# Installation of CCP4 Version 4.2

**Current Version CCP4 4.2.2 November 2002**

## Contents



# Requirements

**Compilers**
Fortran and C compilers are needed to build the suite.

**Disk Space**
To uncompress the standard distribution requires ~290Mbytes.

Disk requirements for the fully built and installed suite depends on your system and particularly if you use shared libraries. An IRIX6.5 system with shared libraries and X-windows software requires ~250Mbytes, though this can be reduced to below 200Mbytes after installation by performing *make realclean* (see "Reducing Disk Space Requirements for Executables").
A minumum cut-down version of the suite (consisting of the compressed tarred files for ccp4-main, lib, include, doc, src and etc) will require ~16Mbytes to unpack the distribution.

Some programs use a lot of temporary scratch space to execute - you will need 10s of MBytes free in the directory pointed at by the environment variable $CCP4_SCR.

# Downloading the Software

Decide where you want to put the software. This might be */usr/local* (following usual Unix convention) or */public/xtal* (mirroring the traditional VMS crystallography setup) - this directory is later referred to by the environment variable *$CCP4_MASTER*.

FTP the compressed tar file *ccp4-4.2.tar.gz* to this directory (or see Alternative Compression Formats)

Uncompress the gzipped file :

**> gzip -dc ccp4-4.2.tar.gz | tar xvf -**

This will create a new *ccp4-4.2* directory, in turn containing about ten new directories (see Directory Structure for details).

# Building the Suite on Unix/Linux

## Editing the Setup File

If you have a previously installed CCP4 Suite it is worthwhile looking at the old *$CCP4/include/ccp4.setup* as it may save re-editing the new file (but do a comparison to check there are no new features - Version 4.2 does have some significant changes).

The setup file is sourced by users (typically from their `.cshrc` file) to create the environment to run CCP4 programs. You will need to customise this file for your site.

**> cd ccp4-4.2/include**

**> cp ccp4.setup-dist ccp4.setup** (to make two copies of the file - one is left unchanged for reference).

Edit ccp4.setup following the instructions in the comments - the minimum that you will need to do is:

- Set environment variables for *CCP4_MASTER* and *CCP4* - all other environment variables should follow correctly from these except:
- Environment variables for the scratch directories *CCP4_SCR* and *BINSORT_SCR* (see Scratch Directories).
- The environment variable *CCP4I_TCLTK* pointing to the directory containing Tcl/Tk executables used by CCP4I (the user interface) - (see the separate CCP4I Installation page).

- Setup for *X-Windows programs* which is highly system-dependent (see X-Windows Graphics Programs for details).
- Declaring your *HTTP_PROXY* environment variable (Optional). Otherwise, leave commented out.

Source this file - the setting of the environment variables is necessary for the rest of the installation:

**> source ccp4.setup**

*NB* if you are using the `bash` shell (which is common on some Linux systems) then you will need to create your ccp4.setup file from the *$CCP4/include/ccp4.setup-bash* file instead (the rest of the instructions above still apply).

If you are unsure whether you are using `bash` then use the following command:

**> echo $SHELL**

which will return something like `/bin/bash`.

## Configure and Make

See Introduction to compilers, configure and make if you are a novice installer.

Make sure that you are in the top CCP4 directory and look at a listing of currently supported systems using the *configure help* option (see Configure Systems for more information - essential if you have multiple operating systems). Then start the configure process proper:

**> cd $CCP4**
**> ./configure help**
**> ./configure *system options***

where the *system* is one of those listed by *configure help*, and must be supplied. There are *options* for setting non-default directories, for controlling whether or not certain parts of the Suite are built, and for overcoming possible problems. The *options* that you are most likely use are *--with-shared-lib* and *--with-x*. See Configure Options for details and a full list of options.

If you are running configure for the first time then before the configuration starts a copy of the current CCP4 licence will be displayed and you will be asked whether you agree to the licensing conditions, and to return a completed and signed copy of the licence, if this has not already been done. A text copy of the licence agreement can be found in the file $CCP4/licence.txt; please note that if a copy of the previous license has been returned then there is no need to send a copy of this new agreement, although it is still necessary to agree to the conditions before the configuration can proceed. To agree to the licence and continue with the configuration, enter "y" at the prompt.

When the configuration is complete be sure that you are in the *$CCP4* directory and then run *make* (to build the executables) and *make install* to move executables from the *src* to the *bin* directory (unless an alternative target directory was specified with the *configure --bindir* option).
A reminder for a few people using the *configure --bindir* option: if you intend to reuse the same executables directory as for a previous CCP4 installation then remove the installed executables with

*make uninstall*.

**> cd $CCP4**
**> make** (or **make > make.log** to keep a log file)
**> make install** (or **make install >> make.log** to append to the log file)

It is generally unnecessary but if you want to test the suite before installation look at Testing.

You can save disk space by removing many files which have been moved elsewhere:

**> make realclean**

or, rather than completely deleting files you can leave dummy copies of the executables so that if the sources get updated, make will only rebuild the relevant ones:

**> make clean empty-targets**

If you have problems see Problems.

### Install CCP4 Interface

If you have tcl/tk/blt installed, and have set the environment variables in `ccp4.setup` correctly, then you should be ready to go. Just type `ccp4i`. Information on tcl/tk/blt and on configuring the interface for local preferences can be found here.

## Building the Suite on VMS

The Suite can be obtained by anonymous FTP in the form of Unix-type ‘compressed tar’ (.tar.Z) files. The ftp area contains a directory VMSutils which has source and compiled versions of the compress and tar utilities for VMS if you don’t already have them on your system. First use compress to produce a .tar file and then unpack it using tar in the CCP4_MASTER directory (alternatively, the ftp server can uncompress files on the fly, but at the expense of slower transfer - see the banner information from the server).

You may need some special incantation to ftp these binary files in the correct form in VMS. For instance you will certainly need to be in the binary mode and may also need nmap. VMS ftp implementations seem to vary - consult a local expert if necessary.

The file called *CINCL:CCP4-SETUP.COM* should be executed as a part of your login to set up various logical names and symbols. See the comments in the example version CCP4-SETUP-DIST.COM about what you need to edit after copying it to CINCL:CCP4-SETUP.COM. In particular, the assignment of CCP4_MASTER needs to be edited and the scratch area CCP4_SCR should also be assigned appropriately. After editing this file, execute it (as *@ccp4-setup*) before compiling the suite, as follows.

The CCP4 sub-directories [LIB.CCIF], [LIB.SRC], [LIB.DATA], [SRC] and [UNSUPPORTED.SRC] contain source which needs to be compiled. Each sub-directory contains a file MAKEFILE.COM (or MAKE.COM in the case of [LIB.CCIF]) to run the compilation. You might want to change these to alter the compilation options, *e.g.* to add debugging information. MAKEFILE.COM in the CCP4 directory can be used to control compilation in the sub-directories. Thus, having executed CCP4-SETUP.COM, go to the top of the distribution using `set def ccp4` and just type @*makefile* to build the library and the programs, putting them into the CBIN directory. This will build the 'unsupported' programs; if you don't want these, modify the .com file appropriately or run @*makefile* in the appropriate directories individually. If you now execute CCP4-SETUP.COM again, it will search the CBIN directory for executables and set up symbols for each. You should now be able to execute the programs as in the examples.

# Installing Pre-Built CCP4

Pre-compiled binary versions of the programs are available for some platforms. It is still necessary to obtain the source code distribution, which contains various data files required by the programs at run-time, and to edit and source the setup file, see above. The set-up script **BINARY.setup** must also be run to complete the installation.

The procedure is as follows:

1. Download and unpack the main distribution, see "Downloading the Software"
2. Edit and source the ccp4.setup file, see "Editing the Setup File"
3. Download the appropriate binaries for your system and unpack in the top CCP4 directory *i.e.* *$CCP4*. This should create the **bin** subdirectory containing the executables.
4. Run the *BINARY.setup* script in *$CCP4* to create binary data file and complete the installation.

The *$CEXAM/unix/runnable/run-all* script can be run to test the installation, see Testing the Suite.

# Appendix A - Directory Structure

The directory structure of the CCP4 distribution under the top **ccp4-<*version-number*>** directory:

**bin** default destination for installed program binaries
**ccp4i** CCP4 graphical user interface
**doc** plain text documentation files (generated from the html files)
**etc** Unix shell scripts and VMS DCL procedures
**examples**
  example scripts/DCL procedures
  **data** miscellaneous data required by some of the runnable example scripts
  **toxd** example data and model coordinates for a small protein (alpha-dendrotoxin from green mamba venom: "toxd")
  **rnase** example data and model coordinates for Ribonuclease from Streptomyces aureofaciens: "rnase")
  **unix**
    **non-runnable**
      scripts which can't be run for want of data
    **runnable**
      scripts which can be run with the data in toxd or rnase
  **vms** some DCL procedures

**tutorial2000**
tutorial material for using the CCP4 suite for Data Processing, MAD phasing, Molecular Replacement and Refinement
**data** data files with which to run the tutorials
**html** instructions on running the tutorials
**results**
results files from the tutorials

**html** library and program documentation in HTML form

**include**
setup scripts and the .def files needed for program startup

**lib** default destination for installation of the binary library files
**ccif** C library for low level operations on mmCIF files
**data** machine-independent ‘library’ files such as the symmetry operators
**lapack**
**blas/src**
source code for netlib reference BLAS maths routines
**src** source code for netlib version of LAPACK maths routines
**rxdispencer**
a modified version of Henry Spencer’s regular expression library
**src** source code for the CCP4 library

**manual**
LATEX source for the manual and corresponding PostScript file

**src** program source

**unsupported**
source for the ‘unsupported’ programs

**x-windows**
X-Windows programs (xloggraph *etc.*)

# Appendix B: Alternative Compression Formats

We believe contemporary Unix/Linux systems all support gzip compression - if yours does not then:

a) Let us know or we are liable to stop distributing alternatives.
b) FTP the ccp4-all.tar.Z file and uncompress:

```
> zcat ccp4-all.tar.gz | tar xf -
```

# Appendix C: Introduction to compilers, configure and make

CCP4 programs are written in either the C or the Fortran programming languages (and the source files are called either *foo.c* or *foo.f*) and so need either the C compiler (usual called *cc*) or Fortran compiler (usually called *f77*) to compile and link them. Compilers are really just programs which process the code to make other programs. ’Compiling’ is converting the program source code (human readable) into machine code (computer readable) in an object file called *foo.o*.

Large programs are written in several separate source files and so will be made up from several object files. Programs also require some standard 'libraries' of commonly used utilities - these are similar to object file and have file names like *foo.a*. There are some 'system' libraries which are a standard part of the operating system of your computer and some CCP4 libraries such as mtzlib and symlib which provide utilities to handle MTZ files and symmetry which are used by many programs. The CCP4 libraries also start out as source code in $CCP4/lib/src and need compiling too. The next stage after compiling is *linking* (this is still done by the 'compiler') which is pulling together all the object files and libraries needed for the program to work and creating the 'executable' file (usually just called *foo*).

There is an alternative approach to linking programs with the libraries and that is using 'shared libraries'. Linking programs to the default archive library `foo.a` involves including the necessary library routines in the program executable. Over the suite, therefore, the same code is included many times, and a lot of extra disk space is used. In contrast, when a program is linked to a shared library `foo.so` (created from `foo.a` by the link editor *ld*) only a reference to the library routine is included - it is not actually loaded until runtime. This means that the program executable takes up less disk space, but also that the program must be able to find the library routine at runtime. For this to work requires that the environment variable LD_LIBRARY_PATH is set correctly. This variable is a list of the directories that should be searched to find shared object libraries; it may include the names of several 'system' libraries or libraries for non-CCP4 programs but should also include *$CCP4/lib*.

The process of compiling and linking a complex system such as the CCP4 Suite is best done by a utility called *make* which interprets the instructions in a *makefile*. The great usefulness of *make* is that the system 'understands' the dependencies between the various levels of the programming system so that, for example, if you change one file which is used to build many different programs than the make procedure 'knows' to recompile and relink all the programs that are affected by the change. The *make* procedure will also control other necessary functions such as installing the software (which is basically moving files from where they are first built to where you want them to be used) or any configuring of data or documentation files. It will also clean up by deleting intermediate files.

It is usual in a complex program system to have several makefiles, one in each directory, always called Makefile. Within CCP4 there are separate Makefiles in several directories. There is a master Makefile in the $CCP4 directory which 'calls' all the other makefiles and so to run the make should only require typing the command *make* in the $CCP4 directory - this command will, by default, use the file called Makefile. The make utility can be got to perform some of its other functions by commands such as *make install* or *make clean*.

There is one more important part of the process, *configure*. The differences in operating systems and user requirements means it is impossible to write a makefile which would work for everybody. The way round this is to edit the makefile before it is run to customise it for your system. This process is done by a script called *configure* (which is in $CCP4 directory) which converts a standard initial file, *Makefile.in*, into a customised *Makefile* for your system. The *configure* process will do a lot of automatic checks on your system (do you have compilers? what system libraries are present?) and has a large number of options for you to set your own requirements.

So the minimal instructions you will actually need to type for a standard Unix installation are the trilogy:

**> configure**
**> make**
**> make install**

(but see Building the Suite on Unix/Linux before trying it on CCP4!).

# Appendix D: Configure Systems

If you wish to do multiple installations for multiple machine types, then you should run the script *duptree* first, which is in the $CCP4 directory. This will produce a further script called *duptree_ccp4* containing information on the source directory tree. This second script should then be run once for each machine type - in each case a duplicate source tree is created with soft links to the original source. The *configure* script should then be run within each duplicate source tree. The script *duptree* contains further details.

If your system does not appear on the list of supported systems for configure then use generic.

Note that the latest IRIXs support both 32-bit and 64-bit ABIs, and two systems for configure are provided to cover this, namely **irix** and **irix64**.

# Appendix E: Configure Options

The main configuration parameters (apart from the system type) are controlled by command-line flags to configure of the form

> **--<flag>=<value>**

where **<flag>** may be abbreviated uniquely and = may be replaced by whitespace. The currently-implemented flags of this form are are:

**--srcdir**
: The CCP4 source directory (referred to as $CCP4)

**--bindir**
: Where the program binaries get installed

**--libdir**
: Where the library files get installed

Other common options are controlled by flags of the form **--with-feature**:

**--with-f2c**
: Use the free f2c compiler rather than a ‘native’ one.

**--with-shared-lib**
: Build shared versions of the libraries libccp4 and libccif, and link against these (this saves disk space, see Reducing Disk Space Requirements).

**--with-x**
: Build the X-Windows-based programs on some platforms.

**--with-rxdispencer**
: Build the distributed version of Henry Spencer’s regular expression library (held in $CCP4/lib/rxdispencer). Most unix systems should have their own regular expression library, and this option should only be used if this isn’t the case. Currently, the regular expression library is

only needed by the CCIF library.

Other options are:

**--disable-ccif**
: Disable installation of the libccif library (held in $CCP4/lib/ccif). This is purely a safety feature in case the new libccif fails on some systems (it shouldn't!). Disabling the ccif library will mean that data harvesting won't work for some programs. One likely cause of failure of the CCIF build is the lack of a local regular expression package, in which case check out the --with-rxdispencer option first.

**--disable-lapack**
: Disable configuration of the LAPACK linear algebra package. By default configure will search for LAPACK libraries already installed in some form on the system, if none are found then otherwise it will build the netlib 'reference' libraries which are distributed with the suite (in $CCP4/lib/lapack).
To force building of the reference LAPACK libraries regardless of those already on the system, use the **--with-netlib-lapack** option. Disabling the LAPACK is NOT RECOMMENDED as the compilation of a number of programs now use the libraries.
See the MODLIB documentation for more details about LAPACK.

**--disable-mapslicer**
: Disable installation of the MapSlicer Tcl command library. Without this library the MapSlicer viewer will not work.

**--with-warnings**
: Include extra compilation flags which output all compiler warnings (normally compilation is silent). This option may be useful for some developers.

**--non_shared**
: Don't allow linking against shared libraries (including system libraries) when building the program executables. This results in significantly larger executables but does make them more portable. Not compatible with **--with-shared-lib**.
*N.b. This option is only implemented for OSF1 systems.*

**--onlylib**
: Build only the object libraries and not the CCP4 programs (See Installing only the Object Library).

configure currently assumes that the shareable (plain-text) library files remain in $CCP4/lib/data rather than being installed elsewhere.

If the build is not done in $CCP4, configure creates src and lib directories mirroring those in $CCP4 and does the compilations there.

Configure examines the environment variables necessary for running the suite as defined by ccp4.setup and tries to check them as far as reasonable, but these are not actually used in the build process. There are a number of configuration parameters that you don't usually want to change, but whose defaults can be over-ridden by the values of environment variables when configure is run:

**CC**
: the C compiler (default is system's native compiler; another possibility might be gcc )

**COPTIM**
: C compiler optimisation flags (default is the highest optimisation which is considered safe or

unagressive)

**XCFLAGS**
: any extra flags you need to give to the C compiler, apart from those for optimisation (these are system dependent)

**FC**
: the Fortran compiler (default is system's native compiler); there isn't currently any support for other than native compilers

**FOPTIM**
: Fortran compiler optimisation flags. The default varies, but will have no debugging extras. In some cases (notably IRIX), the default is used because there are so many problems encountered with the optimiser. You may want to try higher optimisation levels and see if there's a significant performance improvement and that the code still works

**XFFLAGS**
: any extra flags necessary for the Fortran compiler, apart from those for optimisation; you might want to change this, for instance to make smaller binaries at the expense of debugging ability

**XLDFLAGS**
: any extra flags needed for `ld`, typically extra libraries

**RANLIB**
: dummy on SysV Unix, ranlib on BSD, depending on whether 'ar -r' builds a symbol table itself

**LNS**
: indicates how to make symbolic links in the filesystem (usually 'ln -s'), else 'cp' to copy rather than link. 'ln' (hard links) could be used in the absence of symbolic links if you don't need to operate across file systems

**M4**
: how to run the m4 macroprocessor and define a symbol to indicate the system type (see configure source)

**MAKE**
: the name of the 'make' program for use in recursive Makefiles if the system's make doesn't define the symbol MAKE. Usually null

**SETFLAGS**
: set appropriately if individual programs need special Fortran compiler flags (most likely to suppress optimisation) - see the configure source

**INSTALL_PROGRAM**
: a command to install executables. Uses 'install -c' if a BSD version is available (to avoid interfering with running programs), else 'cp'

**INSTALL_DATA**
: a command to install non-executables. If BSD install is available, uses 'install -m 644', else 'cp'

There are corresponding Makefile variables that you can override, *e.g.*:

**make FOPTIM=-g**

Suppose you support several types of system and share the CCP4 source and system-independent data files across a network with a distributed file system such as NFS, AFS or RFS. Then it may be convenient to have a subdirectory in $CCP4 for each system in which to build and, possibly, in which to keep the resulting binaries. Directories for binaries can be anywhere, though.

# Appendix F: X-windows Graphics Programs

The most used X-Windows-based programs are xplot84driver (used to view map sections and other xplot84 graphics files) and xloggraph (used to view graphs in log files but now superceded by loggraph which is part of ccp4i). The X-windows programs will run on SGIs, DECs and Linux but xloggraph will not run on VMS (requires Athena widgets which are not available).

To build X-windows programs you must use the *configure --with-x* option and you must ensure that the programs have access to the resources' relevant application defaults files. This is installation specific - there is an example for Unix users in ccp4.setup.

# Appendix G: Reducing Disk Space Requirements for Executables

The program executables are big, and you may want to make them smaller. There are various possible ways to do this which are system-dependent:

*Increasing the optimisation level of the compiler* might make the executables bigger or smaller. However, under agressive optimisation the programs may not work at all. To do this see either Configure Options for ways to change the configuration parameters or use parameters to *make* , (for example *make FOPTIM=-O* ).

*Using shared libraries* will definitely make executables smaller - use the configure argument *--with-shared-lib* to build a shared version of the CCP4 library to link against. You must make sure you have the correct setup, with LD_LIBRARY_PATH defined, to find the shared CCP4 library at runtime. Beware, using shared libraries has caused problems in the past when upgrading the operating system or moving executables.
All systems supporting the *dlopen* mechanism should be able to use shared libraries.

In extremis you can probably save some space by *removing all debugging information* from the executables using the *strip* program or installing using *install -s* (see Unix man pages for *strip* or *install*). This will prevent diagnosing some problems, though.

# Appendix H: Testing the Suite

If you want to test the suite before installation then it is necessary to have the *$CCP4/src* directory, which currently contains the executables, on the path, for eaxmple, for csh:

**> set path=($CCP4/src:$path)**

You will find two example datasets in *$CCP4/examples/toxd* and *$CCP4/examples/rnase*, and some runnable scripts in *$CCP4/examples/unix/runnable* or *$CCP4/examples/vms*. The script *$CCP4/examples/unix/runnable/run-all* can also be used to automatically run the example scripts in sequence and report any failures.

# Appendix I: Problems Building the Suite

The amount of testing we can do is limited and there may be installation problems particularly on less widely-used systems. If you have a problem..

First check our problem page (http://www.dl.ac.uk/CCP/CCP4/problems.html)

If there is no reported fix then please try to solve the problem - if you suspect it relates to the library try building the library and running the test program for it with *make testlib*. If the problem is in building some of the programs (*make* fails on the *srcdir* target, having done the *libdir* target successfully) you can build as much as possible using:

| | |
|---|---|
| **> cd $CCP4** | |
| **> make -i** | |
| **> cd $CCP4/src** | (or **cd $CCP4/unsupported/src**) |
| **> make instsome** | (This will install the executables which are built and ignore what is missing) |

Finally if you can not solve the problem contact CCP4 (ccp4@ccp4.ac.uk).

If you find a solution please let CCP4 know; sending patches (made with *diff -c*) is very helpful. If you successfully use a configure option which reports itself as ‘untested’ or ‘not properly tested’ please let CCP4 know.

# Appendix J: Scratch Directories

You will need at least 40Mbytes of scratch space to run many of the programs. The conventional Unix scratch area of /tmp or /usr/tmp is rarely setup with this much space. Many programs run more efficiently if the space is local to the machine that they are running on. But a potential problem is that if each machine in your laboratory has its own scratch directory then it is often confusing for users to find files on the scratch directory on a remote machine.

```
setenv CCP4_SCR /scratch
setenv BINSORT_SCR /scratch
```

Alternatively users can just use the subdirectory in their $HOME area so:

```
setenv CCP4_SCR   $HOME/temporary
setenv BINSORT_SCR   $HOME/temporary
```

Note that the user interface, CCP4i, also has the idea of a TEMPORARY directory which by default is the same as CCP4_SCR but this may need modifying for some setups.

# Appendix K: Installing only the Object Library

To build the CCP4 library to support other software systems and avoid the *make* procedure attempting to access the *src* directory use the configure option **--onlylib**. Otherwise follow the same procedure for Building the Suite on Unix/Linux outlined above.

# Appendix L: Installing Updates to Individual Programs

For bug-fix updates distributed as a patch file, you should copy this into $CCP4_MASTER and use the patch program to apply them. There is a short guide to applying patches on the CCP4 web site.

To rebuild individual programs - put the new source code in $CCP4/src

**> cd $CCP4/src**
**> make *foo***

Alternatively if you used *make empty-targets* when you did the original installation then a simple *make* will just rebuild the necessary. When you have installed the changes to your satisfaction remove the .orig files left behind by patch.

**> make clean**

# Appendix M: Installing Updates to the Whole Suite

Patch releases of the whole suite may be made from time to time - these collect together all bug-fix updates made since the previous release, and do not introduce any new functionality to the programs.

Patch releases are made available in a number of ways:

1. as complete new source code in a compressed tar file;
2. as precompiled binaries;
3. as a source code patch file.

If downloading the suite for the first time it is best to take the most recent available version. Otherwise, the source code patch file can be applied to a previously installed copy of the suite as follows:

| | |
|---|---|
| **> cp *patchfile* $CCP4** | (Move the patch file to the top CCP4 directory) |
| **> cd $CCP4** | |
| **> patch -p1 -N -i *patchfile* >& patch.log** | (Apply the patch to the source code using the patch program, and save the output to a log file) |

If you wish to rebuild any updated libraries and programs then you further need to:

| | |
|---|---|
| **> ./config.status** | (Rerun configure with the original settings to regenerate Makefiles) |
| **> make [install]** | (Rebuild [and install] any updated programs) |

Finally, `patch` generates files with the extension *.orig* (one *.orig* file for each patched file). These files can be removed by running **make clean** afterwards.

If using the precompiled binaries you can simply download the latest executables as outlined in the section on Installing Pre-Built CCP4. (Please note that it is still a good idea to apply the source code patch in this case, as this will address bugs in the uncompiled elements of the suite, such as CCP4i and the documentation.)

Finally, we recommend that you read the notes below before applying patches for the first time, and to report any problems with the procedure to CCP4 (ccp4@ccp4.ac.uk).

### Notes on applying patches to the whole suite

i. Patch files from CCP4 will have names of the form `ccp4-`*`oldversion-newversion`*`.diff`, which indicates that the patch will upgrade version *old-version* to version *new-version*. Applying a patch file to the wrong version will have unpredictable (and most likely undesirable) results.
ii. It is intended that patches should be *incremental*, so if you have *e.g.* CCP4 4.2 and wish to upgrade to 4.2.2 via the patches, then it will be necessary to first apply the upgrade from 4.2 to 4.2.1, and then that for 4.2.1 to 4.2.2. (It is not necessary to rebuild the suite inbetween applying the patches however.)
iii. The patch files cannot upgrade binary files (such as MTZ data files, images, and precompiled Java files) which are included in the distribution.
iv. It is sensible to keep a log file from the patching procedure (see instructions above), in case the patching is unsuccessful. In the event of a problem, please send the details along with the log file to ccp4@ccp4.ac.uk.
v. To reverse the patch follow the instructions above, including an additional **-R** option on the `patch` command.

## Appendix N: Uninstalling the Suite

If it is necessary or desirable to undo the installation and start again from scratch (for example you wish to run the configuration with different options) then the following prescription should restore the original distribution:

| | |
|---|---|
| **> cd $CCP4** | |
| **> make uninstall** | (Delete all the installed files, e.g. libraries in **$CCP4_LIB** and binaries in **$CBIN**) |
| **> make distclean** | (Delete all the files created in configuring and building the suite, including the Makefiles) |

It will then be necessary to re-run configure etc as if starting from scratch. This procedure should also work for individual sections of the suite, for example if you want to uninstall and remake the libraries then **cd $CLIBS** instead before executing *make uninstall* and *make distclean*.