Clipper status (July 2002)

Kevin Cowtan

July 22, 2002

1 Overview

Clipper has the following features:

- Objects to store and manipulate all crystallographic data, specialised towards computation. (i.e. performance, convenience for common tasks).
- An optional mechanism to organize those objects.
- Built on a 'Doctrine of fallibility': Everything extendable, Everything replaceable, Everything optional.

  Extensive use has been made of *Generic Programming* through C++ templates.

  In this way provision is made for unforseen tasks, for the preferences of developers, and for future developments (e.g. offset FFT grids).
- Conformance to core modern C++ style guidelines.
- Tiny (11K lines core).

Initially it was envisaged that Clipper would also provide a new means of data representation (and eventually storage), but the EBI work is superior for this purpose, with the potential to represent a more general data model which is easily maintained with a minimum of coding. Clipper will provide i/o methods for EBI data objects early in the development of that project.

2 Status

Handling of reflections, maps, and models have been demonstrated for common crystallographic tasks, e.g. sigmaa calculation, structure factor calculation (iso/aniso, sum/fft), skeletonisation. These cover a broad range of tasks and test the handling of reflection data, crystallographic maps, and coordinate models.

Density modification can be implemented in the current system, however averaging will stress the NXmap system, with possible bug-fixing or API changes.

Refinement can be implemented in the current system, if the appropriate gradient methods are added to AtomSF (a simple task). Construction of restraints requires MMDB bond and dictionary functionality - this is handled at the MMDB level using the wrapper pass-throughs.

MTZ i/o uses the MTZ format in a rather different way to existing software, columns are grouped, with the groupings marked by extended column names. Eventually this functionality should probably be reflected in the user interface. This overlaps with Martyn's two-character column types, but I don't know how yet.

Note that Clipper is designed for crystallographic computation, and not for some of the low level bookkeeping tasks. The 'cad' application is a pathological task for Clipper, however a crude demo of the functionality has been implemented.

The following table gives my subjective feeling concerning the maturity of the API, and the level of testing which has been applied to various components (10=good).

Package	Object	API	Testing
core	Cell	9	9
core	Spacegroup	9	9
core	Coords etc	9	9
core	HKL_info/data	10	10
core	Xmap	10	9
core	NXmap	5	3
core	Container forms	9	8
core	I/O	9	8[1]
core	Message	4	6
contrib	AtomSF	9	9
contrib	MMDB (wrapper)	7[2]	7
contrib	SFcalc func	7	7
contrib	Skeleton func	6	4

Notes:

1: UMTZ needs a small patch for old MTZs.

2: Currently I'm the only one using the wrapper.

The MMDB wrapper seamlessly handles conversion of individual numbers in MMDB to Clipper types, e.g. Coord_orth, U_aniso_orth, dramatically simplifying the interaction with other Clipper objects. However most people programming with both started before the wrapper was available, and are still coding the interchange by hand.

3 Documentation

Current documentation is extensive:

- 7 essays
- brief documentation for 1366 classes/methods
- full documentation for 484 most common classes/methods
- Total of 500 pages of reference documentation

Probably the most pressing need is for some complete documented examples. e.g. document existing FFT, add sfcalc application.

4 Applications

Simple methods can be packaged as function-objects acting on a few data objects. More complex methods which require a variable number of data objects will be passed the top node of a container sub-tree.

These may be repidly be wrapped as applications using whatever design is currently in fashion. If it changes, rewriting the application wrapper is an easy task. Once the data model arrives, function objects will probably be wrapped to work on sections of the data model directly. (Formal interface specification might allow automatic generation of GUI, CLI and documentation, but this might be better left until the data model work is established).

My next priority is starting a Baysian phase improvement program.