mac OS >= 10.4.

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 'libs'. Example names from other systems 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 63Mb, 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 that 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 pkg(s)
```

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.

On Linux, there is an alternative mechanism for mixing 32-bit and 64-bit libraries known as *multilib*. If a 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 chosen using the `setarch` command. For example, a 32-bit build may be chosen 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).

At present there is a protential problem with packages using Java, as the post-install for a 'i386' 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.

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 Windows 2000 or later on ix86 CPUs (including AMD64/EM64T chips and Windows x64).

Your file system must allow long file names (as is likely except perhaps for some network-mounted systems).

Installation is *via* the installer '`R-2.7.0-win32.exe`'. Just double-click on the icon and follow the instructions. You can uninstall R from the Control Panel. (Note that you will probably (depending on the Windows language settings) 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 for more details.

3.1 Building from source

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 E [The Windows toolset], page 52 (and perhaps updates in `www.murdoch-sutherland.com/Rtools`) for details.

The '`Rtools.exe`' executable installer described in Appendix E [The Windows toolset], page 52 also includes some additions 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.

To avoid warnings you may want to set the environment variable `CYGWIN` to '`nodosfilewarning`'.

3.1.2 Getting the source files

You need to collect the following sets of files:

- Get the R source code '`R-2.7.0.tar.gz`' from CRAN. Open a command window (or another shell) at directory *R_HOME*, and run

  ```
  tar zxvf R-2.7.0.tar.gz
  ```

  to create the source tree in *R_HOME*. **Beware**: do use `tar` to extract the sources rather than tools such as WinZip that do not understand about symbolic links.

  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 using

  ```
  make rsync-recommended
  ```

- Optionally, you can install a version of ATLAS (`math-atlas.sourceforge.net`) tuned to your system for fast linear algebra routines. Pre-built '`Rblas.dll`' for various CPUs are available in the '`windows/contrib/ATLAS`' area on CRAN. If you are building R from source, there are macros `USE_ATLAS` and `ATLAS_PATH` in the file '`MkRules`'. Set `USE_ATLAS = YES` and `ATLAS_PATH` to where the ATLAS libraries are located. You will need to make the libraries yourself[1]: none of the binaries we have seen are compiled for the correct compiler. Since R has its own '`xerbla`' it is necessary to delete that in ATLAS by

[1] We do this using the Cygwin compilers, often with some difficulty.

```
ar d /path/to/libf77blas.a xerbla.o
```

There used to be support for AMD's AMD Core Math Library (ACML) and Kazushige Goto's BLAS, but neither is currently available for use with the current compilers used.

The following additional items are normally installed by '`Rtools.exe`'. If instead you choose to do a completely manual build (or a cross-build), you will also need

- Get '`iconv.dll`' from http://www.stats.ox.ac.uk/pub/Rtools/iconv.dll and put it in '*R_HOME*`/src/gnuwin32/unicode`'.
- The Tcl/Tk support files are in a zip file at http://www.stats.ox.ac.uk/pub/Rtools/: unzip this in *R_HOME*, and it will add directories '*R_HOME*`/Tcl`', '*R_HOME*`/Tcl/bin`', etc.
- You need `libpng` and `jpeg` sources (available, e.g., from www.libpng.org, ftp://ftp.uu.net/graphics/[png,jpeg], http://www.libtiff.org. You will need files '`libpng-1.2.18.tar.gz`', '`jpegsrc.v6b.tar.gz`', '`tiff-3.8.0.tar.gz`' or later.

  Working in the directory '*R_HOME* `/src/gnuwin32/bitmap`', install the `libpng` and `jpeg` sources in sub-directories. The `libpng` sub-directory must be named '`libpng`' (as required by the `libpng` documentation). The `jpeg` sub-directory for version 6b is named '`jpeg-6b`'; if you use a different version, edit '`Makefile`' and change the definition of `JPEGDIR`.

  Example:

```
> tar xzvf libpng-1.2.20.tar.gz
> mv libpng-1.2.20 libpng
> tar xzvf jpegsrc.v6b.tar.gz
> tar xzvf tiff-3.8.2.tar.gz
> mv tiff-3.8.2/libtiff .
> rm -rf libtiff-3.8.2
```

3.1.3 Building the core files

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`'. Edit '`MkRules`' to set the appropriate paths as needed and to set the type(s) of help that you want built. **Beware**: '`MkRules`' contains tabs and some editors (e.g., WinEdt) silently remove them. Then run

```
make all recommended
```

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

Notes:

- The file '`bin/Rchtml.dll`' is only built if CHM help is specified in '`MkRules`'. Its source is in the help directory, and you need the HTML Help Workshop files to build it. You can just copy this from a binary distribution.
- 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.
- By default Doug Lea's `malloc` in the file '*R_HOME*`/src/gnuwin32/malloc.c`' is used for R's internal memory allocations. You can opt out of this by commenting the line `LEA_MALLOC=YES` in '`MkRules`', in which case the `malloc` in '`msvcrt.dll`' is used. This does work but imposes a considerable performance penalty.
- You can run a parallel make by e.g.

```
make -j2 all
make recommended
```

  but this is only likely to be worthwhile on a dual-processor/dual-core machine with ample (at least 384Mb) of memory. (On a dual AthlonMP it reduced the build time by about 30%.) Note that this may sometimes stop and have to be restarted.

3.1.4 Building the bitmap files

The file '*R_HOME*/bin/Rbitmap.dll' is not built automatically.

Running `make` in '*R_HOME*/src/gnuwin32/bitmap' or `make bitmapdll` in '*R_HOME*/src/gnuwin32' should build 'Rbitmap.dll' and install it in '*R_HOME*/bin'.

3.1.5 Checking the build

You can test a build by running `make check`. You may need to set `TMPDIR` to the absolute path to a suitable temporary directory: the default is 'c:/TEMP'. (Use forward slashes and do not use a path including spaces.)

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 `check-devel` and `check-recommended`.

3.1.6 Building the manuals

The PDF manuals can be made by

```
make manuals
```

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

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

To make DVI versions of the manuals use

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

(all assuming you have `tex` and `latex` installed and in your path).

See the Section 2.2 [Making the manuals], page 4 section in the Unix section for setting options such as the paper size.

3.1.7 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, as well as the recommended packages and Inno Setup (see Section E.4 [The Inno Setup installer], page 53).

Once everything is set up

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

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

`Rpwd.exe` (a utility needed in the build)
`rbuild` (the executables, the FAQ docs etc.)
`rpackage` (the base packages)
`htmldocs` (the HTML documentation)
`bitmapdll` (the bitmap support files)
`recommended` (the recommended packages)
`vignettes` (the vignettes in package **grid**:
only needed if building from svn checkout)

`manuals` (the PDF manuals)
`rinstaller` (the install program)
`crandir` (the CRAN distribution directory)

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

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 a full installation of R from the standard installer (that is, select '`Full Installation`' from the '`Select Components`' screen), then add packages and make other customizations to that installation. Then in '`src/gnuwin32/installer`' run

```
make myR IMAGEDIR=rootdir
```

where '`rootdir`' is the path to the root of the customized installation (forward slashes and no spaces, please). This creates an executable with the standard name, '`R-2.7.0-win32.exe`', so please rename it to indicate that it is customized.

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.

3.1.8 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 available from http://wix.sourceforge.net/: we tested version 2.0. (This needs the .NET 1.1 framework installed: it ran on a vanilla Windows XP SP2 machine. Probably Windows 2000 or later is required by some of the features used.) Once WiX is installed, set the path to its home directory in '`MkRules`'.

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. Then

```
cd installer
make msi
```

which will results in a file of about 40Mb with a name like '`R-2.6.0-win32.msi`'. This can be double-clicked to be installed, but those who need it will know what to do with it.

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

3.1.9 Cross-building on Linux

It is possible to cross-build R or packages on (at least) '`ix86`' and '`x86_64`' Linux, and the '`ix86`' cross-compilers have also been used successfully on '`x86_64`' Linux.

The preferred build environment is to use `gcc` 4.2.1: this can easily be built from the sources as a cross-compiler, but the MinGW-specific patches are not yet stable (and not needed to build R).

You will need suitable cross-compilers installed and in your path. We do not at present distribute suitable cross-compilers.

You will need Perl, `zip` and `unzip` installed and (to make the manuals) `makeinfo` version 4.7 or later (part of GNU `texinfo`) as well as '`texinfo.tex`'.

You also need the R source ('`R-2.7.0.tar.gz`'), the Tcl/Tk support files and '`iconv.dll`' (see above).

Then: untar '`R-2.7.0.tar.gz`' somewhere, unpack '`R_Tcl.zip`' at the top level and put '`iconv.dll`' in '`src/gnuwin32/unicode`', then

```
cd /somewhere/R-2.7.0/src/gnuwin32
```

Edit '`MkRules`' to set `BUILD=CROSS` and the appropriate paths (including `HEADER` if needed).

Edit '`MkRules`' to set the type(s) of help that you want built. (You will not be able to cross-build '`.chm`' files, so `WINHELP` is automatically set to `NO`.)

You also need a working copy of *this version* of R on Linux: uncomment and set `R_EXE` in '`MkRules`' to point to it.

Then run `make` (and parallel make works reliably, unlike on Windows).

Packages can be made in the same way as natively: see Section 6.3.4 [Customizing package compilation under Windows], page 17, via the '`Makefiles`' but not via '`R CMD INSTALL`'. So care is needed where packages have dependencies: Linux versions of the dependencies must be installed in a library in the search path. So for example to cross-build the **MCMCpack** package we used

```
# MCMCpack depends on coda, so point to the library containing it
export R_LIBS=/R/library
make PKGDIR=/mysources pkg-MCMCpack
make PKGDIR=/mysources lazyload-MCMCpack
cd ../../library
zip -r9X /dest/MCMCpack_0.7-4.zip MCMCpack
```

Even so, packages which depend on others that need to run compiled code to load may not work (**methods** is a special exception).

To distribute a cross-build (or just to transfer it to a Windows machine for testing) use

```
make all recommended manuals
cd installer
make imagedir
zip -r9X R-2.7.0.zip R-2.7.0  # or something similar
```

Note that '`.chm`' help files (the default for a vanilla binary installation) will not be made when cross-building.

Also based on this facility is '`Makefile-rcb`' by J. Yan and A. J. Rossini. For details, see the '`Makefile-rcb`' file itself, or `http://cran.r-project.org/doc/contrib/cross-build.pdf`.

4 Installing R under Mac OS X

The '`bin/macosx`' directory of a CRAN site contains binaries for Mac OS X for a base distribution and a large number of add-on packages from CRAN to run on Mac OS X version 10.4.4 or higher.

The simplest way is to use '`R-2.7.0.dmg`'. Just double-click on the icon and the disk image file will be mounted. Read the '`ReadMe.txt`' inside the disk image and follow the instructions.

See the R for Mac OS X FAQ for more details.

4.1 Building from source on Mac OS X

If you want to build this port from the sources, you can read the above mentioned R for Mac OS X FAQ for full details. You will need to collect and install some tools as explained in the document. Then you have to expand the R sources and configure R appropriately, for example

```
tar zxvf R-2.7.0.tar.gz
cd R-2.7.0
./configure --with-blas='-framework vecLib' --with-lapack \
  --with-aqua --enable-R-framework
make
```

and then sit back and wait. The first two options are the default (and strongly recommended), and with some toolsets have been essential. The second line of options is also default on Mac OS X, but needed only if you want to build R for use with `R.app` Console, and imply '`--enable-R-shlib`' to build R as a shared/dynamic library.

These options configure R to be built and installed as a framework called '`R.framework`'. The default path for '`R.framework`' is '`/Library/Frameworks`' but this can be changed at configure time specifying the flag '`--enable-R-framework[=`*`DIR`*`]`' or at install time as

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

the '`R.framework`' has not to be specified in the path.

Note that building the '`R.app`' GUI console is a separate project: see the FAQ for details.

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*.

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 34). As from R 2.4.0 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.

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 running 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 three or more. 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.

As from R 2.5.0 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()`'.

As from R 2.5.0 users can have one or more libraries, normally specified by the environment variable `R_LIBS_USER`. This has a default value (use '`Sys.getenv("R_LIBS_USER")`' within an R session to see what it is), but only is 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 requires that compilers and tools (including Perl 5.8.0 or later) 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 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 looked once that files has been read, so users cannot easily change it.

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.

To install packages from source in Unix use

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

The part '`-l /path/to/library`' can be omitted, in which case the first library in `R_LIBS` is used if set, otherwise the main library '*`R_HOME`*`/library`' is used. (`R_LIBS` is looked for in the environment: note that '`.Renviron`' is not read by `R CMD`.) Ensure that the environment variable `TMPDIR` is either unset (and '`/tmp`' exists and can be written in and executed from) or points to a valid temporary directory.

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 set the option `CRAN` to 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 user 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 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 by setting argument `repos` to `NULL`.

`install.packages` can look in several repositories, specified as a character vector by the argument `repos`: these can include a CRAN mirror, Bioconductor, Omegahat, local archives, local files, ...).

6.3.1 Windows

What `install.packages` does by default is different on Unix 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) at the list of *binary* versions of packages available for your version of R and downloads the latest versions (if any), although optionally it will also download and install a source package by setting the `type` argument.

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

`R CMD INSTALL` works in Windows to install source packages if you have the source-code package files (option "Source Package Installation Files" in the installer) and toolset (see Appendix E [The Windows toolset], page 52) installed. Installation of binary packages must be done by `install.packages`. `R CMD INSTALL --help` will tell you the current options under Windows (which differ from those on a Unix-alike): in particular there is a choice of the types of documentation to be installed.

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 win-builder.r-project.org.

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

6.3.2 Mac OS X

On Mac OS X `install.packages` works as it does on other Unix-like systems, but there is an additional type `mac.binary` that can be passed to `install.packages` in order to download and install binary packages from a suitable repository, and is the default if running from the GUI console. These Macintosh binary package files have the extension '`tgz`'. The R GUI provides for installation of either binary or source packages, from CRAN or local files.

6.3.3 Customizing package compilation under Unix

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`', 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`.

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.

6.3.4 Customizing package compilation under Windows

This section describes ways to customize package compilation using the standard C, C++ and FORTRAN compilers and tools. For instructions on using non-standard tools, see the '`README.packages`' file.

The Makefiles can be customized: in particular the name of the DLL can be set (for example we once needed `integrate-DLLNM=adapt`), the compile flags can be set (see the examples in '`MakeDll`') and the types of help (if any) to be generated can be chosen (variables `HELP`, `HELPTYPES` and `WINHELP`). The simplest way to customize the compilation steps is to set variables in a file '`src/Makevars.win`', which will automatically be included by '`MakeDLL`'. For example, for RODBC '`src/Makevars.win`' could include the line

```
DLLLIBS+=-lodbc32
```

or, equivalently,

```
RODBC-DLLLIBS=-lodbc32
```

but in fact contains the single line

```
PKG_LIBS=-lodbc32
```

If you have a file '`src/Makefile.win`', that will be used as the makefile for source compilation in place of our makefile and '`MakeDll`' and '`src/Makevars.win`' will be ignored.

Package-specific compilation flags can be overridden or added to using the personal file '`$HOME/.R/Makevars.win`', or if that does not exist, '`$HOME/.R/Makevars`'. (See the '`rw-FAQ`' for the meaning of `$HOME`.) For the record, the order of precedence is (last wins)

- '`MakeDll`' and '`MkRules`'
- '`src/Makevars.win`' if it exists, otherwise '`src/Makevars`'
- '`$HOME/.R/Makevars.win`' if it exists, otherwise '`$HOME/.R/Makevars`'.
- '`src/Makefile.win`' if present causes all but the last of the above to be ignored.

Beware: references to variables in '`R.dll`' are converted to the right form by using the header files. You must include them.

For additional control, '*`R_HOME`*`/src/gnuwin32/Makefile`' contains additional make targets corresponding to various options to `R CMD INSTALL`. These assume that package `foo`'s source code has been installed in directory '*`R_HOME`*`/src/library/foo`'. Then `make pkg-foo` is similar to `R CMD INSTALL foo` (but the latter would require '*`R_HOME`*`/src/library`' to be the current directory). Other targets are

- `ziponly-foo`, to use zip to compress the help files after building the package.
- `ziphelp-foo` to both compress the help files and to keep the originals.
- `zipdata-foo` to compress the data files. This is recommended if you have either many small data files (as in package **Devore5**) or a few large data files.
- `pkgcheck-foo` to check the package (like `R CMD check foo`).

Using this approach allows variables to be set during the build, e.g.

```
make PKGDIR=/mysources RLIB=/R/library pkg-foo
```

Some variables that may be used include:

- `DEBUG=T` to compile with debugging information for `gdb`.
- `PKG_CFLAGS=` to specify options to the C compiler.
- `PKG_CPPFLAGS=` to specify options to the preprocessor.
- `PKG_CXXFLAGS=` to specify options to the C++ compiler.
- `PKG_FFLAGS=` to specify options to the FORTRAN 77 compiler.
- `PKG_FCFLAGS=` to specify options to the Fortran 95 compiler (if specified).
- `PKG_LIBS=` to specify options to the linking step making the DLL.
- `PKGDIR=/path/to/source` to specify the path to the package source files.
- `RLIB=/path/to/library` to specify the path to the library where the package should be installed.

For a complete list of variables, see the '`M*`' files in '*`R_HOME`*`/src/gnuwin32`'. The `PKG_*` flags are those typically included in '`Makevars`' files.

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. Set the `repos` argument as in the previous section. The `update.packages()` 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.

6.5 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, in most installations one can just remove the package directory from the library.

Note: only `remove.packages` can remove package *bundles*.

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.omegahat.org/R`: this must be an URL scheme that `download.packages` supports (which also includes '`ftp://`' and '`file://`'). 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.
- `"win.binary"`: located at '`bin/windows/contrib/x.y`' for R versions *x.y.z* and containing '`.zip`' files.
- `"mac.binary"`: located at '`bin/macosx/universal/contrib/x.y`' for R versions *x.y.z* and containing '`.tgz`' files. If the repository contains only packages for a specific architecture, the package distribution type can be set to `"mac.binary.xxx"` where *xxx* specifies the architecture, replacing `universal` by *xxx* in the path above.

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 (provided there are no bundles), 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.

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

As from R 2.7.0 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
```

The `write_PACKAGES` utility in package **tools** can help prepare the '`PACKAGES`' and '`PACKAGES.gz`' files.

7 Internationalization and Localization

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

R long worked in the ISO Latin-1 8-bit character set and so covered English and most Western European languages (if not necessarily their currency symbols). Since R 2.1.0 it has supported (where possible) multi-byte character sets such as UTF-8 and others used in Chinese, Japanese and Korean.

Full internationalization of the character sets is enabled unless R is built under Unix-alikes using `configure` option '`--disable-mbcs`' provided the OS can support it: see Appendix B [Configuration on Unix], page 33. Under Windows, support for Windows' own MBCS is always included.

All builds of R support all single-byte 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. Fully internationalized builds can also handle most multi-byte locales, in which a single character is represented by one, two or more consecutive bytes: examples of such locales are those using UTF-8 (becoming standard under Linux but non-existent under Windows) and those for Chinese, Japanese and Korean.

The other aspect of the internationalization is support of 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.
- A 'territory', used mainly to specify the currency. These are generally specified by an upper-case two-character abbreviation following ISO 3166. Sometimes the combination of language and territory is used to specify the encoding, for example to distinguish between traditional and simplified Chinese.
- A charset encoding, which determines both how a byte stream should be divided into characters, and which characters the subsequences of bytes represent.
- Optionally, a modifier, for example to indicate that Austria is to be considered pre- or post-Euro.

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, and most OSes have only one charset each for many languages. Note too the remark above about Chinese.

7.1.1 Locales under Linux

Modern Linux uses the XPG locale specifications which have the form '`en_GB`', '`en_GB.utf8`', '`aa_ER.utf8@saaho`', '`de_AT.iso885915@euro`', the components being in the order listed above. (See `man locale` and `locale -a` for more details.) Similar schemes (but often in different cases) are used by most Unix-alikes.

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 msdn.microsoft.com/library/default.asp?url=/library/en-us/vccore98/html/_crt_language_and_country_strings.asp.

7.1.3 Locales under Mac OS X

Mac OS X supports locales in its own particular way, but the R GUI tries to make this easier for users. See 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`'.

Internally Mac OS X uses a form similar to Linux but without specifying the encoding (which is `UTF-8`). It is based on ICU locales (http://icu.sourceforge.net/userguide/locale.html) and not POSIX ones.

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 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*[1] the language you are changing to can be output in the current character set.

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 basic C-level error messages.
- 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.

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 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. 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.

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

Translations in the right language but the wrong charset be made use of by on-the-fly re-encoding (on almost all systems). 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 English/American error messages with pairs of 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 is used.

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

8 Choosing between 32- and 64-bit builds

Many current CPUs have both 32- and 64-bit sets of instructions: this has long been true for UltraSparc and more recently for MIPS, PPC and '`x86_64`' (AMD Opteron and Athlon64, Intel Xeon and Pentium/'Core' supporting EM64T). Many 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) 64-bit is.

All current versions of R use 32-bit integers and 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[3] is limited to 4GB, and on most OSes to 3GB (or even 2GB). The limits for 64-bit processes are much larger.

  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 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).
- 32-bit OSes by default limit file sizes to 2GB. This can often be worked around: and `configure` selects suitable defines if this is possible. (We have also largely worked around that limit on Windows.) 64-bit builds have much larger limits.
- Because the pointers are larger, R's basic structure (the cons cell) is larger (normally twice the size). This means that R objects take more space and (usually) more time to manipulate. So 64-bit versions of R will typically run slower than 32-bit versions. (On Sparc Solaris the difference was 15-20%, on Linux on Opteron around 10%. The pattern is not universal on Intel Core 2 Duo the 64-bit version is around 10% faster on both Linux and Mac OS X.)

So, for speed you may want to use a 32-bit build, but to handle large datasets (and perhaps large files) a 64-bit build. You can build both and install them in the same place: See Section 2.5 [Sub-architectures], page 7.

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). On all versions of R, the maximum length (number of elements) of a vector is $2^{31} - 1$, about 2 billion, and on 64-bit systems the size of a block of memory allocated is limited to $2^{34} - 1$ bytes (8GB). It is anticipated these will be raised eventually but routine use of 8GB objects is (in 2005) several years off.

8.1 Windows

Currently the Windows build of R is a 32-bit executable. This runs happily on Windows 64 on AMD64 and EM64T, but is limited to (we are told) a 2GB address space. It will not be possible to provide a native version for Windows 64 until suitable compilers are available, and currently (mid-2007) that is not imminent.[4]

[1] also known as IEC 559 and IEEE 754

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

[3] until recently this limit applied to all processes, not just to one process

[4] It is likely that commercial 64-bit compilers could be used, but those we have looked at do not work in the same way as the MinGW compilers so extensive changes to the build system would be needed. Also, end users would need to have the same compilers to install extension packages. There is an experimental version of MinGW for 64-bit Windows, but it is not yet good enough to build a working R.

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 Unix 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 `"2.6.0"`.

There is full access to R's handling of `NaN`s, `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 will 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 facilties 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

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

Then

```
make
```

will make standalone libraries '`libRmath.a`' and '`libRmath.so`'. '`make static`' and `make shared` will create just one of them.

NB: certain compilers are unable to do compile-time IEEE-754 arithmetic and so cannot compile '`mlutils.c`' and several other files. The known example is earlier versions of Sun's `cc` (e.g. Forte 6 and 7): the Sun Studio 11 suite does work.

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

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

```
#define MATHLIB_STANDALONE
#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.

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 almost all the tools to make R and then run

```
(cd ../../include; make -f Makefile.win config.h Rconfig.h Rmath.h)
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, 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,dynamic
-Wl,-Bdynamic -lRmath
```

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

If you make use of dynamic linking you should use

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

to ensure that the constant like `NA_REAL` are linked correctly. (Auto-import will probably work, but it is better to be sure.)

Appendix A Essential and useful other programs under Unix

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 deb) 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

You need a means of compiling C and FORTRAN 77 (see Section B.5 [Using FORTRAN], page 35). Some add-on packages also need a C++ compiler. Your C compiler should be IEC 60059[1], POSIX 1003.1 and C99-compliant if at all possible. R tries to choose suitable flags for the C compilers it knows about, but you may have to set `CC` or `CFLAGS` suitably. For recent versions of `gcc` with `glibc` this means including '`-std=gnu99`'[2]. If the compiler is detected as `gcc`, `-std=gnu99` will be appended to `CC` unless it conflicts with a setting of `CFLAGS`.

Unless you do not want to view graphs on-screen you need '`X11`' installed, including its headers and client libraries. (On Fedora Core 3 and SuSE 9.x Linux this meant the '`xorg-x11-devel`' and '`xorg-x11-libs`' RPMs. For Fedora Core 5 and 6 it means (at least) '`libX11`', '`libX11-devel`', '`libXt`' and '`libXt-devel`'. On Debian 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 depends on the `readline` library available from any GNU mirror: 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).

The use of multi-byte characters, conversion between encodings (including for translated messages) and the R `iconv` function depend on having the system `iconv` function: this is part of recent versions of `glibc` and many Unixes. You can also install GNU `libiconv` (which is not the same as that in `glibc`), possibly as a plug-in replacement: see www.gnu.org/software/libiconv. Note that the R usage requires `iconv` to be able to translate between `"latin1"` and `"UTF-8"`, to recognize `""` as the current encoding and to translate to and from the Unicode wide-character formats `"UCS-[24][BL]E"` – this is not true of most commercial Unixes. This is regarded as essential from R 2.5.0: if you do not have it will need to configure with '`--without-iconv`' (or equivalent), and `make check` (and other checks) are likely to fail.

Perl version 5.8.0 or later, available via www.perl.com/CPAN is essential.

You will not be able to build most of the manuals unless you have `makeinfo` version 4.7 or later installed, and if not some of the HTML manuals will be linked to CRAN. (Version 4.6 is known to create incorrect HTML files.) To make DVI or PDF versions of the manuals you will also need '`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` (part of the GNU **texinfo** distribution).

The DVI and PDF documentation and building vignettes needs `tex` and `latex`, or `pdftex` and `pdflatex`.

If you want to build from the R Subversion repository you need both `makeinfo` and `pdflatex`.

[1] also known as IEEE 754

[2] '`-std=c99`' excludes POSIX functionality, but '`config.h`' will turn on all GNU extensions include the POSIX functuonality.

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 `X11`, `jpeg()`, `png()` and `tiff()` uses the `cairo` and (optionally) `Pango` libraries. Cairo version 1.0 or later is required[3], and some features reguire 1.2 or later (and may not work before 1.4). 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, and are likely to fail if `cairo` was built statically (unusual). Most systems with `Gtk+` 2.8 or later installed will have suitable libraries, but some (e.g. Solaris 10) may need `cairo` added separately. Mac OS X comes with none of these libraries, but `cairo` support has been added to the binary distribution.

For the best font experience with these devices you need suitable fonts installed: Linux users will want the `urw-fonts` package. Another useful set of fonts is the 'liberation' truetype fonts available at https://www.redhat.com/promo/fonts/, 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 (http://en.wikipedia.org/wiki/Liberation_fonts).

The bitmapped graphics devices `jpeg()`, `png()` and `tiff()` need the appropriate headers and libraries installed: `jpeg` (version 6b or later) or `libpng` (version 1.2.3 or later) and `zlib` (version 1.1.3 or later) or `libtiff` (any recent version – 3.8.2 was tested) respectively.

The `bitmap` and `dev2bitmap` devices and also `embedFonts()` use ghostscript (www.cs.wisc.edu/~ghost).

If you have them installed (including the appropriate headers and of recent enough versions), `zlib`, `libbz2` and PCRE will be used if specified by '`--with-system-zlib`', '`--with-system-bzlib`' or '`--with-system-pcre`': otherwise versions in the R sources will be compiled in. As the latter suffice and are tested with R you should not need to change this. In particular, the version of `zlib` 1.2.3 in the R sources has enhancements to work with large file systems on 32-bit platforms.

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 Fedora Core 5/6 require the '`libXmu`' and '`libXmu-devel`' RPMs.

A.2.1 Tcl/Tk

The **tcltk** package needs Tcl/Tk >= 8.3 installed: the sources are available at 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`'

[3] We recommend that people with cairo 1.0.x, e.g. on Fedora Core 5, install 1.4.x from the sources which works well

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, setting the paths to the correct config files may be necessary to avoid confusion between them.

Versions of Tcl/Tk from 8.3 to 8.5.0 have been used successfully.

A.2.2 Java support

`configure` looks for Java support on the host system, and if it finds it sets some settings which are useful for Java-using packages. `JAVA_HOME` can be set during the `configure` run to point to a specific JRE/JDK.

Principal amongst these 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 Sun Java is

```
/usr/java/jdk1.5.0_06/jre/lib/amd64/server:/usr/java/jdk1.5.0_06/jre/lib/amd64
```

Note that this unfortunately depends on the exact version of the JRE/JDK installed, and so will need updating if the Java installation is updated. This can be done by running `R CMD javareconf`. The script re-runs Java detection in a manner similar to that of the `configure` script and updates settings in both '`Makeconf`' and '`R_HOME/etc/ldpaths`'. See `R CMD javareconf --help` for details.

Another alternative of overriding those setting is to set `R_JAVA_LD_LIBRARY_PATH` (e.g. in '`~/.Renviron`'), or use '`/etc/ld.so.conf`' to specify the Java runtime library paths to the system. Other settings are recorded in '`etc/Makeconf`' (or a sub-architecture-specific version), e.g.

```
JAVA = /usr/bin/java
JAVAC = /usr/bin/javac
JAVA_HOME = /usr/java/jdk1.5.0_06/jre
JAVA_LD_LIBRARY_PATH = $(JAVA_HOME)/lib/amd64/server:$(JAVA_HOME)/lib/amd64:\
$(JAVA_HOME)/../lib/amd64:/usr/local/lib64
JAVA_LIBS = -L$(JAVA_HOME)/lib/amd64/server -L$(JAVA_HOME)/lib/amd64
            -L$(JAVA_HOME)/../lib/amd64 -L/usr/local/lib64 -ljvm
```

where '`JAVA_LIBS`' contains flags necessary to link JNI programs. Some of the above variables can be queried using `R CMD config`.

A.3 Linear algebra

A.3.1 BLAS

The linear algebra routines in R can make use of enhanced BLAS (Basic Linear Algebra Subprograms, www.netlib.org/blas/faq.html) routines. However, as from R 2.4.0[4] 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"
```

[4] Earlier versions of R searched for external BLASes, but this caused frequent difficulties.

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[5], 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 (`libsunperf` on Sun Sparc[6], `libessl` on IBM, `vecLib` on Mac OS X). The correct incantation for these is usually 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.

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** will need to be re-installed.

A.3.1.1 ATLAS

ATLAS (`math-atlas.sourceforge.net`) is a "tuned" BLAS that runs on a wide range of Unix-alike platforms. Unfortunately it is usually built as a static library that on some platforms cannot be used with shared libraries 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 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 systems with multiple processors it is possible to use a multi-threaded version of ATLAS, by specifying

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

Consult its file '`INSTALL.txt`' for how to build ATLAS with position-independent code (at least on version 3.8.0): this also describes how to build ATLAS as a shared library.

ATLAS can also be used on Windows: see see Section 3.1.2 [Getting the source files], page 8 when building from source, and R Windows FAQ for adding pre-compiled support to binary versions.

A.3.1.2 ACML

For '`x86_64`' processors under Linux and Solaris 10 there is the AMD Core Math Library (ACML) `www.amd.com/acml`. For the `gcc` version we could use

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

if the appropriate library directory (such as '`/opt/acml4.0.1/gfortran64/lib`') is in the `LD_LIBRRARY_PATH`. For other compilers, see the ACML documentation. There is a multithreaded Linux version of ACML available for `gfortran` which needs `gcc` >= 4.2.0 (or some RedHat versions of 4.1.x). To make use of this you will need something like

[5] unless FORTRAN double complex is not supported on the platform

[6] Using the Sun Studio `cc` and `f95` compilers

```
--with-blas="-L/opt/acml4.0.1/gfortran64_mp/lib -lacml_mp"
```

See see Section A.3.1.5 [Shared BLAS], page 30 for an alternative (and in many ways preferable) way to use ACML.

Earlier versions of ACML are available for 32-bit Linux systems and for those using `g77`.

A.3.1.3 Goto BLAS

Kazushige Goto has written another tuned BLAS which is available for several processors and OSes.

This has been made available in several formats, but is currently available only as source code. For *academic use only* (after registering) it can be obtained via `www.tacc.utexas.edu/resources/software/software.php`. Once this is built and installed, it can be used by configuring with

```
--with-blas="-lgoto"
```

See see Section A.3.1.5 [Shared BLAS], page 30 for an alternative (and in many ways preferable) way to use recent versions of the Goto BLAS.

Note that currently (Nov 2007) a multi-threaded Goto BLAS will be built by default if and only if the building is on a multi-processor system (counting multiple cores and hyperthreading), and at run time the default number of threads is the number of CPUs detected.

It has been reported that on some RedHat-based Linux systems it is necessary to set `GOTO_NUM_THREADS=1` or `OMP_NUM_THREADS=1` (to disable multiple threads) in the environment when using a multi-threaded Goto BLAS, but ours run happily with multiple threads.

A.3.1.4 Intel MKL

For Intel processors under Linux, Intel's Math Kernel Library (`www.intel.com/software/products/mkl`) version 9 can be used by

```
--with-blas="-lmkl -lguide -lpthread"
```

This is multi-threaded, but the number of threads defaults to 1 (and can be increased by setting `OMP_NUM_THREADS`). (Thanks to Andy Liaw for the information.)

For version 10 on '`x86_64`', Ei-Ji Nakama used GCC compilers and

```
MKL_LIB_PATH=/opt/intel/mkl/10.0.1.014/lib/em64t
MKL="   -L${MKL_LIB_PATH}                                  \
        -Wl,--start-group                                  \
                ${MKL_LIB_PATH}/libmkl_gf_lp64.a           \
                ${MKL_LIB_PATH}/libmkl_gnu_thread.a        \
                ${MKL_LIB_PATH}/libmkl_core.a              \
        -Wl,--end-group                                    \
        -liomp5 -lguide -lpthread -lgomp"

./configure --with-lapack="$MKL" --with-blas="$MKL"
```

See see Section A.3.1.5 [Shared BLAS], page 30 for an alternative (and in many ways preferable) way to use MKL.

A.3.1.5 Shared BLAS

Note that 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 is 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 (e.g., '`-O3`') of compiler optimization are used.
- It is easy to change the BLAS without making 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 '`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/acml4.0.1/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 recent versions of the Goto BLAS and for 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 `libsunperf` on Solaris, `vecLib` on Mac OS X and ACML on '`ix86`'/'`x86_64`' Linux and Solaris). 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/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`'.

Intel's Math Kernel Library suppies a full LAPACK which can be used via

```
--with-lapack="-L/path/to/libs -lmkl_lapack64"
```

or '`-lmkl_lapack32`': library `mkl_lapack` is static and not PIC. However, the version 8.0.1.006 we tested failed a regression test in the complex LAPACK.

If you do use '`--with-lapack`', be aware of potential problems with bugs in the LAPACK 3.0 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
```

(seen with the Debian `-llapack` which was current in late 2002, Fedora Core 4 Extras `-llapack` in September 2005 and 64-bit `libsunperf` in Forte 7). Other potential problems are incomplete versions of the libraries: for example `libsunperf` from Sun Forte 6.x was missing the entry point for `DLANGE` and `vecLib` has omitted the BLAS routine `LSAME`. For problems compiling LAPACK using recent versions of `gcc` on '`ix86`' Linux, see Section C.11 [New platforms], page 48: these problems have surfaced in Fedora Core 3's distribution, for example.

Please **do** bear in mind that using '`--with-lapack`' is 'definitely **not** recommended': it is provided **only** because it is necessary on some platforms.

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 so `libsunperf` can be used.

On some systems it is necessary that an external BLAS/LAPACK was built with the same FORTRAN compiler used to build R: known problems are with R built with `gfortran`, see Section B.5.1 [Using gfortran], page 36.

Appendix B Configuration on Unix

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

'`--enable-BLAS-shlib`'
: build the BLAS as a shared/dynamic library [no]

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.

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

B.2 Internationalization support

R can be compiled with support for multi-byte character sets (MBCS), in particular for UTF-8 locales (which are usually identified by suffix '`.utf8`' or '`UTF-8`', something like '`en_GB.utf8`'. UTF-8 is an encoding of Unicode and in principle covers all human languages simultaneously: however, a given system may not have fonts capable of displaying more than a few of these languages.

Support for MBCS is selected if possible at `configure` time (unless disabled with '`--disable-mbcs`'). This will check for a large number of features, notably support for the C99/UNIX98 wide character functions, for UTF-8 or MBCS support in X11 and for `iconv` with a rich enough functionality. If enough of these are found, `MBCS` will be listed as one of the

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

"Additional capabilities". Then if R is started in a UTF-8 locale it assumes that the terminal will supply and display UTF-8-encoded characters[2]. If run in a single-byte locale, R behaves almost exactly as if it was configured with '`--disable-mbcs`'.

A version of R built with MBCS support can also be run in other multi-byte locales, for example those using the EUC-JP, EUC-KR and EUC-TW encodings on Unix-alikes and the code pages for Chinese, Japanese and Korean on Windows.

Translation of messages is supported via GNU `gettext` unless disabled by the configure option '`--disable-nls`' or the underlying OS has insufficiently standard C functions to support it. 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 all the variables you might want to set) 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). To force a single file to be read, set the environment variable `CONFIG_SITE` to the location of the file.

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 and 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 papersize when making DVI and 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 browser

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

[2] You may have to set this with `luit`, but it should be the default in a window manager session started in UTF-8.

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. 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 on recent systems[3] ''`-Bdirect,--hash-style=both,-Wl,-O1`'' is recommended at http://lwn.net/Articles/192624/

B.3.4 Making manuals

The default settings for making the manuals are controlled by `R_RD4PDF`, `R_RD4DVI` and `R_PAPERSIZE`.

B.4 Using make

To compile R, you will most likely find it easiest to use GNU `make`. On Solaris 2.6/7/8 in particular, you need a version of GNU `make` different from 3.77; 3.79.1 and later work fine, as does the Sun `make`. The native `make` is reported to fail on SGI Irix 6.5 and Alpha/OSF1 (aka Tru64).

To build in a separate directory you need a `make` that uses the `VPATH` variable, for example GNU `make`, or Sun `make` on Solaris 2.7 or later.

`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
```

Note the comment above about using a parallel `make`: this works for building R but not for installing nor checking.

B.5 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)[4], 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-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.

[3] e.g. Fedora Core 6

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

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[5], but you may need to avoid compiler flags[6] asserting FORTRAN 77 compliance.

For performance reasons[7] you may want to choose a FORTRAN 90/95 compiler.

It is possible to use `f2c`, the FORTRAN-to-C converter (`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 may 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 not on a 64-bit platform. If your compiler is not `gcc` you will need to set `FPICFLAGS` appropriately.

B.5.1 Using gfortran

`gfortran` is the F95 compiler that is part of `gcc` 4.*x*.*y*. There were problems compiling R with the first release (`gcc` 4.0.0) and more with pre-releases, but these are resolved in later versions.

On Linux '`x86_64`' systems there is an incompatibility in the return conventions for double-complex functions between `gfortran` and `g77` which results in the final example in `example(eigen)` hanging or segfaulting under external BLASs built under `g77`. This should be detected by a `configure` test.

B.6 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 libraries

`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 libraries

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

[6] In particular, avoid `g77`'s '`-pedantic`', which gives confusing error messages.

[7] e.g., to use an optimized BLAS on Sun/Sparc

`MAIN_LDFLAGS`
: additional flags for the main link

`SHLIB_LDFLAGS`
: additional flags for linking the shared libraries

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

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

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

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

`FPICFLAGS`
: special flags for compiling Fortran 95 code to be turned into a shared library

`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 library. 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.

Appendix C Platform notes

This section provides some notes on building R on different Unix-like 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.

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 and in particular the fonts provided by the X server. So if you sometimes use R at a console and sometimes remotely from an X11 session running on a Windows machine, you may have to setup the fonts differently for the two usages.

When X11 was designed, most displays were around 75dpi, whereas today they are of the order of 100dpi or even higher. 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 Core 5's

```
xorg-x11-fonts-75dpi
xorg-x11-fonts-100dpi
xorg-x11-fonts-truetype
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 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-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/X11R6/lib/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.utf8`' but not when running in '`en_GB.utf8`' on the same machine (although on some systems many UTF-8 X11 locales are

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

aliased to '`en_US.utf8`' 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 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).)

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 for 64-bit CPUs such as that for AMD Opteron. 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.

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 Core 5 and Fedora 8 we used

```
CC="gcc -m32"
CXXFLAGS="-m32 -O2 -g"
FFLAGS="-m32 -O2 -g"
FCFLAGS="-m32 -O2 -g"
LDFLAGS="-L/usr/local/lib"
LIBnn=lib
```

Fedora Core 3 also needed

```
--x-libraries=/usr/X11R6/lib
```

and various '`i386`' '`-devel`' RPMs had to be added.

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.

R used to include the compiler flag '`-mieee-fp`', but it seems this was really an alias for the linker flag '`-lieee`'. Neither are needed for a modern Linux (e.g. using `glibc` 2.2/3/4) but could conceivably be needed on an older version. `glibc` 2.1 required '`-D__NO_MATH_INLINES`' to achieve IEC 60059-compliance for `exp`, and this is included in `R_XTRA_CFLAGS` if required.

Several Linux distributions have shipped unreleased versions of `gcc` 4.0.0 and its FORTRAN compiler `gfortran` (see the separate comments). Some versions of `gcc` 4 (such as that in Fedora Core 3) produce incorrect code. In our experiments `gcc` 3.4.x always produced faster and more reliable code. It seems that `gcc` 4.0.x cannot compile '`src/main/plot.c`' when building R as a shared library on '`ix86`' unless the optimization level is changed from the default '`-O2`' ('`-O3`' works, as does `gcc` 4.1.0 with the default settings).

It has been reported that using `gcc` 4.0.3 on '`ppc64`' needed the compiler flag '`-mminimal-toc`' to avoid errors when linking R as a shared library.

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 problems linking against '`libnmath.a`' and when linking R as a shared library.

On some[2] `glibc` systems, '`-fgnu89-inline`' needs to be added to '`CFLAGS`' when using the (currently unreleased) `gcc` 4.3.0, since '`wchar.h`' is not set up properly for C99-style inlines.

C.2.1 Intel compilers

Intel compilers have been used under '`ix86`' and '`x86_64`' Linux and R contains code to set the FPU options suitably. Brian Ripley tried version 9.0 of the compilers for '`ix86`' on Fedora Core 3 *via*

```
CC=icc
F77=ifort
CXX=icpc
ICC_LIBS=/opt/compilers/intel/cc/9.0/lib
IFC_LIBS=/opt/compilers/intel/fc/9.0/lib
LDFLAGS="-L$ICC_LIBS -L$IFC_LIBS -L/usr/local/lib"
SHLIB_CXXLD=icpc
```

and adding optimization flags failed: at least '`src/main/regex.c`' and '`src/modules/lapack/dlamc.f`' needed to be compiled without optimization. For '`x86_64`' on Fedora Core 5 he used

```
CC=icc
CFLAGS="-g -O3 -wd188 -ip"
F77=ifort
FLAGS="-g -O3"
CXX=icpc
CXXFLAGS="-g -O3"
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. R will add '`-mp`' in `R_XTRA_{C,F,CXX}FLAGS` to maintain correct IEC 60559 arithmetic. The flag '`-wd188`' suppresses a

[2] e.g. Fedora Core 5

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.

For some comments on building on an Itanium ('ia64') Linux system with gcc or the Intel compilers see www.nakama.ne.jp/memo/ia64_linux.

C.2.2 PGI compilers

Jennifer Lai used the Portland Group compilers on 'x86_64' to build pre-2.2.0. Updated versions of the settings she used are

```
PG_HOME=/usr/pgi/linux86-64/6.0
CC=pgcc
CFLAGS="-g -O2 -Kieee"
CPPFLAGS="-I$PG_HOME/include -I$PG_HOME/include/CC"
F77=pgf77
FFLAGS="-g -O2 -Kieee"
CXX=pgCC
CXXFLAGS="-g -O2 -Kieee"
FC=pgf95
FCFLAGS="-g -O2 -Kieee"
SHLIB_CXXLDFLAGS=-shared
SHLIB_LDFLAGS=-shared
LDFLAGS="-L$PG_HOME/libso -L/usr/lib64"
```

Note particularly the last, which is needed to ensure that a shared version of libc is found. The flag '-Kieee' ensures strict compliance to IEC60659. Also, http://www.amd.com/us-en/assets/content_type/DownloadableAssets/dwamd_PGI_nov603.pdf suggests that '-pc64' may be desirable.

C.2.3 SunPro compilers

Brian Ripley tested the SunPro Studio 12 compilers (http://developers.sun.com/sunstudio/index.jsp) on 'x86_64' Linux with

```
CC=cc
CFLAGS="-xO5 -xc99 -xlibmil -nofstore"
CPICFLAGS=-Kpic
F77=f95
FFLAGS="-O5 -libmil -nofstore"
FPICFLAGS=-Kpic
CXX=CC
CXXFLAGS="-xO5 -xlibmil -nofstore"
CXXPICFLAGS=-Kpic
FC=f95
FCFLAGS=$FFLAGS
FCPICFLAGS=-Kpic
LDFLAGS=-L/opt/sunstudio12/lib/amd64
SHLIB_LDFLAGS=-shared
SHLIB_CXXLDFLAGS="-G -lCstd"
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 resulting build of R was not quite as fast as that built with gcc 4.2.1 at -O3.

C.3 Mac OS X

You can build R as a Unix application on Mac OS X using the Apple Developer Tools ('Xcode') and `g77` or `gfortran`. You will also need to install an X sub-system or configure with '`--without-x`'. The X window manager is part of the standard Mac OS X distribution since Mac OS X version 10.3 (Panther), but it is typically not pre-installed prior to 10.5 (Leopard).

For more information on how to find these tools please read the R for Mac OS X FAQ.

If you use the X window manager and prefer `Terminal.app` to `xterm`, you should be aware that R, like many Unix tools, uses the existence of a `DISPLAY` environment variable to determine whether an X system is running. This affects the default graphics device for the commmand-line version of R and the behaviour of the `png` and `jpeg` devices.

The `vecLib` library can be used *via* the (default) configuration options

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

to provide higher-performance versions of the BLAS and LAPACK routines. Building R without these options *via*

```
--without-blas --without-lapack
```

used not to work with earlier versions of `gcc`, but can be done with `gcc` 3.3 and later.

C.3.1 64-bit builds

64-bit builds are supported as from R 2.7.0 on 10.5 (Leopard). All that is needed is to select suitable compiler options, e.g. for recent Intel Macs

```
CC='gcc -arch x86_64'
CXX='g++ -arch x86_64'
F77='gfortran -arch x86_64'
FC='gfortran -arch x86_64'
OBJC='gcc -arch x86_64'
```

in '`config.site`' or on the `configure` command line.

C.4 Solaris

C.4.1 Solaris 10 and Open Solaris

R has been built successfully on Solaris 10 (both Sparc and '`x86`') using `gcc` 3/`g77`, `gcc` 4/`gfortran` and the (zero cost) Sun Studio 11/12 compilers. Sun packages for R are available from `http://www.sunfreeware.com/` for both architectures. (Recent Sun machines are Opterons ('`x86-64`') rather than '`x86`', but 32-bit '`x86`' executables are the default.)

There are also reports of success on OpenSolaris (aka Solaris Express Community Edition, and sometimes as Solaris 11) on '`x86`'.

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.

Modern Solaris systems allow a large selection of Open Source software to be installed via `pkg-get`: a Sparc Solaris 10 system came with `libreadline` and `libiconv` and a choice of `gcc3` and `gcc4` compilers, installed under '`/opt/csw`'. (You will need GNU `libiconv`: the Solaris version of `iconv` is not sufficiently powerful.)

If using `gcc`, do 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. A version of `gcc` optimized for Sparc (using technology from Sun's compilers) is available from Sun.[3]

[3] `http://www.sun.com/download/index.jsp?cat=Application%20Development&tab=3&subcat=Development%20Tools`

When using the Sun compilers[4] do *not* specify '`-fast`', as this disables IEEE arithmetic and `make check` will fail.

To compile for a 64-bit Sparc target with `gcc` 4 we used

```
CC="gcc -m64"
F77="gfortran -m64"
CXX="g++ -m64"
FC="gfortran -m64"
LDFLAGS="-L/opt/csw/gcc4/lib/sparcv9 -L/opt/csw/lib/sparcv9"
```

replacing '`gfortran`' with '`g77`' for `gcc` 3.x.y. Note that paths such as '`/opt/csw/gcc4/lib/sparcv9`' may need to be in the `LD_LIBRARY_PATH` during configuration.

For the Sun Studio compilers a little juggling of paths was needed to ensure GNU `libiconv` is used rather than the Solaris `iconv`: we used

```
CC="cc -xc99"
CPPFLAGS=-I/opt/csw/include
CFLAGS="-O -xlibmieee"
F77=f95
FFLAGS=-O4
CXX=CC
CXXFLAGS=-O
FC=f95
FCFLAGS=$FFLAGS
LDFLAGS=-L/opt/csw/lib
SHLIB_CXXLDFLAGS=-G -lCstd
```

to ensure that the `libiconv` version of '`iconv.h`' was found. For a 64-bit target add `-xarch=v9` (or `v9b`, or `amd64`) to each of the compiler commands, or for Sun Studio 12 add `-m64`. You can target specific Sparc architectures for (slightly) higher performance: Sun recommend

```
32-bit: -xtarget=ultra3 -xarch=v8plusa
64-bit: -xtarget=ultra3 -xarch=v9a
```

(in `CFLAGS` etc.) as a good compromise for recent Sparc chipsets.

By default the Sun Studio compilers do not conform to the C99 standard (appendix F 8.9) on the return values of functions such as `log`: use `-xlibmieee` to ensure this. Also, errors have been reported on '`x86_64`' if `-xc99` is omitted.

On '`x86`' you will get marginally higher performance *via*

```
CFLAGS="-xO5 -xc99 -xlibmieee -xlibmil -nofstore"
FFLAGS="-O5 -libmil -nofstore"
CXXFLAGS="-xO5 -xlibmil -nofstore"
SAFE_FFLAGS="-O5 -libmil -fstore"
```

Building on '`x86`' with `gcc` 4 failed `make check` in the complex LAPACK tests: using Sun Studio 11 worked correctly.

There is limited support for '`x86-64`' builds in the pre-built GNU software repositories, with 64-bit libraries (which are in '`/opt/csw/lib/amd64`' and so on) being scarce (and not including the `gcc` support libraries). The Sun Studio compilers do support 64-bit builds via `-xarch=amd64` (or, for version 12, `-m64`).

The Sun performance library `libsunperf` is available with the Sun Studio compilers. If selected as a BLAS, it must also be selected as LAPACK via

```
./configure --with-blas=sunperf --with-lapack
```

However, our tests were none too successful: Sparc 64-bit builds crashed.

[4] including `gcc` for Sparc from Sun.

C.4.2 Sparc Solaris 9 and earlier

These are now obsolete operating systems, so this subsection refers to equally old compiler versions.

Sun packages for R are available from http://www.sunfreeware.com/ for both Sparc and 'x86'.

R 2.6.2 was built successfully on Sparc Solaris 8 (aka Solaris 2.8 aka SunOS 5.8) using gcc 3/g77, gcc 4/gfortran and the 'Sun ONE Studio 7 Compiler Suite' (aka Forte 7).

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.

gcc 3.2.1 and 3.2.2 generate incorrect code on 32-bit Solaris builds with optimization, but versions 3.2.3 and later work correctly. (The symptom was that make check failed at the first attempt to plot.)

If using gcc, do 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 so (for example) gcc compiled under Solaris 2.6 will not compile R under Solaris 2.7. Also, do ensure that it was compiled for the assembler/loader in use: if you download gcc from www.sunfreeware.com then you need to download binutils too. To avoid all these pitfalls we recommended you compile gcc from the sources yourself.

It was reported by Mike Pacey that Sun Forte 9 requires '-xopenmp=stubs' added to LDFLAGS.

When using the Sun compilers do *not* specify '-fast', as this disables IEEE arithmetic and make check will fail. The maximal set of optimization options known to work on Sparc Solaris 8 is

```
-xlibmil -xO5 -dalign
```

('x86' versions do not need '-dalign', and some do not support it.) To get correct results for log requires -xlibmieee, but R works around that.

We have found little performance difference between gcc and cc but considerable benefit from using a SunPro Fortran compiler: the gcc/f77 combination worked well. For many C++ applications Forte 7 requires '-lCstd', which the configure script will add to SHLIB_CXXLDFLAGS if it identifies the compiler correctly.

To compile for a 64-bit target on Sparc Solaris (which needs an UltraSparc chip and for support to be enabled in the OS) with the Forte 7 compilers we used

```
CC="cc -xarch=v9 -xc99"
CFLAGS="-xO5 -xlibmil -dalign"
F77="f95 -xarch=v9"
FFLAGS="-xO5 -xlibmil -dalign"
CXX="CC -xarch=v9"
CXXFLAGS="-xO5 -xlibmil -dalign"
FC="f95 -xarch=v9"
FCFLAGS="-xO5 -xlibmil -dalign"
```

in 'config.site'.

For 64-bit compilation with gcc 3.4.x we used

```
CC="gcc -m64"
F77="g77 -m64"
CXX="g++ -m64"
FC="gfortran -m64"
LDFLAGS="-L/usr/local/lib/sparcv9 -L/usr/local/lib"
```

replacing 'g77' with 'gfortran' for gcc 4.x.y. Note that '/usr/local/lib/sparcv9' may need to be in the LD_LIBRARY_PATH during configuration. (configure will append -std=gnu99 to CC.)

Solaris on Sparc CPUs need '`PIC`' and not '`pic`' versions of `CPICFLAGS` and `FPICFLAGS` since the '`pic`' version only allows 1024 symbols on a 64-bit build (and 2048 on a 32-bit build).

Note that using `f95` allows the Sun performance library `libsunperf` to be selected: it may not work[5] with `f77`, and will not with `g77`. `libsunperf` contains both BLAS and LAPACK code, and '`--with-lapack`' may be required if you use it. On our test system using `libsunperf` failed for 64-bit builds with both Forte 7 and Sun Studio 11, albeit in different ways. Our experience has been that ATLAS's BLAS is faster than `libsunperf`, especially for complex numbers.

For a 64-bit build, 64-bit libraries must be used. As the configuration process by default sets `LDFLAGS` to '`-L/usr/local/lib`', you may need to set it to avoid finding 32-bit addons (as in the `gcc -m64` example above). It is possible to build Tcl/Tk as 64-bit libraries with the configure option `--enable-64bit`, but only with the Sun compilers (and not with `gcc`) as of Tcl/Tk 8.4.5.

There have been alignment issues, with Sun libraries requiring 8-byte alignment of doubles (which `gcc` generated by default, but `cc` did not).

C.5 HP-UX

% R-help 2007 Nov 1/2, and follow up to BDR Nov 6. Fan Long built R 2.6.0 on HP-UX 11.23 on '`ia64`' using the native compilers via

```
CC=cc
CFLAGS="+z +DD64"
CXX=aCC
CXXFLAGS="-b -lxnet +z +DD64"
FC=f90
FCFLAGS=+DD64
F77=f90
FFLAGS=+DD64
LDFLAGS="-L/usr/lib/hpux64 -L/opt/fortran90/lib"
```

Here '`+z`' selects PIC code and `+DD64` a 64-bit build.

He found that '`stdbool.h`' was detected but was non-standard, so '`HAVE_STDBOOL_H`' had to be undefined in '`src/include/config.h`'. In addition, the compiler objected to

```
static const pthread_once_t fresh_once = {PTHREAD_ONCE_INIT};
```

(line 362 of '`src/extra/intl/lock.c`') and needed the braces removed.

The remaining reports on HP-UX here predate R 2.0.0.

R has been built successfully on HP-UX 11.0 using both native compilers and `gcc`. By default, R is configured to use `gcc` and `g77` on HP-UX (if available). Some installations of `g77` only install a static version of the `g2c` library that cannot be linked into a shared library since its files have not been compiled with the appropriate flag for producing position independent code (PIC). This will result in `make` failing with a linker error similar to

```
ld: CODE_ONE_SYM fixup to non-code subspace in file foo.o -
shared library must be position independent. Use +z or +Z to recompile.
```

('`+z`' and '`+Z`' are the PIC flags for the native compiler `cc`.) If this is the case you either need to modify your `g77` installation or configure with

```
F77=fort77
```

to specify use of the native POSIX-compliant FORTRAN 77 compiler.

You may find that `configure` detects other libraries that R needs to use as shared libraries but are only available as static libraries. If you cannot install shared versions you will need to tell `configure` not to use these libraries, or make sure they are not in the library path. The

[5] recent versions have `f77` as a wrapper for `f95`, and these do work.

symptom will be the linker error shown in the last paragraph. Static libraries that might be found and would cause problems are

BLAS	use '--without-blas'
Tcl/Tk	use '--without-tcltk'
libpng	use '--without-libpng'
jpeg	use '--without-jpeglib'
zlib	use '--without-system-zlib'

and bzip2 and pcre are problematic when building 'libR.so', only. These can be avoided by '--without-system-bzlib' and '--without-system-pcre' respectively, but these are the defaults.

Some versions of gcc may contain what appears to be a bug at the '-O2' optimization level that causes

```
> 2 %/% 2
[1] 1
> 1:2 %/% 2
[1] 0 0     # wrong!!
```

which will cause make check to fail. If this is the case, you should use CFLAGS to specify '-O' as the optimization level to use.

You can configure R to use both the native cc and fort77 with

```
./configure CC=cc F77=fort77
```

f90 insists on linking against a static 'libF90.a' which typically resides in a non-standard directory (e.g., '/opt/fortran90/lib'). Hence, to use f90 one needs to add this directory to the linker path via the configure variable LDFLAGS (e.g., ./configure F77=f90 LDFLAGS=/opt/fortran90/lib).

C.6 IRIX

R 2.1.0 has been successfully built on IRIX64 6.5 using both gcc and the native (MipsPro 7.4) compiler. However, neither version has passed make check due to a problem with time zones (see below). A 64-bit executable has not been successfully built.

It appears that some (but not all) versions of IRIX have broken wide-character header files and so may need '--disable-mbcs'.

To build R with gcc use something like the following configuration flags

```
CPPFLAGS="-I/usr/freeware/include"
LDFLAGS="-L/usr/freeware/lib32"
```

To build the Tcl/Tk package you will most likely need to add

```
--with-tclconfig=/usr/freeware/lib/tclConfig.sh
--with-tkconfig=/usr/freeware/lib/tkConfig.sh
```

since these configuration scripts are not on your path.

To build R with the native compilers, use something like the following configuration flags

```
CC=cc F77=f77 CXX=CC
CPPFLAGS="-I/usr/freeware/include" LDFLAGS="-L/usr/freeware/lib32"
CFLAGS="-O2" FFLAGS="-O2" CXXFLAGS="-O2"
--with-system-bzlib=yes
```

The MipsPro compiler will not build the bzlib library, so you must use the external one provided by SGI as a freeware package.

After configuration, it is necessary to use gmake instead of the native make to build R.

There is a problem with the time zones on IRIX (originally reported by George N. White III for 1.9.0) which will cause the `strptime` tests to fail unless Arthur Olson's timezone data ftp://elsie.nci.nih.gov/pub/ has been installed (see also cspry.co.uk/computing/Indy_admin/TIMEZONE.html) and `-ltz` is added to the list of libraries (for example, in environment variable `LIBS`).

The flag '`-OPT:IEEE_NaN_inf=ON`' is added for the native compilers.

C.7 Alpha/OSF1

R has been built successfully on an Alpha running OSF1 V4.0 / V5.1 using `gcc`/`g77` and `cc`/`f77`. Mixing `cc` and `g77` fails to configure. The `configure` option '`--without-blas`' was used since the native blas seems not to have been built with the flags needed to suppress `SIGFPE`'s. Currently R does not set a signal handler for `SIGFPE` on platforms that support IEEE arithmetic, so these are fatal.

At some point in the past using `cc` required '`-std1`' to be set so '`__STDC__`' was defined. As far as we know this is no longer needed, and `configure` no longer sets it, but it does set '`-ieee_with_inexact`' for the C compiler and '`-fpe3`' for the FORTRAN compiler (and '`-mieee-with-inexact`' and '`-mieee`' for `gcc`/`g77`) (in the appropriate `R_XTRA_*` flags).

C.8 Alpha/FreeBSD

Attempts to build R on an Alpha with FreeBSD 4.3 have been only partly successful. Configuring with '`-mieee`' added to both `CFLAGS` and `FFLAGS` builds successfully, but tests fail with `SIGFPE`'s. It would appear that '`-mieee`' only defers these rather than suppressing them entirely. Advice on how to complete this port would be greatly appreciated.

C.9 AIX

We no longer support AIX prior to 4.2, and `configure` will throw an error on such systems. The recent testing has been under AIX 5.2 on '`powerpc`', where Ei-ji Nakama was able to build pre-2.5.0 with `gcc 4.0.3` in several configurations.

Mr Nakama found 32-bit versions of R could be built with `configure --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"
```

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.)

C.10 Cygwin

The Cygwin emulation layer on Windows can be treated as a Unix-alike OS. This is unsupported, but experiments have been conducted and a few workarounds added for R 2.6.0.

Only building as a shared library works,[6] so use

```
./configure --disable-nls --disable-mbcs --enable-R-shlib
make
```

MBCS does not work—`wcstod` is missing—but would only be of any use in a CJK locale. NLS does work, although '`--with-included-gettext`' is preferable. You will see many warnings about the use of auto-import.

It almost passes `make check`: FIFOs are supposedly supported but hung in '`example(fifo)`'.

Note that this gives you a command-line application using `readline` for command editing. The '`X11`' graphics device will work if a suitable X server is running, and the standard Unix-alike ways of installing source packages work. There was a bug in the '`/usr/lib/tkConfig.sh`' script in the version we looked at, which needs to have

```
TK_LIB_SPEC='-ltk84'
```

The overhead of using shell scripts makes this noticeably slower than a native build of R on Windows.

C.11 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 know 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 requires 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 SunPro compilers causes R's `NaN` to be set incorrectly.

Shared Libraries: There seems to be very little agreement across platforms on what needs to be done to build shared libraries. 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.)

[6] DLLs need to have all links resolved at build time and so cannot resolve against '`R.bin`'.

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 Enabling search in HTML help

There is a search engine available from the front page of the HTML help system, the page that is displayed by `help.start()`. The search engine is written in Java and invoked by Javascript code, so the first thing to do is to ensure that both are enabled in your favourite browser. Then try it and see: with most browsers you should see

```
Applet SearchEngine started
```

displayed in the status bar. (Internet Explorer shows `Applet started`.) Then click on one of the keywords and after a short delay (several seconds) you should see a page of search results.

If this fails you should double-check that Java is enabled in your browser by visiting a page such as www.java.com/en/download/help/testvm.jsp. Java 1.4 or later is needed.

On Mozilla-based browsers the links on the results page will become inactive if you return to it: to work around this you can open a link in a new tab or window.

Many thanks to Marc Schwartz in tracking down many of these issues with enabling the Java search engine.

D.1 Java Virtual Machines on Linux

We are aware of problems with certain older Java installations: Sun's Java Run-time Environment version 1.5.0 or later is strongly recommended for Mozilla-based browsers. In particular, Sun's Java Run-time Environment `j2re` 1.4.2_02 to _05 do not work under '`ix86`' Linux.

Note that there appears not to be a Sun Java plugin for 64-bit browsers on '`x86_64`' Linux: (forum.java.sun.com/thread.jspa?threadID=568127&tstart=75) (although one is promised for Java 7) but 32-bit browsers have been used on that platform.

Marc Schwartz reports that the 'IcedTea' JVM that ships with Fedora 8 does not work with the search applet. On '`ix86`' the Sun JRE and plugin can be instaled: see www.fedorafaq.org/#java. (The same FAQ documents the use of 32-bit Firefox on '`x86_64`'.)

Other Java installations, for example those from Blackdown and IBM, have been used.

Older but still useful links are for Firefox/Mozilla, plugindoc.mozdev.org/faqs/java.html, for Konqueror www.konqueror.org/javahowto/, for Opera www.opera.com/support/search/supsearch.dml?index=459 and for Debian GNU/Linux www.debian.org/doc/manuals/debian-java-faq/.

D.2 Java Virtual Machines on Unix

We have much less experience than under Linux, but we do know that Sun's Run-time Environment `j2re` 1.4.2_03 does not work under Sparc Solaris 8, whereas `jre` 1.5.0 and `j2re` 1.4.2_01 (available from java.sun.com/products/archive) do.

D.3 Java Virtual Machines on Windows

We have not seen any problems on Windows provided a Java Virtual Machine has been installed and is operational: Sun's current `j2re` 1.6.0 works in Internet Explorer, Mozilla 1.7 and Mozilla Firefox on Windows XP/Vista. Note that a recent Windows system may not have a JVM installed at all. Visit java.sun.com/getjava/manual.html to install a Sun JVM. Which (if any) JVM is enabled can be set in 'Set Program Access and Defaults' in Windows XP (SP1 or later), and which JVM is used by browser plugins may also be controlled by the Sun Java applet in the Control Panel.

Recent versions of Internet Explorer may block the use of Java applets and need the block removed *via* the *information bar*.

Apple's Safari 3.0.4 also works.

D.4 Java Virtual Machines on Mac OS X

Java 1.5.0 ships with recent versions of Mac OS X, and the HTML search engine works with Safari under Mac OS 10.5.

Appendix E 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 www.murdoch-sutherland.com/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 15 for details.)

As from R 2.7.0, only building with `gcc 4.y.z` is supported, and that compiler set works out-of-the-box on Windows Vista.

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 section contains a lot of prescriptive comments. They are here as a result of bitter experience. Please do not report problems to R-help 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 http://www.murdoch-sutherland.com/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.

- Perl (in '`Rtools.exe`')
- The command line tools (in '`Rtools.exe`')
- The MinGW compilers (in '`Rtools.exe`')

For building simple packages containing data or R source but no compiled code, only the first two of these are needed.

A complete build of R including compiled HTML help files and PDF manuals, and producing the standalone installer '`R-2.7.0-win32.exe`' will also need the following:

- The Microsoft HTML Help Workshop
- LaTeX
- The Inno Setup installer

It is important to set your `PATH` properly. The '`Rtools.exe`' optionally sets the path to components that it installs.

Your `PATH` may include '`.`' first, then the '`bin`' directories of the tools, Perl, MinGW and LaTeX, as well as the Help Workshop directory. 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, all on one line,

```
PATH=c:\Rtools\bin;c:\Rtools\perl\bin;c:\Rtools\MinGW\bin;c:\texmf\miktex\bin;
     c:\progra~1\htmhe~1;c:\R\bin;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 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.

Edit '*`R_HOME`*`/src/gnuwin32/MkRules`' to set the appropriate paths as needed and to set the type(s) of help that you want built. **Beware**: '`MkRules`' contains tabs and some editors (e.g., WinEdt) silently remove them.

[1] For example, the Cygwin version of `make 3.81` fails to work correctly.

Set the appropriate environment variables.

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.

E.1 Perl

You will need a Windows port of `perl5` (but only the basic functionality, not any of the third-party Win32 extensions). The Vanilla Perl package is included in '`Rtools.exe`'. A more full-featured distribution is available from `www.activestate.com/Products/ActivePerl`, and this was used in releases of R up to R 2.5.1. Alternatives are listed at `win32.perl.org`.

Beware: you do need a *Windows* port and not the Cygwin one. Users of 64-bit Windows can use a Win64 Perl (such as that from ActiveState) if they prefer.

E.2 The Microsoft HTML Help Workshop

To make compiled html ('`.chm`') files you will need the Microsoft HTML Help Workshop, currently available for download at `msdn.microsoft.com/library/en-us/htmlhelp/html/hwmicrosofthtmlhelpdownloads.asp` and `www.microsoft.com/office/ork/xp/appndx/appa06.htm`. This is *not* included in '`Rtools.exe`'.

You may need this on the same drive as the other tools. (Although we regularly use it on a different drive, problems have been reported in the past.)

To skip building compiled html help when building R, set `WINHELP=NO` in '`MkRules`'. In this case the Help Workshop will not be needed (but it will be needed for installing packages by `R CMD INSTALL` with the default settings).

E.3 LaTeX

The '`MiKTeX`' (`www.miktex.org`) distribution of LaTeX includes a suitable port of `pdftex`. The 'basic' version of '`MiKTeX`' almost suffices (the **grid** vignettes need '`fancyvrb.sty`'), but it will install the 15Mb '`lm`' package if allowed to (although that is not actually used). The '`Rtools.exe`' installer does *not* include any version of LaTeX.

Please read Section 2.2 [Making the manuals], page 4 about how to make '`refman.pdf`' and set the environment variables `R_RD4DVI` and `R_RD4PDF` suitably; ensure you have the required fonts installed.

E.4 The Inno Setup installer

To make the installer package ('`R-2.7.0-win32.exe`') we require Inno Setup 5.1.7 or later (including 5.2.x) from `jrsoftware.org`. This is *not* included in '`Rtools.exe`'.

Edit file '`src/gnuwin32/MkRules`' and change `ISDIR` to the location where Inno Setup was installed.

E.5 The command line tools

This item and the next are 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`, `diff`, `echo`, `egrep`, `expr`, `find`, `gawk`, `grep`, `gzip`, `head`, `ls`, `make`, `makeinfo`, `mkdir`, `mv`, `rm`, `rsync`, `sed`, `sh`, `sort`, `texindex` and `touch`; we use those from the Cygwin distribution (`www.cygwin.com`) or compiled from the sources. You will also need `zip` and `unzip` from the Info-ZIP project (`www.info-zip.org`). All of these tools are in '`Rtools.exe`'.

Beware: 'Native' ports of make are **not** suitable (including that at the mingw site). There were also problems with several earlier 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.

E.6 The MinGW compilers

This version of R is set up to use `gcc 4.2.1` for which MinGW compilers were released in August 2007. The '`Rtools.exe`' installer currently includes the `-sjlj` version of `4.2.1` of the MinGW port of `gcc` from http://sourceforge.net/project/showfiles.php?group_id=2435.

If you would like to install your own copy, we recommed downloading from the URL above, as the download links from www.mingw.org have led to obsolete versions. See the notes on www.murdoch-sutherland.com/Rtools for updates.

To download the components individually, currently you need

```
mingw-runtime-3.13.tar.gz
w32api-3.10.tar.gz
binutils-2.17.50-20060824-1.tar.gz
gcc-core-4.2.1-sjlj-2.tar.gz
gcc-g++-4.2.1-sjlj-2.tar.gz
gcc-gfortran-4.2.1-sjlj-2.tar.gz
```

(and `gcc-objc-4.2.1-sjlj-2.tar.gz` if you want Objective C support). Unpack these into the same directory (using `tar zxf tarball_name`). (You may need to copy '`bin/gcc-sjlj.exe`' to '`bin/gcc.exe`' and a few badly-written packages need '`bin/g++-sjlj.exe`' copied to '`bin/g++.exe`'.) This compiler should work on Windows Vista without any workarounds.

Note that `mingw-runtime-3.13.tar.gz` or later and `gcc-4.2.1` or later are needed to get a correct build of R itself. (The 'Snapshot' `binutils-2.17.50-20070129-1.tar.gz` has also be tested.) There are known problems with using other compiler sets on Windows Vista (http://www.nabble.com/environment-hosed-during-upgrade-tf3305745.html#a9195667 and that a suitable `PATH` needs to be set to include the path to '`cc1`'.

Other builds of `gcc` 4 are available from http://gcc.gnu.org/wiki/GFortranBinaries and http://www.tdragon.net/recentgcc/: these need the `PATH` workaround on Vista.

Function and variable index

Concept index

Environment variable index