

R Installation and Administration

Version 2.3.0 (2006-04-24)

R Development Core Team

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

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

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

Copyright © 2001–2005 R Development Core Team

ISBN 3-900051-09-7

Table of Contents

1	Obtaining R	1
1.1	Getting and unpacking the sources	1
1.2	Getting patched and development versions	1
1.2.1	Using Subversion and rsync	1
2	Installing R under Unix-alikes	3
2.1	Simple compilation	3
2.2	Making the manuals	4
2.3	Installation	5
2.4	Uninstallation	6
2.5	Sub-architectures	6
3	Installing R under Windows	8
3.1	Building from source	8
3.1.1	Getting the tools	8
3.1.2	Getting the source files	8
3.1.3	Building the core files	9
3.1.4	Building the bitmap files	9
3.1.5	Checking the build	10
3.1.6	Building the manuals	10
3.1.7	Building the installer	10
3.1.8	Cross-building on ix86 Linux	11
4	Installing R under Mac OS X	12
4.1	Building from source on Mac OS X	12
5	Running R	13
6	Add-on packages	14
6.1	Installing packages	14
6.1.1	Customizing package compilation under Unix	15
6.1.2	Customizing package compilation under Windows	15
6.2	Updating packages	16
6.3	Removing packages	16
6.4	Setting up a package repository	17
7	Internationalization and Localization	18
7.1	Locales	18
7.1.1	Locales under Linux	18
7.1.2	Locales under Windows	19
7.1.3	Locales under Mac OS X	19
7.2	Localization of messages	19
8	Choosing between 32- and 64-bit builds	21
8.1	Windows	21

Appendix A	Essential and useful other programs under Unix	22
A.1	Essential programs	22
A.2	Useful libraries and programs	22
A.2.1	Tcl/Tk	23
A.2.2	Linear algebra	23
Appendix B	Configuration on Unix	26
B.1	Configuration options	26
B.2	Internationalization support	26
B.3	Configuration variables	27
B.3.1	Setting paper size	27
B.3.2	Setting the browser	27
B.3.3	Compilation flags	27
B.3.4	Making manuals	28
B.4	Using make	28
B.5	Using FORTRAN	28
B.5.1	Using gfortran	29
B.6	Compile and load flags	29
Appendix C	Platform notes	31
C.1	X11 issues	31
C.2	Linux	32
C.3	Mac OS X	34
C.4	Solaris on Sparc	34
C.5	HP-UX	35
C.6	IRIX	36
C.7	Alpha/OSF1	37
C.8	Alpha/FreeBSD	37
C.9	AIX	37
C.10	New platforms	38
Appendix D	Building the GNOME console	40
Appendix E	Enabling search in HTML help	42
E.1	Java Virtual Machines on Linux	42
E.2	Java Virtual Machines on Unix	42
E.3	Java Virtual Machines on Windows	42
E.4	Java Virtual Machines on Mac OS X	43
Appendix F	The Windows toolset	44
F.1	The command line tools	44
F.2	Perl	45
F.3	The MinGW compilers	45
F.4	The Microsoft HTML Help Workshop	45
F.5	L ^A T _E X	45
F.6	The Inno Setup installer	46
	Function and variable index	47
	Concept index	48

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.tgz’ file, and unpack it with

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

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.tgz | 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.tgz-split.*’ files and paste them together at the destination with (Unix)

```
cat R-x.y.z-split.* > R-x.y.z.tgz
```

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).

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/](http://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-patches/>’ (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*.

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 -ll --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’ and ‘sparc-sun-solaris’, as well as probably (it is tested less frequently on these platforms) ‘i386-*-freebsd’, ‘i386-*-netbsd’, ‘i386-*-openbsd’, ‘i386-sun-solaris’, ‘mips-sgi-irix’, ‘alpha-dec-osf*’, ‘rs6000-ibm-aix’, and ‘hppa-hp-hpux’.

In addition, binary distributions are available for some common Linux distributions and for Mac OS X (on PowerPC). 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 22, 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 27.) Issue the following commands:

```
./configure
make
```

(See [Section B.4 \[Using make\]](#), page 28 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, but you should look carefully at any reported discrepancies. 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

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

and so on, as described further below. This has the advantage of always keeping your source tree “clean”. (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`). Note that you need Perl version 5: if this is not available on your system, you can obtain PDF versions of the documentation files via CRAN.

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 the info files unless you have `makeinfo` version 4.7 or later installed.

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:

<code>times</code>	(The default.) Using standard PostScript fonts. This works well both for on-screen viewing and for printing, and is the default. 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 T_EX distribution, but can be 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** 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. This was the default in R version 1.x.y.

The default can be overridden by setting the environment variable `R_RD4PDF`. (On Unix, this will be picked up at install time.) The default value is `times,hyper`: omit `hyper` if you do not want hyperlinks, e.g. for printing.

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
```

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.

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`.

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

```
libdir=EPREFIX/${LIBnn}
```

All of `bindir` and `mandir` and `libdir` can also be specified on the `make install` command line.

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-2.3.0`.

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.

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`.

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`.

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`’ and ‘`i386`’) under MacOS 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 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 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 build you can select which build to run by

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

and running ‘`R`’ will run the last build that was installed.

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

```
R --arch=name CMD INSTALL --libs-only pkg(s)
```

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 95, 98, NT4, 2000, ME, XP and 2003 (at least) on Intel x86 and clones (including AMD64/EM64T chips and Windows 64).

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

Installation is *via* the installer ‘R-2.3.0.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 F \[The Windows toolset\]](#), page 44 (and perhaps updates in www.murdoch-sutherland.com/Rtools) for details.

Be sure to set your path in the order given in the appendix.

3.1.2 Getting the source files

You need to collect the following sets of files:

- Get the R source code ‘R-2.3.0.tgz’ from CRAN. Open a command window (or another shell) at a directory *whose path does not contain spaces*. We will call this directory *R_HOME* below. Run

```
tar zxvf R-2.3.0.tgz
```

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.

- Extract the international character support file ‘iconv.dll’ from <http://www.murdoch-sutherland.com/Rtools/iconv.zip> and put it in ‘*R_HOME*/src/gnuwin32/unicode’.
- The TclTk support files are in www.murdoch-sutherland.com/Rtools/R_Tcl.zip: 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]. You will need files ‘libpng-1.2.8.tar.gz’ and ‘jpegsrc.v6b.tar.gz’ or later.
- 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 `rsync` and 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 Pentium and AthlonXP chips are available in the ‘windows/contrib/ATLAS’ area on CRAN.

Another possibility is to use AMD's AMD Core Math Library (ACML) www.amd.com/acml. To use the Cygwin-compiled version (Cygwin itself is not needed, at least currently), install 'acml-3-0-0-win32-g77.exe' (or later) and edit '*R_HOME/src/gnuwin32/MkRules*' to define `USE_ACML=YES` and set `ACML_PATH` to the path to 'libacml.a'. If you also set `USE_ACML_LAPACK=YES`, ACML will be used to provide the LAPACK library for R itself and any packages using `$LAPACK_LIBS` built under that setting.

There used to be support for Kazushige Goto's BLAS, but this is no longer available for Windows, and it seems that it will in future only to be offered via a source-code licence.

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/Rhtml.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. 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-1.2.8.tar.gz
> mv libpng-1.2.8 libpng
> tar xzvf jpegsrc.v6b.tar.gz
```

Once everything is set up in directory '*bitmap*', `make` in that directory or `make bitmapdll` in the parent directory 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 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 F.6 \[The Inno Setup installer\]](#), [page 46](#)).

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-2.3.0.exe’, so please rename it to indicate that it is customized.

3.1.8 Cross-building on ix86 Linux

It is possible to cross-build R or packages on ‘ix86’ Linux, and the ‘ix86’ cross-compilers have also been used successfully on ‘x86_64’ Linux.

You will need i386-mingw32 cross-compilers installed and in your path. There is currently a complete set of tools at www.stats.ox.ac.uk/pub/Rtools/mingw-cross5.tar.bz2 (Just unpack this somewhere and put its ‘bin’ directory in your path.)

You will need Perl, zip and unzip installed and makeinfo version 4.7 or later (part of GNU texinfo).

You also need the R source (‘R-2.3.0.tgz’).

Then: untar ‘R-2.3.0.tgz’ somewhere, and

```
cd /somewhere/R-2.3.0/src/gnuwin32
```

Edit ‘MkRules’ to set BUILD=CROSS and the appropriate paths (including HEADER) as 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 Chapter 6 [Add-on packages], page 14.

(It is possible to cross-build the installers using WINE, which we leave as an exercise for the reader.)

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

```
cd installer
make imagedir
zip -r9X R-2.3.0.zip R-2.3.0 # or something similar
```

We have not found a reliable way to convert base to lazy-loading when cross-building, so it is left in the old format.

4 Installing R under Mac OS X

The ‘`bin/macosx`’ directory of a CRAN site contains binaries for MacOS X for a base distribution and a large number of add-on packages from CRAN to run on Mac OS X version 10.3.0 or higher.

The simplest way is to use ‘`R.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-2.3.0.tgz
cd R-2.3.0
./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.

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 27](#)).

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.

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 Installing packages

Packages may be distributed in source form or compiled binary form. Installing source packages requires that compilers and tools (including Perl 5.004 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.

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.

The same command works in Windows if you have the source-code package files (option "Source Package Installation Files" in the installer) and toolset (see [Appendix F \[The Windows toolset\]](#), page 44) installed.

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"))
```

Unless the library is specified (argument `lib`) the first library in the library search path is used. 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)
```

What `install.packages` does by default is different on Unix and Windows. On Unix-alikes (include MacOS X unless running from the GUI console) 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.

`install.packages` can install a source package from a local '`.tar.gz`' file by setting argument `repos` to `NULL`.

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`.

`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, ...).

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.package` in order to download and install binary packages from CRAN. 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.

6.1.1 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.

6.1.2 Customizing package compilation under Windows

This section describes ways to customize package compilation using the standard C / C++ / 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+=-lodb32
```

or, equivalently,

```
RODBC-DLLLIBS=-lodb32
```

but in fact contains the single line

```
PKG_LIBS=-lodb32
```

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/Makefile`’
- ‘`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 of the above to be ignored.

Beware: files ‘`src/Makefile`’ or ‘`src/Makevars`’ will be used if they exist and the ‘`.win`’ equivalents do not. Such files included in package sources are usually designed for use under Unix and are best removed.

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.

6.2 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.

6.3 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*.

6.4 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/powerpc/contrib/x.y` or `bin/macosx/i686/contrib/x.y` for R versions `x.y.z` and containing ‘.tgz’ files.

Each terminal directory must also contain a ‘PACKAGES’ file. This can be a concatenation of the ‘DESCRIPTION’ files of the packages separated by blank lines (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’.

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

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 has 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). What characters are valid in names was taken from the current locale. In general other locales with single-byte encodings worked, but e.g. `postscript()` and `pdf()` needed to be told about the encoding in use.

Full internationalization can be enabled when R is built under Unix-alikes by the (default) `configure` option ‘`--enable-mbcs`’: see [Appendix B \[Configuration on Unix\]](#), page 26. Under Windows, it is enabled by default in source builds, but support for ‘East Asian’ (Chinese/Japanese/Korean) languages is only enabled in the binary install if it is selected in the installer.

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) and those for Chinese, Japanese and Korean. Note that only some of the graphics devices can handle multi-byte or even non-Latin-1 character sets.

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.

Internally Mac OS X uses a form similar to Linux but without specifying the encoding (which is UTF-8).

7.2 Localization of messages

The preferred language for messages is by default taken from the locale. This can be overridden first by the setting of the environment variable `LANGUAGE` and then 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.)

Note that you should not expect to be able to change the language once R is running, and so `LC_MESSAGES` is not supported by `Sys.setlocale`.

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.

Translations in the right language but the wrong charset be made use of by on-the-fly re-encoding (on systems that support it). 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’) 64-bit is.

All current versions of R use 32-bit integers and IEC 60559¹ double-precision reals, and so compute to the same precision² 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 process³ 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 more time to manipulate, especially during garbage collection. So 64-bit versions of R will typically run slower than 32-bit versions. (On Sparc Solaris the difference was 15-20%, on ‘x86_64’ around 10%.)

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 6.

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 build a native version for Windows 64 until suitable compilers are available, and currently (mid-2006) that is not in prospect.

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

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 28](#)). Some add-on packages also need a C++ compiler. Your C compiler should be IEC 60059¹, 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 `CFLAGS` suitably. For recent versions of `gcc` with `glibc` this means including `-std=gnu99` in `CFLAGS` (`-std=c99` excludes POSIX functionality).

Unless you do not want to view graphs on-screen you need ‘X11’ installed, including its headers and client libraries. (On Fedora Core Linux this means the ‘`xorg-x11-devel`’ and ‘`xorg-x11-libs`’ RPMs, for example. Older Linuxes used ‘`XFree86-`’. Some Debian derivatives apparently also require ‘`libxt-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: you will need a fairly recent version. Otherwise you will need to configure with ‘`--with-readline=no`’ (or equivalent).

You will need Perl version 5.004 or later, available via www.perl.com/CPAN, to build any of the on-line documentation.

You will not be able to build the info files 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.)

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

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

A.2 Useful libraries and programs

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.

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 bitmapped graphics devices `jpeg()` and `png()` 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) respectively.

¹ also known as IEEE 754

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.

A.2.1 Tcl/Tk

The `tcltk` package needs Tcl/Tk 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'
```

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 will most likely be necessary to avoid confusion between them.

Versions of Tcl/Tk from 8.3 to 8.4.13 have been used successfully: 8.0 is no longer supported.

A.2.2 Linear algebra

The linear algebra routines in R can make use of enhanced BLAS (Basic Linear Algebra Subprograms, www.netlib.org/blas/faq.html) routines. Some are compiler-system-specific (`libsunperf` on Sun Sparc², `libessl` on IBM, `vecLib` on Mac OS X) but `ATLAS` (math-atlas.sourceforge.net) is a “tuned” BLAS that runs on a wide range of Unix-alike platforms. If no more specific library is found, a `libblas` library in the library path will be used. 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'`. (Alternatively, the environment variable `BLAS_LIBS` can be set, for example in `'config.site'`.)

For systems with multiple processors it is possible to use a multi-threaded version of `ATLAS`. An 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 version of `ATLAS`. You can use a multi-threaded `ATLAS` by specifying

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

Users of Fedora Core 4 Extras will want to use

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

since the basic BLAS is incomplete.

Another tuned BLAS which is available for some processors under Linux is by Kazushige Goto, currently available *for academic use only* (after registering) via <http://www.tacc.utexas.edu/resources/software/software.php>. (Recent versions are multi-threaded only.) Once this is installed, it can be used by configuring with

² Using the SunPro aka Forte aka Sun ONE cc and f95 compilers

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

It has been reported that on some RedHat-based Linux systems it is necessary to set `OMP_NUM_THREADS=1` (to disable multiple threads) in the environment when using a multi-threaded Goto BLAS, but others run happily with multiple threads. That environment variable should be set when using R profiling. (In some applications, allowing multiple threads slowed the computations down several-fold, so it can be helpful to restrict the number of threads.)

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

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

with the same caveat on multi-threading (but the number of threads defaults to 1). (Thanks to Andy Liaw for the information.)

For 'x86_64' and 'ix86' processors under Linux there is the AMD Core Math Library (ACML) www.amd.com/acml. For the gcc version (which is single-threaded only) we could use

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

if the appropriate library directory such as '/opt/acml3.1.0/gnu64/lib' is in the `LD_LIBRARY_PATH`. For other compilers, see the ACML documentation. Although the documentation is unclear, 32-bit versions of ACML do run on other 'ia32' chips such as AthlonMP and P4.

Note that the BLAS library will be used for several 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. In any case, the BLAS library must be usable with dynamically-loadable code: this can be a problem with ATLAS on some platforms (including 'x86_64') as it is not by default built with position-independent code.

You will need double-precision and double-complex versions of the BLAS, but not single-precision nor complex routines. This is checked by `configure`, as some incomplete BLAS libraries have been found (e.g., the generic BLAS supplied with Fedora Core 3 and 4 Extras is missing `drotm`, `drotmg` and `zdrot`).

On some systems it is necessary that an external BLAS 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 29.

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). 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 supplies a full LAPACK which can be used via

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

or `-lmkl_lapack32`: `-lmkl_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.10 \[New platforms\]](#), page 38: 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.

As with all libraries, you need to ensure that they and R were compiled with compatible compilers and flags. For example, this means that on Sun Sparc using the native compilers the flag ‘`-dalign`’ is needed so `libsunperf` can be used.

An ATLAS ‘tuned’ BLAS 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. Other BLASs can also be used when building from source.

Note that under Unix (but not under Windows) if R is compiled against a non-default BLAS, then all BLAS-using packages must also be. So if R is re-built after ATLAS is installed, then packages such as **quantreg** will need to be re-installed.

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]
```

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 performance¹ 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 a R system linked with `'--enable-R-shlib'` are linked against the dynamic library and so such packages cannot be used from a R system built in the default way.

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`, something like `en_GB.utf8`². 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 “Additional capabilities”. Then if R is started in a UTF-8 locale it assumes that the terminal will supply and display UTF-8-encoded characters³. If run in a single-byte locale, R behaves almost exactly as if it was configured with `'--disable-mbcs'`.

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

² AIX has to be different: it has `EN_US.UTF-8`!

³ You may have to set this with `luit`, but it should be the default in a window manager session started in UTF-8.

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.

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.

B.3.3 Compilation flags

If you have libraries and header files, e.g., for GNU readline, in non-system directories, use the variables `LD_FLAGS` (for libraries, using `‘-L’` flags to be passed to the linker) and `CPP_FLAGS` (for header files, using `‘-I’` flags to be passed to the C/C++ preprocessors), respectively, to specify these locations. These default to `LD_FLAGS=-L/usr/local/lib` (`/usr/local/lib64` on most 64-bit Linux OSes) and `CPP_FLAGS=-I/usr/local/include` 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).

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 2.7/8/9 (but not earlier).

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

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

B.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`, `f77`, `pgf77`, `cf77`, `fort77`, `f132`, `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.

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 time⁵, but you may need to avoid compiler flags⁶ asserting FORTRAN 77 compliance.

For performance reasons⁷ 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.

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

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

B.5.1 Using gfortran

`gfortran` is the F95 compiler that is part of `gcc 4.x.y`.

There was a problem with the version in the `gcc 4.0.0` release and most pre-releases. At least on ‘`ix86`’ and ‘`x86_64`’ Linux and MacOS X the dynamic version of the FORTRAN runtime `libgfortran` could not be used: if this is loaded redirection of C ‘`stdin`’ (which R uses in many of its scripts) becomes non-functional. A workaround is to set the environment variable `GFORTRAN_STDIN_UNIT` to `-1`, but versions before 10 April 2005 had another bug causing the setting to be ignored. Both problems have been fixed for `gcc 4.0.1`. The version of `gfortran` shipping with Fedora Core 4 seems to have the fixes in place.

Some versions of `g95` (from www.g95.org) have the same problem.

On `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`, as well as with Dr Goto’s BLAS. 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

LDLFLAGS 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

MAIN_LDFLAGS
additional flags for the main link

SHLIB_LDFLAGS
additional flags for linking the shared libraries

LIBBn 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

Library paths specified as ‘-L/lib/path’ in **LDLFLAGS** 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 reporting¹ 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 do 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'` we used

```
CFLAGS="-m32 -O2 -g -Wall -pedantic -std=gnu99"
CXXFLAGS="-m32 -O2 -g"
FFLAGS="-m32 -O2 -g"
LDFLAGS=-m32
LIBnn=lib
--x-libraries=/usr/X11R6/lib
```

and had to build `readline` ourselves as Fedora Core 3 only shipped a 64-bit version.

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 `'-liieee'`. 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.

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'` *via*

```
CC=icc
F77=ifort
CXX=icc
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"
```

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

```
CC=icc
CFLAGS="-g -O2 -wd188"
F77=ifort
CXX=icc
ICC_LIBS=/opt/compilers/intel/cce/9.0/lib
IFC_LIBS=/opt/compilers/intel/fce/9.0/lib
LDFLAGS="-L$ICC_LIBS -L$IFC_LIBS -L/usr/local/lib"
```

R will add `-mp` in `R_XTRA_{C,F,CXX}FLAGS` to maintain correct IEC 60559 arithmetic. The flag `'-wd188'` suppresses a large number of warnings about the enumeration type `'Rboolean'`. Because the Intel C compiler sets `'__GNUC__'` without complete emulation of `gcc`, we suggest adding `CPPFLAG=-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.

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.3 Mac OS X

You can build R as a Unix application on Mac OS X using the Apple Developer Tools 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).

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 command-line version of R and the behaviour of the `png()` and `jpeg` devices.

The `vecLib` library of Mac OS X $\geq 10.2.2$ 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.4 Solaris on Sparc

R has been 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), and less regularly on Solaris 9 and 10. GNU `make` was needed prior to Solaris 2.7 for building other than in the source tree, and is sometimes needed to establish the correct dependencies when rebuilding. R 2.2.1-patched has been built on Solaris 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 strongly 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 SunPro 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 is

```
-xlibmil -x05 -dalign -xc99
```

(‘x86’ versions do not need ‘`-dalign`’, and some do not support it.)

We have found little performance difference between `gcc` and `cc` but considerable benefit from using a SunPro FORTRAN compiler: the `gcc/f77` combination works 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 ‘`--without-blas`’ and

```
CC="cc -xarch=v9"
CFLAGS="-x05 -xlibmil -dalign -xc99"
```



```
F77="f95 -xarch=v9"
FFLAGS="-x05 -xlibmil -dalign"
CXX="CC -xarch=v9"
CXXFLAGS="-x05 -xlibmil -dalign"
FC="f95 -xarch=v9"
FCFLAGS="-x05 -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 4.0.x. Note that '/usr/local/lib/sparcv9' may need to be in the 'LD_LIBRARY_PATH' during configuration.

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 work² 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 6U1 and 7, albeit in different ways. Our experience has been that ATLAS's BLAS is faster than `libsunperf`, especially for complex numbers.

Some care is needed to ensure that libraries found by `configure` are compatible with the R executable and modules, as the testing process will not detect many of the possible problems. For 32-bit builds under `cc` the flag '-dalign' is needed for some of the Sun libraries: fortunately the equivalent flag for `gcc`, '-mno-unaligned-doubles', is the default. In theory, libraries such as `libpng`, `libjpeg`, `zlib` and the ATLAS libraries need to be built with a `pic` or `PIC` flag, which could be a problem if static libraries are used. In practice this seems to give little problem for 32-bit builds, especially as 'ix86' Linux accepts non-PIC code in shared libraries.

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 add-ons (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 Forte compiler (and not with `gcc`) as of Tcl/Tk 8.4.5.

C.5 HP-UX

The reports on HP-UX here predate R 2.0.0.

R has been built successfully on HP-UX 10.2 and HP-UX 11.0 using both native compilers and `gcc`. However, 10.2 has not been tested since R 1.4.0. 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

² recent versions have f77 as a wrapper for f95, and these do work.

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

BLAS	use ‘ <code>--without-blas</code> ’
Tcl/Tk	use ‘ <code>--without-tcltk</code> ’
libpng	use ‘ <code>--without-libpng</code> ’
jpeg	use ‘ <code>--without-jpeglib</code> ’
zlib	use ‘ <code>--without-system-zlib</code> ’

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.

Some systems running HP-UX 11.0 may have a `gcc` that was installed under HP-UX 10.2. Between versions 10.2 and 11.0 HP-UX changed its support functions for IEEE arithmetic from the recommended functions of the IEEE standard to the ones specified in the C9x draft standard. In particular, this means that `finite` has been replaced by `isfinite`. A `gcc` configured for HP-UX 10.2 run on 11.0 will not find `isfinite`, and as a result `configure` does not recognize the machine as fully supporting IEEE arithmetic and so will not complete. The best solution is to install a properly configured `gcc`. An alternative work-around is to add ‘`-DIEEE_754`’ to the `CFLAGS` variable.

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

On AIX 4.3.3 and AIX 5.1/2 (and probably latter), it was found that the use of "run time linking" (as opposed to normal AIX style linking) was required. For this, the R main program must be linked to the runtime linker with the `'-brtl'` linker option, and shareable objects must be enabled for runtime linking with the `'-G'` linker option. Without these options, the AIX linker would not automatically link to any shared object with a `'.so'` extension. Also, the R main program would be unable to dynamically load modules (such as X11) with the `dlopen` call.

When setting `MAIN_LDFLAGS` and `SHLIB_LDFLAGS` accordingly, note that linker flags must be escaped using `'-Wl,'` if `gcc` is used for linking: use `'MAIN_LDFLAGS="-Wl,brtl"'` and `'SHLIB_LDFLAGS="-Wl,-G"'` in this case.

Tim Hoar <thoar at cgd dot ucar dot edu> reported success building R 1.9.0 under AIX 5.1 in 64-bit mode with

```

OBJECT_MODE=64
CC=/usr/bin/xlc_r
F77=/usr/bin/xlf_r
CXX=/usr/bin/xlC_r
LDFLAGS='-brtl'
CFLAGS='-O -qstrict'
FFLAGS='-O -qstrict'
CXXFLAGS='-O -qstrict'

```

and the X11() device worked. [His system required the 'Makeconf' file to be edited to replace '/lib/crt0.o' by '/lib/crt0_64.o' in R_XTRA_LIBS, but `configure` now tries to detect this.]

Paul Boutros reported success building R 2.0.1 under AIX 5.2 with gcc 3.3.2 using

```

OBJECT_MODE=64
MAIN_LDFLAGS=-Wl,-brtl
SHLIB_LDFLAGS=-Wl,-G

```

(note it is Wl (W ell) not W1 (W one)).

We understand that '--enable-R-shlib' does not work under AIX.

Ei-ji Nakama has reported that 'src/main/regex.c' in R 2.2.0 has to be compiled with '-D_LINUX_SOURCE_COMPAT' in CFLAGS in AIX 5.x (and does not work correctly with AIX 4.x). We believe that the patch applied since obviates the need for this. He tested R 2.2.1 beta on AIX 5.2.

Pre-release versions of R 2.3.0 have been built on AIX 5.3. (This defines the macro 'SOCKLEN_T' and so earlier versions of R cannot be compiled on that version of AIX.)

C.10 New platforms

There are a number of sources of problems when installing R on a new hardware/OS platform. These include

Floating Point Arithmetic: R requires arithmetic compliant with IEC 60559, also known as IEEE 754. This mandates the use of plus and minus infinity and NaN (not a number) as well as specific details of rounding. Although almost all current FPUs can support this, selecting such support can be a pain. The problem is that there is no agreement on how to set the signalling behaviour; Sun/Sparc, SGI/IRIX and 'ix86' Linux require no special action, FreeBSD requires a call to (the macro) `fpsetmask(0)` and OSF1 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.)

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 Building the GNOME console

This interface is experimental and incomplete. The console offers a basic command line editing and history mechanism, along with tool and button bars that give a point-and-click console to some R commands. Many of the features of the console are currently stubs, and the console is no longer under development: it has been kept available as an example of adding a front-end to R.

A `gtk()` graphics device (a port of the `x11()` device to GDK (the GIMP Drawing Kit)) is available from CRAN as package **gtkDevice**: this cooperates rather better with the console than the `x11()` device.

The sources for the GNOME console for R are now available as package **gnomeGUI** on CRAN and *via* Subversion by

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

You need to have built R first with the `--enable-R-shlib` option, and installed R to where you are going to use it from.

Please check you have all the requirements. You need at least the following packages or later installed:

```
audiofile-0.2.1
esound-0.2.23
glib-1.2.10
gtk+-1.2.10
imlib-1.9.10
ORBit-0.5.12
gnome-libs-1.4.1.2
libxml-1.8.16
libglade-0.17
```

It is preferable to have a complete installation of the GNOME desktop environment. If you use Linux, then this should be provided with your distribution. (Note: `libglade2` is not a suitable version of `libglade`.)

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`. If you use a pre-packaged version of GNOME then you must have the developer versions of the above packages in order to compile the R-GNOME console.

It is possible to install the front-end in the same way as an R package, via `R CMD INSTALL` or `install.packages`.

For greater control, it can be configured and built independently of R. Create a build directory, and from there run

```
/path/to/gnomeGUI/configure R_HOME=/path/to/R/installation
make
make install
```

This installs the two files `bin/exec/Rgnome` and `share/glade/gnome-interface.glade` in `R_HOME`.

The full list of options to this `configure` is

```
'R_HOME'    the directory (containing 'bin/R') of the R installation
'--with-gnome'
              specify the prefix for the GNOME dirs
```

```
'--with-gnome-includes=DIR'  
    specify location of GNOME headers  
'--with-gnome-libs=DIR'  
    specify location of GNOME libs  
'--with-libglade-config=LIBGLADE_CONFIG'  
    specify location of libglade-config
```

How to invoke the GNOME console is described under [section “Invoking R”](#) in *R Introduction and Administration*: in brief, use command-line option ‘--gui=gnome’.

Appendix E 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 (although that will fail for earlier versions of Java such as the Microsoft JVM which do work with R). Java 1.1 is sufficient.

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.

E.1 Java Virtual Machines on Linux

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

This and `j2re 1.4.2_01` do work: the latter can be found in Sun's archive at java.sun.com/products/archive/.

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

Other useful links are for Mozilla, plugindoc.mozdev.org/faqs/java.html and www.mozilla.org/releases/mozilla1.7/installation-extras.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/.

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) but 32-bit browsers have been used on that platform, and a Blackdown Java plugin is available.

E.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 Solaris, whereas `jre 1.5.0` and `j2re 1.4.2_01` (available from java.sun.com/products/archive/) do.

E.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 1.5.0` works in Internet Explorer, Netscape 7.x, Mozilla 1.6/7 and Mozilla FireFox on Windows XP. Note that a recent Windows system may not have a JVM installed at all. For Netscape/Mozilla/FireFox 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*.

E.4 Java Virtual Machines on Mac OS X

The HTML search engine does not work with Safari under Mac OS X, but `j2re 1.4.x` may work with Mozilla, Firefox and Camino if the Java Embedding Plugin (javaplugin.sourceforge.net) is used.

The Aqua GUI provides an interface to `help.search` that may substitute for the Java search.

Appendix F The Windows toolset

If you want to build R from the sources 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.

Some of these tools are also necessary for building add-on packages from source. (Most Windows users will not need to do that; see [Chapter 6 \[Add-on packages\]](#), [page 14](#) for details.) We have found that the build process for R is quite sensitive to the choice of tools: please follow our instructions **exactly**, even to the choice of particular versions of the tools. 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.

You will certainly need the following items to produce a working copy of R. See the subsections below for detailed descriptions.

- The command line tools
- Perl
- The MinGW compilers

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-2.3.0.exe’ will also need the following:

- The Microsoft HTML Help Workshop
- L^AT_EX
- The Inno Setup installer

Your path should include ‘.’ first, then the ‘bin’ directories of the tools, perl, minGW, and L^AT_EX, 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). 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.

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 is safest not to have any other version of the Cygwin dlls in your path.

F.1 The command line tools

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`, `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). We have packaged a set of all of these tools at www.murdoch-sutherland.com/Rtools/tools.zip.

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 dll. 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.

F.2 Perl

You will need the Windows port of `perl5`. A package containing this is available from www.activestate.com/Products/ActivePerl.

Beware: you do need the *Windows* port and not the Cygwin one.

F.3 The MinGW compilers

You need a recent version of the MinGW port of `gcc` from www.mingw.org. See the notes on www.murdoch-sutherland.com/Rtools for updates.

In late 2005 there was an installer `MinGW-5.0.0.exe` in the ‘Proposed’ section. Using this, select the ‘Candidate’ package and add the ‘g++’ and ‘g77’ compilers to the ‘MinGW base tools’: components will be downloaded from Sourceforge and unpacked. Note that you do need the ‘Candidate’ compilers (`gcc-3.4.5`).

An alternative is to download the components individually, currently

```
gcc-core-3.4.5-20051220-1.tar.gz
gcc-g++-3.4.5-20051220-1.tar.gz
gcc-g77-3.4.5-20051220-1.tar.gz
binutils-2.16.91-20050827-1.tar.gz
mingw-runtime-3.9.tar.gz
w32api-3.6.tar.gz
```

and unpack these into the same directory (using `tar xzf tarball_name`). Note that `mingw-runtime-3.8.tar.gz` or later is required to build R, `gcc-g77-3.4.4` or later to build some contributed packages, and `gcc-3.4.5` is needed to get a correct build of R itself.

F.4 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.

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

To skip building compiled html help, set `WINHELP=NO` in ‘MkRules’. In this case the Help Workshop will not be needed.

F.5 L^AT_EX

The ‘MiKTeX’ (www.miktex.org) distribution of L^AT_EX includes a suitable port of `pdftex`, as does the ‘ProText’ (www.tug.org/protext) enhanced version. We have also used the ‘fptex’ distribution (available from any CTAN node such as www.ctan.org/tex-archive/systems/windows/fptex and as part of TexLive 2005 (www.tug.org/texlive)). Another actively-maintained version is available from www.fsci.fuk.kindai.ac.jp/kakuto/win32-ptex.

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.

F.6 The Inno Setup installer

To make the installer package (`R-2.3.0.exe`) we require Inno Setup 5.1.5 or later from jrsoftware.org.

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

Function and variable index

C

`configure` 3, 5, 27, 28

H

`HELP` 15

`HELPTYPES` 15

I

`install.packages` 14

M

`make` 28

`MakeDll` 15

`Makevars.win` 15

R

`R_HOME` 3

`remove.packages` 16

U

`update.packages` 16

W

`WINHELP` 15

Concept index

A

AIX 37

B

BLAS library 23, 28, 34, 35

F

FORTRAN 28

H

Help pages 4

HP-UX 35

I

Installation 5

Installing under Unix-alikes 3

Installing under Windows 8

Internationalization 18

IRIX 36

L

LAPACK library 24, 34, 35

Linux 3, 32

Locale 18

Localization 18

M

Mac OS X 3, 12, 34

Manuals 4

Manuals, installing 6

O

Obtaining R 1

P

Packages 14

Packages, installing 14

Packages, removing 16

Packages, updating 16

R

Rbitmap.dll 9

Repositories 17

S

Solaris 34

Sources for R 1

Subversion 1, 4, 22, 40

V

Vignettes 4, 22