

# CCP4 software library: Design Document

**Author:** Martyn Winn

**Revision:** 0.2 **Date:** 03/10/2002

## Form of library

1. Support functions must be available to all partial makes (CMTZ etc.). Possibly some duplication should be allowed to support partial makes.

## Naming conventions

- Fortran interface must of course follow existing naming.
- All new functions must have distinct names. In practice, we have:

CMTZ

Prefix "Mtz" for new functions, and "ccp4\_lr" etc. for those which are closely tied to Fortran functions. Data structures are MTZ, MTZXTAL, MTZSET, MTZCOL.

CMAP

Prefix "ccp4\_cmap\_"

CSYM

Prefix "ccp4spg\_"

Unit cell conversions

Prefix "ccp4uc\_"

CCP4 style

Prefix "ccp4"

System-dependent functions

Prefix "ccp4\_utils\_"

Pedants would argue that these should all conform to "ccp4\_".

- Namespaces are defined for C++ applications, as follows:

CMTZ

namespace CMtz

CMAP

namespace CMap\_io

CSYM

namespace CSym

Unit cell conversions

namespace CCP4uc

CCP4 style and system-dependent functions

namespace CCP4

## Coding conventions

- **Return values:** If a function is not returning a specific object or pointer to an object, then it will be of type int and return an error status. The convention is that 1 represents success, and 0 represents failure.

## Error handling and debugging

*Contract* - Error handling based upon standard C libraries. Information to be made available to calling routines via defined return status and error code. Library to contain error code with accessor functions. The response is left to the calling routines, except in FORTRAN wrappers).

`ccp4_errno.h` defines error state `ccp4_errno`, so are separate from system `errno`. This takes the form:

```
| 12 bits (system) | 4 bits (level) | 16 bits (code) ||
```

and is set in `ccp4_signal()`. `ccp4_errno` is externally visible.

Error message strings are set statically allocated in `library_err.c`, and made accessible by `ccp4_strerror` (cf `strerror`) and `ccp4_error` (cf `perror`).

`ccp4_errno` is set by `ccp4_signal(int error_code, char msg, void (*callback)() )` which allows callback (extend to pass data and return void \*?).

Non-integrated routines `ccperror()`.

## Support for scripting

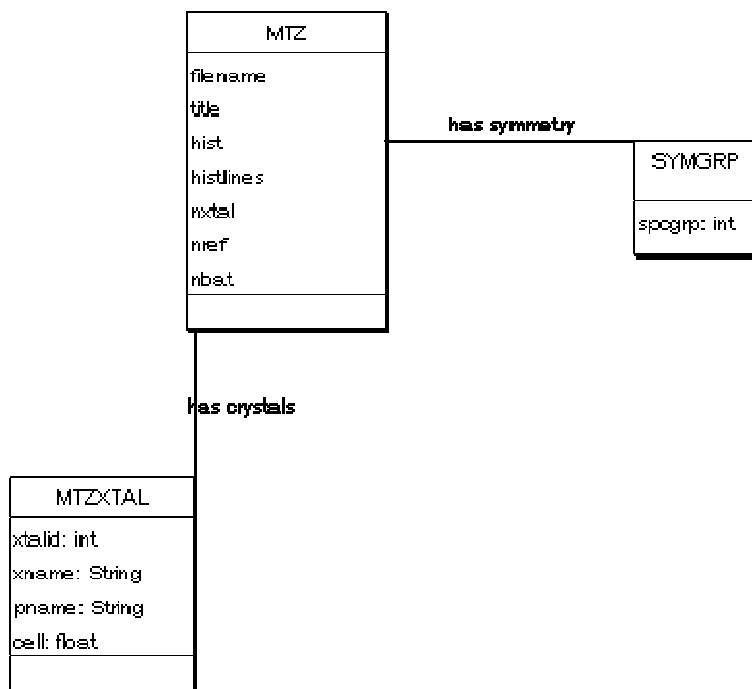
Class and struct members must be made available to scripting languages, and this is done in a variety of ways:

1. Generation of shadow classes, as in python interface to MMDB.
2. Use of SWIG-generated accessor functions for structs (not tried yet).
3. Explicit accessor functions. `cmtzlib.c` has a number of these (e.g. `MtzColType`). When implemented at the C level, these are accessible to all applications. However, it may be that they are only useful for Tcl, in which case they could be implemented as SWIG inline functions.

## CMTZ

- Central role is to read an MTZ file, manipulate the contents in memory, and to write out to a new MTZ file if desired. Applications have access to the data structure either directly or through accessor functions.
- Reflection data can be held in memory or on disk.
- `cmtzlib_f` contains static arrays to link the Fortran MINDX to specific structures in memory.
- Columns of data are in general identified by a full path `/crystal/dataset/column` although a bare `column` can be used provided it is unique.
- The tcl interface provides direct access to MTZ data structures from tcl scripts. The data structures cannot be navigated at the tcl level, and sufficient accessor functions should be provided.

Here is an incomplete diagram of MTZ header structure:



## Comparisons with Fortran API of CCP4 4.2

The aim is to continue to support the legacy Fortran API. However, there are some subtle differences:

- **Column ordering:** Rather than order columns in the MTZ file according to the time they were created, the columns are grouped by crystal and dataset. Thus, when additional columns are "appended" they may in fact be inserted so that they belong to the appropriate dataset. Therefore, avoid assumptions about column ordering in the file.

## CMAF

*Minimum Contract* - C style io access to mapfile. Read data into program specified arrays. Replication of FORTRAN API.

Mapfile described by C-structure (currently flat), representing header plus additional information for parsing. This is accessible by accessor functions.

Logical breakdown of file into header, symops, sections, rows, and data. Read, Write and Seek access to be given to symops, etc ...

## CSYM

1. Extensive spacegroup information is held in a flat file `syminfo.lib`. This file is generated from `cctbx` with the addition of CCP4 spacegroup numbers and real space ASUs where appropriate. The file holds data for non-standard settings.

2. Spacegroups are identified by standard number, CCP4 number, or list of operations, and data for that spacegroup read into memory.
3. Some data is calculated on the fly, e.g. centric and epsilon zones.
4. csymlib\_f contains static arrays to link subroutine calls to structures in memory.

## **Unit cell**

1. Contains manipulations of cell parameters and orthogonal to fractional transformations. Taken mainly from old rwb Brook.

## **CCP4 style**

1. Contains functions to give CCP4-style command line interface, CCP4 banner, etc.
2. Aim to reproduce traditional CCP4 look-and-feel.

## **System-dependent functions.**

### **IO routines**

*Minimum Contract* - extend and rationalise current io library to give additional file information.

Wraps buffered (FILE) and unbuffered (fp) access. All access allowed as items or raw bytes. Maintains information on status of file, including length, location and error status. Adds byte swapping on write to allow appending to a non-native file. File mode is set by fopen() style character mode.

Ability to get/put/unget (not implemented).

Wrapping of FILE based buffer control (not implemented).