CCP4 Release Handbook

Version of 3rd October 2008

CCP4 Release Handbook: Contents

1 Introduction

This document is intended to be a reference manual for making releases of the CCP4 software suite. It is based on various notes accumulated over the years from releases made by the staff at CCLRC Daresbury Laboratory.

1.1 Aims of this document

The aim of this document is to capture key knowledge regarding making releases of the CCP4 software suite, and enable the procedures to be followed by any CCP4 staff member.

The procedures described in this handbook should reflect the process followed for the last release. However the procedures can change between releases as components and ways of doing things also change. Therefore this document will evolve over time.

2 Overview of the release process

Making releases of the CCP4 software suite has long ceased to be a trivial proposition, and much work needs to be performed in order to update the software and ensure that it is ready for release to the user community. The steps involved in doing this include gathering new and updated programs and integrating them into the suite, updating interfaces, adding new features or functions, updating the build system and the documentation, and performing testing. The management and performance of these tasks may vary from project to project, and are outside the scope of this document.

This document is focused on the practical steps that need to be taken in the process of making a new release of the software available to the user community, and which are true regardless of the content of the released software. These steps are only relevant once the major tasks outlined above are completed.

At this stage, certain routine tasks should be performed. Where possible these tasks should be automated. Over time tasks will change, some tasks will no longer be relevant while new tasks or procedures will be identified.

The basic steps are:

1. Prepare the release files by making sure that all version numbers, installation documents and so on are consistent and up-to-date.
2. Tag the components in CVS
3. Export the components and make them available
4. Make binaries for Unix/Linux platforms
5. Update the download area on the server
6. Perform tidying up and review

3 Before starting the release process

There are some things which need to be done before starting on the release process.

- Update the supporting tools
- Add placeholders for new documents

3.1 Ongoing activities during the release process

The following general components should also be updated although this may turn out to be ongoing:

3.1.1 CCP4i User and Library documentation

This is kept as HTML files in subdirectories of the **`$CCP4I_HELP`** directory.

The following things need to be checked and if necessary updated:

- Ensure that the modules and task lists in the documentation are consistent with those in the interface (this includes the **`$CCP4I_HELP/nav.html`** document as well as module-specific documents in **`$CCP4I_HELP/modules/`**)
- Ensure that links are operational, from both the task interface "Help" button and within the documents
- Ensure that new fields in the "configure interface and "preferences" sections are added to the relevant documentation (**`configure.html`** and **`preferences.html`** respectively in **`$CCP4I_HELP/general/`**).
- Ensure that any new "core" features are also added to the relevant documentation.

Also, changes to the CCP4i "library" functions should be reflected in updates to the files held in **`$CCP4I_HELP/programmers`**, specifically those referenced by the **`$CCP4I_HELP/programmers/prognav.html`** document.

3.1.2 Postscript Manual

This is generated from a set of LaTeX source code files in the $CCP4/manual directory. The substance of this document is not now subject to reviews or updates. The minimal updates usually performed are to check and if necessary update the LaTeX source for the following things:

- The program lists in **`proglist.tex`**
- The dates in **`frontmatter.tex`**
- The CCP4 staff named in **`intro.tex`**

The Postscript manual files themselves (A4 and US letter size) are regenerated as part of the process of building the release archives.

3.1.3 Installation instructions in INSTALL.html

`INSTALL.html` is held in the **`$CCP4`** directory, and contains comprehensive installation instructions for installing the suite in various ways on various platforms, and needs to be kept up-to-date with changes to its structure and configuration.

`INSTALL.html` is the master document from which **`INSTALL`** (the text version) and **`INSTALL.ps`** (the Postscript version) are generated during the process of building the release archives – therefore it is only necessary to keep **`INSTALL.html`** up to date.

3.1.4 BINARY.setup

`BINARY.setup` is the setup script that is run on UNIX systems to complete a binary installation of the suite. It is a cut-down version of the **`configure`** script, which performs all the non-compilation tasks (such as checking for the licence agreement file, creating subdirectories, installing data files and so on) that **`configure`** and **`make`** do for a source code build. **`BINARY.setup`** must therefore be updated to include any changes or additions in this process.

3.1.5 CHANGES and ACKNOWLEDGEMENTS Files

The **`CHANGES`** text file in **`$CCP4`** needs to be kept up to date by recording significant changes (particularly those that are relevant for the end users).

Note that the HTML version in **`$CHTML`** is generated from this text version as part of the process of building the release archive files, and so does not need to be kept up to date.

3.1.6 Examples and Tutorials

The distributed logfiles in **`$CEXAM/unix/runnable`** should be updated for the examples (these are the files with the extension **`.log-dist`**).

The tutorials in **`$CEXAM/tutorial/`** should be checked as part of the release preparation process, and the files in the **`results`** subdirectory also updated if necessary.

3.1.7 C Library documentation

These should be updated from the C code header files, although this process is currently undocumented.

4 Preparing the Release

4.1 Introduction

This section outlines the procedures for updating the suite prior to tagging for release – essentially it is a process of ensuring that version numbers and any generated files are all up to date. More significant updates should already have carried out separately before this stage (although this process may sometimes reveal where this has not been done).

4.2 Procedure

The following procedure is suggested:

- CVS checkout a fresh working copy of the branch that the release will be made from – either the main branch (for a major release) or the appropriate patch branch (for patch releases).
- Optionally, tag all the files in this working copy with a "code freeze" tag.
- Set up the CCP4 environment for the working copy, then
- Update the source code version numbers
- Update the CHANGES files
- Update the INSTALL documents
- Update the program documentation, CCP4i documentation and Postscript manual.

These steps are outlined in the subsequent subsections.

4.3 Create release branch

The purpose of setting up a branch is to separate the frozen developments of the release from the ongoing developments in the trunk of CVS. As such, this is typically only necessary for the ccp4 repository.

It is easiest to do this with the *rtag* instruction. The suggested method is to tag the root of the branch, then to branch from this tag. This makes it easier to merge changes from the branch back into the trunk.

```
> cvs -d <cvs CVSROOT dir> rtag series-X_Y-root ccp4
> cvs -d <cvs CVSROOT dir> rtag -r series-X_Y-root -b series-
X_Y ccp4
```

The precise name is not important but it must be unique within the CVS module.

Obviously, everyone needs informed that the branching has occurred.

To work on the branch checkout using the branch name.

```
> cvs -d <cvs CVSROOT dir> co -r series-X_Y ccp4
```

4.4 Acquiring the tools for patch releases

For patch releases, the tools required for updating the release files may be missing from the checked out version. This is because these tools are usually tagged as dont_release_me in CVS and are therefore excluded from the patch branch by default). This shouldn't be an issue for major releases.

During the process, where the tools are missing it is necessary to acquire them by checking out the individual file from the main CVS branch, for example:

```
> cvs checkout -r HEAD <missing_tool_file>
```

The rest of the process should otherwise be the same.

4.5 Update source code version numbers

- **Core library**
  In **$CLIBS/ccp4_program.h**, update the macros CCP4_VERSION_NO and CCP4_PATCH_LEVEL to the correct numbers.
  Note that only CCP4_PATCH_LEVEL is updated for patch releases.

- **CCP4i**
  In **$CCP4I_TOP/bin/*OPSYS*/startup.tcl** (where ***OPSYS*** is **UNIX** or **WINDOWS**), increment the most minor CCP4i version number & ensure that the CCP4 version number matches that of the core library.

- **Setup files**
  In the **$CINCL** directory, update the distributed setup files so that the CCP4 environment variable will point to the correct top-level directory (e.g. **ccp4-6.0.2** for the 6.0.2 release). The files are:
  - **ccp4.setup-dist**
  - **ccp4.setup-bash**
  - **ccp4.setup-sh**
  - **ccp4.setup-zsh**

- **agree2ccp4v*** in **configure** and **BINARY.setup**
  agree2ccp4v* is a file that is created by either configure or BINARY.setup, whenever the installing user agrees to the CCP4 licence agreement, and then updated each time a new version is installed.
  For releases with a new licence, the trailing number i.e. the * must be updated in both files.
  For all releases, the version number that is written to the file must also be updated to reflect the current release.

- **Patch_ccp4.sh script**
  Update the version number to point to the correct FTP directory where patch files will be made available after the release.

4.6 Update the CHANGES files

The ASCII text **$CCP4/CHANGES** file is the master changes document; the **$CHTML/CHANGESinVx_y.html** is derived from it.

x and **y** in the HTML file name are the major and minor release version numbers respectively. Use the **make_CHANGES.tcl** script to generate this file, for example:

```
> make_CHANGES.tcl -i CHANGES -o CHANGESinVx_y.html -v 6.0.2
```

See also the section on "Tools" in Appendix B.

Note that for patch releases the convention is that the patch revision number is not added to the name of the **CHANGESinVx_y.html**.

4.7 Update INSTALL documents

Update the **$CCP4/INSTALL.html** and the derived documents **INSTALL** and **INSTALL.ps**

- **INSTALL.html** is the master document. Things to be updated include version numbers, dates, file sizes for download and installation, components, procedures for installation etc
- **INSTALL** is the text version of **INSTALL.html**. It can be generated using the **lynx** text-based HTML browser, e.g. with the command

  ```
  > lynx [-cfg lynx.cfg] -nolist -dump INSTALL.html > INSTALL
  ```

- **INSTALL.ps** is the Postscript version. It can be generated using the **html2ps** program, e.g.

  ```
  > html2ps -n -u INSTALL.html > INSTALL.ps
  ```

- **$CCP4/ccp4i_installation.html**: update the version number of the interface, and of the suite.
- **$CCP4/README**: update the version number, the name of the HTML **CHANGES** file, and check other details for correctness.

4.8 Documentation

4.8.1 HTML Documentation

Before rebuilding the documentation as described in the following steps, ensure that the following tools are available in the $CHTML directory:

- makeindices.pl
- makereference
- Makefile

The makeindices and makereference scripts should also be updated with the appropriate version numbers.

Then, in the **$CHTML** directory:

- Run "**make indices**" to remake the files **INDEX.html**, **FUNCTION.html** and **whatis**
- Run "**make reference**" to remake **REFERENCES.html**

After running these steps, view the HTML pages in a web browser for a cursory check that the formatting is okay, and do CVS diff to check that any changes since the last update look reasonable.

Where programs appear to have been removed, check whether this is really the case – sometimes the files have been updated and the INDEX HTML comment has been corrupted or lost. This needs to be fixed before proceeding.

4.8.2 Man-page style documentation

The man-page style documentation is held in the **$CDOC** directory and is created entirely from the HTML documents in **$CHTML**. It requires that the **$CDOC/Makefile** is present and that the **lynx** program is available (see Appendix C). It may also be necessary to edit the Makefile to correct the command for running **lynx**.

Then: run "**make**" in **$CDOC** to rebuild the .doc files, and check for any new files that are not already in CVS by doing e.g.

```
> cvs status *.doc | grep -i unknown
```

Scanning through the output of this command to identify any new .doc files which correspond to new programs added since the last release. For any new files, these will need to be added to CVS before they can be committed.

Note that certain files are not included as doc files, for example INDEX and FUNCTION. If in doubt, use your judgement – how likely is it that someone will ever think of looking at a REFERENCES document via the man command?

4.8.3 Core CCP4i Documentation

The core CCP4i documentation is generated from the doc-comments in the CCP4i source code files in **$CCP4I_TOP/src** and **$CCP4I_TOP/utils**.

The update process is performed by should by running :

```
> $CCP4/ccp4i/bin/create_docs
```

The CCP4 environment needs to be set prior to doing this step. The documents should then be generated correctly in the **$CCP4I_HELP/programmers/** directory.

4.8.4 Postscript CCP4 Manual

The manual source files are in **`$CCP4/manual/`**. If there have been changes to the LaTex source since the last release of the manual then **`manual.ps`** (A4 paper size) and **`manual_us.ps`** (US letter size) versions of the Postscript document will have to be remade.

In order to do this the **`GNUmakefile`** must be available; then:

To regenerate the A4 format manual:

- Run **`gmake manual.ps`**

To regenerate the US version:

- In manual.tex, uncomment "letterpaper" and comment out "times" (Latex comment character is "%")
- Run **`gmake manual.ps`** then move **`manual.ps`** to **`manual_us.ps`**

4.8.5 Other Documentation and related files

- **Acknowledgements**
  Check that the **`$CHTML/ACKNOWLEDGEMENTS_V*.html`** document has been updated.

4.9 Preparing components outside of core CCP4

Currently there are no additional procedures required for preparing any of the non-core packages (i.e. **`phaser/cctbx`** and **`chooch`**).

4.10 Preparation Checklist

A checklist for preparing the release is at the end of this handbook, see 14.

5 Tagging and Exporting CCP4 Releases

The procedure for tagging and exporting a CCP4 release is following sections.

5.1 Introduction

Once the release files have been prepared in CVS, they can be exported for release. The procedure is simple:

- Tag the files for release with a new CVS tag
- Remove the release tag from files which should not be included
- CVS export the files using the release tag
- Package into **`tar.gz`** files

5.2 Prepare the release

Follow the steps in "Preparing the Release" to ensure that version numbers, documentation and other components have all been brought up to date in CVS.

5.3 Tag the components

If you are using a checked-out working copy of the release files then ensure that all the files are up to date in the CVS branch that you're using. Then CVS tag the working versions of each module included in the release, e.g.:

```
> cd $CCP4
> cvs tag release-<version_no>
```

where ***`<version_no>`*** is the release tag. Since CVS tags cannot contain periods, these are conventionally replaced with underscores in the version number, e.g. version 6.0.2 will be tagged as **`release-6_0_2`**.

This needs to be done for each module (i.e. ccp4, phaser, cctbx and so on) using the same tag for each.

Note: as of 6.1 it is planned that the Balbes database be release as a separate module. It is, therefore, necessary that a different tag be used for this component, and that the release tag not be applied to the database elements within balbes. Currently the database is tagged as *database*.

Note that it is also possible to use the "**`rtag`**" CVS command, which doesn't require a working copy. This is useful for doing something like cctbx, e.g.:

```
> cvs rtag release-<version_no> cctbx
```

See the CVS documentation for more information on **`rtag`**.

5.4 Untag components that shouldn't be released

Not everything in the CVS archive is intended to be released to the general community, for example the supporting tools used for preparing the release should not be included in the archives. To avoid including these files, the release tag is removed by using e.g.:

```
> cvs rtag -r dont_release_me -d release-<version_no>_alpha ccp4
```

These files (for example release tools) which are not intended for public release should already have had the tag "dont_release_me" added, by doing e.g.:

```
> cvs tag dont_release_me <file1> ...
```

5.5 Exporting the files

The following sections outline how to export the files and package them into tar.gz archives. The current procedure that should be followed is the post-6.0 procedure; the pre-6.0 procedure is included here for information only.

5.5.1 Procedure pre-CCP4 6.0

This was the old way of doing things: the tagged version of the ccp4 module was exported manually to the dist directory /ccpdisk/xtal/dist/:

```
> cd /ccpdisk/xtal/dist
> cvs export -r release-<version_no>_alpha ccp4
> mv ccp4 ccp4-<version.no>_alpha
```

where version.no is the release number, e.g. `4.2'

Then the following manual steps were performed:

1. Change permissions on $CETC files to 755:

   ```
   > cd ccp4-<version.no>_alpha/etc
   > chmod 755 *
   > cd ../..
   ```

2. Create a tar.gz file to place on the ftp server:

   ```
   > tar cvf ccp4-<version.n>_alpha.tar ccp4-<version.no>
   > gzip ccp4-<version.no>_alpha.tar
   > mv ccp4-<version.no>_alpha.tar.gz
/ccpdisk/ccp4/public/ftp/pub/pjx
   ```

   Note: avoid using gtar (i.e. GNU tar) as this can cause problems with unpacking the files on systems that use non-GNU versions of tar. See 13 for details of known problems with different versions of tar.

3. Test by transferring from ftp server and running e.g. auto-build-binaries.csh, then do run-all

5.5.2 Procedure for CCP4 6.+

As of CCP4 6.0 the suite has been split into multiple components represented by separate CVS modules (ccp4, phaser, cctbx and chooch). In order to reduce the amount of work and the room for human error, the script **`build_tars.csh`** will generate source code tar archives based on the tags in CVS.

```
> build_tars.csh <version.string> <repository tag>
```

where ***`<version.string>`*** is something like “5.99.5” or “6.0”. This should only be run on dlpx1 due to the problems using GNU tar (see 13).

5.5.3 Gotchas when reusing existing archives

Even if the component files or source code haven’t changed since the previous release, it is necessary to update the archive files which unpack into versioned directories, for example: if the **`phaser`** sources should unpack into **`ccp4-6.0.2/src/`** then the tar file from CCP4 6.0.1 will need to be updated.

5.6 After exporting the files

This section needs reviewing – PJB 12th October 2006

1. Update the "Release Resources" webpage at http://www.ccp4.ac.uk/dev/releases.html
2. Inform the test sites
    - Try "cvs log -N -d ">18 July 2003" to a list of the revisions since a certain date

6 Making Binary Distributions for UNIX Platforms

6.1 Introduction

Binary distributions are produced for each release (major, minor or patch). Note that this chapter only document covers the production of binaries for UNIX and Linux-based platforms – MS Windows and Mac OS-X binary distributions are also produced but the processes for are different.

6.2 Platforms and machines

UNIX binaries are built on the machines available at DL:

Machine name	Operating System
dlpx1	IRIX 6.5
ccp4h	Linux Red Hat 8.0
hpca1	OSF1 V5.1 (retired)
ccp4l	SunOS 5.8 (retired)
ccp4u	Windows
ccb63mb	OS X intel
ccp4m	OS X ppc

The binaries for the first three are built using the build script "auto-build-binaries-6.0.csh". SunOS traditionally (or perversely) has been built manually.

6.3 Preparations

The relevant binary installation file $CCP4/BINARY.setup should already have been updated as part of the main release preparations described earlier in this handbook.

6.4 Building the binaries

The auto-build-binaries-6.0.csh script (part of the auto-build suite, see the Tools section) can build the distribution and create archives with the correct set of files for CCP4 and for Phaser/CCTBX.

Before running the script:

- Ensure that the auto-build/bin directory is on your `PATH` (nb this must be the absolute and not relative path to the bin directory)
- Ensure that you have tclsh version 8.3 or better on your `PATH` (check this by starting tclsh and then typing the command **`puts $tcl_version`**
- On dlpx1: check that you are using the native version of ld (in /usr/bin) rather than the GNU version (in /usr/local/bin), to prevent problems building the libccp4map.so file. This is most easily done by ensuring that /usr/bin appears on your `PATH` before /usr/local/bin.

Then unpack the files to be built. The script builds using the source code archives previously prepared by the export process. Essentially:

- Unpack the CCP4 source code files for the current release, for example for CCP4 6.0.2 this will unpack into a directory called **`ccp4-6.0.2`**
  If Phaser/CCTBX is also required then unpack this into the same directory. If the files are present then the script will automatically build the binary archive for this without additional instructions.

To perform the build:

- Run the **`auto-build-binaries-6.0.csh`** script. This should be done from the parent directory that the CCP4 files were unpacked into.

  The general syntax is:

  `auto-build-binaries-6.0.csh` ***<directory>*** ***<version>*** **`[options…]`**

  where ***<directory>*** is the relevant CCP4 directory (e.g. **`ccp4-6.0.2`**) and ***<version>*** is the version string for this release (e.g. **`6.0.2`**).

  The script supports a number of options, however these are not normally required except for the **`-onlypack`** option (see below).

Upon completion the script will produce a gzipped tar file containing the binary distribution for the CCP4 suite, and (if Phaser and CCTBX source code was also included) a gzipped tar file with Phaser and selected CCTBX components.

- A special case is Linux, where the build must be done twice for the two possible Unicode formats. To build for a particular Unicode format, ensure that the version of Python that is first on the PATH has been built using that format (see below) – there are no other special steps that need to be taken.

As well as information written to standard output while the script is running, it also creates a directory called e.g. **`Linux_build_log`** (the first part of the name will be appropriate to the platform), which contains the logfile output from each stage of the build (configure, make and make install). If there are problems then these logs may be useful.

Once the files are created it is recommended to perform some tests (see below).

Note: **`auto-build-binaries-6.0.csh`** replaced by **`auto-build-binaries-6.1.csh`** that copies the share directory. Also, libxml2 and gc should be built statically.

Gotcha: currently *rapper* will not build on ccp4h using the 4.2.0 gnu compilers, it gives a "TLS" threading error. The build is successful using the 3.4 compilers, so these should be used instead.

6.5 Notes on Phaser and CCTBX for Linux

The autobuild script will automatically build Phaser and the CCTBX libraries if the source code is present, and will also produce an archive file containing the required files. On Linux platforms this includes installation-specific files which enable the Python bindings for Phaser to work (these are not included on other platforms)

The complication with Linux is that the Python installation can have one of two types of Unicode setups: UTF or USC-4. Phaser Python bindings for one setup will not work with the other, so binaries must be produced for each.

The default Python installation is UTF; to build a UCS-4 version specify the **`--enable-unicode=ucs4`** flag for the Python configure when building.

To test whether a particular version of Python is UTF or UCS-4, start the Python interpreter and try the following commands:

```
>>> import sys
>>> print str(sys.maxunicode)
```

If the resulting number is greater than 65536 then it is a UCS-4 installation; otherwise it's a UTF installation.

To build Phaser with a particular Unicode format, ensure that the version of Python that appears first on the `PATH` has been installed with that format.

UTF vs. UCS python: the chroot setup on ccp4h

In order to build both *utf* and *ucs* versions of the cctbx interface a **chroot** arrangement has been setup on ccp4h in /**mnt/chroot**. Currently the main login is *utf* and the chroot is *ucs*, both having python 2.4.2 in /usr/local.

Chroot jail setup:

- **`mkdir /mnt/chroot`**
- **`rsync –av /usr /mnt/chroot`** `repeat other directories which are required`
- **`mkdir /mnt/disk`** `setup /dev without devfs`
- **`mount –o bind / /mnt/disk`**
- **`rsync –ac /mnt/disk/dev /mnt/chroot`**
- **`umount /mnt/disk`**

Chroot usage:

- **`mkdir /mnt/chroot/home`** `mount /home /var /root`
- **`mount –o /home / /mnt/chroot/home`** `repeat for others`
- **`/usr/sbin/chroot /mnt/chroot`**
- **`su <user>`**

6.6 Repacking binary archives without rebuilding the code

In certain circumstances it is useful to be able to get the autobuild script to rebuild the archive files without having to perform the whole build and install process. In this case, run the script with the **-onlypack** option, e.g.:

```
> auto-build-binaries-6.0.csh ccp4-6.0.2 6.0.2 -onlypack
```

This will construct the gzipped tar files from the files that have already been built.

Reusing Makefiles, or reissuing config.status

Options exist to either just rerun the make and allowing the make rules take care of any dependency changes (for instance when the above chroot is being used to build the alternative cctbx interface)

- **auto-build-binaries-6.0.csh ccp4-6.0.2 6.0.2 –usemake**

Or, alternatively the config.status can be rerun

- **auto-build-binaries-6.0.csh ccp4-6.0.2 6.0.2 –useconfig**

6.7 Gotchas when reusing existing archives

In the case that the sources and other files that make up a binary package haven't changed since the last release, it may still be necessary to update the archive file to accommodate changes in the directory into which the binaries will unpack. For example, the phaser binary should unpack into ccp4-6.0.2/bin/ which means that the binary archive for CCP4 6.0.1 will need to be updated even if the program itself is unchanged.

6.8 Naming conventions for binary archives

See the tables in the "Interactive Downloads Pages" section elsewhere in the handbook.

6.9 Testing the binaries

A cursory test is a good idea. Unpack the binaries in a clean directory and set up the environment, then run BINARY.setup. The following is a minimal set of tests to check that nothing major is wrong:

1. $CEXAM/unix/runnable/run-all runs to completion
2. CCP4i launches (optionally also run a Patterson task and generate a map file)
3. MapSlicer runs and can view the map produced in step 2

4. Rasmol launches (e.g. do rasmol $CEXAM/toxd/toxd.pdb). Note that on most platforms this test fails due to the default display depth being incorrect – ignore this failure.
5. Rotgen launches
6. Mosflm starts (n.b. the command to start Mosflm is "ipmosflm")
7. Run the Phaser tests in $CEXAM/unix/runnable:

    - **`phaser_py_test.exam`**
    - **`cctbx_py_test.exam`**
    - **`cctbx_run_tests.exam`**

6.10 Making the binaries available

Once binary files have been prepared the release coordinator should be informed so that the next stage of the release process can occur. A useful central area to put the files onto is the **`/ccpdisk/xtal/dist`** directory.

Making Binary Distributions for OS X

Introduction

OS X runs on two basic CPU architectures, which are not compatable, PowerPC and x86. It is, therefore, necessary to generate binaries for both of these (plus a G5 vs. G4 division on the ppc side). In the long run Apple (and CCP4) will drop support for ppc.

The mac distribution divides the suite into a series of optional packages.

Tools

lipo

The lipo command creates "universal" (multi-architecture) binaries.

lipo [-info] [-detailed_info] [-arch arch_type input_file] ... [
input_file] ... [-arch_blank arch_type] [-create] [-thin arch_type]
[-replace arch_type filename] ... [-remove arch_type] ... [-extract
arch_type] ... [-extract_family arch_type] ... [-output output_file]
[-segalign arch_type value] ...

install_name_tool

This changes the dynamic share library install names (rpath under linux).

install_name_tool [-change old new] ... [-id name] file

relative paths can be setup using @executable_path .

packagemaker

Build can create the installation package (.pkg), metapackage (.mpkg), or distribution (.mpkg) file specified by destination-path.

Helper scripts

Various helper scripts exist to aid the mac build.

overall.py/overall.g4.py – driver scripts
tidy.pl – automates the install_name_tool changes
definitions.py – most of the version numbering
<component>_copy.py – copy the component to the distribution tree

the overall.py script must be run in the *util* directory.

There are various gotchas in that definitions.py and various of the files must be updated for the program names and versions (list.pl outputs a list of directory contents in the correct format). Also, update the compiler versions in core-copy.py and tidy.pl.

Mactel using ifc/icc

Building the binaries.

The suite is build as for any other platform, although I prefer to use dynamic libraries.

Bundling the suite.

- Copy the desired elements of the suite into the separate hierarchies for each package. These will be pointed at by the packagemaker application and form the basis of the .bom files.
- Copy ccp4.setup-sh and ccp4.setup-csh to bin directory in the ccp4 core tree
- Setup directories to contain the Resource files for the metapackage and individual packages. These will be bundled by packagemaker.
    - InstallationCheck, for metapackage
    - VolumeCheck, for metapackage and others
    - postinstall, for metapackage and others
    - postupgrade, for metapackage and others
    - install_name_tool - required for users who have not installed the developers tools (note. The use of relative paths should get around this)
    - English.lproj, - VolumeCheck and InstallationCheck error strings, for metapackage and others
- Bundle the appropriate runtime libraries.
- run install_name_tool over binaries to update rpath elements (this is largely performed by *tidy.pl*)
- set dependencies in ccp4mapwish, in particular remove the numbering from the tcl and tk libraries
- run packagemaker.app to generate the archive/package, and the metapackage
- use Disk Utility to make disk mountable images, .dmg (effectively .tar), archives

Note: to hide packages create a directory within the metapackage and copy the bundled packages to this directory. Next edit the metapackage info.plit to reflect this change of location.

PowerPC (G4 and G5) using xlf/xlc

As for Mactel, but with the following changes.

Preparation

Note. The mp libraries are part of a technical preview, and permission to distribute

these has not been granted.

libxlomp_ser.A.dylib
libxlsmp.A.dylib

simply removing the softlink is sufficient.

Building the binaries.

The creation of G4 binaries on a G5 machine require the use of the –arch and –mtype compiler options.

Further, for the G4 build generate libBLAS.dylib and libLAPACK.dylib, these will be required by earlier systems which do not include vecLib._--with-netlib-lapack builds the static libraries; the object files should be recompiled with -qpic -qnocommon and linked into .dylibs

ld -dylib -dynamic -flat_namespace -undefined suppress -all_load libblas.a -o libBLAS.dylib -lcc_dynamic -ldylib1.o -dylib_current_version 3.0 -dylib_compatibility_version 1.0

ld -dylib -dynamic -flat_namespace -undefined suppress -all_load liblapack.a -o libLAPACK.dylib -lcc_dynamic -ldylib1.o -dylib_current_version 3.0 -dylib_compatibility_version 1.0 -L. -lBLAS

Bundling the suite.

Require the ibm runtime environment.

libxlf90.A.dylib
libxlfmath.A.dylib
libibmc++.A.dylib

7 Interactive Downloads Pages

This section outlines the update and maintenance of the interactive downloads pages.

The first part gives a detailed overview of the structure and content of the pages, the associated files, and the various naming conventions that are used.

These details are most useful for updating and deploying the pages for a release of the suite, and the procedure for doing this is explained in some detail in the second section.

7.1 Overview of the download pages

7.1.1 Introduction

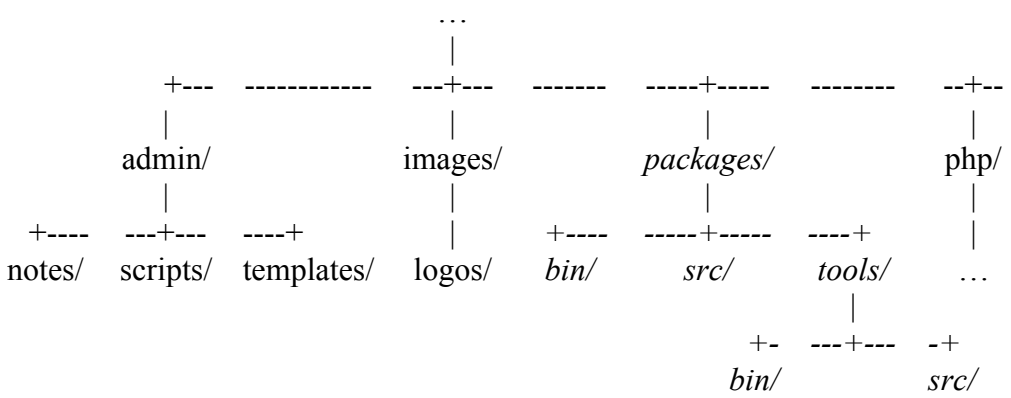
This section gives a general overview of the download pages in terms of the directory structure and contents, PHP scripts, configuration files and configuration and maintenance tools used to deploy the pages for a release.

There are also descriptions of the naming conventions that should be used for directories and files in the packages area.

7.1.2 Structure of the download page area

There are two areas, one for internal testing prior to releases and one external containing the released version. The top-level directory of the test download area is currently in **`/public/ccp4/www/group/download-test/`** on ccp4serv2. This can be considered the "beta staging area". The top-level directory of the official public version is **`/public/ccp4/www/download/`**.

The directory structure looks like this:



The files and directories are part of a CVS module called **`ccp4-download`**, in the subdirectory **`download_beta`**:

- **php** contains the developmental PHP scripts which generate the pages

- **images** contains the various logos for each of the packages that are available for download
- **packages** contains the various component archive files available for download. This is normally soft-linked to `/public2/packages_current`, due to the shortage of disk space elsewhere.
  Within *packages* there are three subdirectories:
  - **bin**: holds the archive files of binaries for each package, arranged by operating system (and operating system subset where appropriate)
  - **src**: holds the archive files of the source code for each package
  - **tools**: holds the archive files of the "tools" (i.e. Tcl/Tk, Python etc). The tools directory is further subdivided into its own bin and src directories, which mirror the arrangement of the packages directory – the platforms referenced in packages/tools/bin should map onto those in packages/bin.
- **admin** contains notes, scripts, templates and data files used in maintaining and updating the pages. These are normally only used when the pages are being configured.

The released area is situated in **`/public/ccp4/www/download/`** on ccp4serv2. It contains only the files that are in the php directory for the test one. The idea is that once you are happy you copy the file from the php directory in the testing area directly in this directory.

7.1.3 Overview of the PHP Scripts and associated configuration files

The scripts in php/public have the following functions:

File name	Function
`downloadman.php`	Starting page. Offers user choice of operating systems. Continue invokes os.php
`os.php`	Package selection. Continue invokes summary.php.
`summary.php`	Summarise the user's choice and warn about possible dependencies. Continue invokes final.php.
`final.php`	Final summary of the user's choice, plus licensing agreement. Continue invokes archive.php. The manifest file is generated at this point.
`archive.php`	Process user's choice (with archive2.php) and bundle install.sh and the generated manifest.sh file. It will also add a record of the user choice and the date in a **`admin/download-tracker.log`** by calling a function declared in script_gen.inc
`archive2.php`	See archive.php.
`declared.inc`	Header file. Sets up PHP arrays which map files in the packages area onto names of operating systems, file sizes and so on. *Note that this is automatically generated using the decgen.tcl script from the contents of the packages directory plus the information in the configuration and template files. It should not be routinely hand-edited.*
`common.inc`	Header file. Sets common PHP variables e.g. directory paths

	and CCP4 version number.
`downloadmanfaq.php`	FAQ page linked from other pages.
`progress.html`	Header file. Common code for drawing the "progress table" at the top of each page.
`browser_det.inc`	Header file. Determine user's browser and operating system.
`footer.html`	Common code for the HTML links which appear at the bottom of each of the PHP pages.
`script_gen.inc`	PHP code that prepares the content of the manifest.sh script for the archive. Script_gen.inc is included in final.php and also the commandline.php script. It also contains the function that records users' choices of downloads for logging purposes.
`install.sh`	The installation script that relies on a manifest file to know what need to be installed. It is bundled with the archives by archive.php
`ccp4_packing.lock`	A simple lock file that gets copied accross during packing and deleted from the packing area once archive is ready. This is used to prevent two user to generate the same combination at the same moment.

There are also a number of configuration files which are used in the generation of the declared.inc file mentioned above. These files are in the **…/admin/templates/** directory and have the following functions:

File name	Function
`dependencies.conf`	For each package, indicates which other package(s) it depends on. Also indicates the minimum version required for packages.
`declared.inc.in`	Template file used to generate the declared.inc file (see above).
`packages.conf`	For each package, contains the text string that is used to describe that package to the user in the download pages e.g. "CCP4 Program Suite v6.0". Changes to version numbers should be made here. Also indicates the name of the image file for the package logo in the "logos" directory.
`platforms.conf`	For each possible platform this has the text string used to describe the platform to the user in the download pages, for example "Red Hat 8.0".

The .conf files contain PHP array names followed by key-value pairs (elements in pairs separated by whitespace) which are used to populate that array. The files also contain comment lines (which start with the hash character #) and blank lines. Both these lines are ignored when the files are processed. Lines in the file starting with "//" (the PHP comment symbol) are echoed verbatim to the PHP code that is generated from the file.

7.1.4 Configuration and maintenance tools

This section outlines the administration and configuration tools used by the download pages.

File name	Function
`decgen.tcl`	Scans the packages area and generates a version of the declared.inc file which is consistent with the contents of that area. This is not normally invoked directly, instead it is wrapped by the update_declared.csh script.
`update_declared.csh`	Wrapper for decgen.tcl which ensures that the script is run in the correct location and that the resulting declared.inc file is moved to the appropriate place.
`export.csh`	Update the publicly visible copies of the web pages.
`loopIt.php`	Generate a set of pre-built download combinations in the cache area.
`checktar.sh`	Script that traverses the cache area on ccp4serv2 and tests the cached archives to ensure that they hold the expected set of files.
`build_tars.csh`	Export source code from the CVS archives. *Note that this script is part of the autobuild area rather than belonging to the download pages.*

7.1.5 Archive file naming conventions in the packages area

The following conventions are used for naming the various archive files. It is important to adhere to these conventions as they are expected by the PHP scripts, the maintenance tools such as the **`decgen.tcl`** script.

Naming conventions for source code archives

File name	Contents
`ccp4-core.tar`	Source code for the core CCP4 suite
`tcl_tk_++.tar`	Source code for Tcl/Tk, BLT, itcl, itk, ...
`Python.tar`	Source code for Python
`phaser-cctbx.tar`	Source code for Phaser and the CCTBX libraries
`chooch.tar`	Source code for Chooch
`clustalw.tar`	Source code for Clustalw
`fasta.tar`	Source code for Fasta
`graphviz.tar`	Source code for Graphviz

Naming conventions for binary archives

File name	Contents
`ccp4-<version>_<platform>.tar`	Binaries for the core CCP4 suite for the specific platform.
`TclTk++bin-<platform>.tar`	Binaries for Tcl/Tk, BLT, itcl, itk, ...
`Python-bin-<platform.tar`	Binaries for Python
`clustalw-bin-<platform>.tar`	Binaries for Clustalw
`Fasta-bin-<platform>.tar`	Binaries for Fasta
`graphviz-bin-<platform>.tar`	Binaries for Graphviz

phaser-bin<*platform*>.tar	Binaries for Phaser for the specific platform
Coot-<*version*>-<*platform*>.tar	Binaries for Coot
chooch-<*platform*>.tar	Binaries for Chooch for the specific platform
ccp4mg-<*version*>-<platform>.tar.g	Binaries for CCP4mg ***Note:*** *the .gz extension here is not a typo, this package has to be gzipped when put on the website. This is to activate the mechanism that reduces combinations of packages.*

7.1.6 Recommended platform naming conventions in the packages area

The table below gives a set of recommended naming conventions for the different platforms as used in the naming schemes above.

System	<platform>
Irix 6.5	**Irix**
Linux	**Lin**
OSF1 5.1	**Osf**
Solaris	**Sol**

The naming conventions for the different Linux flavours are:

Linux Flavour	<subdirectory_name>
Fedora Core	**Fedora**
SuSE 9.0+	**Suse**
Red Hat 8	**RH8**

So for example binary files for SuSE would be in …**/bin/Lin/Suse/**…

Note that packages for different flavours of Linux may have the same name, so they are distinguished by their position in the packages area. The reason that they need to have the same name is to allow the use of a static install script is now static – it is easier to use names with only a top-level O/S specification known to the install script through the manifest file, rather than a combination O/S and sub-Linux specification that the install script can not foresee.

7.2 Deploying the download pages for a new release

7.2.1 Overview of the procedure

New download pages need to be set up for each new release of CCP4. In this case a release is any distinct version (major, minor or patch). This section gives an overview of the steps in the procedure; however each individual step is covered in more detail in the subsequent sections.

It is useful to recognise that the deployment procedure uses two sets of download pagees: ***developmental pages*** and ***public pages***. Both sets use the same packages area but are otherwise separate and distinct.

The developmental pages are checked out from CVS, modified and tested before having the changes committed back to the repository as part of the preparation procedure. Once these pages are working, the public pages are generated either by CVS export or by a direct copy.

The idea is that the developmental pages can be placed in a password-protected area and only made publicly once everything is updated and tested.

The basic steps in the procedure are then:

1. **Populate the download area with the download files**
   Build and populate the packages area with source code and binary archives for all packages and tools that form part of this release

2. **Update the PHP download pages, configuration files and headers with the correct information**
   - Edit the decgen.tcl script and the packages.conf and dependencies.conf file to add new packages and dependencies, and also update details of existing packages if there have been changes
   - Update common.inc with details of packages area, cache area and new release version number
   - Run the update_declared.csh script to synchronise with the packages area, and commit the new version of declared.inc to the CVS archive

3. **Publish the pages on the public area**
   Tag the download pages archive with a release CVS tag, and export the pages to the public area

4. **Clean up and repopulate the download cache area**
   Remove out of date cache files and repopulate the cache area with prebuilt combinations of packages.

Each of these steps is described in more detail in the following sections.

7.2.2 Populating the "packages" download area with download files

The component archive files should be placed in the appropriate areas in the "packages" subdirectory (see above).

Source code archives are generated from the **`build_tars.csh`** script (part of the autobuild suite rather than the download pages) and placed in …**`/packages/src`**.

Binaries of the core packages for Unix/Irix are generated using the auto-build-binaries-6.0.csh script and placed in …**`/packages/bin/OS`** where **`OS`** is e.g. **`Irix`** for IRIX systems, or **`Lin`** for Linux. For Linux there are subdirectories which hold the binaries appropriate for particular Linux flavours when these are different (RedHat 8,

SuSE etc). Binaries that are identical for all Linuxes should be in **`packages/bin/Lin.`**

It is worth noting that both the binary and source code packages (both the CCP4 packages and the tools) are included as uncompressed tar files i.e. they are not gzipped (except for ccp4mg). See above for more information on the file naming conventions.

Note further that binaries for Coot and CCP4mg must be fetched from the appropriate websites, and that the same comments apply for Linux.

Also you need to keep in mind that if you are adding a package that constitute a new download module, you have to update install.sh to add the necessary installation instructions as well as the manifest generation function to add the necessary information concerning the presence of the module in the user choice.

7.2.3 Make the developmental download pages by performing a CVS checkout of the download page archive

Create a clean CVS working copy of the download page archive to make changes in before exporting to a public area. These will be the developmental pages where updates will be made and committed back to CVS.

7.2.4 Update the common.inc header file in the developmental pages

The header file **php/common.inc** must be updated (and the changes committed to CVS) for the following:

- CCP4 version number (this is the version that appears on the web page banner)
- Location of the CCP4 “packages” directory
- Location of the cache area

7.2.5 Update configuration files packages.conf, dependencies.conf and platforms.conf in the developmental pages

Any changes to the version numbers or descriptions for individual packages should be updated in the **`packages.conf`** file (similarly for any changes to package dependencies, minimum versions, available platforms and so on):

- The platforms.conf file must be updated for any changes to platform information.
- The packages.conf file must be updated for any changes to package information (e.g. version numbers changed for individual packages).

7.2.6 Update the decgen.tcl and update_declared.csh scripts and generate the new declared.inc file in the developmental pages

decgen.tcl is a tool used to automatically generate header files for the download pages which are specific to the packages area. update_declared.csh is a wrapper script that

runs decgen.tcl and ensures that the resulting headers are placed in the appropriate location.

The following changes are required:

- The glob matching patterns in the **`decgen.tcl`** script must be updated to account for any changes to the file naming conventions for the component archive files, and patterns for any new packages should be added. Do this by modifying or adding `Define…` calls in the top-level script.

- **`update_declared.csh`** should be updated to account for any change to the location of the "packages" directory.

A new version of the declared.inc file header can then be generated:

- Run the **`update_declared.csh`** script (in the admin/scripts directory) on either dlpx1 or ccp4serv2, to update filenames and sizes in the php directory (nb this script doesn't have to be run from a special location). If this is successful then CVS commit the updated declared.inc.

7.2.7 Export the download pages to the public area

The pages must then be "published" to the publicly accessible area.

First, the updated working copy of the PHP pages must be prepared, by CVS tagging the files in the php directory with the appropriate release tag, e.g. "release-6_0_99e"

Then the **export.csh** script (in the *admin/scripts* directory) should be run the script to update the publically visible copies of the pages:

```
> export.csh <version.string>
```

where <version.string> matches the CVS tag and is of the form "6.0.99e".

Essentially this does a CVS export of the pages from CVS.

7.2.8 Create or clean up or create the cache area and populate with pre-constructed archives

The cache area on ccp4serv2 must be cleaned up in order to remove out-of-date versions of the cached or "prefetched" versions of the archives. (In this context, "cleaned up" means deleting all existing archive packages that are already in the cache area). Alternatively, this directory must be created manually if it doesn't already exist.

The cache area can be populated with some pre-constructed archives. This is normally done on the eve of the release announcement by running a script called **loopIt.php** which builds download archives in the cache area for a predefined set of "popular" combinations.

Which combinations to construct are defined by an array within loopIt.php. This array should be updated to reflect the latest statistics, and also any new combinations that it is suspected will become popular quickly (for example if a new module is added).

Finally, the **`checktar.sh`** script should also be updated to point to the correct cache area.

7.3 Gotchas

There are a number of reasons why updates might not work.

- The cache area doesn't already exist.
- Make sure you are using a distinct packages directory and cache directory as targets when using the test download pages otherwise this can become really ugly!
- Tools files for Linux in the "packages" area have been placed in subdirectories (there are no subdirectories of "Lin" for the tools, only for the packages).

7.4 Administering and maintaining the download area after deployment

In the current implementation it is possible that download archive files may be incomplete. A script **`checktar.sh`** exists which will traverse the area on ccp4serv2 holding the archive files, and test that they hold the expected files. This should be run regularly to perform the checks (does Martyn Winn do this at the moment?).

8 FTP Server

This outlines the structure of the ftp area on ccp4serv, and the manual and automated procedures for generating the ftp area.

Warning: this is not up-to-date at present! See the INSTALL.html in the CCP4 6.0 or the CCP4 CVS archive to see what form the FTP area takes overall.

8.1 Structure of the FTP Area

The ftp area is accessible on dlpx1 from /ccpdisk/ccp4/public/ftp/ and for a particular release x.y.z has the following directory structure:

```
pub/ccp4/x.y.z/
pub/ccp4/x.y.z/binaries
pub/ccp4/x.y.z/packed
pub/ccp4/x.y.z/patches
pub/ccp4/x.y.z/unpacked
```

8.2 Creating the FTP Area Automatically

Use the script "make-ftp-area.sh" to automatically build the ftp area directory tree outlined above, and populate the packed and unpacked subdirectories from the files already exported from CVS into /ccpdisk/xtal/dist/.

Usage: make-ftp-area.sh <subdir_in_dist> <version.no>

e.g. make-ftp-area.sh ccp4-4.2.1 4.2.1

Nb currently the script is interactive, as a fail-safe.

The binaries must be built separately at present (use the script "auto-build-binaries.csh" to build binaries for each platform) and then copied to the appropriate place by hand.

The script will generate a number of htmls README files within the ftp directory, which offer explanations of the contents of each of the directories.

8.3 Creating FTP Area by Hand

This section outlines the procedure for making the ftp area manually, and is intended to elucidate the automatic procedure.

1. Create the ftp directory structure appropriate for the current version as outlined above.
2. Copy the files in dist/ccp4- directly into the "unpacked" directory
3. Generate the compressed tar files in the "packed" directory:
    i. For each subdirectory do:

```
   > /usr/local/bin/gtar -c -Z -f $PACKED/$i.tar.Z -C $CCP4
$i
   > /usr/local/bin/gtar -c -f $PACKED/$i.tar -C $CCP4 $i &&
   > /usr/sbin/gzip --best $PACKED/$i.tar &&
   > rm -f $FTPAREA/$i.tar
```

ii. For the files in the top directory do:

```
   > /usr/local/bin/gtar cZf $PACKED/ccp4-main.tar.Z -C
$CCP4 configure \
     duptree Makefile.in CHANGES COPYING PROBLEMS INSTALL
INSTALL.ps \
     INSTALL.html MAKEFILE.COM README BINARY.readme
BINARY.setup \
     licence.txt
   > /usr/local/bin/gtar cf $PACKED/ccp4-main.tar -C $CCP4
configure \
     duptree ...
   > /usr/sbin/gzip --best $PACKED/ccp4-main.tar &&
     rm -f $PACKED/ccp4-main.tar
```

iii. For the whole distribution:

```
   > ...
```

8.4 Preparing the “patches” subdirectory

A copy of the $CETC file “index.patches.template” called “index.patches” should be edited and placed in the patches subdirectory. Subsequent source code patches should be placed in this directory, and references to the patches should be placed in the index file (this procedure and the relevant tools are described in more detail in the CCP4 Programmers Handbook).

8.5 Mirror Sites

Once the servers have been updated inform the mirror sites:

- Philip Bourne bourne@sdsc.edu, although Kenneth (Ken) Addess addess@sdsc.edu seems to have taken this over now
- Atsushi Nakagawa atsushi@protein.osaka-u.ac.jp

9 Checklist After Official Release

1. **Make the release announcement**
   - Copy "Announcement_*" and update for the current release. Send this to bulletin boards (ccp4bb, o-info) to announce the new release.
2. **Inform the ftp mirrors**
   - See "FTP Server" (see section 8).
3. **Update the webpages**
   - In download.php update the current version, and add the last version to the "obsolete" list
4. **Create the patch branch**
   - See "Patch Branches and Releases" (see 10).
5. **Update the Problems Page**
   - http://www.ccp4.ac.uk/problems.html
   - For major release: make fresh page, keep a copy of the old version.
   - For patch release: extend the existing page.
6. **Update the "Release Resource Page"**
   - http://www.ccp4.ac.uk/dev/releases.html
7. **Update the Libraries-only distribution**
   Make a new "ccp4-onlylibs-.tar.gz" and place on the prerelease ftp area
   - Use the "-tag " option on the export-onlylibs.csh script
8. **Update the project log**

10 Patch Branches and Releases

10.1 Making a patch branch

Split off a branch in CVS, e.g.:

```
> cvs tag -b -r release-4_2_1 release-4_2_1_patch
```

which creates a branch rooted at the tag "release-4_2_1" called "release-4_2_1_patch".

10.2 Adding updates to the Patch Branch

Changes which are reported on the problems pages should also be added to the patch branch. This is the procedure that I use:

1. Make a patch for the change on the main branch, e.g.:

   ```
   > cvs diff -c -r <old_revision> <file> > patch.diff
   ```

2. Switch the file over to the patch branch:

   ```
   > cvs update -r release-4_2_1_patch <file>
   ```

3. Apply the patch:

   ```
   > patch -i patch.diff
   ```

4. Check that the patching has worked.
5. Commit the change (with the same comment as in the main trunk):

   ```
   > cvs commit <file>
   ```

6. Switch back to the main trunk using

   ```
   > cvs update -A <file>
   ```

10.3 Tagging and Exporting Patch Releases

Use the same mechanism as for official releases, e.g.:

```
> cvs tag -r release-4_2_1_patch release-4_2_2
```

will tag the most up-to-date files in the branch with the tag "release-4_2_2". The CVS "export" command can then be used to make releases and so on.

10.4 Making Source Code Patches (Upgrades)

Use the rdiff command of CVS to make diffs suitable for use with Larry Wall's patch program:

```
> cvs rdiff -c -r release-4_2_1 -r release-4_2_2 ccp4 > ccp4-4.2.1-
4.2.1.diff
```

The patch can be applied using patch as follows:

1. cd $CCP4
2. [ftp ccp4-4.2.1-4.2.1.diff to this directory]
3. patch -p1 -N -i ccp4-4.2.1-4.2.1.diff >& patch.log
4. ./config.status ; make [install]

"-p1" means strip one level of directory name from those in the patch file; "-N" means don't try to apply reverse patches.

12 Supporting Tools

The release process is supported by a number of different software tools. These tools are intended to automate some of the procedures in the preceding sections.

Where the tools are part of the main CCP4 CVS archive, these tools should be tagged as "**dont_release_me**" (see "Preparing the release").

Tool	Location	Description
auto-build-binaries-6.0.csh	Autobuild archive*	Make binary archives from source code for Unix/Linux platforms • **auto-build-binaries-6.0.csh dir version [options]** See the auto-build README for more detailed instructions.
build_tars.csh	Autobuild archive*	Export source code to archive files for each component • **build_tars.csh *version [packages]*** *Version* is a string of the form e.g. 6.0.2. *Packages* is a list of one or more of *ccp4*, *phaser* and/or *chooch*. If no packages are specified then all packages are exported.
create_docs	**$CCP4/ccp4i/bin**	Create/update the CCP4i programmer's documentation from the formatted comments in the CCP4i source code • **$CCP4/ccp4i/bin/create_docs**
GNUMakefile	**$CCP4/manual**	Remake the postscript version of the CCP4 manual from the LaTeX source.
Makefile	**$CDOC**	Generate the *.doc files (man pages) from the HTML program documentation in the $CHTML directory • **make**
Makefile	**$CHTML**	Invoke makeindices.pl and makereference (see below). • **make indices** • **make reference**
make_CHANGES.tcl	**$CCP4**	Generate the CHANGESinV*.html file from the text CHANGES file • **make_CHANGES.tcl [-i *input*] [-o *output*] [-v *version*]**
makeindices.pl	**$CHTML**	Regenerate the INDEX.html, FUNCTION.html and whatis files based on the contents of the HTML documentation (invoked via Makefile, see above).
makereference	**$CHTML**	Regenerate the REFERENCES.html file based on the contents of the HTML documentation (invoked via Makefile, see above).
export.csh	Download page script**	Export tagged download pages from the PHP area to the public area.

update_declared.csh	Download page script**	Regenerate declared.inc file in the "php" directory for the download pages. This is a wrapper for decgen.tcl
decgen.tcl	Download page script**	Generate declared.inc file; do not invoke directly, instead use update_declared.csh.
checktar.sh	Download page script**	Check for and report if download archive files do not contain the expected files.
commandline.php	Download page script**	Generate a download archive from the command line, given a string indicating the target platform and the list of packages to be included.
loopIt.php	Download page script**	Generate a number of download archives for different platforms and package combinations, by iterating the commandline.php script.

*12.1 Autobuild Archive**

The Autobuild archive contains the automatic build scripts and other useful tools.

The archive has its own CVS module (called **auto-build**) on ccpdisk; to check out a working copy do:

```
> cvs -d /ccpdisk/xtal/CVSROOT checkout auto-build
```

or equivalent, depending on how you wish to access CVSROOT. There is also a working copy already available at **/ccpdisk/xtal/auto-build**.

The scripts themselves are in the bin subdirectory of the Autobuild archive. An ASCII text file autobuild.README contains documentation.

*12.2 Download Page Scripts***

Like the Autobuild archive, the scripts for the download page operations also have a dedicated CVS module on ccpdisk, called **ccp4-download**.

There is also a working copy on ccp4serv2 – see the section on the interactive download pages for more details.

13 External Tools

External tools required for the release process include:

- html2ps
- lynx

12.1 Html2ps

"html2ps" is a HTML to Postscript converter, which is available from http://user.it.uu.se/~jan/html2ps.html. It is used within the release process to generate Postscript documents from the HTML masters.

A typical command to run the program for this purpose might look like the following example:

% html2ps –n –u document.html > document.ps

The –n flag indicates that page numbering should be used in the output file. The –u flag indicates that links within the original document should be shown as underlined in the output file.

12.2 Lynx

"lynx" is a text-based HTML browser, which is available from http://lynx.isc.org. It is used within the release process to generate text documents from the HTML masters.

On some systems lynx may already be installed, otherwise you will need to install it before preparing the release. Note also that if lynx is not installed in /usr/local then you may need to add the –cfg option to specify the location of the lynx configuration file.

A typical command to run the program for our purposes might look like the following example:

% lynx [-cfg lynx.cfg] –nolist –dump document.html > document.txt

The –cfg flag tells lynx where to find the configuration file, in the event that it is not installed in /usr/local.

13 Known Problems and Workarounds

This appendix lists the known problems in various stages of the process.

Date	Stage	Description
12/12/2006	**Building binaries of the core CCP4 on dlpx1**	There is likely to be a problem with creating the libccp4map.so file (a part of MapSlicer) on dlpx1, if the GNU version of ld appears on the PATH ahead of the native IRIX version. If this is the case then the file will not be created and the logfile will have a warning issued from ld of the form: `unrecognised option `-n32` The solution is to ensure that the native ld (/usr/bin/ld) appears on the PATH before the GNU version (/usr/local/bin/ld).
12/10/2006	**Reusing existing archive files for a new (patch) release**	Even if the source code for a particular package e.g. phaser has not been updated between releases (most likely for patch releases), it is still necessary to update the source and binary archive files for subsequent releases. This is to avoid the problem that the package distributed with e.g. CCP4 6.0.1 will unpack into ccp4-6.0.1, and so will be incorrect for CCP4 6.0.2.
10/11/2005	**Making archive files on dlpx1**	If the GNU version of tar is used to make the archive files, then there may be problems with long file pathnames when using a non-GNU tar to unpack. The current solution is to create the archives using a non-GNU version of tar (i.e. /usr/bin/tar on dlpx1).
Pre-10/11/2005	**Making binary distributions on Sun platforms**	On SunOS, tar -C doesn't work as advertised so the build script fails to create a proper tar.gz file with the right contents.

14 Checklist for preparing a release

Release version	
CVS tag	
Date (start)	
Date (finished)	
Status (e.g. public, beta, etc)	

Task	**Done**	**Checked**
INSTALL documents updated		
ccp4i_installation.html updated		
README updated		
CHANGES files updated		
Version of core library updated		
CCP4i version updated		
Setup files updated		
HTML documentation updated: • INDEX.html • FUNCTION.html • REFERENCES.html		
Doc files updated		
CCP4i documentation updated		
Manual updated		
Acknowledgements updated		
C library documentation updated		
Example logfiles updated		
Tutorial logfiles updated		
BINARY.setup updated		

Tagging release components		
Component	**Tagged**	**Removed dont_release_me**
CCP4		
PHASER		
CCTBX		
CHOOCH		

15 Test Sheet for CCP4 installations

Date			
Software package		**Version**	
Download method			
Components			

Installation Details	
Machine	
Platform	
Directory	
Install method	

Testing Installed Software		
Program/test	***Performed***	***Outcome***
Run-all		
CCP4i starts		
Mapslicer		
Mosflm		
Rasmol		
CCP4mg		
Coot		
Cpirate		
Phaser		
Chooch		

Comments/notes