Clipper

Using the Clipper libraries.

Kevin Cowtan
cowtan@ysbl.york.ac.uk

Clipper libraries

Why use a crystallographic library?

- Because it will save you a huge amount of work:
  - 3-10x increase in productivity.
  - Common algorithms are already built-in.
  - Well designed classes prevent common coding errors.
- Because it has been extensively debugged.
  - by use in other programs.
  - by test suites of varying degrees of formality.

Why not?

- Because its more to learn, and more to build.

Kevin Cowtan, cowtan@ysbl.york.ac.uk Sienna/Clipper

Clipper libraries

Why use a Clipper in particular?

- Purpose designed for phase improvement and interpretation, i.e. good for:
  - Phasing
  - Phase improvement
  - Refinement
  - Model building

Why not?

- No facilities for un-merged data.
- Limited facilities for anything before fixing origin.

Kevin Cowtan, cowtan@ysbl.york.ac.uk Sienna/Clipper

Clipper libraries

Why use a Clipper in particular?

- Mainly for C++ use.
- For Fortran or scripting purposes, write a wrapper which does most of the task and communicates in an appropriate way with the problem concerned.

Why not?

- No general scripting interface.
  - (Although it certainly works with SWIG.)

Kevin Cowtan, cowtan@ysbl.york.ac.uk Sienna/Clipper

Clipper libraries

How to get Clipper:

- From my website:
  - http://www.ysbl.york.ac.uk/~cowtan/clipper/clipper.html
  - Simple script based build system.
- With CCP4
  - GNU autoconf build system.
- With CCTBX
  - SCONS build system.

I'm not an expert on build systems!

Kevin Cowtan, cowtan@ysbl.york.ac.uk Sienna/Clipper

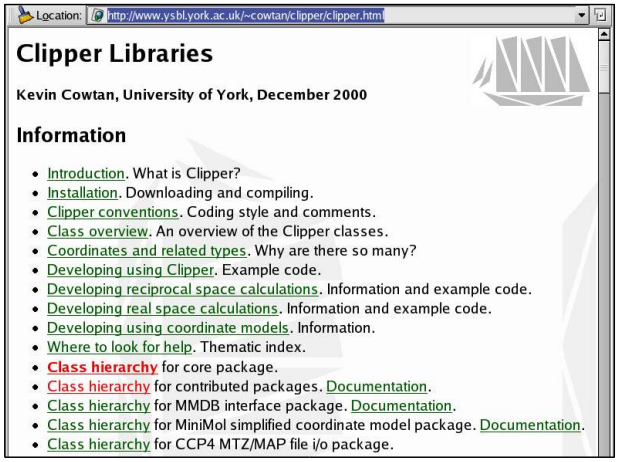
Clipper libraries

How to get information on Clipper:

- From my website:
  - http://www.ysbl.york.ac.uk/~cowtan/clipper/clipper.html
  - Comprehensive 'doxygen' documentation, including:
    - class codumentation
    - a range of tutorials
    - a few examples: more are in the 'examples' directory.
  - Aside: When you write code:
    - document it.
    - write test code.
      - (Proper test classes are better than stand-alone test programs – I'm working on it right now).

Kevin Cowtan, cowtan@ysbl.york.ac.uk Sienna/Clipper

Clipper libraries



Clipper libraries

Building Clipper applications:

- When compiling, `-I$INCLUDEDIR`
  - Usually `$PREFIX/include`
- When linking, `-L$LIBDIR`
  - Usually `$PREFIX/include`
- Using my build scripts, to make a simple program put a single `.cpp` file in the `examples` directory, and use

  `make` *`program_name`*

Clipper libraries

Clipper conventions:

- Main classes in '`clipper`' namespace.
- Data may be held at either precision: data classes in '`clipper::data32`', '`clipper::data64`'
- Small objects use 'double' by default, but 'float' is also possible.
- Terminology:
  - fractionals are (u,v,w).
  - $4\sin^2 / ^2$ is 'invresolsq'

Clipper

Brief overview of classes

Clipper libraries

Overview of classes:

- Helper classes
  - `Clipper::String`
    - An interchangeable extension to std::string, with additional parsing and manipulation methods (e.g. split, strip).
  - `Clipper::Util`
    - A collection of useful static functions providing common crystallographic functions (e.g. I_1/I_0, B-U conversion) and portability code.

Clipper libraries

Overview of classes:

- Crystal information
  - cell and symmetry
- Ordinates, derivatives, operators and grids
  - HKLs, coordinates etc.
- Data classes
  - reflection data, maps (crystallographic and otherwise)
- Method objects
  - common tasks, e.g. data conversion, sigmaa, etc.
- Input/output classes

Clipper libraries

Crystal information:

- A crystal is defined by two main classes: a unit cell (`clipper::Cell`) and a spacegroup (`clipper::Spacegroup`).
- These are complex classes which store derived information and provide optimised methods for handling it.
- Two smaller 'descriptor' objects provide a more compact representation for storage and transmission: The cell descriptor (`clipper::Cell_descr`) holds just the cell edges and angles, and the spacegroup descriptor (`clipper::Spgr_descr`) holds the 'signature' of the spacegroup.

Clipper libraries

Ordinates and grids:

- Ordinates include Miller indices (`clipper::HKL`), orthogonal and fractional coordinates (`clipper::Coord_orth, clipper::Coord_frac`), grid coordinates (`clipper::Coord_grid`), and others.
  - The ordinates have methods to convert to any other related form.
- Operators for transforming coordinates. Rotation matrices and rotation translation operators (e.g. `clipper::RTop_orth, clipper::RTop_frac, clipper::Symop`).
  - Operators also contain transformation methods.
- Gradients, curvatures, grids (`clipper::Grid_sampling`), etc.

Clipper libraries

Data objects:

- Data objects hold the actual crystallographic data. They include reciprocal space data (`clipper::HKL_info, clipper::HKL_data`), crystallographic and non-crystallographic maps (`clipper::Xmap, clipper::NXmap`), and FFT maps (`clipper::FFTmap`)
- The primary design goal of the data objects is that they hide all the bookkeeping associated with crystallographic symmetry (and in real space, cell repeat). Data can be written to and read from any region of real or reciprocal space, and the unique stored copy of the data will be modified correctly. This is all achieved in a computationally efficient manner.

Clipper libraries

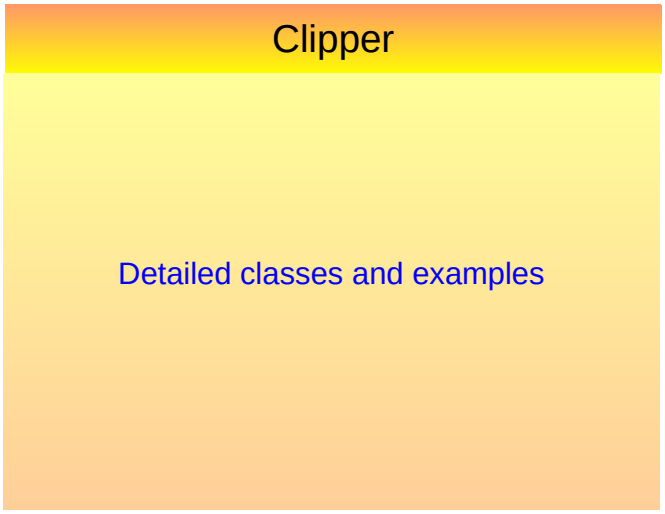
Method Objects:

- For common crystallographic tasks:
  - calculating functions of resolution.
  - calculating structure factors from coordinates.
  - sigma-a, likelihood maps.
  - map filtering.
  - map alignment, feature recognition, skeletonisation.
- Constructor creates a (tiny) method object setting any parameters for the calculation.
- `operator(...)` method does the actual calculation.

Clipper libraries

Input/Output objects

- Input/Output objects are used to record the contents of an object in a file or restore the contents from a file.
- Different objects are used for different file types, but the interfaces are as similar as the file format allows.
- (`CCP4MTZfile, CCP4MAPfile, MMCIFfile, CNSfile, PHSfile`)



Clipper Libraries: In detail

Crystal Information: Cell and Spacegroup

- We almost never make objects from scratch, but we can using the compact 'descriptions':

```
Cell_descr celld(30.0,40.0,50.0);
Cell cell( celld );

Spgr_descr spgrd( "P 2ac 2ab" );
Spacegroup spgr( spgrd );
```

- Cell description takes up to 6 arguments (a,b,c,alpha,beta,gamma), degrees or radians.
- Spacegroup description can be H-M or Hall symbol, spacegroup number, or list of operators.

Clipper Libraries: In detail

Crystal Information: Cell

- We can access cell properties:

```
double a     = cell.a();
double astar = cell.a_star();
double vol   = cell.volume();
Mat33<> mat = cell.matrix_orth();
```

- Many clipper objects have an 'null' state, which they are in if not initialised, and an 'init' method:

```
if ( cell.is_null() )
  cell.init( clipper::Cell_descr( ... ) );
```


Clipper Libraries: In detail

Crystal Information: Spacegroup

- We can access symmetry operators:

```
int nsym  = spgr.num_symops();
int nsymp = spgr.num_primitive_symops();
Symop op  = spgr.symop[i];
```

- Asymmetric units:

```
if ( spgr.in_asu( hkl ) )
  ...
```

- Reflection centric/absence/multiplicity:

```
HKL_class cls = spgr.hkl_class( hkl );
```

- And much more.

Clipper Libraries: In detail

Coordinates:

- Can construct from 3 numbers, or Vec3<>.
- Can convert orthogonal <-> fractional using `Cell`.
- Can convert fractional <-> grid using `Grid_sampling`.
- Can transform using `coord.transform(rtop)` or `rtop*coord`

Clipper libraries

	Coord_orth (const **ftype** &x, const **ftype** &y, const **ftype** &z) *constructor: from x,y,z*
	Coord_orth (const **Coord_orth** &x1, const **Coord_orth** &x2, const **Coord_orth** &x3, const **ftype** &length, const **ftype** &angle, const **ftype** &torsion) *constructor: from 3 coords and bond length, angle, torsion*
const **ftype** &	**x** () const *get x*
const **ftype** &	**y** () const *get y*
const **ftype** &	**z** () const *get z*
ftype	**lengthsq** () const *return square of length of vector in Angstroms*
Coord_frac	**coord_frac** (const **Cell** &cell) const *orthogonal-fractional coordinate conversion*
Coord_orth	**transform** (const **RTop_orth** &op) const *return transformed coordinate*
String	**format** () const *return formatted **String** representation*

Clipper Libraries: In detail

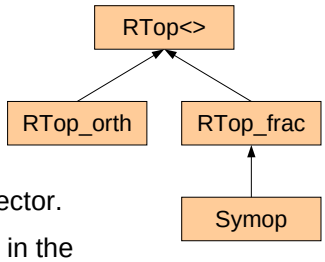
Coordinates:

- Example transformations...

```
// Make grid coord from ints
Coord_grid cg( u, v, w );
// convert to fractional using grid
Coord_frac cf = cg.coord_frac( grid );
// transform using symop
cf = spgr.symop(2) * cf;
// convert to orthogonal using cell
Coord_orth co = cf.coord_orth( cell );
// format and print the result
std::cout << co.format() << "\n";
```

Clipper Libraries: In detail

Operators:



- Rotation-translation operator most common, consists of rotation matrix and translation vector.
- Construct from matrix and vector.
- Can apply to any coordinate in the same system.
- Can combine multiple RTop-s by multiplication.

Clipper Libraries: In detail

Operators:

- Rotations may also be represented by Euler angles (CCP4 or full 24 conventions), polar angles (CCP4 convention), or quaternion (interchange format).

```
//Make euler rotation (radians)
Euler_ccp4 euler( pi/3, pi/4, pi/5 );
// convert to quaternion
Rotation rot( euler );
// convert to polar
Polar_ccp4 polar = rot.polar_ccp4();
// make vector
Coord_orth co( 1.0, 2.0, 3.0 )
// make RTop_orth
Rtop_orth op( rot.matrix(), co );
```


Clipper Libraries: In detail

HKLs:

- Construct from integers, access h, k, l:

```
HKL hkl(1,2,3);
int h = hkl.h();
```

- Calculate resolution:

```
double s = hkl.invresolsq( cell );
```

- Transform:

```
HKL equiv1 = spgr.symop(i) * hkl;
HKL equiv2 = spgr.isymop(i) * hkl;
```

- Calculate phase shift, etc:

```
double dphi = hkl.sym_phase_shift( symop );
```


Clipper libraries

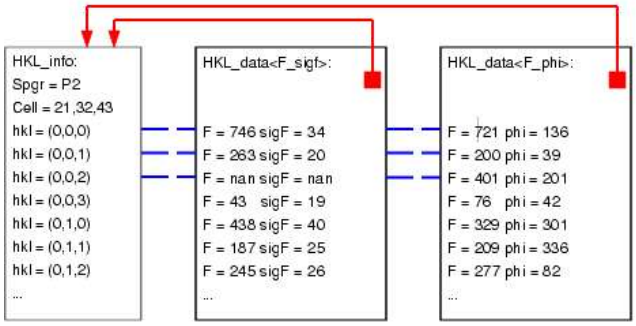
Data objects:

- Reciprocal space:
  - We often want to handle many lists of related reflection data, handling all the data connected with one HKL at once.
  - We often want to add new data during the course of the calculation.
  - Some data are tied together.
- Clipper implements a system of data lists, holding data of crystallographic types, using a common indexing defined by a parent object.

Clipper libraries

Data objects:

- Reciprocal space:



Note: Data types may be complex, e.g. F+/F-, ABCD

Clipper libraries

Data objects:

- Reciprocal space:
  - `clipper::HKL_info` manages the list of HKLs for all the objects.
  - `clipper::HKL_data<type>` holds a single list of data of some crystallographic type. (e.g. F/sigF, A/B/C/D, I(+)/I(-)/sigI(+)/sigI(-))

- Note this is the scheme in Clipper version 1. In version 2 (coming soon), `clipper::HKL_info` objects will disappear, being created and destroyed in the background automatically as required. Existing code is not affected!

Clipper libraries

Data objects:

- Reciprocal space:

```
// make crystal objects
Spacegroup spgr( ... );
Cell       cell( ... );
Resolution reso( 3.0 );
// make reflection list
HKL_info hklinf( spgr, cell, reso, true );
// make data lists using reflection list
HKL_data<data32::F_sigF>  fsig( hklinf );
HKL_data<data32::Phi_fom> phiw( hklinf );
HKL_data<data32::ABCD>    abcd( hklinf );
```


Clipper libraries

Data objects:

- Reciprocal space: Can access data by index in list, or by HKL. If the HKL requested isn't in the list, the correct symmetry equivalent will be chosen, and the data transformed automatically (including Friedel and phase shift).

```
// get 23rd data by index
double f = fsig[23].f();
double sig = fsig[23].sigf();
// get (1,2,3) data, (-1,-2,-3) data
double phi1 = fphi[ HKL(1,2,3) ];
double phi2 = fphi[ HKL(-1,-2,-3) ];
// by definition, phi1 = -phi2
```


Clipper libraries

Data objects:

- Data types are complex objects, from which individual items may be accessed. They know how to transform themselves about reciprocal space. The user can define new data types.
- Operators (+,-,*,&,|,!) are available to add, subtract, or scale entire lists when these make sense.
- 'Compute operators' are available for applying common operations and conversions to data, e.g.:
  - ABCD <-> phi/fom
  - Scaling by scale, overall B
  - F -> semi-normalised E

Clipper libraries

Data objects:

- Conversions are available for transforming between types, individually or as a list. (Lambda operators).

```
HKL_data<data32::F_sigF>  fsig( hklinf );
HKL_data<data32::ABCD>    abcd( hklinf );
HKL_data<data32::Phi_fom> phiw( hklinf );
HKL_data<data32::F_phi>   fphi( hklinf );
// convert abcd to phi/fom
phiw.compute( abcd, Compute_phifom_from_abcd() );
// convert F/sigF and phi/fom to weighted F/phi
fphi.compute( fsig, phiw,
              Compute_fphi_from_fsigf_phifom() );
```


Clipper libraries

Data objects:

- These methods provide convenient access when performance isn't an issue. But access by HKL involves symmetry search, so can be slow. When performance is critical, an alternative approach is adopted, using 'reference' objects, which are slightly related to STL iterators. These are designed to optimise common access pattens:
- When looping over all data sequentially: `HKL_data<*>::HKL_reference_index`
- When accessing data by HKL rather than by index: `HKL_data<*>::HKL_reference_coord`

Clipper libraries

Data objects:

- The index is like a normal array index, but can also return resolution and other information. It may be used for any `HKL_data` with the same `HKL_info`.

```
HKL_data<data32::F_phi>::HKL_reference_index ih;
for ( ih = fphi.first(); !ih.last(); ih.next() ) {
   HKL hkl = ih.hkl();
   double s = ih.invresolsq();
   data32::F_phi fp = fphi[ih];
}
```

- The coordinate reference type allows fast access to neighbouring and nearby reflections (for which the symmetry operator for the stored ASU is usually conserved).

Clipper libraries

Data objects:

- Indices and references may be used across any lists which share the same `HKL_info`.
- To transfer data between differently indexed lists, you must go back to the underlying `HKL`.
  - Loop over the target `HKL_data` and fetch data from the source `HKL_data` by HKL.

Clipper libraries

Data objects:

- Crystallographic and non-crystallographic maps (`clipper::Xmap, clipper::NXmap`)
  - The data objects are templates which can hold data of any type. In the case of a map, this type will usually be `double' or `float'.
  - Xmap-s have crystallographic symmetry and lattice repeat.
  - Nxmap-s have neither, and define a bounded region in the coordinate space.

Clipper libraries

Data objects:

- Crystallographic maps (`clipper::Xmap`)
  - Construct from Spacegroup, Cell, Grid_sampling.
  - Usually construct Grid_sampling from Spacegroup, Cell, Resolution.

```
Grid_sampling grid( spgr, cell, reso );
Xmap<float>   xmap( spgr, cell, grid );
```

  - Can the calculate maps by FFT from HKL_data<F_phi>

```
xmap.fft_from( fphi );
xmap.fft_to( fphi );
```


Clipper libraries

Data objects:

- Crystallographic maps (`clipper::Xmap`)
  - Access map by index, or coordinate:

```
// allowed but discouraged
float rho1 = xmap.get_data(1234);
float rho2 = xmap.get_data(Coord_grid(1,2,3));
```

  - Fine for some tasks. However:
    - Indices are not sequential, owing to irregular shape of ASU.
    - Access by coodinate requires symmetry lookup to find value in stored ASU.

Clipper libraries

Data objects:

- Crystallographic maps (`clipper::Xmap`)
  - Access map by two reference types when performance is important:

```
// preferred: reference index
Xmap::Map_reference_index ix;
for ( ix = xmap.first(); !ix.last(); ix.next() )
  float rhox = xmap[ix];

// preferred: reference coord
Xmap::Map_reference_coord iy( xmap, Coord_grid(1,2,3) );
float rho1 = xmap[iy];
iy.next_u();
iy.prev_w();
float rho2 = xmap[iy];
// now iy -> Coord_grid(2,2,2)
```

  - References may be shared across similar maps.

Clipper libraries

Data objects:

- Crystallographic maps (`clipper::Xmap`)
  - Methods are provided for interpolation, and also gradient and curvature calculation:

```
// get interpolated density
Coord_frac cf( 0.1, 0.2, 0.3 );
float rho1 = xmap.interp<Interp_linear>( cf );
float rho2 = xmap.interp<Interp_cubic> ( cf );
```

  - Also sorting, statistics, etc.

Clipper libraries

Data objects:

- Indices and references may be used across any maps which share the same space-group and grid (similar to reflection data).
- To transfer data between differently indexed maps, you must go back to the underlying `Coord_grid`, or `Coord_orth` if cells are different (interpolate).
  - Loop over the target `Xmap` and fetch data from the source `Xmap` by `Coord_grid`.

Clipper libraries

Data objects:

- Atoms:
  (`clipper::Atom, clipper::Atom_list`)
  - `clipper::Atom`: The simplest definition necessary for electron density calculation.
  - `clipper::Atom_list`: Derived from `std::vector<Atom>`.
  - For more complex atom manipulation, see the MMDB interface and the MiniMol package.

Clipper libraries

Method objects:

- There are a range of common calculations provided in to the Clipper packages:
  - Resolution functions – used to calculate smoothly varying estimates of things in reciprocal space, e.g. |F(|h|)| for normalisation of E's.
  - Electron density/structure factor calculation from atoms.
  - Map filtering, e.g. for calculating local mean of variance of electron density.
  - Skeletonisation.
  - Likelihood weighting and map calculation.

Clipper libraries

Method objects:

- There are a range of common calculations provided in to the Clipper packages:
  - Electron density/structure factor calculation from atoms.

```
SFcalc_iso_fft sfcalc;
Atom_list atoms;
...
sfcalc( fphi, atoms );
```

  - Constructor for `SFcalc_iso_fft` can take optional arguments to control it's behaviour.
  - The actual calculation is done by the () operator. The result is the first argument.
    - Note: there are several implementations (e.g. slow/fft/iso/aniso).

Clipper libraries

Method objects:

- Other examples:
  - Map filtering: first we define the filter function to apply to the map, and then we apply it using a given filter implementation.

```
MapFilterFn_step fn( filter_radius );
MapFilter_fft<float> fltr( fn, 1.0, Relative );
Xmap<float> filtered;
fltr( filtered, xmap );
```

  - This filters with a step function of a given radius, using the scaling parameters 1.0 and 'Relative', and puts the result in the new map.

Clipper libraries

Method objects:

- Other examples:
  - Resolution functions are more complex:
    - First, we define the curve we want to fit.
    - Secondly, we define the function the curve must minimise.
    - Thirdly, we define a set of initial parameters.
    - Fourthly, we feed both of these to an evaluator.
  - e.g. fitting a spline function to scale two datasets together:

```
BasisFn_spline basisfn( 6 );
TargetFn_scaleF1F2<data32::F_sigF,data32::F_sigF>
  targetfn( fsig1, fsig2 );
std::vector<double> params( 6, 1.0 );
clipper::ResolutionFn
  rfn( hklinf, basisfn, targetfn, params );
```

Clipper libraries

Method objects:

- Other examples:
  - e.g. fitting a spline function to scale two datasets together:

```
// fitting the function
BasisFn_spline basisfn( 6 );
TargetFn_scaleF1F2<data32::F_sigF,data32::F_sigF>
  targetfn( fsig1, fsig2 );
std::vector<double> params( 6, 1.0 );
clipper::ResolutionFn
  rfn( hklinf, basisfn, targetfn, params );

// scaling the data
for ( ih = fsig1.first(); !ih.last(); ih.next() )
  fsig1[ih].scale( sqrt( rfn(ih) ) );
```


Clipper libraries

Input/output objects:

- Open an I/O object for a particular file type, for read or write, and then import/export the data oject concerned using that object.
  - e.g. For a crystallograpic map:

```
Xmap<float> map;
CCP4MAPfile mapfile;
mapfile.open_read( "1ajr.map" );
mapfile.import_xmap( map );
mapfile.close();
```

  - For export, open_write() and then export_xmap().
  - Can also import/export NXmap-s from the same object.

Clipper libraries

Input/output objects:

- Reflection files vary. PHS files are simple. CNS and mmCIF files intermediate. MTZ files contain multiple sets of data of various types, and accompanying information.
  - e.g. Import a set of F-s and σ-s:

```
CCP4MTZfile mtzfile;
mtzfile.open_read( "1ajr.map" );
mtzfile.import_hkl_info( hklinf );
mtzfile.import_hkl_data( fsig, "/*/*/[FP,SIGFP]" );
mtzfile.close();
```

  - Can also import/export spacegroup, cell, and dataset information, where the format supports it.

Clipper libraries

Trivial examples:

- Read some data, calculate a map.
  - Assume F and phi.
    (We've seen how to get these from phi/fom, ABCD.)
- Expand data to P1
  - Including all appropriate symmetry transformations.
- Rotating density from an Xmap into an Nxmap

Clipper libraries

Calculate a map from structure factors:

```
clipper::HKL_info hkls;            // define hkl objects
clipper::HKL_data<clipper::data32::F_phi> fphidata( hkls );
// READ DATA
clipper::CCP4MTZfile mtzin;
mtzin.open_read( "my.mtz" );       // open new file
mtzin.import_hkl_info( hkls );     // read sg, cell, reso, hkls
mtzin.import_hkl_data( fphidata, "/*/*/[FCAL,PHICAL]" );
mtzin.close_read();
// DEFINE MAP
clipper::Grid_sampling mygrid( hkls.spacegroup(), hkls.cell(),
                               hkls.resolution() );  // grid
clipper::Xmap<float> mymap( hkls.spacegroup(), hkls.cell(),
                            mygrid );              // map
mymap.fft_from( fphidata );         // fill map
// OUTPUT MAP
clipper::CCP4MAPfile mapout;
mapout.open_write( "my.map" );      // write map
mapout.export_xmap( mymap );
mapout.close_write();
```


Clipper libraries

Expanding data to lower symmetry:

```
// make new objects
clipper::HKL_info newhkls( newspacegroup,
                           hkls.cell(), hkls.resolution() );
clipper::HKL_data<clipper::data32::f_phi> newfphidata(newhkls);
// and fill them
HKL_info::HKL_reference_index ih;
for ( ih = newhkls.first(); !ih.last; ih.next() )
  newfphidata[ih] = fphidata[ih.hkl()];


// same thing to expand a map
// make new object
newmap.init( newspacegroup, map.cell(), map.grid_sampling() );
// and fill it
clipper::Xmap_base::Map_reference_index ix;
for ( ix = newmap.first(); !ix.last(); ix.next() )
  newmap[ ix ] = map.get_data( ix.coord() );
```

Clipper libraries

Expanding data to lower symmetry:

```
// make new objects
clipper::HKL_info newhkls( newspacegroup,
                           hkls.cell(), hkls.resolution() );
clipper::HKL_data<clipper::data32::f_phi> newfphidata(newhkls);
// and fill them
HKL_info::HKL_reference_index ih;
for ( ih = newhkls.first(); !ih.last; ih.next() )
  newfphidata[ih] = fphidata[ih.hkl()];
```

Important rule of thumb for reflections and maps:

- When moving data into an object with organization, always loop through the target object, and set each element by fetching data from the source object by coordinate.

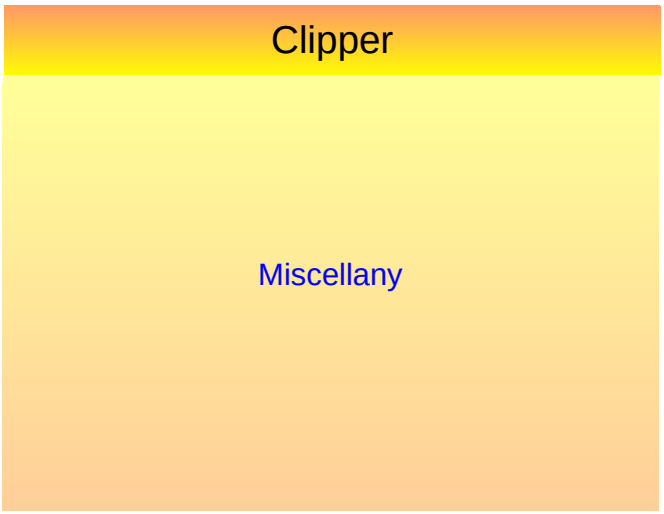
Clipper libraries

Rotating a map:

```
// make objects
clipper::Xmap<float> xmap;
clipper::NXmap<float> nxmap;
clipper::RTop_orth rtop

// initialise objects
...

// do the rotation
clipper::NXmap_base::Map_reference_index ix;
for ( ix = nxmap.first(); !ix.last(); ix.next() )
  nxmap[ix] = xmap.interp<Interp_cubic>( rtop*ix.coord_orth() );
```

Clipper libraries

CCP4 Extras:

- A mini-package of tools for building CCP4 command line programs:
  - A parser class, which combines input from both command line and standard input.
  - A CCP4-program class, which calls all the normal functions at the beginning of a program, and tidies up at the end.
    - Note: don't use exit. Continue to the end of a block.
- Currently just a source file which you can link to your program.
  - Possible part of libclipper-ccp4 in future?

Clipper libraries

Learning more:

- Documentation contains several tutorials, plus extensive reference material (~1000 pages)
- Lots of material in the 'examples' directory:
  - map calculation
  - structure factor calculation
  - ML weighting and maps
  - data analysis
  - data conversion
  - scaling
  - etc.

Clipper libraries

Conclusions:

- Clipper contains a great many building blocks for the rapid construction of crystallographic calculations.
- It is suitable for a certain range of problems, based on language and stage of structure solution.
- Convenience methods provide very simple ways to do complex tasks.
- Optimised methods provide near-optimal performance in terms of CPU and memory usage.
- Lots of documentation and examples...