Clipper

Generated by Doxygen 1.6.1

Mon Jan 4 12:00:20 2010

Contents

Chapter 1

The Clipper Project

Author:

Kevin Cowtan

There are currently two major pressures on crystallographic computing:

- Increased automation to increase throughput in line with genomics applications.

- Increased data complexity, as data from more sources is combined and carried through the whole calculation to solve more difficult problems.

The Clipper project is an initiative to address these pressures.

The aim of the project is to produce a set of object-oriented libraries for the organisation of crystallographic data and the performance of crystallographic computation. The libraries are designed as a framework for new crystallographic software, which will allow the full power of modern programming techniques to be exploited by the developer. This will lead to greater functionality from simpler code which will be easier to develop and debug.

1.1 Object oriented programming

The evolution of high level programing may be very imprecisely caricatured as follows:

- Early high-level languages (Fortran): There is no structure to data or code. Variables are not grouped in any way or specially associated with any code.
- Structured programming (C, Pascal, etc): Data structures are introduced, grouping related variables into compound objects, which can completely specify the state of some object.
- Object-oriented programming (Smalltalk, C++, etc): Data structures may now contain code, and now describe not only the state of the object, but also its behavior and interactions. (In the general case, all code becomes part of an object.)

Clipper is object-oriented. The main benefit of this approach is that code becomes much more reusable, since objects are self-contained, and may be reused, rewritten, or replaced without affecting other code. Additionally, the organisation of the code and data is generally much clearer.

1.2 Clipper objects

Clipper defines a wide range of objects. These fall into a number of groups, including:

- **Crystallographic objects:** (p. 2) : Cell, Spacegroup, Metric, R/T operator, etc.
- **Data objects:** (p. 2) : Reflection data, Crystallographic map, Non-crystallographic map, Coordinate model.
- **Method objects:** (p. 3) : FFT arrays, Resolution functions, Conversion objects, Import/-export objects

The coordinate object is a substantial task in itself and is addressed by an interface to the CCP4 'MMDB' library, developed by Dr Eugene Krissinel.

Some of the objects will be discussed in more detail:

1.2.1 Crystallographic objects:

These implements the fundamental properties of a crystal.

The Cell object: This object describes a unit cell. It holds the cell parameters, and derived information including coordinate conversion matrices and metrics. Any cell object may be used to convert coordinates between orthogonal and fractional forms, and calculate distances in real space and resolutions in reciprocal space.

The Spacegroup object: This object describes a spacegroup, using information from the 'cctbx' library of Dr Ralph Grosse-Kunstleve. It can be use to generate symmetry coordinates and reflections, phase shifts, centricity, asymmetric units, and so on.

1.2.2 Data objects:

These hold actual data. They are written as templates which can hold whatever type of data the developer requires.

The reflection data object: It is commonly necessary to store several related items of reflection data. Therefore this object is split into two parts; a parent object which holds a list of Miller indices and related data, and then several data objects which hold the actual data associated with each Miller index. The data objects can hold data of arbitrary types: these types will usually consist of several values. For example, a structure factor magnitude and its variance, or all four Hendrickson-Lattman coefficients, are usually held in a single data object.

To the user, the data appears to cover the whole of reciprocal space, however in practice only an asymmetric unit is stored. Data is transformed about reciprocal spaces as required. When a new data type is defined, its behavior under transformation is also be defined so that this mapping can be performed.

The crystallographic map object: This object also implements crystallographic symmetry, and also cell repeat, in a manner which is transparent to the user. It may also hold arbitrary data types: common examples would include bits, real values, complex values, or orthogonal or fractional gradients.

The non-crystallographic map object: This object is used for map data which does not have symmetry or cell repeat, for example an NCS averaging mask.

1.2.3 Method objects:

These are used to provide additional functionality commonly required in crystallographic calculations. Examples include:

FFT map: This object holds data which may be represented in either real or reciprocal space. The data may be accessed in either form, and may be transformed between spaces as required.

The resolution function evaluator: This object creates an arbitrary function of position in reciprocal space, by optimising the parameters of some basis function in order to minimise some target function. This is an extreme generalisation of the idea of 'resolution bins', and can be used for anything from $<|F|^2>_S$ to sigma-a and beyond.

Import/export objects: The only import/export objects implemented so far are the MTZfile and MAPfile objects. These stand as proxies for an external file, and can transfer data between the Clipper objects and the file.

1.3 Scripting

Automation of crystallographic tasks depends on being able to communicate between successive tasks, and by being able to execute control code to activate tasks and make protocol decisions. This functionality is provided through a scripting interface. A Python interface will be provided through the boost.python library. It is possible that interfaces to C and a range of scripting languages will be provided through other means. All the data and the full functionality of the methods will be available from the scripting layer, allowing full automation and full communication between tasks.

Eventually the individual programs should disappear, rather exposing their functionality directly to the scripting layer. At the same time, data will have to move from traditional files into a database, so that each task has immediate access to all the information currently available.

1.4 Partial class hierarchy (as of 1/2/2001)

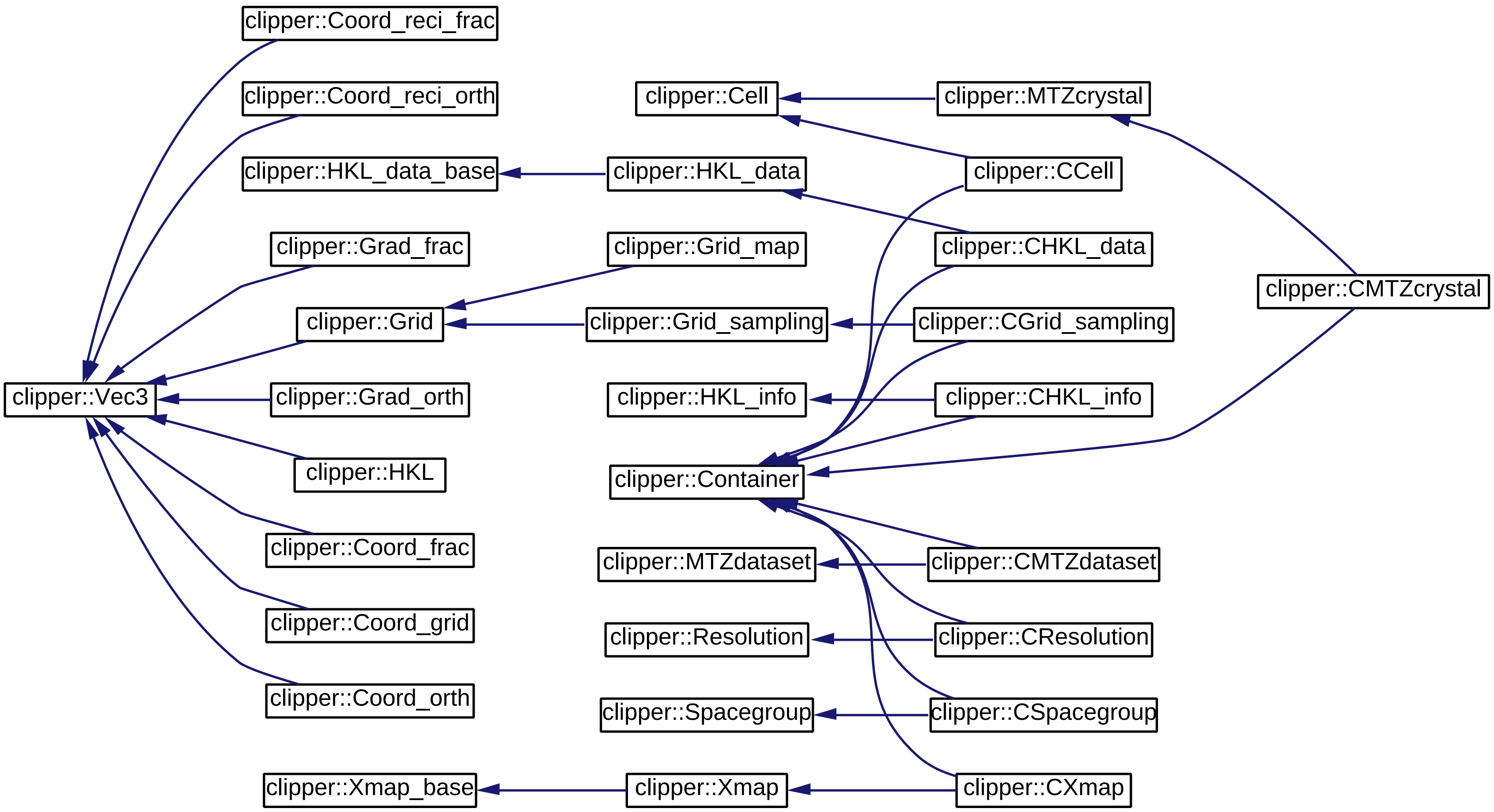


Figure 1.1: width=15cm

1.5 Acknowledgments:

I would like to thank Ralph Grosse-Kunstleve, Airlie McCoy, Eugene Krissinel, Jan Zelinka and the CCP4 staff for their many and varied contributions to this effort.

Clipper stands for 'Cross-crystal Likelihood Phase Probability Estimation and Refinement', which is what I hope to use it for.

1.6 References:

- 'Clipper' code and documentation (K. Cowtan): `http://www.ysbl.york.ac.uk/~cowtan/clipper/clipper.html`
- 'mmdb' coordinate library (E. Krissinel): `http://www.ebi.ac.uk/~keb/cldoc/`
- 'cctbx' crystallography toolbox (R. Grosse-Kunstleve): `http://cctbx.sourceforge.net`

See the following pages in the Clipper documentation for further information:

- **Class Overview** (p. 7) (brief guide to classes)
- **Conventions** (p. 15) (coding style)
- **Coordinates and related types** (p. 19) (Coodinate types)
- **Developing using Clipper** (p. 23) (initial example)
- **Developing using Reflection Data** (p. 29) (reflection data handling and examples)
- **Developing using Crystallographic Maps** (p. 37) (map handling and examples)
- **Developing using Atomic Models** (p. 45) (model handling)
- **Where to Look for help on specific tasks.** (p. 53) (thematic index)
- **Installation** (p. 49) (and dependencies)

Chapter 2

Class Overview

2.1 Introduction

This page contains a brief outline of the types of classes provided by the Clipper libraries, followed by a more detailed description of the different classes.

1. **Ordinates and grids:** (p. 9)

   Ordinates include Miller indices (**clipper::HKL** (p. 261)), orthogonal and fractional coordinates (**clipper::Coord_orth** (p. 182), **clipper::Coord_frac** (p. 172)), and grid coordinates (**clipper::Coord_grid** (p. 175)).

   Grids describe the storage of 3D data: The basic grid class (**clipper::Grid** (p. 250)) describes the dimensions and indexing of a 3D rectangular arrays. The unit cell grid (**clipper::Grid_sampling** (p. 255)) also provides methods for converting between fractional and grid coordinates, and the map grid (**clipper::Grid_range** (p. 252)) describes a general sampled map, which is bounded by lower and upper grid coordinates.

2. **Crystal information** (p. 9)

   A crystal is defined by two main classes: a unit cell (**clipper::Cell** (p. 133)) and a spacegroup (**clipper::Spacegroup** (p. 413)).

   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** (p. 137)) holds just the cell edges and angles, and the spacegroup descriptor (**clipper::Spgr_descr** (p. 421)) hold the Hall code of the spacegroup.

3. **Data objects** (p. 10)

   Data objects hold the actual crystallographic data. They include reciprocal space data (**clipper::HKL_info** (p. 284), **clipper::HKL_data** (p. 267)), crystallographic and non-crystallographic maps (**clipper::Xmap** (p. 463), **clipper::NXmap** (p. 357)), and FFT maps (**clipper::FFTmap** (p. 215))

   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.

   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', however in the case of reciprocal space data the types are often more complex, for example 'magnitude and phase' (F_phi) or Hendrickson-Lattman coefficient (ABCD).

4. **Input/Output objects** (p. 12)

   Input/Output objects are used to record the contents of an object in a file or restore the contents from a file. (MTZfile, MAPfile)

   The crystal descriptors and data objects may be used as stand-alone objects. However, the data objects are related to a particular crystal, and may be related to each other. In order to facilitate the organisation of information, all the objects have 'container' variants.

2.2 Classes

The classes which a developer will commonly encounter are as follows:

2.2.1 Ordinates and grids:

- **clipper::HKL** (p. 261)

  Reflection/Miller index. Constructed with 3 integers, h, k and l. Read-write access is provided to these members through the h(), k() and l() methods.

- **clipper::Coord_orth** (p. 182)

  Orthogonal Angstrom coordinate, defined on some unspecified coordinate frame. Constructed from 3 real numbers, x, y, and z. Read access is provided to these members through the x(), y() and z() methods.

- **clipper::Coord_frac** (p. 172)

  Fractional coordinate, defined in fractions of the cell edges. The coordinates are therefore not necessarily orthogonal. Constructed from 3 real numbers, u, v, and w, being fractions of the a, b, and c axes. Read access is provided to these members through the u(), v() and w() methods.

- **clipper::Coord_grid** (p. 175)

  Grid coordinate, defined on some unspecified grid. (Ideally this object should define its own grid, but these things are used so frequently that the overhead would be unacceptable.) Constructed with 3 integers, u, v and w, being grid positions along the a, b and c axes. Read-write access is provided to these members through the u(), v() and w() methods.

- **clipper::Grid** (p. 250)

  A generic grid definition. Constructed with 3 integers, nu, nv and nw, defining the size of a grid along 3 direction. Read access is provided to these members through the nu(), nv() and nw() methods. Methods are provided for simple indexing of a 3D array stored as a 1D list (Like a multidimensional local array in C, or a Fortran array with the indices reversed).

- **clipper::Grid_sampling** (p. 255)

  Derived from **clipper::Grid** (p. 250), this class is used for a grid which defines the sampling of a unit cell. Constructor as for **clipper::Grid** (p. 250). It provides additional methods for converting between fractional and grid coordinates, and for reducing a coordinate from anywhere in crystal space to the unit cell (0-1,0-1,0-1).

- **clipper::Grid_range** (p. 252)

  Derived from **clipper::Grid** (p. 250), this class is used for defining a map grid. The map grid need not cover a unit cell, and need not be based at the origin. It is constructed by two **clipper::Coord_grid** (p. 175)'s, representing the lower and upper bounds of the volume in 3D space. Methods are provided for simple indexing of a 3D array stored as a 1D list.

2.2.2 Crystal information

- **clipper::Cell_descr** (p. 137)

  The cell descriptor class (**clipper::Cell_descr** (p. 137)) contains the basic information required to describe a unit cell, i.e. the cell edges a, b, and c, and the cell angles, alpha, beta and gamma. It is constructed by providing these 6 values in order. The angles may be given in degrees or radians, if they are less than pi they are assumed to be in radians. If they angles are omitted they are assumed to be 90 degrees.

- **clipper::Cell** (p. 133)

  The cell class (**clipper::Cell** (p. 133)) is constructed from a cell descriptor (**clipper::Cell_descr** (p. 137)), but additionally contains derived information, including the cell volume, the orthogonalising and fractionalising matrices (for converting between orthogonal and fractional coordinates) and the metric tensors (for computing the distance between points in real space, or the resolution of a reflection in reciprocal space).

It provides methods for conversion between fractional and orthogonal coordinates (frac_to_orth, orth_to_frac), calculating distances in real space (distancesq) and reflection resolutions (invresolsq).

- **clipper::Spgr_descr** (p. 421)

  The spacegroup descriptor class (**clipper::Spgr_descr** (p. 421)) contains the Hall symbol, which uniquely describes the spacegroup. It can be constructed from a spacegroup number, a traditional Hermann-Maugnuin symbol, or a list of symmetry operators (not implemented).

- **clipper::Spacegroup** (p. 413)

  The spacegroup class (**clipper::Spacegroup** (p. 413)) is constructed from a spacegroup descriptor (**clipper::Spgr_descr** (p. 421)), but additionally contains a list of all the symmetry operators, and tables for looking up inverses of symmetry operators.

  It provides methods for returning symmetry operators, along with other information such as the point group and Laue group.

  Symmetry operators are represented as a 3x3 matrix and 3x1 vector of either real numbers (Rsymop) or integers (Isymop). For integer symops the translation components are scaled by a factor of 12. Both real and integer symops provide methods for transforming Miller indices (sym_hkl) and fractional coordinates (sym_coord), and for calculating the phase shift associated with the symmetry transformation of a reflection (sym_phase_shift).

2.2.3 Data objects

Reciprocal space data

The reciprocal space data object is divided into two parts: the first part (**clipper::HKL_info** (p. 284)) handles indexing of reflections and information derived from the Miller indices, and the second part contains the actual data. The reasons for this division are that it is common to have several lists of information for each reflection, so a single list of Miller indices can be used for several lists of data. This also allows the use of optimised tables for looking up reflections, without wasting space duplicating them for several data lists.

However, if reciprocal space data are stored which do not have the same list of Miller indices, it is quite possible to provide several **clipper::HKL_info** (p. 284) objects, one for each list of data.

- **clipper::HKL_info** (p. 284)

  The reflection indexing class (**clipper::HKL_info** (p. 284)) contains a cell, a spacegroup, and a list of Miller indices. Reflection data is most commonly accessed by looping through each reflection in turn, and so the reflections can be referred to by number, or by coordinate-reference types provided for that purpose. However, it is occasionally necessary to refer to a reflection directly by its Miller index. For this purpose, a fast lookup table is also maintained, allowing efficient access to any reflection. This operation may however require searching through several symmetry equivalents, so there is still some overhead. The overhead is reduced by using the appropriate coordinate-reference type (**clipper::HKL_info::HKL_reference_coord** (p. 293)), which stores the last use symmetry operator as a guess at which operator will be required next.

  clipper::HKL_info (p. 284) is constructed by providing a **clipper::Spacegroup** (p. 413) and **clipper::Cell** (p. 133) (unless the container version is used). A method is provided to generate a list of unique non-absent reflections to a given resolution (generate_hkl_list), or alternatively a list may be imported using an I/O object.

Coordinate-Reference Types are provided for looping through the list of reflections (**clipper::HKL_info::HKL_reference_index** (p. 296)) or accessing reflections by Miller index (**clipper::HKL_info::HKL_reference_coord** (p. 293)). The latter is optimised for the case where reflections are taken consecutively from the same region of reciprocal space.

- clipper::HKL_data<T>

  The reflection data object is a template object, where the template type is the type of crystallographic information to be stored in the object. It simply stores a list of data of that type.

  Data types typically include several values. Examples include measured X-ray intensity and its standard deviation (I_sigI), structure factor magnitude and phase (F_sigF), and Hendrickson-Lattman coefficients (ABCD). Data types are derived from a base type (Datatype_base), and should override all the methods of that type. This will allow the data to be automatically transformed about reciprocal space, and imported or exported to a file, as required.

  The class provides methods for accessing the data by index number, by coordinate-reference type, or by Miller index. The last method should avoided for speed critical tasks.

  The coordinate-reference types are typedef'ed from those in **clipper::HKL_info** (p. 284), and may be used interchangeably with them. However clipper::HKL_data<T> provides an additional method (next_data) for accessing only the non-missing data in a list.

Real space data

- clipper::Xmap<T>

  The crystallographic map data object is a template object, with the template type being the type of the data; usually double or float for maps, or int or bool for masks. (In future a specialisation for complex data will be available, with a complex fft method.)

  The map object is constructed by providing a **clipper::Spacegroup** (p. 413) and **clipper::Cell** (p. 133) (unless the container version is used), and the unit cell sampling (**clipper::Grid_sampling** (p. 255)). Methods are provided to set and get data by grid coordinate (**clipper::Coord_grid** (p. 175)), however it is more efficient to use coordinate-reference types for this purpose.

  Coordinate-Reference Types are provided for looping through the unique map grid points (**clipper::Xmap_base::Map_reference_index** (p. 328)) or accessing map grid points by coordinate (**clipper::Xmap_base::Map_reference_coord** (p. 323)). The latter is optimised for the case where map values are taken consecutively from the same region of crystal space. Methods are also provided for moving one grid step along any of the cell axes.

- clipper::NXmap<T>

  The non-crystallographic map data object is a template object, with the template type being the type of the data; usually double or float for maps, or int or bool for masks. It contains a finite bounded grid of data with no symmetry or repeat. A rotation-translation operator allows the grid coordinates to be related to an arbitrary orthogonal coordinate frame. This will often relate to the orthogonal coordinate representation of a portion of some unit cell to which the map is related.

  Coordinate-Reference Types are provided for looping through the map grid points (**clipper::NXmap_base::Map_reference_index** (p. 330)) or accessing map grid points by coordinate (**clipper::NXmap_base::Map_reference_coord** (p. 326)).

Other objects

- **clipper::FFTmap** (p. 215)

  The FFT map (**clipper::FFTmap** (p. 215)) is a map used for calculating Fast Fourier transforms. It can hold represent data in either real or reciprocal space, and transform between representations. The data is stored covering the whole of the P1 unit cell in real space, or the L>0 hemisphere in reciprocal space, however the crystallographic symmetry of the data is maintained at all times. This is achieved by setting every symmetry equivalent value when writing to the object. When reading the symmetry is assumed and only the requested value is returned.

  In general it is not necessary to use an FFT map directly, since one will be created whenever an FFT is required. However if data is being transformed very frequently, or if very fast random access to reflection or map values is required, then it may be beneficial to use an **clipper::FFTmap** (p. 215) for this purpose.

- **clipper::ResolutionFn** (p. 392)

  The resolution function evaluator is a class for the calculation of arbitrary functions of resolution or position in reciprocal space. It can be used for the generation of Wilson plots, or for the calculation of simple likelihood function such as sigma-a. It is a superior replacement for any calculation which would previously have been performed using resolution bins.

2.2.4 Input/Output objects

- clipper::MTZfile

  Import/export class for MTZ files.

- clipper::MAPfile

  Import/export class for CCP4 maps.

2.2.5 Coordinate-reference types.

Coordinate-reference types are provided for accessing data in both real and reciprocal space. Data is commonly referred to in one of two ways:

1. by index into a list of unique data, or
2. by 3-dimensional coordinate. The coordinate-reference types are designed to make switching between the two forms of reference easier. Two coordinate-reference types are provided in each space - one which behaves more like an index, and one which behaves more like a coordinate. However, either may be asked to return either an index or a coordinate on request. References may also return other useful information, for example the HKL_-reference types will return the resolution and reflection class of a reflection (i.e. centricity, phase restriction, multiplicity).

The coordinate reference types are as follows:

```
clipper::HKL_info::HKL_reference_index
clipper::HKL_info::HKL_reference_coord
clipper::XMap_base::Map_reference_index
clipper::XMap_base::Map_reference_coord
clipper::NXMap_base::Map_reference_index
clipper::NXMap_base::Map_reference_coord
```

The index-like reference is simply an index, and a pointer to the parent object, which allows the corresponding coordinate to be looked up. It provides a convenient way to access either every unique reflection in a list, or every unique map grid point in the asymmetric unit, in turn. The coordinate-reference type is usually initialised by assignment from the 'first' method in the parent object. It may then be incremented by its own 'next' method. Completion is tested by the 'last' method. For example, if hklinfo is an object of type **clipper::HKL_info** (p. 284) and xmap is an object of type clipper::Xmap<double>, then the following loops could be used to access all the data:

```
for ( clipper::HKL_info::HKL_reference_index ih = hklinfo.first(); !ih.last();
    ih.next )
```

However, if the Miller indices of a reflection are are required, the reference type may be queried directly. Other information can also be obtained:

```
HKL hkl = ih.hkl();
int index = ih.index();
ftype s = ih.invresolsq();
HKL_class cls = ih.hkl_class();
```

Similarly, if hkldata is an object of type clipper::HKL_data<I_sigI>, then the following will loop over all non-missing data in the list:

```
for ( clipper::HKL_info::HKL_reference_index ih = hkldata.first_data(); !ih.las
    t(); ih.next_data(hkldata) )
```

The reference types in real space are equivalent:

```
for ( clipper::Xmap_base::Map_reference_index ix = xmap.first(); !ix.last(); ix
    .next ) {
  int i = ix.index()
  Coord_grid coord = ix.coord();
}
```

The coordinate-like types are more sophisticated. They hold a coordinate and a pointer to the parent object. However, the corresponding index is also stored (if it exists), along with the symmetry operator (and in reciprocal space the Friedel operator) used to obtain it. If an new coordinate is assigned to the reference, then this symmetry operator will be tried first, providing a significant performance gain.

The differences between the index-like and coordinate-like reference types can be summarised as follows:

- index-like types can only refer to the position of a stored datum, i.e. a coordinate in the stored ASU or a non-absent reflection in the reciprocal ASU.
- coodinate-like types can refer to any possible position, and therefore also store the symmetry transformations required to get back to the stored data.

Generally index-like types are used to loop through the stored data, whereas coordinate-like types are used to interact with data stored in a different way.

Chapter 3

Conventions

3.1 Introduction

This page discusses the conventions of coding style and memory management adopted in the Clipper libraries.

3.2 Notation

- Miller indices have componented h, k, l.
- Fractional coordinates have components u, v, w.
- Orthogonal coordinates have components x, y, z.
- The squared distance from the origin in reciprocal space, in A^{-2} , is usually referred to as 'invresolsq'. This is equal to $4\sin^2\theta/\lambda^2$
- Temperature factors and anisotrpic displacement parameters are always given as U-values.
- Angles are always given in radians.

3.3 Coding Style

3.3.1 Dialect

Clipper is written in C++, with extensive use of modern constructs such templates, and in particular the Standard Template Library (STL). The coding style of the library, and its API, both reflect this.

In particular, a developer writing code using Clipper objects should be able to do so with minimal or no use of pointers, or of the built-in `new` and `delete` operators. References are used in preference to pointers where possible. Heap objects, with the corresponding dangers of memory leakage, are avoided. (However, for the cases where the developer needs to use heap objects, an optional mechanism is provided for automatic garbage collection).

Clipper objects should generally be passed by reference rather than by value.

3.3.2 Floating-point representation.

The package may be compiled to use either `float` or `double` for the representation of floating-point numbers. This is acheived by defining floating-point values with a user-defined type `ftype`. This is set by a `typedef` in lib/clipper_util.h.

Data may have a different type. Since data objects may be large, these may be individually typed to save memory or provide additional precision as required. Data types are based on templates in lib/clipper_datatypes.h.

The type for Fast-Fourier transforms is dependent on the FFTw library to which Clipper is linked. FFTw may be compiled for `float` or `double`.

Since crystallographic FFTs generally have quite short dimensions (< 1000), `float` is recommended.

3.3.3 Namespaces

All Clipper objects belong to the `Clipper` namespace, and can be accessed by using the `Clipper::` prefix or

```
using namespace Clipper;
```

Templates for reciprocal space datatypes are provided in the `Clipper::datatypes` namespace. Instantiations of these types for `float` and `double` data are provided in `Clipper::data32` and `Clipper::data64`.

Classes are all named with an initial capital letter. Method names are all lowercase, with words separated by underscore ('_'). A handful of top level functions are implemented; these have an initial capital letter to distinguish them from standard library functions.

3.4 Memory management

All Clipper objects are designed to be created on the stack. Such objects are destroyed automatically when they go out of scope. Large or variable-sized data items within an object are generally held in STL vectors, and so are stored on the heap, and automatically destroyed with the associated object.

Sometimes it may be useful to create Clipper container objects on the heap. In this case, the developer can arrange for the object to be destroyed when it is no-longer needed. However an alternative, automatic, method for memory management is provided, in the form of a `destroyed_with_parent` flag. If in object is created on the heap (i.e. with `new`), this flag may be set as follows:

```
  Container* new_container = new Container( parent, "A container" );
  new_container->set_destroyed_with_parent();
```

Then, as soon as `parent` is destroyed, either by going out of scope, or by a `delete` operator, the new container will also be destroyed.

3.4.1 Passing objects

Clipper objects should generally be passed by reference rather than by value. In particular passing a container type by value will lead to a container which is orphaned from the tree.

Passing or assigning an object always results in a 'deep copy', i.e. all of the data associated with the object is copied.

3.5 Coding standards

Code in the Clipper library should obey the following standards. Some of these are stylistic, but most are simply good practice. Code which does not obey these standards will be rejected from my tree.

1. No C-style casts.
2. No reinterpret_cast<> except for conversion of external types.
3. No static_cast<> without a very strong performance argument.

4. No const_cast<>.
5. No preprocessor macros, except for a stand-alone portability library.
6. No global functions. Useful functions are contained in the Util class.
7. Absolutely strict use of const, both for parameters and methods.
8. Parameters are passed by reference.
9. No pointers in public APIs.
10. No pointers internally, expect for
    - testing for NULL after a dynamic_cast
    - member variables which point to other objects
11. No 'new' or 'malloc'. Use STL containers. (Exception: for explicitly memory managed objects. There are still a couple of 'new's in the MTZ i/o layer).
12. 'explicit' is used where necessary to prevent accidental type conversion.
13. Access specifiers are ordered public, protected, private.

There are a few places where I haven't yet finished implementing all of these standards, but I'm working on it.

Chapter 4

Coordinates and related types

4.1 Introduction

There are a large number of coordinate types used in crystallographic calculations, and a significant number of other small data types also in common use. These types, their uses, and the conversions between them will be discussed in the following sections.

The coordinate types are divided into two classes:

- Reciprocal space coordinates. These are used when handling data in reciprocal space, e.g. structure factor data. Generally only the HKL types is required, however for fast rotation functions other forms may be required.

- Real space coordinates. These are used when data is being manipulated in real space, e.g. electron density maps or coordinate models.

In addition, data types are defined for map gradients and curvatures and for anisotropic atomic displacement parameters (B or U values).

4.1.1 3-vector and matrix types.

All of the coordinate types are derived from a template class for a vector of 3 values; clipper::Vec3<T>. Generally T will be the floating point type, but in some cases vectors of `int` are used. **Vec3**<> (p. 461) also defines various simple arithmetic operations, including addition, subtraction, negation, and scaling. Any coordinate type may be constructed from a **Vec3**<> (p. 461) by using an explicit constructor. (Implicit conversion is not allowed, since this would allow accidental type conversion between coordinate types). The elements of the vector may be accessed as v[0], v[1], and v[2], however the coordinate types all define type-specific names for the members.

Similarly, clipper::Mat33<T> provides a template class for a 3x3 matrix of values. Matrix-vector and matrix-matrix products are defined. If a vector pre-multiplies a matrix, it is assumed that the vector is a row rather than column vector.

Symmetric matrices are common on crystallography, so a type is defined for these; clipper::Mat33sym<T>. This does not define any operations except conversion to or from Mat33<T>. This form is therefore used mainly for packed storage of some quantities, such as anisotropic U values.

Rotation-translation operators are provided by the clipper::RTop<T> template class. A rotation-translation operator consists of a rotation defined by a Mat33<T>, and a translation defined by a Vec3<T>. The matrix and vector components may be accessed through the rot() and trn() methods. RTop's may be applied to vectors or combined using the '*' operator.

4.2 Coordinate types

4.2.1 Reciprocal space coordinate types

HKL

The integral index (sometimes referred to as a Miller index) of a reflection. It has components h, k, and l.

Coord_reci_frac

Reciprocal space fractional coordinate. This is identical to an HKL, except that the values are floating point. This would commonly be used in rotation function calculations when interpolation is required in reciprocal space.

Coord_reci_orth

Reciprocal space orthogonal coordinate. This is a coordinate defined in inverse Angstroms along three orthogonal axes. The exact convention for selection of these axes is defined by the **clipper::Cell** (p. 133) class.

4.2.2 Real space coordinate types

Coord_orth

Orthogonal coordinate in Angstroms. This is a coordinate defined in Angstroms along three orthogonal axes. The exact convention for selection of these axes is defined by the **clipper::Cell** (p. 133) class.

Coord_frac

Fractional (crystal) coordinate. This is defined as the fractional position in the unit cell, with each component traversing a unique cell over the range (0..1).

Coord_map

Map coordinate. This is a floating point coordinate representing a position in the frame of a map. In the case of a crystallographic map, this will simply be the fractional coordinate scaled by the grid sampling of the unit cell (not the asymmetric unit). However in most cases Coord_frac is used in preference for crystallographic maps. In the case of a non-crystallographic map, fractional coordinates are undefined, but the NXmap will define a conversion between orthogonal and map coordinates.

Coord_grid

Map grid coordinate. This is the integer version of a Coord_map, and represents the index of a grid point in a map. The origin is always at the origin of the coordinate frame of the map, thus for crystallographic maps the origin is the crystallographic origin. For non-crystallographic maps, the origin is the first grid point in the map.

4.3 Derivative types

Derivatives and curvatures may be with respect to orthogonal, fractional, or map coordinates. When performing calculations on a crystallographic map, derivatives are usually with respect to fractional coordinates, however map coordinates may be used if preferred. The resulting derivatives may then be converted to orthogonal coordinates if necessary. When performing calculations on a non-crystallographic map, derivatives are calculated with respect to map coordinates.

Derivatives are stored as **Vec3<>** (p. 461) with the template type matching the type of the map. Curvatures are stored as a Mat33<>.

4.4 Operator Types

Two types of rotation-translation operators are defined as special cases of **RTop<>** (p. 402) with a floating point template type.

RTop_frac

This is a rotation-translation operator defined on fractional coordinates. A Crystallographic symmetry operator is an example of this type, and in fact Symop is derived from this class.

RTop_orth

This is a rotation-translation operator defined on orthogonal coordinates. A non-crystallographic symmetry operator is an example of this type.

4.5 Other types

Anisotropic atomic displacement parameters (ADPs) are also defined. These are derived from **Mat33sym<>** (p. 336) with a floating point template type. There are two types:

U_aniso_orth

Anisotropic atomic displacement parameters defined on orthogonal Angstrom coordinates, as used in PDB files. The terms have units of inverse angstroms squared.

U_aniso_frac

Anisotropic atomic displacement parameters defined on fractional coordinates.

4.6 Conversion of coordinates and related types

Coordinates and related types are converted by the following means:

- Any fraction coordinate, gradient, curvature, operator, or ADP (with the `_frac` suffix) may be converted to an orthogonal coordinate, etc. (with the `_orth` suffix) or vice-versa using the appropriate method of the Cell class. The name of the required method is always the return type in lower case. e.g. to convert a **clipper::Coord_orth** (p. 182) to a **clipper::Coord_frac** (p. 172), call the clipper::Cell::coord_frac() method of the appropriate **clipper::Cell** (p. 133) object.

- A real space fractional coordinate, gradient, or curvature may be converted to a map coordinate, gradient, or curvature or vice-versa, by using the appropriate method of the **clipper::Grid_sampling** (p. 255) class. The name of the required method is always the return type in lower case. Fractional coordinates may also be converted directly to grid coordinates and vice-versa.

- Coord_grid can be converted directly to Coord_map using the appropriate constructor. Coord_map can be converted to Coord_grid using a method specifying the type of rounding required.

- Coord_map for a non-crystallographic map may be converted to a Coord_orth or vice-versa using the methods of the **clipper::NXmap** (p. 357) object concerned.

Chapter 5

Developing using Clipper

5.1 An FFT example

Most crystallographic calculations are based upon the observed data, i.e structure factors, phase probability estimates, and associated data. This page describes how to read observed structure factors and phases from an MTZ file, and perform a Fourier transformation to obtain an electron density map.

For reflection data to be meaningful, a cell, spacegroup, and resolution limit must first be defined. This can be achieved by defining a Spacegroup object (**clipper::Spacegroup** (p. 413)), a Cell object (**clipper::Cell** (p. 133)), and a resolution object (**clipper::Resolution** (p. 389)), and using them to create a reflection list (**clipper::HKL_info** (p. 284)). However, in this case the cell, spacegroup and resolution will be imported from the MTZ file, so it is not necessary to set their values. Therefore an uninitialised **clipper::HKL_info** (p. 284) object is created using the null constructor.

```
clipper::HKL_info myhkl();
```

If we want to calculate a map, we need a structure factor and phase. So we'll create a data list of type `F_phi` . The data list must refer to a reflection list, so we pass our **clipper::HKL_info** (p. 284) object to the constructor.

```
clipper::HKL_data<clipper::data32::F_phi> fphidata( myhkl );
```

Next, we want to open an MTZ file. I'll assume it is called `my.mtz` . To do this we create an `clipper::MTZfile` object. This is an i/o object which can be linked to an MTZ file by an open command, and can then fill Clipper data objects using the information from that file.

```
clipper::CCP4MTZfile mtzin;
mtzin.open_read( "my.mtz" );   // open new file
```

When we created the reflection list object, we did not provide a cell or spacegroup, or generate a reflection list. So we'll get those from the file now.

```
mtzin.import_hkl_info( myhkl );    // read sg, cell, reso, hkls
```

Now we want some actual data. We read a couple of MTZ columns into the data list we created earlier.

```
mtzin.import_hkl_data( fphidata, "/native/CuKa/[FCAL,PHICAL]" );
```

Note that we create the data-list as a child of the reflection list `myhkl` . (Actually, any descendent will do). The import operation will actually move it, by inserting the appropriate `CCrystal` and `CDataset` objects, according to the information on the file.

Note also that we give specify the MTZ columns by giving the full path: crystal, dataset, and column names. The column names are grouped in square brackets. (For older MTZ files, which do not have crystal and dataset entries, the dummy names "unnamed_crystal" and "unnamed_-dataset" are used.)

Now we close the MTZ file. **NOTE:** No data is actually read until you close the file!

```
mtzin.close_read();
```

Next, we create a map. For this we need to provide a grid. The grid depends on the spacegroup, cell, and resolution, which we get from the reflection list:

```
clipper::Grid_sampling mygrid( myhkl.spacegroup(), myhkl.cell(), myhkl.
      resolution() );  // define grid
```

Now we make the map itself, using the spacegroup, cell, and grid.

```
clipper::Xmap<float> mymap( myhkl.spacegroup(), myhkl.cell(), mygrid );  // defin
      e map
```

Next, we calculate the map by FFT.

```
mymap.fft_from( fphidata );                  // generate map
```

Finally, then map is written out to an output file.

```
clipper::CCP4MAPfile mapout;
mapout.open_write( "my.map" );      // write map
mapout.export_xmap( mymap );
mapout.close_write();
```

To summarise, here is the complete program:

```
  clipper::HKL_info myhkl;
  clipper::HKL_data<clipper::data32::F_phi> fphidata( myhkl );

  clipper::CCP4MTZfile mtzin;
  mtzin.open_read( "my.mtz" );        // open new file
  mtzin.import_hkl_info( myhkl );     // read sg, cell, reso, hkls
  mtzin.import_hkl_data( fphidata, "/native/CuKa/[FCAL,PHICAL]" );
  mtzin.close_read();

  clipper::Grid_sampling mygrid( myhkl.spacegroup(), myhkl.cell(), myhkl.
      resolution() );  // define grid
  clipper::Xmap<float> mymap( myhkl.spacegroup(), myhkl.cell(), mygrid );  // def
      ine map
  mymap.fft_from( fphidata );                  // generate map

  clipper::CCP4MAPfile mapout;
  mapout.open_write( "my.map" );      // write map
  mapout.export_xmap( mymap );
  mapout.close_write();
```

5.2 Libraries and Include files

Which header and library files should you include in your application? The libraries are as follows, and there is an aggregated header file for each library.

Contents	**Library**	**Include**
Reflection, maps, models	$PREFIX/lib/libclipper.a	$PREFIX/include/clipper/-clipper.h
Function objects (e.g. SFcalc)	$PREFIX/lib/libclipper-contrib.a	$PREFIX/include/clipper/clipper-contrib.h
CCP4 mtz/map i/o	$PREFIX/lib/libclipper-ccp4.a	$PREFIX/include/clipper/clipper-ccp4.h
SHELX phs i/o	$PREFIX/lib/libclipper-phs.a	$PREFIX/include/clipper/clipper-phs.h
cctbx interface	$PREFIX/lib/libclipper-cctbx.a	$PREFIX/include/clipper/clipper-cctbx.h

Alternatively, you can include individual header files from the subdirectories of include/clipper/. The precise header files can be identified by the following include tree:

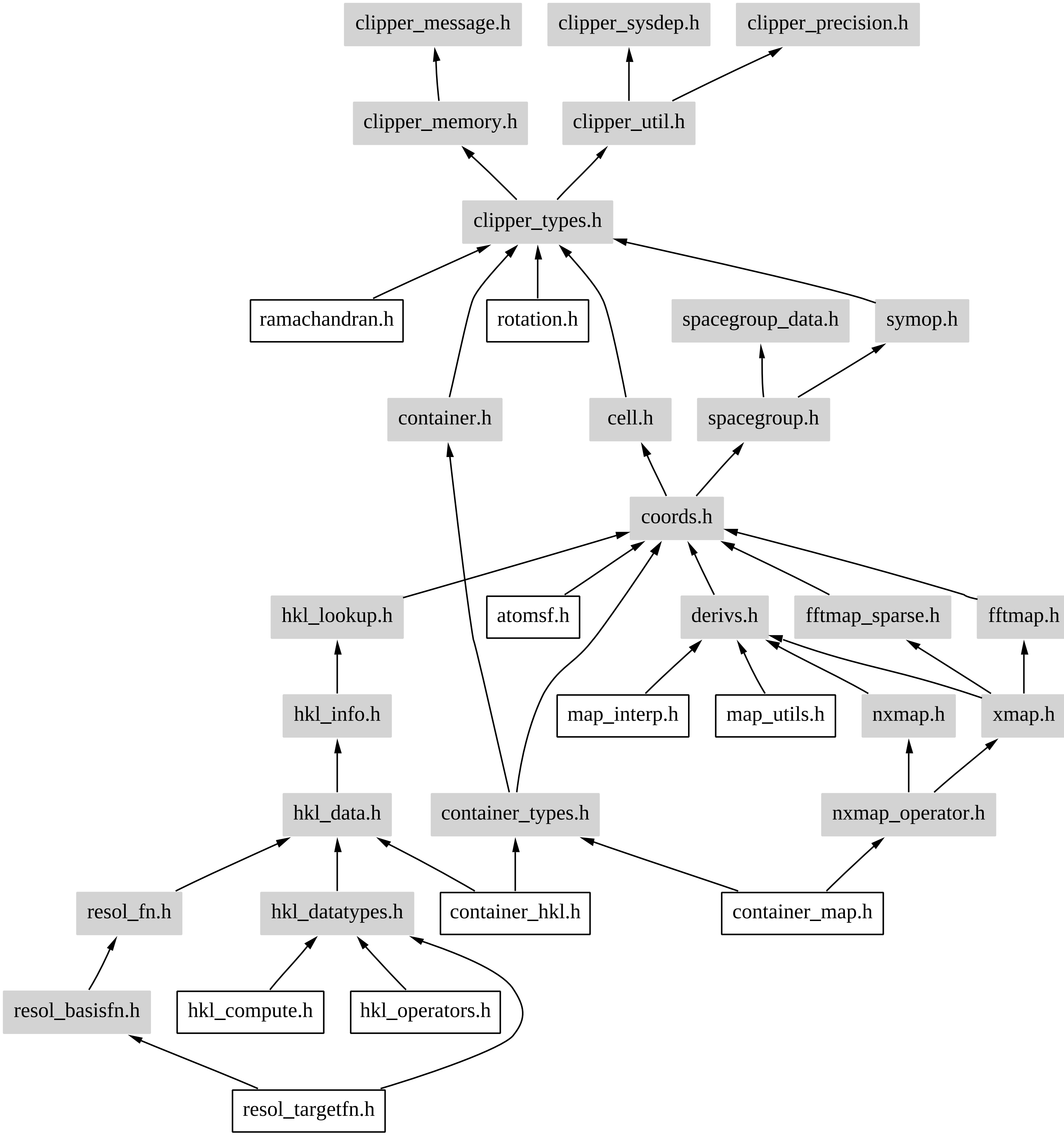


Figure 5.1: =10cm

Chapter 6

Developing using Reflection Data

6.1 Understanding HKL_info and HKL_data<>

Storage of reciprocal space data in Clipper involved two objects: HKL_info and HKL_data<>. The first stores a list of reflections (HKLs), and the second stores a list of data of some type for each reflection.

Why is this division made? Because often, many types of data will be stored for the same reflection list, and duplicating the list of reflections for each data list would be wasteful. The relationship of an HKL_info object and several HKL_data<> objects is shown below.

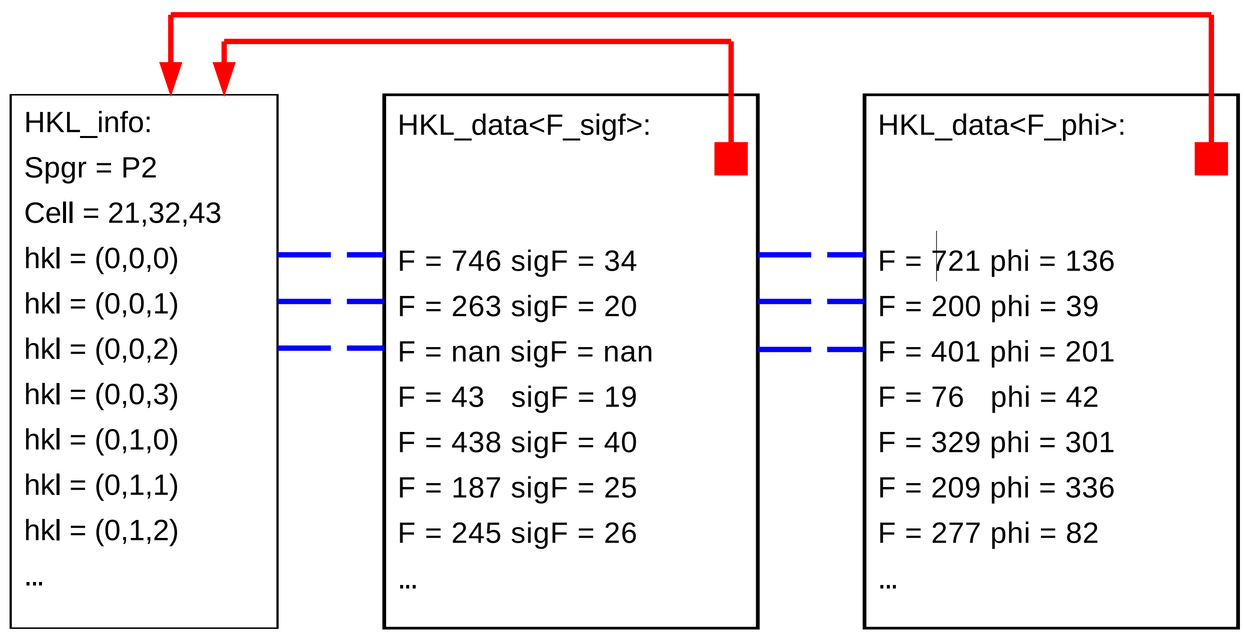


Figure 6.1: width=8cm

These objects handle all the crystallographic symmetry operations associated with storing data in reciprocal space. To the programmer, the data appears to fill a sphere of reciprocal space. However, only a unique set of data is stored, and any changes to a data values are automatically reflected in the symmetry and Friedel related reflections.

6.2 The HKL_info object

The HKL_info object stores a list of reflections, transformed into a standard reciprocal space asymmetric unit. For example:

- **P1:** data is stored for l>0 or (l=0 and (h>0 or (h=0 and k>=0)))
- **P2:** data is stored for k>=0 and (l>0 or (l=0 and h>=0))
- **P212121:** data is stored for h>=0 and k>=0 and l>=0

However, the programmer should never need to know which asymmetric unit is being used, as data in other parts of reciprocal space are generated automatically.

The HKL_info object also stores a lookup table, which is used to rapidly determine the location in the list of a given HKL. It also stores lookup tables for the reflection class (HKL_class) and resolution (invresolsq).

The reflection list depends upon the spacegroup (which determines the reciprocal asymmetric unit and systematic absences), the cell parameters, and the resolution limit. Therefore to initialise an HKL_info object, objects of types Spacegroup, Cell and Resolution are passed to the constructor or initialiser.

Reflection classes

The 'class' of a reflection is a function of its HKL and the spacegroup, and describes whether it is centric, systematically absent, and what its allowed phase and symmetry enhancement factor or multiplicity are (i.e. epsilon). The class of a reflection is described by the class **clipper::HKL_-class** (p. 265).

Reflection resolution

The resolution of a reflection is a function of its HKL and the unit cell parameters. It is generally handled in terms of the inverse resolution squared, or 'invresolsq'. This is equal to $4\sin^2\theta/\lambda^2$.

6.3 The HKL_data<> object

The data object is not much more than an array of data, but it has a number of special methods to make crystallographic operations on that data more convenient. It is defined as a template class, where the template type is the type of crystallographic information to be stored in the object. It simply stores a list of data of that type, and a pointer to the parent HKL_info object, which defined the HKL for each element in the list. Additionally a pointer to a Cell object is stored, which may optionally be used for the case where different data comes from slight different unit cells (e.g. RT and frozen data). Therefore to initialise an HKL_data object, an HKL_info object and optionally a Cell object are passed to the constructor or initialiser.

Data types typically include several values. Examples include measured X-ray intensity and its standard deviation (I_sigI), structure factor magnitude and phase (F_sigF), and Hendrickson-Lattman coefficients (ABCD). Data types are derived from a base type (Datatype_base), and should override all the methods of that type. This will allow the data to be automatically transformed about reciprocal space, and imported or exported to a file, as required.

Methods are provided to access the data by index or by HKL. Any transformations which must be applied to the data in obtaining a symmetry or Friedel related value are applied automatically.

In order to use the class efficiently, some important difficulties must be borne in mind.

A problem arises when we wish to apply some transformation to the values stored in a data list. In this case, we must access every unique value in the asymmetric unit once and once only, applying the desired transformation. Only then will the entire data list have been transformed correctly.

A second problem arises if we want to access the stored value of the data at some position in reciprocal space, for example to expand to a lower spacegroup. Then it is necessary to search through all the symmetry operators, applying each one in turn to the desired HKL to find the operator which brings the HKL into the stored asymmetric unit, with a Friedel inversion if necessary. Clearly this can be time consuming, especially if there are many symmetry operators.

Both these problems are addressed by the use of HKL reference types. These come in two forms:

- index-like references (**clipper::HKL_info::HKL_reference_index** (p. 296))
- coordinate-like references (**clipper::HKL_info::HKL_reference_coord** (p. 293))

The index-like reference behaves like an index: it stores a reference to an HKL_info object and a position in the reflection list. It is used to loop over all the values in the reflection list, using the HKL_info::first(), and HKL_reference_index::last() and HKL_reference_index::next() methods. The HKL corresponding to the index, its resolution, and reflection class can be returned at any point.

The coordinate-like reference behaves like an HKL coordinate: it stores a reference to an HKL_info and an HKL. However to enhance performance it also stores the position in the list corresponding to that HKL, and the number of the symmetry operator used to get back into the stored asymmetric unit, along with a flag to signify Friedel inversion. Since reflections are usually accessed systematically, the next HKL used will commonly require the same symmetry operator, and so that operator is tried first. Methods are provided for incrementing and decrementing along the h, k, and l directions.

The differences between the index-like and coordinate-like reference types can be summarised as follows:

- index-like types can only refer to the position of a stored datum, i.e. reflection in this list.
- coordinate-like types can refer to any possible position, and therefore also store the symmetry transformations required to get back to the stored data.

HKL reference types may be shared between any data lists which have the same reflection list. It is the responsibility of the programmer to ensure this restriction is obeyed.

6.4 HKL_data code fragments

6.4.1 Importing and exporting HKL_data

Currently objects are provided for import and export to CCP4 '.mtz' files and XtalView/SHELX '.phs' files. (More will be added later).

To import a datalist from an MTZ files we need an HKL_info object to hold the reflection list and an HKL_data object to hold the actual data. We also need MTZdataset and MTZcrystal objects to hold the additional information which will be returned from the MTZ file. Then we create a CCP4MTZfile object, open it onto a file, and read the reflection list and data.

```
clipper::HKL_info myhkl;            // define objects
clipper::MTZdataset myset;
clipper::MTZcrystal myxtl;
clipper::HKL_data<clipper::data32::F_phi> fphidata( myhkl, myxtl );

clipper::CCP4MTZfile mtzin;
mtzin.open_read( "my.mtz" );        // open new file
mtzin.import_hkl_info( myhkl );     // read sg, cell, reso, hkls
mtzin.import_hkl_data( fphidata, myset, myxtl, "native/CuKa/[FCAL PHICAL]" );
mtzin.close_read();
```

Similar functions are used for accessing XtalView/SHELX .phs files, although the interface is much simpler since these files do not contain labelled data.

6.4.2 Expanding reflection data to a lower symmetry

To expand a list of data to a lower symmetry, we need two reflection lists, one for each spacegroup; and two datalists, one for each reflection list. The lower symmetry list is then filled by looping over all the reflections in that list an requesting the value from the other list for that HKL.

```
clipper::HKL_info oldhkl( .... );
clipper::HKL_data<clipper::data32::f_phi> olddata(oldhkl);
// ---- fill the objects here ----
clipper::HKL_info newhkl( Spacegroup( Spgr_descr( 1 ) ),
                          oldhkl.cell(), oldhkl.resolution() );
clipper::HKL_data<clipper::data32::f_phi> newdata(oldhkl);
HKL_info::HKL_reference_index ih;
for ( ih = newhkl.first(); !ih.last; ih.next() ) {
  newdata[ih] = olddata[ih.hkl()];
}
```

Note that the '.hkl' is vital, as we want the data with the corresponding hkl, not the data from the corresponding position in the list. If efficiency is paramount, using an HKL_reference_coord to access the old list will save some searches over the symmetry operators:

```
clipper::HKL_info::HKL_reference_index ih;
clipper::HKL_info::HKL_reference_coord ik( oldhkl );
for ( ih = newhkl.first(); !ih.last; ih.next() ) {
  ik.set_hkl( ih.hkl() );
  newdata[ih] = olddata[ik];
}
```

6.4.3 Applying simple operations to a whole data list

While it is simple to loop through a reflection list and apply some transformation on the data, some simple operations have been automated using built-in C++ arithmetic operators for data of specific types, logical operators for data or any type, comparison operators for flags, and function 'Computation operators' for more complex operations.

Arithmetic operators.

Arithmetic operators are defined for the addition, subtraction, and scaling of map coefficients (i.e. HKL_data<datatypes::F_phi>), and for the addition and scaling of Hendrickson-Lattman coefficients (class HKL_data<datatypes::ABCD>). Thus, to add two lists of map coefficients, the following code is required:

```
clipper::HKL_data<clipper::data32::F_phi> fphi1, fphi2, fphi3;
// ---- set data here ----
fphi3 = fphi2 + fphi1;
```

The columns are added using vector addition. If any values in either list are missing, then the result is missing. Subtraction is similar. Multiplication by a scalar scales the magnitude of every non-missing element in the list.

Logical operators.

Standard C/C++ bitwise logical operators (&, |, ^, !) may be applied to any data list. For each data in the list, the value 'true' will be returned if the data is not missing, or false if it is missing.

The result of the operation is a new data list of type HKL_data<Flag_bool>, containing the results of the logical operation. This may be used in further logical operations, or may be used as a mask to eliminate data from a list using the HKL_data::mask() method.

Comparison operators.

Comparison operators (==, !=, >, <, >=, <=) may be applied to a data lists of flags (i.e. HKL_-data<datatypes::Flag>), to compare the values in the list with a single integer. This is commonly used in the handling of Free-R test sets. The result is a list of HKL_data<Flag_bool>, where the value of the flag for each reflection is the result of the compraison of the frag for that reflection and this given integer. So, for example, to make a list of data containing only the values for which the test set is numbered 18 or greater, use the following code:

```
clipper::HKL_data<clipper::data32::F_sigF> fsigf, fsigftest;
clipper::HKL_data<clipper::data32::Flag> flag;
// ---- set data here ----
fsigftest = fsigf;
fsigftest.mask( flag >= 18 );
```

Computation operators

Computation operators handle more complex crystallographic tasks, and will be discussed in more detail.

To use a computation operator, call the compute() method of the destination datalist. This method must be supplied with one or two source datalists, and a computation operator. This is an object which performs the computation for an individual reflection, and is usually constructed on the fly.

Some computation operators simply convert a datalist of one type to a datalist of another type. For example, you can convert a phase and weight to Hendrickson Lattman coefficients. (Of course C and D will be 0, because a phase and weight can only describe a unimodal distribution).

```
clipper::HKL_data<clipper::data32::Phi_fom> myphifom;
// ---- set data here ----
clipper::HKL_data<clipper::data32::ABCD> myabcd;
myabcd.compute( myphifom,
  clipper::data32::Compute_abcd_from_phifom() );
```

Some computation operators take data from two datalists. For example, you can calculate map coefficient (magnitude and phase) from a set of observed magnitudes and a phase and weight:

```
clipper::HKL_data<clipper::data32::F_sigF> myfsig;
clipper::HKL_data<clipper::data32::Phi_fom> myphifom;
// ---- set data here ----
clipper::HKL_data<clipper::data32::F_phi> myfphi;
myfphi.compute( myfsig, myphifom,
  clipper::data32::Compute_fphi_from_fsigf_phifom() );
```

Computation operators may operate on a datalist itself, and can also take parameters. These parameters are passed to the constructor of the computation operator. For example, to apply a scale factor of 2.0 and and U-value of 0.5 to a list of reflections, use the following code:

```
clipper::HKL_data<clipper::data32::F_sigF> myfsig;
// ---- set data here ----
myfsig.compute( myfsig,
  clipper::data32::Compute_scale_u<data32::F_sigF>( 2.0, 0.5 ) );
```

Computation operators are provided for performing addition, subtraction and scaling of map coefficients (i.e. F_phi), addition and computation of Hendrickson Lattman coefficients, and for scaling data. It is fairly simple to define new computation operators, see **core/hkl_compute.h** (p. ??).

6.5 Defining a new reflection datatype

Several data types are defined in the file `hkl_datatypes.h` (p. ??) . Defining a new type proceeds as follows:

1. Define a struct containing the data which needs to be stored for each reflection. A default constructor should be supplied which initialises all the data to NaN for floats, or an illegal value for ints (e.g. -1 for Free-R flag).
2. Defined a member function 'void friedel()' which changes the values of the data to the values of the Friedel opposite of the data. (e.g. a magnitude is unchanged, a phase will be negated).
3. Defined a member function 'void shift_phase(const float)' which chages the values of the data to the value of a symmetry equivalent with the given phase shift from the original. (e.g. a magnitude is unchanged, a phase will have the shift added to it).
4. Define a member function 'static const string type()' for that struct which returns a 'type name' string for this type. This is used to identify the data type and to infer column names in an mtz file.

For example, an F_phi group are defined as follows:

```
struct F_phi
{
  float f,phi;
  F_phi() { f=phi=Nan(); }
  static const String type() { return "F_phi"; }
  void friedel() { phi=-phi; }
  void shift_phase(const ftype dphi) { phi+=dphi; }
  const bool missing() const { return (isnan(f) || isnan(phi)); }
};
```

The datalist types are constructed from the individual data type by a template class.

If you need to store your new datatype in an MTZ file, you must also define an MTZ_iotype by derivation from clipper::MTZ_iotype_base, create a static instance of the new MTZ_iotype, and add it to the mtz_iotype_registry.

Chapter 7

Developing using Crystallographic Maps

7.1 Understanding Xmaps

The most commonly used map object in Clipper is the crystallographic map, or Xmap. This is the map class which is used to describe any function of position in a crystal which obeys the crystal cell repeat and symmetry. Electron density is the most common example.

The aim of the Xmap is to provide a fast and efficient way of storing some property, in such a way that the crystallographic symmetry and cell repeat are imposed without any effort from the programmer. Therefore, a **clipper::Xmap** (p. 463) appears to be infinite in every direction, allowing the map to be queried at any position in crystal space. However, only a unique subregion of a single cell is stored. If a value in the map is changed, then every copy of that value throughout crystal space also changes.

7.1.1 Grids

Representing the electron density across the whole unit cell is simple enough. A sampling is chosen which is sufficient to represent the electron density to the desired level of detail. The unit cell is then divided into a 3-dimensional grid with the given sampling. The first point is at the origin, and the first point along each line through the array lies on the surface of the unit cell. The last point along each line is the point before the surface of the next cell. Since the cell repeats, there is no point storing each surface twice.

There are generally some restrictions on the grid sampling, imposed by the cell dimensions, space-group, and FFT requirements. The **clipper::Grid_sampling** (p. 255) class can be used to select an appropriate grid sampling for any particular problem.

The following image shows a slice of a unit cell which has been sampled on a (6x8) grid. 48 points are stored, with fractional coordinates 0...5/6 on the horizontal axis and 0...7/8 on the vertical axis.

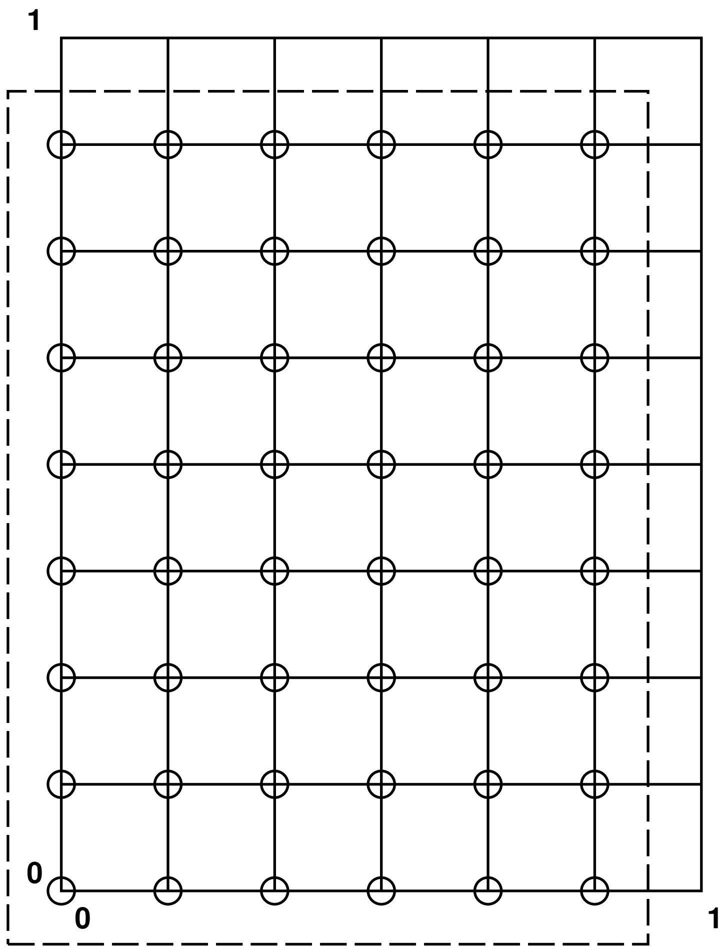


Figure 7.1: width=5cm

The fraction coordinate of any map grid point may therefore be determined by dividing the zero-based grid coordinate by the grid sampling along each direction.

7.1.2 Asymmetric units

Unfortunately the general case in crystallography is rather more complex. In addition to the unit cell repeat, most molecules crystallise with some sort of symmetry within the unit cell. Thus the unique region of crystal space, referred to as an asymmetric unit (ASU), is smaller than the the unit cell by a factor equal to the number of symmetry operators.

A naive approach to this problem would be to store only an oblong 'brick' within the unit cell. Unfortunately in the general case this does not work, since some symmetry operators are not aligned along grid axes. Even simple cases, such as a single 2-fold rotation axis, can cause problems, as illustrated in the figure below:

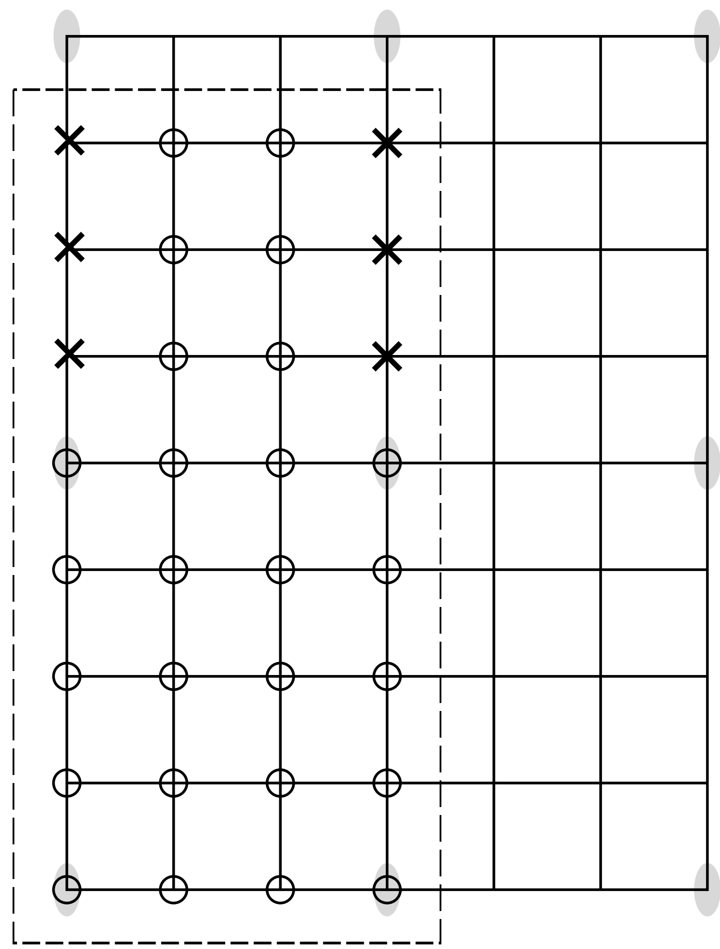


Figure 7.2: width=5cm

The map section shows a sampling of a P2 unit cell. An attempt has been made to select an asymmetric unit using a rectangular subregion of the section. However it can be seen that the origin is related to itself, and the 2-fold axis also relates the points on the left and right-hand edges of the oblong to other points on the same edge. Therefore 26 points are required to store a complete asymmetric unit section, and the brick containing those points is just over half the unit cell wide and contains 32 points.

The approach used by the Xmap class is therefore to store a compact brick of density, along with a flag array marking which points within that brick are considered to be part of the ASU, and which are considered to be outside the ASU.

Note:

In actual fact, **clipper** (p. 81) stores a border of 1 extra cell around the ASU brick. This extra row is used to speed up systematic searches through the density by flagging the edge of the ASU. In addition these extra points cache the number of the symop required to get back into the ASU, making systematic density searches very efficient.

7.1.3 Coordinates

There are three kinds of coordinates which are commonly used in real space. They are:

- Orthogonal (Angstom) coordinates (**clipper::Coord_orth** (p. 182))
- Fractional (cell) coordinates (**clipper::Coord_frac** (p. 172))
- Grid coordinates (**clipper::Coord_grid** (p. 175))

Grid coordinates are always integers, the others are floating point values.

Conversion between orthogonal and fractional coordinates is performed using the **clipper::Coord_orth::coord_frac()** (p. 184) and **clipper::Coord_frac::coord_orth()** (p. 173) methods, supplying the cell as an argument. e.g. corth.coord_frac(xmap.cell())

Conversion between fractional and grid coordinates is performed using the **clipper::Coord_-map::coord_frac()** (p. 181) and **clipper::Coord_frac::coord_map()** (p. 174) methods, supplying the grid as an argument. e.g. cfrac.coord_map(xmap.grid_sampling())

7.2 The Xmap class

The **clipper** (p. 81) Xmap class therefore holds a flag array and data array corresponding to a compact brick containing at least complete asymmetric unit.

The Xmap is constructed by providing a spacegroup, cell, and grid sampling. These are used to determine an appropriate ASU, and allocate an array to store the map data. The data may then be read or written by providing the appropriate grid coordinate. Alternatively, interpolated values and gradients at non-grid locations may be read by providing a fraction coordinate. A variety of interpolation methods are provided for this purpose.

In order to use the class efficiently, some important difficulties must be borne in mind.

A problem arises when we wish to apply some transformation to the values stored in the map. In this case, we must access every unique value in the asymmetric unit once and once only, applying the desired transformation. Only then will the entire map have been transformed correctly.

A second problem arises if we want to access the stored value of the density at some position in crystal space, represented by a grid coordinate. Then it is necessary to search through all the symmetry operators, applying each one in turn to the coordinate to find the operator which brings the coordinate into the stored asymmetric unit. Any cell translations must also be taken into account. The value of the map for that coordinate can then be returned. Clearly this can be time consuming, especially if there are many symmetry operators.

Both these problems are addressed by the use of map reference types. These come in two forms:

- index-like references (**clipper::Xmap_base::Map_reference_index** (p. 328))
- coordinate-like references (**clipper::Xmap_base::Map_reference_coord** (p. 323))

The index-like reference behaves like an index: it stores a reference to a map and an index into that map. It is used to loop over all the values in the asymmetric unit of a map, using the Xmap<>::first(), and Map_reference_index::last() and Map_reference_index::next() methods. The coordinate corresponding to the index can be returned at any point.

The coordinate-like reference behaves like a coordinate: it stores a reference to a map and a grid coordinate into that map. However to enhance performance it also stores the index corresponding to that coordinate, and the number of the symmetry operator used to get back into the stored asymmetric unit. Since maps are usually accessed systematically, the next coordinate used will commonly require the same symmetry operator, and so that operator is tried first. An efficient caching mechanism makes incrementing or decrementing the coordinate along the u, v, or w directions particularly fast.

The differences between the index-like and coordinate-like reference types can be summarised as follows:

- index-like types can only refer to the position of a stored datum, i.e. a coordinate in the stored ASU.
- coordinate-like types can refer to any possible position, and therefore also store the symmetry transformations required to get back to the stored data.

Use of map reference types is always preferred over requesting a coordinate directly. The accessor methods for the map are designed to encourage efficient usage.

Map reference types may be shared between any maps which have the same symmetry and grid sampling. It is the responsibility of the programmer to ensure this restriction is obeyed.

7.3 Xmap code fragments

7.3.1 Importing and exporting Xmaps

Importing and exporting Xmaps to or from CCP4 map files couldn't be easier. To import a map, use:

```
clipper::CCP4MAPfile file;
file.open_read( "my.map" );
file.import_xmap( xmap );
file.close_read();
```

The extent and axis order of the input map do not matter, the resulting map will always cover the preferred ASU. However, if there is insufficient information in the input map to obtain an ASU, the remaining portion of the map will be untouched.

To export a map, use:

```
clipper::CCP4MAPfile file;
file.open_write( "my.map" );
file.export_xmap( xmap );
file.close_write();
```

7.3.2 Expanding a map to a lower symmetry

Sometimes it is necessary to expand a map to a lower symmetry. The ASU for the new spacegroup will be larger than the ASU for the old spacegroup, so the additional density values must be generated by applying the symmetry operators from the old spacegroup. This can be handled automatically by the map class.

The new map is constructed to share a grid and cell with the old map, but with the new spacegroup. We must ensure that every value in the new map is set, so we loop over the new map using a **clipper::Xmap_base::Map_reference_index** (p. 328). The we request the corresponding density from the old map, using the coordinate of the Map_reference_index:

```
  clipper::Xmap<float> oldmap, newmap;
...
  newmap.init( newspacegroup, oldmap.cell(), oldmap.grid_sampling() );
  clipper::Xmap_base::Map_reference_index ix;
  for ( ix = newmap.first(); !ix.last(); ix.next() ) {
    newmap[ ix ] = oldmap.get_data( ix.coord() );
  }
```

This works, but it would be faster to use a Map_reference_coord to access the second map, so that the symmetry operators do not need to be searched every time. The coordinate of the Map_reference_coord can be set from the Map_reference_index into the first map. i.e.:

```
  clipper::Xmap<float> oldmap, newmap;
...
  newmap.init( newspacegroup, oldmap.cell(), oldmap.grid_sampling() );
  clipper::Xmap_base::Map_reference_index ix;
  clipper::Xmap_base::Map_reference_coord iy(oldmap);
  for ( ix = newmap.first(); !ix.last(); ix.next() ) {
    iy.set_coord( ix.coord() );
    newmap[ ix ] = oldmap[ iy ];
  }
```

7.3.3 Looping over a small block of density

Sometimes it is necessary to access all of the map values in a cubic or oblong block of the unit cell. To perform this operation efficiently, coordinate-like map references must be used to eliminate the need to search over symmetry operators. Either of the following approaches may be used, depending on the situation:

Three map references can be used, one each for looping along the u, v, and w directions. Suppose the loop must run over the box bounded by Coord_grid g0 and Coord_grid g1. This leads to code of the following form:

```
  clipper::Xmap_base::Map_reference_coord i0, iu, iv, iw;
  i0 = clipper::Xmap_base::Map_reference_coord( xmap, g0 );
  for ( iu = i0; iu.coord().u() <= g1.u(); iu.next_u() )
    for ( iv = iu; iv.coord().v() <= g1.v(); iv.next_v() )
      for ( iw = iv; iw.coord().w() <= g1.w(); iw.next_w() ) {
        // ---- access xmap[iw] here ----
      }
```

Alternatively, in most cases it is only necessary to optimise the innermost loop.

```
  int u, v;
  clipper::Xmap_base::Map_reference_coord ix( xmap );
  for ( u = g0.u(); u <= g1.u(); u++ )
    for ( v = g0.v(); v <= g1.v(); v++ )
      for ( ix.set_coord(Coord_grid(u,v,g0.w())); ix.coord().w() <= g1.w(); ix.ne
      xt_w() ) {
        // ---- access xmap[ix] here ----
      }
```

Chapter 8

Developing using Atomic Models

8.1 Core model handling facilities

Requirements for the manipulation of atomic models in crystallographic calculations are very varied, and often the objects required must be tailored to the specific calculation. Therefore, Clipper provides only the minimum tools required for atomic models to interact with structure factors and maps. More advanced facilities are provided through optional packages.

The most common interaction between atomic models and structure factors or maps is through electron density or mask calculation. For this purpose, only a subset of atomic properties are required: element, coordinate, occupancy and temperature factor (U-value). Clipper therefore provides a simple class, **clipper::Atom** (p. 107), which contains this information. It has a template constructor which allows it to be initialised from any Atom-like object.

Electron density and mask calculations require a list of atoms, therefore a simple class for holding a list of atoms, **clipper::Atom_list** (p. 109) is provided. This is a trivial extension of std::vector<clipper::Atom>, and provides a template constructor to build an atom list from any vector-like object of atom-like objects. Atom lists can also be constructed using the standard vector methods, e.g. push_back(), insert().

Clipper methods requiring an atom list, e.g. a `structure factor calculation` may be given a Clipper::Atom_list, any class derived from that class, or a std::vector<clipper::Atom> as an argument.

8.2 Optional model handling packages

Two optional packages are provided for model manipulation. Both require the the 'MMDB' coordinate library by E. Krissinel to be installed (see `http://msd.ebi.ac.uk/~keb/cldoc/`).

8.2.1 MMDB interface.

The `MMDB interface` is a lightweight interface to allow MMDB to be used with Clipper. It consists of one main class: clipper::MMDBAtom_list. This is a trivial extension of **clipper::Atom_list** (p. 109) which may be used anywhere an Atom_list would be used. It provides an additional constructor which will build an atom list from an MMDB atom selection (i.e. PPCAtom + count).

Other classes are provided which may optionally be used to access information in the MMDB hierarchy in the form of Clipper objects. These are trivial extensions of MMDB objects. The MMDB objects may therefore be cast to the Clipper types to access the additional conversion methods.

8.2.2 MiniMol package.

The `MiniMol package` is a lightweight, STL based implementation of a small subset of the MMDB functionality. It provides simpler tools for manipulating single models and their constituents. The use of STL-style code means that no memory management is required, and all common operations such as assignment and passing variables have the most obvious effect.

The minimum necessary information is stored for each object in the hierarchy, however additional information of any type may be added to any object through use of the **clipper::PropertyManager** (p. 379).

Basic searching and selection tools are provided, and logical 'and' and 'or' operators may be used to combine models, polymers, and monomers.

MiniMol provides an MMDBfile class which may be used to extract models from and store models back into an MMDB, or a file.

8.3 Choosing a model package.

If you have very specific model requirements, you may be better off writing your own objects tailored to those requirements, and then deriving an Atom_list class to allow your objects to be communicated to Clipper. Otherwise, consider the following points:

- MMDB is a very powerful package, with an enormous range of facilities. However, it requires a good knowledge of C++, and the API can be quite complex.

- MiniMol is a very simple package with a concise and intuitive interface. It has a simple API accessible to non-C++ programmers.

MiniMol will do simple tasks much more easily than MMDB, but it won't do hard tasks at all. If MiniMol does everything you will need (or can easily be extended to do so), then it is probably a good choice, otherwise use MMDB.

Chapter 9

Installation

9.1 Installing Clipper and dependencies

On Linux systems you will need at least gcc 3.0.

Clipper has the following dependencies:

- The 'fftw' FFT library: http://www.fftw.org/
- (Alternatively, a kiss-fft overlay is available).

It may optionally be used with:

- The CCP4v6.1 and mmdb libraries: ftp://ftp.ccp4.ac.uk/ccp4/6.1.1/ccp4-6.1.1-core-src.tar.gz
- The 'cctbx' Crystallographic toolbox: http://cctbx.sourceforge.net/

The following instructions should download and install Clipper and all its dependencies on a properly configured Linux machine.

```
#!/bin/csh -f
wget ftp://ftp.ccp4.ac.uk/ccp4/6.1.1/ccp4-6.1.1-core-src.tar.gz
wget http://www.ysbl.york.ac.uk/~cowtan/clipper/clipper21.latest.tar.gz
#wget http://www.ysbl.york.ac.uk/~cowtan/clipper/clipper-kiss.latest.tar.gz
tar xvfz ccp4-6.1.1-core-src.tar.gz
setenv CCP4_MASTER      $PWD/
setenv CCP4             $CCP4_MASTER/ccp4-6.1.1/
setenv CCP4_SCR         /tmp/$USER
setenv BINSORT_SCR      $CCP4_SCR
setenv CCP4_OPEN        UNKNOWN
setenv CLIB             $CCP4/lib
setenv CLIBD            $CCP4/lib/data
setenv CLIBS            $CCP4/lib/src
setenv CINCL            $CCP4/include
setenv PUBLIC_FONT84    $CLIB/font84.dat
cd $CCP4
sed -i.orig 's#^\(.*passwd_struct.*\)$#//\1#' lib/src/library_utils.c
./configure --enable-fftw --disable-clipper --disable-ccif --disable-lapack --dis
      able-ssm --disable-pdb_extract --disable-cctbx --disable-diffractionImg --onlylib

gmake -k install

cd $CCP4_MASTER
mkdir clipper
cd clipper
tar xvfz ../clipper21.latest.tar.gz
#tar xvfz ../clipper-kiss.latest.tar.gz
setenv OPT -O1
./configure --ccp4dir=$CCP4
gmake examples
```

This file is available as a shell script from the Clipper homepage.

9.1.1 Problems

Likely problems may include:

- Failure to agree to CCP4 v6 license agreement. If you have not installed CCP4 v6 before, you should read the agreement at http://www.ccp4.ac.uk/ and then use the following before attempting the install:

```
echo $user `date` "V5.0" > ${HOME}/.agree2ccp4v6
```

- Non-Linux platforms: Change the CCP4 configure line to have the name of your platform, instead of 'linux', on the end. The fortran example will fail.
- 'wget' fails: Your firewall may block wget. In which case, fetch the tar files by hand and then run the rest of the script.
- If you haven't installed cctbx, the cctbx example will fail.

Chapter 10

Where to Look for help on specific tasks.

This document provides a thematic index to some of the Clipper documentation. It should provide a starting point to help you to find out how to tackle a particular task using Clipper.

It is divided into two main sections, the first indexing information be theme, and the second providing useful links for performing specific tasks.

10.1 Thematic index.

This section describes the documentation which might help when addressing a particular type of problem.

10.1.1 Documentation common to all problems.

Some information is common to all tasks in Clipper. Most fundamental are the coordinate types. There are described in the following essay: **Coordinates and related types** (p. 19). The most commonly used classes are **clipper::Coord_orth** (p. 182), **clipper::Coord_frac** (p. 172), **clipper::Coord_grid** (p. 175) and **clipper::HKL** (p. 261).

All crystallographic tasks depend upon the spacegroup and cell classes, so read their documentation. The cell class is **clipper::Cell** (p. 133), and the spacegroup class is **clipper::Spacegroup** (p. 413). You may want to look at their parent classes, Cell_descr, and **clipper::Spgr_descr** (p. 421), which are used to initialise the classes if the information is not coming from a file.

If you are going to need to perform operations on coordinates, then you may want to look at the **clipper::RTop_orth** (p. 407) documentation. Its counterpart, **clipper::RTop_frac** (p. 404) is the base class of the symmetry operator **clipper::Symop** (p. 428).

Often, the parent classes of a class also contain useful methods. For example, the coordinate classes are all derived from clipper::Vec3<T>, and the operators are derived from clipper::RTop<T>.

The **clipper::Util** (p. 456) class may contain some helpful utility functions. The **clipper::String** (p. 426) class is a trivial extension of the STL std::string. You can use either interchangeably.

10.1.2 Documentation for reflection-related problems.

First, look at the essay **Developing using Reflection Data** (p. 29), which describes how Clipper handles reciprocal space data.

The fundamental types which hold lists of HKL's and lists of associated reflection data (e.g. F's, phases, etc.) are **clipper::HKL_info** (p. 284) and clipper::HKL_data<T>.

clipper::HKL_data<T> is a template class which can be used to hold any kind of data. A range of common datatypes are provided which you can store in this object: Look at the derived classes of **clipper::Datatype_base** (p. 200) for a list. Alternatively, you can add your own.

Most common reflection data conversions have built in methods. Look at the **clipper::HKL_data<T>::compute()** (p. 268) method. For operators to use with it, look at the `class listing` for classes labelled clipper::datatypes::Compute_∗.

Scaling of data is also built in. See **clipper::ResolutionFn** (p. 392) and the examples on that page, and the derived classes of **clipper::BasisFn_base** (p. 119) and **clipper::TargetFn_base** (p. 434).

10.1.3 Documentation for map-related problems.

First, look at the essay **Developing using Crystallographic Maps** (p. 37), which describes how Clipper handles real space map data.

Crystallographic maps are handled by the clipper::Xmap<T> class. Calculation of FFT's to and from reciprocal space is build in to this class, as are interpolation methods. The **clipper::Map_-stats** (p. 332) class provides tools for calculating map statistics.

Calculation of maps and masks from atomic coordinates is handled in the `contrib` package by the EDcalc- and MASKcalc- classes. Filtering of maps and calculation of masks is aided by the MapFilter- classes in the `contrib` package.

Non-crystallographic maps (i.e. maps with no symmetry or cell repeat) are handled by the clipper::NXmap<T> class. Non-crystallographic maps may be referred into a crystallographic frame using the clipper::NXoperator class.

10.1.4 Documentation for model-related problems.

First, look at the essay **Developing using Atomic Models** (p. 45), which describes how Clipper handles atomic models.

The **clipper::Atom** (p. 107) and **clipper::Atom_list** (p. 109) classes provide basic support for atoms and lists of atoms, or you can use the more advanced `MMDB` or `MiniMol` packages.

To calculate structure factors, maps, or masks from a coordinate model, look at the SFcalc-, EDcalc-, and MASKcalc- classes in the `contrib` package.

10.2 Task index

This section describes the documentation which might help when addressing a particular crystallographic task.

10.2.1 Molecular Replacement.

Rotations are handled in a generic form as Quaternions by the **clipper::Rotation** (p. 398) class. These may be converted to and from Polar angles, Euler angles (24 conventions), and matrix representations. See the **clipper::Polar_ccp4** (p. 374), **clipper::Euler_ccp4** (p. 207), clipper::Euler<T>, and clipper::Mat33<T> classes.

Rotation-translation operators are handled by the derived classes of clipper::RTop<T>, i.e. **clipper::RTop_orth** (p. 407) and **clipper::RTop_frac** (p. 404).

10.2.2 Phase improvement (Density modification)

Filtering of maps and calculation of masks is aided by the MapFilter- classes in the `contrib` package.

Data scaling and sigmaa estimation can be performed using the **clipper::ResolutionFn** (p. 392) class and its related classes.

10.2.3 Refinement

Agarwal coefficients for calculation of refinement residual gradients may be generated using the **clipper::AtomShapeFn** (p. 111) class.

Data scaling and sigmaa estimation can be performed using the **clipper::ResolutionFn** (p. 392) class and its related classes.

10.2.4 Model building

FFFear-style translation functions may be calculated using the FFFear- classes in the `contrib` package.

Chapter 11

Deprecated List

Class clipper::AtomSF (p. **110**) This class has been replaced by AtomShapeFn, which is smaller, faster, and more capable. This class is now a wrapper for that class.

Member clipper::AtomShapeFn::rho_grad (p. **115**)**(const Coord_orth &xyz, std::vector< ftype > &grad) const** return Agarwal density gradients as a function of coordinate

Class clipper::datatypes::Compute_scale_u< T > (p. **165**) Apply scale and U to any scalable datatype

Class clipper::datatypes::D_sigD< dtype > (p. **198**) Anomalous difference data type: D + sigD

Class clipper::TargetFn_sigmaa< T > (p. **443**) simple sigma_a target function

Class clipper::TargetFn_sigmaa_omegaa< T > (p. **444**) simple sigma_a target function

Chapter 12

Directory Hierarchy

12.1 Directories

This directory hierarchy is sorted roughly, but not completely, alphabetically:

Chapter 13

Namespace Index

13.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 14

Class Index

14.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 15

Class Index

15.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 16

File Index

16.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 17

Directory Documentation

17.1 clipper/ Directory Reference

Directories

- directory **core**

17.2 clipper/core/ Directory Reference

Files

- file **atomsf.cpp**
- file **atomsf.h**
- file **cell.cpp**
- file **cell.h**
- file **clipper_instance.cpp**
- file **clipper_instance.h**
- file **clipper_memory.cpp**
- file **clipper_memory.h**
- file **clipper_message.cpp**
- file **clipper_message.h**
- file **clipper_precision.h**
- file **clipper_stats.cpp**
- file **clipper_stats.h**
- file **clipper_sysdep.h**
- file **clipper_test.cpp**
- file **clipper_test.h**
- file **clipper_thread.cpp**
- file **clipper_thread.h**
- file **clipper_types.cpp**
- file **clipper_types.h**
- file **clipper_util.cpp**
- file **clipper_util.h**
- file **container.cpp**
- file **container.h**
- file **container_hkl.cpp**
- file **container_hkl.h**
- file **container_map.cpp**
- file **container_map.h**
- file **container_types.cpp**
- file **container_types.h**
- file **coords.cpp**
- file **coords.h**
- file **derivs.cpp**
- file **derivs.h**
- file **fftmap.cpp**
- file **fftmap.h**
- file **fftmap_sparse.cpp**
- file **fftmap_sparse.h**
- file **hkl_compute.cpp**
- file **hkl_compute.h**
- file **hkl_data.cpp**
- file **hkl_data.h**
- file **hkl_datatypes.cpp**
- file **hkl_datatypes.h**
- file **hkl_info.cpp**
- file **hkl_info.h**

- file **hkl_lookup.cpp**
- file **hkl_lookup.h**
- file **hkl_operators.cpp**
- file **hkl_operators.h**
- file **map_interp.cpp**
- file **map_interp.h**
- file **map_utils.cpp**
- file **map_utils.h**
- file **nxmap.cpp**
- file **nxmap.h**
- file **nxmap_operator.cpp**
- file **nxmap_operator.h**
- file **ramachandran.cpp**
- file **ramachandran.h**
- file **resol_basisfn.cpp**
- file **resol_basisfn.h**
- file **resol_fn.cpp**
- file **resol_fn.h**
- file **resol_targetfn.cpp**
- file **resol_targetfn.h**
- file **rotation.cpp**
- file **rotation.h**
- file **spacegroup.cpp**
- file **spacegroup.h**
- file **spacegroup_data.cpp**
- file **spacegroup_data.h**
- file **symop.cpp**
- file **symop.h**
- file **test_core.cpp**
- file **test_core.h**
- file **test_data.cpp**
- file **test_data.h**
- file **xmap.cpp**
- file **xmap.h**

Chapter 18

Namespace Documentation

18.1 clipper Namespace Reference

Namespaces

- namespace **datatypes**
- namespace **data32**
- namespace **data64**

Classes

- class **AtomShapeFn**

  Atomic shape function object.

- class **AtomSF**

  Atomic scattering factor object.

- class **Metric_tensor**

  Metric tensor.

- class **Cell_descr**

  cell description (automatically converts to radians)

- class **Cell**

  Cell *(*p. *133) object.*

- class **ClipperInstance**
- class **ClipperInstantiator**
- class **Property_base**

  Base class for properties of arbitrary types.

- class **Property**

  Template for a property holding an arbitrary type.

- class **PropertyManager**

Class for holding a list of labelled properties of arbitrary types.

- class **ObjectCache**

  Object Cache manager.

- class **Message**

  ***Message** (p. 340) handler class.*

- class **Message_base**

  Base type for messages.

- class **Message_generic**

  Generic message.

- class **Message_fatal**

  Fatal message (level = 9).

- class **Message_warn**

  Warning message (level = 5).

- class **Message_info**

  Info message (level = 1).

- class **Message_ctor**

  Constructor message (level = 2).

- class **Message_dtor**

  Destructor message (level = 2).

- class **Range**

  ***Range** (p. 384) - upper and lower bounds of some type.*

- class **Range_sampling**

  ***Range** (p. 384) sampling: discrete sampling of a real range.*

- class **Histogram**

  General histogram class.

- class **Generic_ordinal**

  Generic ordinal gernerator.

- class **Test_base**

  *Base class for **clipper** (p. 81) self-test classes.*

- class **Mutex**

  ***Mutex** (p. 351) class: used for locking and unlocking shared resources.*

- class **Thread_base**

  Thread base class: Override this to create new threads.

- class **String**

  String *(p. 426) extension with simple parsing methods.*

- class **Vec3**

  3-vector class

- class **Mat33**

  3x3-matrix class

- class **Mat33sym**

  Compressed form for 3x3 symmetric matrix class.

- class **RTop**

  Rotation-translation operator.

- class **Array2d**

  Simple 2-d array class.

- class **Matrix**

  General matrix class: like ***Array2d*** *(p. 105) but with numerical methods.*

- class **Util**

  Utility class.

- class **Container**

  Definition for a generic container Object.

- class **CHKL_info**

  HKL *(p. 261) list and indexing object container.*

- class **CHKL_data**

  Reflection data list container.

- class **CXmap**

  Crystallographic map container.

- class **CNXmap**

  Non-Crystallographic map container.

- class **CNXmap_operator**

  Non-Crystallographic map operator container.

- class **CResolution**

  Resolution *(p. 389) container.*

- class **CSpacegroup**

  Spacegroup *(p. 413) container.*

- class **CCell**

  CCell *(p. 132) container.*

- class **CGrid_sampling**
  CGrid_sampling *(p. 140) container.*
- class **Resolution**
  Resolution *(p. 389) in angstroms.*
- class **HKL_class**
  reflection class
- class **RTop_orth**
  Orthogonal operator class.
- class **HKL**
  reflection 'Miller' index
- class **Coord_reci_orth**
  orthogonal reciprocal coordinate (length of which is invresolsq)
- class **Coord_reci_frac**
  fractional reciprocal coordinate (i.e. non-integer hkl)
- class **Coord_grid**
  Grid *(p. 250) coordinate.*
- class **Coord_orth**
  orthogonal (Angstrom) coordinates
- class **Coord_frac**
  fractional (cell) coordinates
- class **Coord_map**
  map coordinate: this is like ***Coord_grid*** *(p. 175), but non-integer*
- class **U_aniso_orth**
  Anisotropic orthogonal atomic displacement parameters.
- class **U_aniso_frac**
  Anisotropic fractional atomic displacement parameters.
- class **Grid**
  generic grid
- class **Grid_sampling**
  Grid *(p. 250) sampling of a unit cell.*
- class **HKL_sampling**
  HKL *(p. 261) sampling of reciprocal space.*
- class **Grid_range**

Grid *(*p. *250) range class: defines array limits for a grid.*

- class **Atom**

  Atom *(*p. *107) class.*

- class **Atom_list**

  Atom *(*p. *107) list class.*

- class **Grad_orth**

  orthogonal (Angstom) gradient, with respect to orthogonal x,y,z

- class **Grad_frac**

  fractional (cell) gradient, with respect to fractional u,v,w

- class **Grad_map**

  map coordinate gradient, with respect to grid u,v,w

- class **Curv_orth**

  orthogonal (Angstom) curvatures, with respect to orthogonal x,y,z

- class **Curv_frac**

  fractional (cell) curvatures, with respect to fractional u,v,w

- class **Curv_map**

  map coordinate curvatures, with respect to grid u,v,w

- class **FFTmap_base**
- class **FFTmap_p1**

  FFTmap_p1 *(*p. *222): low level P1 map used for calculating FFTs.*

- class **FFTmap**

  FFTmap *(*p. *215): P1 map with symmetry used for calculating FFTs.*

- class **FFTmap_sparse_p1_base**

  base type for sparse P1 fft maps

- class **FFTmap_sparse_p1_hx**

  FFTmap_sparse_p1_hx *(*p. *230): low level sparse P1 map used for calculating FFTs.*

- class **FFTmap_sparse_p1_xh**

  FFTmap_sparse_p1_xh *(*p. *234): low level sparse P1 map used for calculating FFTs.*

- class **HKL_data_cacheobj**
- class **Datatype_base**

  Reflection data type objects.

- class **HKL_data_base**

  HKL_data_base *(*p. *277).*

- class **HKL_data**

HKL_data<>.

- class **HKL_info**

  HKL *(*p. *261) list container and tree root.*

- class **HKL_lookup**

  Fast reflection lookup object.

- class **LogPhaseProb**

  Log phase probability distribution object.

- class **Interp_nearest**

  Wrapper class for zeroth-order (nearest neighbour) interpolation fns.

- class **Interp_linear**

  Wrapper class for first-order (linear) interpolation fns.

- class **Interp_cubic**

  Wrapper class for third-order (cubic) interpolation fns.

- class **Map_stats**

  Generic map statistics class.

- class **Map_index_sort**

  Generic map sorting class.

- class **NXmap_base**

  NXmap_base *(*p. *363): base for non-crystallographic map class.*

- class **NXmap**

  NXmap<T>: actual non-crystallographic map class.

- class **NX_operator**

  NX_operator *(*p. *352): non-crystal map operator.*

- class **NXmap_operator**

  NXmap_operator *(*p. *368): non-crystal map operator referencing a particular* ***NXmap*** *(*p. *357).*

- class **Prob_phi_2d**

  2-d angular probability distibution class

- class **Ramachandran**

  Ramachandran *(*p. *382) plot class.*

- class **Resolution_ordinal**

  Resolution *(*p. *389) ordinal gernerator.*

- class **BasisFn_binner**

  simple binning basis function

- class **BasisFn_linear**
  simple linear basis function
- class **BasisFn_spline**
  simple smooth basis function
- class **BasisFn_gaussian**
  simple Gaussian basis function
- class **BasisFn_aniso_gaussian**
  simple anisotropic Gaussian basis function
- class **BasisFn_log_gaussian**
  simple log Gaussian basis function
- class **BasisFn_log_aniso_gaussian**
  simple anisotropic Gaussian basis function
- class **BasisFn_expcubic**
  simple Expcubic basis function
- class **BasisFn_base**
  abstract base class for resolution function basis functions
- class **TargetFn_base**
  abstract base class for least-squares resolution function target functions
- class **ResolutionFn**
  2nd order resolution function evaluator
- class **ResolutionFn_nonlinear**
  2nd order resolution function evaluator
- class **TargetFn_meanFnth**
  simple mean $|F|^n$ *target*
- class **TargetFn_meanEnth**
  simple mean $|E|^n$ *target*
- class **TargetFn_scaleF1F2**
  $|F|^2$ *scaling target*
- class **TargetFn_scaleLogF1F2**
  log $|F|^2$ *scaling target*
- class **TargetFn_scaleI1I2**
- class **TargetFn_scaleLogI1I2**
  log $|I|$ *scaling target*

- class **TargetFn_scaleEsq**

  $|E|^2$ scaling target

- class **TargetFn_sigmaa_omegaa**
- class **TargetFn_sigmaa**
- class **Euler**

  Euler *(p. 206) angle class.*

- class **Euler_ccp4**

  Euler_ccp4 *(p. 207) angle class.*

- class **Polar_ccp4**

  Polar_ccp4 *(p. 374) angle class.*

- class **Rotation**

  Rotation *(p. 398) class.*

- struct **Compare_grid**
- class **Spgr_descr**

  spacegroup description

- class **Spgr_cacheobj**
- class **Spacegroup**

  Spacegroup *(p. 413) object.*

- class **RTop_frac**

  Fractional operator class.

- class **Symop**

  Crystallographic symmetry operator.

- class **Isymop**

  Integerised symmetry matrix.

- class **Symop_code**

  Compressed encoded symmetry operator.

- class **Test_core**
- class **Xmap_cacheobj**
- class **Xmap_base**

  Xmap_base *(p. 472): base for crystallographic map class.*

- class **Xmap**

  Xmap<T>: actual crystallographic map class.

Typedefs

- typedef float **ftype32**
- typedef double **ftype64**
- typedef ftype64 **ftype**

  ftype definition for floating point representation

- typedef ftype64 **xtype**

  xtype definition for import/export of data

- typedef int **itype32**
- typedef unsigned int **uitype32**
- typedef long long **itype64**
- typedef unsigned long long **uitype64**
- typedef **Grid_range Grid_map**

  Obsolete form for ***Grid_range*** *(p. 252).*

- typedef float **ffttype**

Functions

- template<class T >
  int **operator==** (const **Vec3**< T > &v1, const **Vec3**< T > &v2)
- template<class T >
  int **operator!=** (const **Vec3**< T > &v1, const **Vec3**< T > &v2)
- template<class T >
  Vec3< T > **operator-** (const **Vec3**< T > &v)
- template<class T >
  Vec3< T > **operator+** (const **Vec3**< T > &v1, const **Vec3**< T > &v2)
- template<class T >
  Vec3< T > **operator-** (const **Vec3**< T > &v1, const **Vec3**< T > &v2)
- template<class T >
  Vec3< T > **operator*** (const T &s, const **Vec3**< T > &v1)
- template<class T >
  Vec3< T > **operator*** (const **Vec3**< T > &v1, const T &s)
- template<class T >
  T **operator*** (const **Vec3**< T > &v1, const **Vec3**< T > &v2)
- template<class T >
  Vec3< T > **operator*** (const **Mat33**< T > &m, const **Vec3**< T > &v)
- template<class T >
  Vec3< T > **operator*** (const **Vec3**< T > &v, const **Mat33**< T > &m)
- template<class T >
  Mat33< T > **operator*** (const **Mat33**< T > &m1, const **Mat33**< T > &m2)
- template<class T >
  Mat33< T > **operator+** (const **Mat33**< T > &m1, const **Mat33**< T > &m2)
- template<class T >
  Mat33< T > **operator-** (const **Mat33**< T > &m)
- template<class T >
  Vec3< T > **operator*** (const **Mat33sym**< T > &m, const **Vec3**< T > &v)
- template<class T >
  Mat33sym< T > **operator+** (const **Mat33sym**< T > &m1, const **Mat33sym**< T > &m2)

- template< class T >
  Mat33sym< T > **operator-** (const **Mat33sym**< T > &m)
- template< class T >
  Vec3< T > **operator*** (const **RTop**< T > &r, const **Vec3**< T > &v)
- template< class T >
  RTop< T > **operator*** (const **RTop**< T > &r1, const **RTop**< T > &r2)
- template< class T >
  std::vector< T > **operator*** (const **Matrix**< T > &m, const std::vector< T > &v)
- **Rotation operator*** (const **Rotation** &r1, const **Rotation** &r2)
- bool **reci_asu** (const **Spgr_descr::Symop_codes** &ops, data::ASUfn asufn)
- **Vec3 real_asu** (const **Spgr_descr::Symop_codes** &ops)

Variables

- **Message message**
- **Message_fatal message_parent_of_root** ("Container: attempt to access parent of root")
- **Message_fatal message_child_out_of_range** ("Container: child index out of range")
- **Message_fatal message_duplicate_path** ("Container: attempt to create duplicate path")
- const **HKL_info NullHKL_info**
- const **Resolution NullResolution**
- const **Spacegroup NullSpacegroup**
- const **Cell NullCell**
- const **Grid_sampling NullGrid_sampling**
- **Message_fatal message_fftmap_get_real_space_error** ("FFTmap: get_real_-data in reciprocal space")
- **Message_fatal message_fftmap_set_real_space_error** ("FFTmap: set_real_data in reciprocal space")
- **Message_fatal message_fftmap_get_reci_space_error** ("FFTmap: get_recip_-data in real space")
- **Message_fatal message_fftmap_set_reci_space_error** ("FFTmap: set_recip_-data in real space")
- **Message_ctor message_ctor_fftmap** (" [FFTmap: constructed]")
- **Message_ctor message_ctor_hkl_data** (" [HKL_data: constructed]")
- **Message_fatal message_recip_asu_error** ("HKL_info: find_sym reciprocal space ASU error")
- **Message_ctor message_ctor_hkl_info** (" [HKL_info: constructed]")
- **Message_ctor message_ctor_nxmap** (" [NXmap: constructed]")
- **Message_ctor message_ctor_xmap** (" [Xmap: constructed]")

18.1.1 Detailed Description

All normal **clipper** (p. 81) objects except data objects are defined in this namespace.

18.1.2 Typedef Documentation

18.1.2.1 typedef ftype64 clipper::ftype

ftype definition for floating point representation This type is used for all floating point values except data. The whole package could be templatised on ftype. However for now it is typedef'ed

18.1.3 Function Documentation

18.1.3.1 Rotation clipper::operator* (const Rotation & *r1*, const Rotation & *r2*)

Note: This multiplication operator combines rotations in left-to-right order, rather than the right-to-left order used for matrices. Thus, the rotation which arises from applying rotation r1 followed by rotation r2 is given by r1*r2. Similarly, the rotation which arises from applying rotation r1 followed by rotation r2 and rotation r3 is given by (r1*r2)*r3.

18.2 clipper::data32 Namespace Reference

Typedefs

- typedef **datatypes::Compute_phifom_from_abcd**< ftype32 > **Compute_-phifom_from_abcd**

  operator

- typedef **datatypes::Compute_abcd_from_phifom**< ftype32 > **Compute_abcd_-from_phifom**

  operator

- typedef **datatypes::Compute_fphi_from_fsigf_phifom**< ftype32 > **Compute_-fphi_from_fsigf_phifom**

  operator

- typedef **datatypes::Compute_EsigE_from_FsigF**< ftype32 > **Compute_EsigE_-from_FsigF**

  operator

- typedef **datatypes::Compute_mean_fsigf_from_fsigfano**< ftype32 > **Compute_-mean_fsigf_from_fsigfano**

  operator

- typedef **datatypes::Compute_diff_fsigf_from_fsigfano**< ftype32 > **Compute_-diff_fsigf_from_fsigfano**

  operator

- typedef **datatypes::Compute_neg_fphi**< ftype32 > **Compute_neg_fphi**

  operator

- typedef **datatypes::Compute_add_fphi**< ftype32 > **Compute_add_fphi**

  operator

- typedef **datatypes::Compute_sub_fphi**< ftype32 > **Compute_sub_fphi**

  operator

- typedef **datatypes::Compute_add_abcd**< ftype32 > **Compute_add_abcd**

  operator

- typedef **datatypes::Compute_scale_u_iso**< **datatypes::I_sigI**< ftype32 > > **Compute_scale_u_iso_isigi**

  operator

- typedef **datatypes::Compute_scale_u_iso**< **datatypes::F_sigF**< ftype32 > > **Compute_scale_u_iso_fsigf**

  operator

- typedef **datatypes::Compute_scale_u_iso**< **datatypes::F_sigF_ano**< ftype32 > > **Compute_scale_u_iso_fsigfano** *operator*

- typedef **datatypes::Compute_scale_u_iso**< **datatypes::F_phi**< ftype32 > > **Compute_scale_u_iso_fphi**

  operator

- typedef **datatypes::Compute_scale_u_aniso**< **datatypes::I_sigI**< ftype32 > > **Compute_scale_u_aniso_isigi**

  operator

- typedef **datatypes::Compute_scale_u_aniso**< **datatypes::F_sigF**< ftype32 > > **Compute_scale_u_aniso_fsigf**

  operator

- typedef **datatypes::Compute_scale_u_aniso**< **datatypes::F_sigF_ano**< ftype32 > > **Compute_scale_u_aniso_fsigfano**

  operator

- typedef **datatypes::Compute_scale_u_aniso**< **datatypes::F_phi**< ftype32 > > **Compute_scale_u_aniso_fphi**

  operator

- typedef **datatypes::Compute_FsigF**< ftype32, **datatypes::F_sigF**< ftype32 > > **Compute_FsigF**

  operator

- typedef **datatypes::Compute_FsigF**< ftype32, **datatypes::F_sigF_ano**< ftype32 > > **Compute_FsigF_from_ano**

  operator

- typedef **datatypes::Compute_scale_u**< **datatypes::I_sigI**< ftype32 > > **Compute_scale_u_isigi**
- typedef **datatypes::Compute_scale_u**< **datatypes::F_sigF**< ftype32 > > **Compute_scale_u_fsigf**
- typedef **datatypes::Compute_scale_u**< **datatypes::F_sigF_ano**< ftype32 > > **Compute_scale_u_fsigfano**
- typedef **datatypes::Compute_scale_u**< **datatypes::F_phi**< ftype32 > > **Compute_scale_u_fphi**
- typedef **clipper::datatypes::I_sigI**< ftype32 > **I_sigI**

  datatype

- typedef **clipper::datatypes::I_sigI_ano**< ftype32 > **I_sigI_ano**

  datatype

- typedef **clipper::datatypes::F_sigF**< ftype32 > **F_sigF**

  datatype

- typedef **clipper::datatypes::F_sigF_ano**< ftype32 > **F_sigF_ano**

  datatype

- typedef **clipper::datatypes::E_sigE**< ftype32 > **E_sigE** *datatype*

- typedef **clipper::datatypes::F_phi**< ftype32 > **F_phi**
  datatype
- typedef **clipper::datatypes::Phi_fom**< ftype32 > **Phi_fom**
  datatype
- typedef **clipper::datatypes::ABCD**< ftype32 > **ABCD**
  datatype
- typedef **clipper::datatypes::Flag Flag**
  datatype
- typedef **clipper::datatypes::Flag_bool Flag_bool**
  datatype
- typedef **clipper::datatypes::D_sigD**< ftype32 > **D_sigD**
  datatype

18.2.1 Detailed Description

This namespace contains instantiations of template classes for storage and manipulation of float data types.

18.3 clipper::data64 Namespace Reference

Typedefs

- typedef **datatypes::Compute_phifom_from_abcd**< ftype64 > **Compute_-phifom_from_abcd**

  operator

- typedef **datatypes::Compute_abcd_from_phifom**< ftype64 > **Compute_abcd_-from_phifom**

  operator

- typedef **datatypes::Compute_fphi_from_fsigf_phifom**< ftype64 > **Compute_-fphi_from_fsigf_phifom**

  operator

- typedef **datatypes::Compute_EsigE_from_FsigF**< ftype64 > **Compute_EsigE_-from_FsigF**

  operator

- typedef **datatypes::Compute_mean_fsigf_from_fsigfano**< ftype64 > **Compute_-mean_fsigf_from_fsigfano**

  operator

- typedef **datatypes::Compute_diff_fsigf_from_fsigfano**< ftype64 > **Compute_-diff_fsigf_from_fsigfano**

  operator

- typedef **datatypes::Compute_neg_fphi**< ftype64 > **Compute_neg_fphi**

  operator

- typedef **datatypes::Compute_add_fphi**< ftype64 > **Compute_add_fphi**

  operator

- typedef **datatypes::Compute_sub_fphi**< ftype64 > **Compute_sub_fphi**

  operator

- typedef **datatypes::Compute_add_abcd**< ftype64 > **Compute_add_abcd**

  operator

- typedef **datatypes::Compute_scale_u_iso**< **datatypes::I_sigI**< ftype64 > > **Compute_scale_u_iso_isigi**

  operator

- typedef **datatypes::Compute_scale_u_iso**< **datatypes::F_sigF**< ftype64 > > **Compute_scale_u_iso_fsigf**

  operator

- typedef **datatypes::Compute_scale_u_iso**< **datatypes::F_sigF_ano**< ftype64 > > **Compute_scale_u_iso_fsigfano** *operator*

- typedef **datatypes::Compute_scale_u_iso**< **datatypes::F_phi**< ftype64 > > **Compute_scale_u_iso_fphi**

  operator

- typedef **datatypes::Compute_scale_u_aniso**< **datatypes::I_sigI**< ftype64 > > **Compute_scale_u_aniso_isigi**

  operator

- typedef **datatypes::Compute_scale_u_aniso**< **datatypes::F_sigF**< ftype64 > > **Compute_scale_u_aniso_fsigf**

  operator

- typedef **datatypes::Compute_scale_u_aniso**< **datatypes::F_sigF_ano**< ftype64 > > **Compute_scale_u_aniso_fsigfano**

  operator

- typedef **datatypes::Compute_scale_u_aniso**< **datatypes::F_phi**< ftype64 > > **Compute_scale_u_aniso_fphi**

  operator

- typedef **datatypes::Compute_FsigF**< ftype64, **datatypes::F_sigF**< ftype64 > > **Compute_FsigF**

  operator

- typedef **datatypes::Compute_FsigF**< ftype64, **datatypes::F_sigF_ano**< ftype64 > > **Compute_FsigF_from_ano**

  operator

- typedef **datatypes::Compute_scale_u**< **datatypes::I_sigI**< ftype64 > > **Compute_scale_u_isigi**
- typedef **datatypes::Compute_scale_u**< **datatypes::F_sigF**< ftype64 > > **Compute_scale_u_fsigf**
- typedef **datatypes::Compute_scale_u**< **datatypes::F_sigF_ano**< ftype64 > > **Compute_scale_u_fsigfano**
- typedef **datatypes::Compute_scale_u**< **datatypes::F_phi**< ftype64 > > **Compute_scale_u_fphi**
- typedef **clipper::datatypes::I_sigI**< ftype64 > **I_sigI**

  datatype

- typedef **clipper::datatypes::I_sigI_ano**< ftype64 > **I_sigI_ano**

  datatype

- typedef **clipper::datatypes::F_sigF**< ftype64 > **F_sigF**

  datatype

- typedef **clipper::datatypes::F_sigF_ano**< ftype64 > **F_sigF_ano**

  datatype

- typedef **clipper::datatypes::E_sigE**< ftype64 > **E_sigE** *datatype*

- typedef **clipper::datatypes::F_phi**< ftype64 > **F_phi**
  datatype
- typedef **clipper::datatypes::Phi_fom**< ftype64 > **Phi_fom**
  datatype
- typedef **clipper::datatypes::ABCD**< ftype64 > **ABCD**
  datatype
- typedef **clipper::datatypes::Flag Flag**
  datatype
- typedef **clipper::datatypes::Flag_bool Flag_bool**
  datatype
- typedef **clipper::datatypes::D_sigD**< ftype64 > **D_sigD**
  datatype

18.3.1 Detailed Description

This namespace contains instantiations of template classes for storage and manipulation of double data types.

18.4 clipper::datatypes Namespace Reference

Classes

- class **Compute_phifom_from_abcd**

  *Compute from **ABCD** (p. 103) to **Phi_fom** (p. 372) by phase integration (loses bimodality).*

- class **Compute_abcd_from_phifom**

  *Compute from **Phi_fom** (p. 372) to **ABCD** (p. 103) (C = D = 0).*

- class **Compute_fphi_from_fsigf_phifom**

  *Compute from F_sigF+Phi_fom to **F_phi** (p. 208).*

- class **Compute_EsigE_from_FsigF**

  *Compute from **F_sigF** (p. 210) to **E_sigE** (p. 202).*

- class **Compute_mean_fsigf_from_fsigfano**

  *Compute from F_sigF_anom to **F_sigF** (p. 210) (mean structure factor).*

- class **Compute_diff_fsigf_from_fsigfano**

  *Compute from F_sigF_anom to **F_sigF** (p. 210) (difference structure factor).*

- class **Compute_neg_fphi**

  *Negate **F_phi** (p. 208) (i.e. advance phase by pi).*

- class **Compute_add_fphi**

  *Add two **F_phi** (p. 208) datalists.*

- class **Compute_sub_fphi**

  *Subtract two **F_phi** (p. 208) datalists.*

- class **Compute_add_abcd**

  *Add two **ABCD** (p. 103) datalists.*

- class **Compute_scale_u**
- class **Compute_scale_u_iso**

  Apply scale and U to any scalable datatype.

- class **Compute_scale_u_aniso**

  Apply scale and U to any scalable datatype.

- class **Compute_FsigF**

  *Compute from **F_sigF** (p. 210) to **F_sigF** (p. 210).*

- class **I_sigI**

  Reflection data type: I + sigI.

- class **I_sigI_ano**

  Reflection data type: I(+) I(+) sigI(+) sigI(-) cov+-.

- class **F_sigF**
  Reflection data type: F + sigF.
- class **F_sigF_ano**
  Reflection data type: F(+) F(+) sigF(+) sigF(-) cov+-.
- class **E_sigE**
  Reflection data type: E + sigE.
- class **E_sigE_ano**
  Reflection data type: E(+) E(+) sigE(+) sigE(-) cov+-.
- class **F_phi**
  Reflection data type: F + phi model or map coeff (e.g. Fcalc, Fbest).
- class **Phi_fom**
  Reflection data type: best phi + fom.
- class **ABCD**
  Reflection data type: Hendrickson-Lattman coeff.
- class **Flag**
  Reflection data type: Free-R flag.
- class **Flag_bool**
  Reflection data type: boolean (false = missing).
- class **D_sigD**

Functions

- **HKL_data< Flag_bool > operator&** (const **HKL_data_base** &d1, const **HKL_data_base** &d2)
- **HKL_data< Flag_bool > operator|** (const **HKL_data_base** &d1, const **HKL_data_base** &d2)
- **HKL_data< Flag_bool > operator^** (const **HKL_data_base** &d1, const **HKL_data_base** &d2)
- **HKL_data< Flag_bool > operator!** (const **HKL_data_base** &d1)
- **HKL_data< Flag_bool > operator==** (const **HKL_data< Flag >** &d1, const int &n)
- **HKL_data< Flag_bool > operator!=** (const **HKL_data< Flag >** &d1, const int &n)
- **HKL_data< Flag_bool > operator>=** (const **HKL_data< Flag >** &d1, const int &n)
- **HKL_data< Flag_bool > operator<=** (const **HKL_data< Flag >** &d1, const int &n)
- **HKL_data< Flag_bool > operator>** (const **HKL_data< Flag >** &d1, const int &n)
- **HKL_data< Flag_bool > operator<** (const **HKL_data< Flag >** &d1, const int &n)
- template<class dtype >
  F_phi< dtype > **operator+** (const **F_phi**< dtype > &d1, const **F_phi**< dtype > &d2)

- template<class dtype >
  F_phi< dtype > **operator-** (const **F_phi**< dtype > &d1, const **F_phi**< dtype > &d2)
- template<class dtype >
  F_phi< dtype > **operator-** (const **F_phi**< dtype > &d1)
- template<class dtype >
  ABCD< dtype > **operator+** (const **ABCD**< dtype > &d1, const **ABCD**< dtype > &d2)
- template<class dtype >
  HKL_data< **F_phi**< dtype > > **operator+** (const **HKL_data**< **F_phi**< dtype > > &d1, const **HKL_data**< **F_phi**< dtype > > &d2)
- template<class dtype >
  HKL_data< **F_phi**< dtype > > **operator-** (const **HKL_data**< **F_phi**< dtype > > &d1, const **HKL_data**< **F_phi**< dtype > > &d2)
- template<class dtype >
  HKL_data< **F_phi**< dtype > > **operator*** (const **HKL_data**< **F_phi**< dtype > > &d1, const **ftype** &s)
- template<class dtype >
  HKL_data< **F_phi**< dtype > > **operator-** (const **HKL_data**< **F_phi**< dtype > > &d1)
- template<class dtype >
  HKL_data< **ABCD**< dtype > > **operator+** (const **HKL_data**< **ABCD**< dtype > > &d1, const **HKL_data**< **ABCD**< dtype > > &d2)
- template **F_phi**< ftype32 > **operator+** (const **F_phi**< ftype32 > &d1, const **F_phi**< ftype32 > &d2)
- template **F_phi**< ftype32 > **operator-** (const **F_phi**< ftype32 > &d1, const **F_phi**< ftype32 > &d2)
- template **F_phi**< ftype32 > **operator-** (const **F_phi**< ftype32 > &d1)
- template **ABCD**< ftype32 > **operator+** (const **ABCD**< ftype32 > &d1, const **ABCD**< ftype32 > &d2)
- template **F_phi**< ftype64 > **operator+** (const **F_phi**< ftype64 > &d1, const **F_phi**< ftype64 > &d2)
- template **F_phi**< ftype64 > **operator-** (const **F_phi**< ftype64 > &d1, const **F_phi**< ftype64 > &d2)
- template **F_phi**< ftype64 > **operator-** (const **F_phi**< ftype64 > &d1)
- template **ABCD**< ftype64 > **operator+** (const **ABCD**< ftype64 > &d1, const **ABCD**< ftype64 > &d2)
- template **HKL_data**< **F_phi**< ftype32 > > **operator+** (const **HKL_data**< **F_phi**< ftype32 > > &d1, const **HKL_data**< **F_phi**< ftype32 > > &d2)
- template **HKL_data**< **F_phi**< ftype32 > > **operator-** (const **HKL_data**< **F_phi**< ftype32 > > &d1, const **HKL_data**< **F_phi**< ftype32 > > &d2)
- template **HKL_data**< **F_phi**< ftype32 > > **operator*** (const **HKL_data**< **F_phi**< ftype32 > > &d1, const **ftype** &s)
- template **HKL_data**< **F_phi**< ftype32 > > **operator*** (const **ftype** &s, const **HKL_data**< **F_phi**< ftype32 > > &d1)
- template **HKL_data**< **F_phi**< ftype32 > > **operator-** (const **HKL_data**< **F_phi**< ftype32 > > &d1)
- template **HKL_data**< **ABCD**< ftype32 > > **operator+** (const **HKL_data**< **ABCD**< ftype32 > > &d1, const **HKL_data**< **ABCD**< ftype32 > > &d2)
- template **HKL_data**< **F_phi**< ftype64 > > **operator+** (const **HKL_data**< **F_phi**< ftype64 > > &d1, const **HKL_data**< **F_phi**< ftype64 > > &d2)
- template **HKL_data**< **F_phi**< ftype64 > > **operator-** (const **HKL_data**< **F_phi**< ftype64 > > &d1, const **HKL_data**< **F_phi**< ftype64 > > &d2)

- template **HKL_data**< **F_phi**< ftype64 > > **operator*** (const **HKL_data**< **F_phi**< ftype64 > > &d1, const **ftype** &s)
- template **HKL_data**< **F_phi**< ftype64 > > **operator*** (const **ftype** &s, const **HKL_data**< **F_phi**< ftype64 > > &d1)
- template **HKL_data**< **F_phi**< ftype64 > > **operator-** (const **HKL_data**< **F_phi**< ftype64 > > &d1)
- template **HKL_data**< **ABCD**< ftype64 > > **operator+** (const **HKL_data**< **ABCD**< ftype64 > > &d1, const **HKL_data**< **ABCD**< ftype64 > > &d2)
- template<class dtype >
  HKL_data< **F_phi**< dtype > > **operator*** (const **ftype** &s, const **HKL_data**< **F_phi**< dtype > > &d1)

18.4.1 Detailed Description

This namespace is used for templates which will later be instantiated for float or double precision data. You would not normally reference a class from this namespace directly.

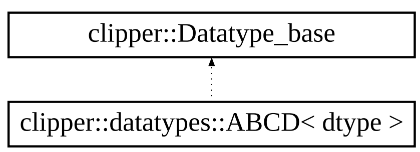
Chapter 19

Class Documentation

19.1 clipper::datatypes::ABCD< dtype > Class Template Reference

Reflection data type: Hendrickson-Lattman coeff.

`#include` <`hkl_datatypes.h`>Inheritance diagram for clipper::datatypes::ABCD< dtype >::



Public Member Functions

- **ABCD** (const dtype &a, const dtype &b, const dtype &c, const dtype &d)
- void **set_null** ()

  initialise to 'missing' (all elements are set to null)

- void **friedel** ()

  apply Friedel transformation to the data

- void **shift_phase** (const **ftype** &dphi)

  apply phase shift to the data

- bool **missing** () const

  return true if data is missing

- void **data_export** (**xtype** array[ ]) const

  conversion to array (for I/O)

- void **data_import** (const **xtype** array[ ])

  conversion from array (for I/O)

- const dtype & **a** () const
- const dtype & **b** () const
- const dtype & **c** () const
- const dtype & **d** () const
- dtype & **a** ()
- dtype & **b** ()
- dtype & **c** ()
- dtype & **d** ()

Static Public Member Functions

- static **String type** ()

  return the name of this data type

- static int **data_size** ()

  return number of data elements in this type

- static **String data_names** ()

  return names of data elements in this type

19.1.1 Detailed Description

template<class dtype> class clipper::datatypes::ABCD< dtype >

Reflection data type: Hendrickson-Lattman coeff.

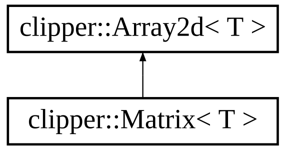
The documentation for this class was generated from the following file:

- hkl_datatypes.h

19.2 clipper::Array2d< T > Class Template Reference

Simple 2-d array class.

#include <clipper_types.h>Inheritance diagram for clipper::Array2d< T >::



Public Member Functions

- **Array2d** ()
  null constructor
- **Array2d** (const int &d1, const int &d2)
  constructor
- **Array2d** (const int &d1, const int &d2, T val)
  constructor
- void **resize** (const int &d1, const int &d2)
  resize
- void **resize** (const int &d1, const int &d2, const T &val)
  resize
- int **size** () const
  size
- const int & **rows** () const
  number of rows
- const int & **cols** () const
  number of cols
- const T & **operator()** (const int &i1, const int &i2) const
  read accessor
- T & **operator()** (const int &i1, const int &i2)
  write accessor

Protected Attributes

- std::vector< T > **data**
- int **d1_**
- int **d2_**

19.2.1 Detailed Description

template<class T = ftype> class clipper::Array2d< T >

Simple 2-d array class.

The documentation for this class was generated from the following file:

- clipper_types.h

19.3 clipper::Atom Class Reference

Atom (p. 107) class.

`#include <coords.h>`

Public Member Functions

- **Atom** ()
  null constructor
- template<class T >
  Atom (const T &atom)
  Constructor: from atom-like object.
- const **String** & **element** () const
  get atom element name: e.g. "C", "N", "Zn2+"
- const **Coord_orth** & **coord_orth** () const
  get atom orthogonal (Angstrom) coordinate
- const **ftype** & **occupancy** () const
  get atom occupancy
- const **ftype** & **u_iso** () const
  get atom orthogonal isotropic U value
- const **U_aniso_orth** & **u_aniso_orth** () const
  get atom orthogonal anisotropic U value
- void **set_element** (const **String** &s)
  set element
- void **set_coord_orth** (const **Coord_orth** &s)
  set coord_orth
- void **set_occupancy** (const **ftype** &s)
  set occupancy
- void **set_u_iso** (const **ftype** &s)
  set u_iso
- void **set_u_aniso_orth** (const **U_aniso_orth** &s)
  set u_aniso
- void **transform** (const **RTop_orth** rt)
  *apply a rotation-translation operator (**RTop** (p. 402)) to the atom*
- bool **is_null** () const
  test for null atom: atom is null is coord is null

Static Public Member Functions

- static **Atom null** ()
  return null atom

19.3.1 Detailed Description

Atom (p. 107) class. This class defines a minimal atom object providing only those properties required for an electron density calculation. A template constructor allows it to be constructed from any other object with appropriate properties.

19.3.2 Member Function Documentation

19.3.2.1 void clipper::Atom::transform (const RTop_orth *rt*)

apply a rotation-translation operator (**RTop** (p. 402)) to the atom The coordinates and **U_-aniso_orth** (p. 454) are transformed. The sigmas are not, since without the full variance-covariance matrix this transformation is impossible.

Parameters:

rt The operator to apply.

References coord_orth(), set_coord_orth(), set_u_aniso_orth(), and u_aniso_orth().

The documentation for this class was generated from the following files:

- coords.h
- coords.cpp

19.4 clipper::Atom_list Class Reference

Atom (p. 107) list class.

`#include <coords.h>`

Public Member Functions

- **Atom_list** ()
  null constructor
- **Atom_list** (const std::vector< **Atom** > &list)
  constructor: from std::vector<Atom>
- template<class T >
  Atom_list (const T &list)
  Constructor: from vector-like list of atom-like objects.

19.4.1 Detailed Description

Atom (p. 107) list class. This class defines a minimal atom list object providing only those properties required for an electron density calculation. It is a trivial derivation from std::vector<Atom>. In addition a template constructor allows it to be constructed from any other object with appropriate properties.

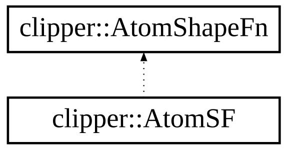
The documentation for this class was generated from the following file:

- coords.h

19.5 clipper::AtomSF Class Reference

Atomic scattering factor object.

#include <atomsf.h>Inheritance diagram for clipper::AtomSF::



Public Member Functions

- **AtomSF** (const **String** &type, const **ftype** u_iso=0.0, const **ftype** occ=1.0)
- **AtomSF** (const **String** &type, const **U_aniso_orth** &u_aniso, const **ftype** occ=1.0)
- void **init** (const **String** &type, const **ftype** u_iso=0.0, const **ftype** occ=1.0)
- void **init** (const **String** &type, const **U_aniso_orth** &u_aniso, const **ftype** occ=1.0)
- **ftype f_iso** (const **ftype** &s) const
- **ftype f_aniso** (const **Coord_reci_orth** &rfl) const
- **ftype rho_iso** (const **ftype** &d2) const
- **ftype rho_aniso** (const **Coord_orth** &uvw) const

19.5.1 Detailed Description

Atomic scattering factor object.

Deprecated

This class has been replaced by **AtomShapeFn** (p. 111), which is smaller, faster, and more capable. This class is now a wrapper for that class.

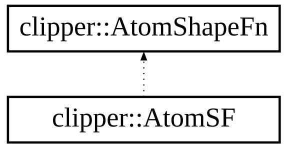
The documentation for this class was generated from the following files:

- **atomsf.h**
- atomsf.cpp

19.6 clipper::AtomShapeFn Class Reference

Atomic shape function object.

`#include <atomsf.h>`Inheritance diagram for clipper::AtomShapeFn::



Public Types

- enum **TYPE** {
  X, **Y**, **Z**, **Uiso**,
  Occ, **U11**, **U22**, **U33**,
  U12, **U13**, **U23** }

Public Member Functions

- **AtomShapeFn** ()

  null constructor

- **AtomShapeFn** (const **Atom** &atom)

  constructor: from atom object

- **AtomShapeFn** (const **Coord_orth** &xyz, const **String** &element, const **ftype** u_-iso=0.0, const **ftype** occ=1.0)

  constructor: from coord, element, isotropic U, occupancy

- **AtomShapeFn** (const **Coord_orth** &xyz, const **String** &element, const **U_aniso_orth** &u_aniso, const **ftype** occ=1.0)

  constructor: from coord, element, anisotropic U, occupancy

- void **init** (const **Atom** &atom)

  initialiser: from atom object

- void **init** (const **Coord_orth** &xyz, const **String** &element, const **ftype** u_iso=0.0, const **ftype** occ=1.0)

  initialiser: from coord, element, isotropic U, occupancy

- void **init** (const **Coord_orth** &xyz, const **String** &element, const **U_aniso_orth** &u_-aniso, const **ftype** occ=1.0)

  initialiser: from coord, element, anisotropic U, occupancy

- **ftype f** (const **Coord_reci_orth** &rfl) const

  return scattering factor as a function of reflection posn

- **ftype rho** (const **Coord_orth** &xyz) const

  return electron density as a function of coordinate

- bool **rho_grad** (const **Coord_orth** &xyz, **ftype** &rho, std::vector< **ftype** > &grad) const

  return Agarwal density gradients as a function of coordinate

- bool **rho_curv** (const **Coord_orth** &xyz, **ftype** &rho, std::vector< **ftype** > &grad, **Matrix**< **ftype** > &curv) const

  return Agarwal density gradient/curvature as a function of coordinate

- bool **rho_grad** (const **Coord_orth** &xyz, std::vector< **ftype** > &grad) const
- **ftype f** (const **ftype** &invresolsq) const

  return (isotropic) scattering factor as a function of resolution

- **ftype rho** (const **ftype** &rsq) const

  return (isotropic) electron density as a function of radius

- std::vector< TYPE > & **agarwal_params** ()

  define parameters for Agarwal gradient/curvature calcs

19.6.1 Detailed Description

Atomic shape function object. The atomic scattering factor object is instantiated for each atom in turn, giving the atom parameters: position, element, occupancy and the isotropic or anisotropic U-value. (See **clipper::Util** (p. 456) for conversion from B-factors.). The methods of the class may then be called to return the scattering in reciprocal space or density in real space using either isotropic or anistropic models as required.

If the atom only has an isotropic U, the faster isotropic methods will be used where available.

This implementation uses the coefficients from Waasmaier & Kirfel (1995), Acta Cryst. A51, 416-431. The source data can be found at: `ftp://wrzx02.rz.uni-wuerzburg.de/pub/local/Crystallography/sfac.dat`

19.6.2 Constructor & Destructor Documentation

19.6.2.1 clipper::AtomShapeFn::AtomShapeFn (const Atom & *atom*)

constructor: from atom object If the atom has an anisotropic U (even if the values are isotropic), then it is initialised as anisotropic, otherwise it is isotropic.

Parameters:

atom The atom object.

References init().

19.6.2.2 clipper::AtomShapeFn::AtomShapeFn (const Coord_orth & *xyz*, const String & *element*, const ftype *u_iso* = 0.0, const ftype *occ* = 1.0)

constructor: from coord, element, isotropic U, occupancy The atom is initialised as isotropic.

Parameters:

- ***xyz*** The atom coordinate.
- ***element*** The atom element.
- ***u_iso*** The isotropic U-value.
- ***occ*** The occupancy.

References init().

19.6.2.3 clipper::AtomShapeFn::AtomShapeFn (const Coord_orth & *xyz*, const String & *element*, const U_aniso_orth & *u_aniso*, const ftype *occ* = 1.0)

constructor: from coord, element, anisotropic U, occupancy The atom is initialised as anisotropic.

Parameters:

- ***xyz*** The atom coordinate.
- ***element*** The atom element.
- ***u_aniso*** The anisotropic U-value.
- ***occ*** The occupancy.

References init().

19.6.3 Member Function Documentation

19.6.3.1 void clipper::AtomShapeFn::init (const Atom & *atom*)

initialiser: from atom object If the atom has an anisotropic U (even if the values are isotropic), then it is initialised as anisotropic, otherwise it is isotropic.

Parameters:

- ***atom*** The atom object.

References clipper::Atom::coord_orth(), clipper::Atom::element(), clipper::Util::is_nan(), clipper::Mat33sym< T >::is_null(), clipper::Atom::occupancy(), clipper::Atom::u_aniso_orth(), and clipper::Atom::u_iso().

Referenced by AtomShapeFn(), and init().

19.6.3.2 void clipper::AtomShapeFn::init (const Coord_orth & *xyz*, const String & *element*, const ftype *u_iso* = 0.0, const ftype *occ* = 1.0)

initialiser: from coord, element, isotropic U, occupancy The atom is initialised as isotropic.

Parameters:

xyz The atom coordinate.

element The atom element.

u_iso The isotropic U-value.

occ The occupancy.

References init().

19.6.3.3 void clipper::AtomShapeFn::init (const Coord_orth & *xyz*, const String & *element*, const U_aniso_orth & *u_aniso*, const ftype *occ* = 1.0)

initialiser: from coord, element, anisotropic U, occupancy The atom is initialised as anisotropic.

Parameters:

xyz The atom coordinate.

element The atom element.

u_aniso The anisotropic U-value.

occ The occupancy.

References clipper::Mat33sym< T >::det(), init(), clipper::Mat33sym< T >::mat00(), clipper::Mat33sym< T >::mat01(), clipper::Mat33sym< T >::mat02(), clipper::Mat33sym< T >::mat11(), clipper::Mat33sym< T >::mat12(), clipper::Mat33sym< T >::mat22(), clipper::Util::pi(), clipper::Util::twopi(), and clipper::U_aniso_orth::u_iso().

19.6.3.4 ftype clipper::AtomShapeFn::f (const Coord_reci_orth & *rfl*) const

return scattering factor as a function of reflection posn Return the scattering factor as a function of position in reciprocal space in electrons.

Parameters:

rfl Position in reciprocal space.

Returns:

The scattering factor in electrons.

References clipper::Coord_reci_orth::invresolsq(), clipper::Mat33sym< T >::quad_form(), and clipper::Util::twopi2().

19.6.3.5 ftype clipper::AtomShapeFn::rho (const Coord_orth & *xyz*) const

return electron density as a function of coordinate Return the density as a function of position in real space in electrons.

Parameters:

xyz Position in real space.

The density in electrons.

References clipper::Coord_orth::lengthsq().

Referenced by clipper::Test_core::operator()().

19.6.3.6 bool clipper::AtomShapeFn::rho_grad (const Coord_orth & *xyz*, ftype & *rho*, std::vector< ftype > & *grad*) const

return Agarwal density gradients as a function of coordinate Return the Agarwal gradients of the density with respect to tha atomic parameters as a function of position in real space in electrons. The parameter list is defined by assignment to **agarwal_params()** (p. 112).

Parameters:

xyz Position in real space. rho The density in electrons.

grad Vector gradient in electrons (pre-size for best performance).

References clipper::Coord_orth::lengthsq(), clipper::Coord_orth::x(), clipper::Coord_orth::y(), and clipper::Coord_orth::z().

19.6.3.7 bool clipper::AtomShapeFn::rho_curv (const Coord_orth & *xyz*, ftype & *rho*, std::vector< ftype > & *grad*, Matrix< ftype > & *curv*) const

return Agarwal density gradient/curvature as a function of coordinate Return the Agarwal gradients of the density with respect to tha atomic parameters as a function of position in real space in electrons. The parameter list is defined by assignment to **agarwal_params()** (p. 112).

Parameters:

xyz Position in real space. rho The density in electrons.

grad Vector gradient in electrons (pre-size for best performance).

curv **Matrix** (p. 338) curvature in electrons (pre-size for best performance).

References clipper::Coord_orth::lengthsq(), clipper::Message::message(), clipper::Util::nan(), clipper::Coord_orth::x(), clipper::Coord_orth::y(), and clipper::Coord_orth::z().

Referenced by clipper::Test_core::operator()().

19.6.3.8 bool clipper::AtomShapeFn::rho_grad (const Coord_orth & *xyz*, std::vector< ftype > & *grad*) const

Deprecated

return Agarwal density gradients as a function of coordinate

References clipper::Coord_orth::lengthsq(), clipper::Message::message(), clipper::Coord_-orth::x(), clipper::Coord_orth::y(), and clipper::Coord_orth::z().

19.6.3.9 ftype clipper::AtomShapeFn::f (const ftype & *invresolsq*) const

return (isotropic) scattering factor as a function of resolution Return the scattering factor as a function of position in reciprocal space in electrons.

Parameters:

invresolsq Inverse resolution squared in inverse Angstroms squared.

Returns:

The scattering factor in electrons.

References clipper::Util::u2b().

19.6.3.10 ftype clipper::AtomShapeFn::rho (const ftype & *rsq*) const

return (isotropic) electron density as a function of radius Return the density as a function of position in real space in electrons.

Parameters:

invresolsq Radius squared in Angstroms squared.

The density in electrons.

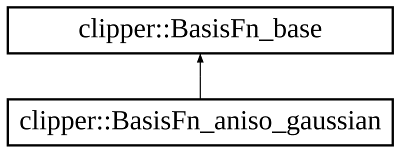
The documentation for this class was generated from the following files:

- **atomsf.h**
- atomsf.cpp

19.7 clipper::BasisFn_aniso_gaussian Class Reference

simple anisotropic Gaussian basis function

#include <resol_basisfn.h>Inheritance diagram for clipper::BasisFn_aniso_gaussian::



Public Member Functions

- **BasisFn_aniso_gaussian** ()

  constructor:

- const **BasisFn_base::Fderiv** & **fderiv_coord** (const **Coord_reci_orth** &xs, const std::vector< **ftype** > &params) const

  the derivatives of the resolution function w.r.t. the parameters

- const **BasisFn_base::Fderiv** & **fderiv** (const **HKL** &hkl, const **Cell** &cell, const std::vector< **ftype** > &params) const

  the value of the resolution function (override for speed)

- **ftype scale** (const std::vector< **ftype** > &params) const

  return the scale factor corresponding to the Gaussian parameters

- **U_aniso_orth u_aniso_orth** (const std::vector< **ftype** > &params) const

  return the anisotropic U corresponding to the Gaussian parameters

19.7.1 Detailed Description

simple anisotropic Gaussian basis function This class provides a anisotropic Gaussian basis function.

19.7.2 Member Function Documentation

19.7.2.1 const BasisFn_base::Fderiv& clipper::BasisFn_aniso_gaussian::fderiv (const HKL & *hkl*, const Cell & *cell*, const std::vector< ftype > & *params*) const `[inline, virtual]`

the value of the resolution function (override for speed) the derivatives of the resolution function w.r.t. the parameters

Implements **clipper::BasisFn_base** (p. 120).

References clipper::HKL::coord_reci_orth(), and fderiv_coord().

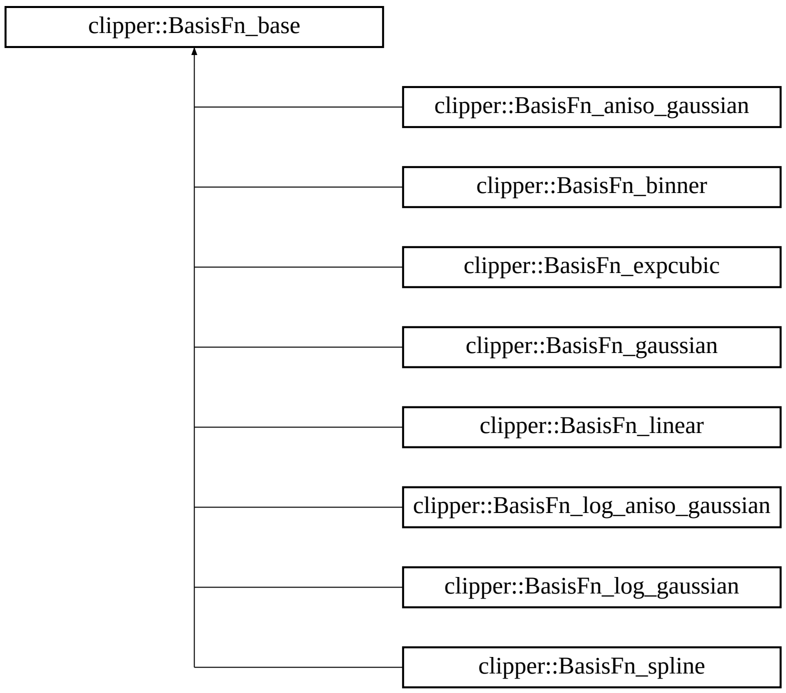
The documentation for this class was generated from the following files:

- resol_basisfn.h
- resol_basisfn.cpp

19.8 clipper::BasisFn_base Class Reference

abstract base class for resolution function basis functions

#include <resol_fn.h>Inheritance diagram for clipper::BasisFn_base::



Classes

- class **Fderiv**

  object holding the basis function and its first two derivatives

Public Types

- enum **FNtype** { **GENERAL**, **LINEAR** }

  enumeration of function types: optionally used to improve convergence

Public Member Functions

- **BasisFn_base** ()

  null constructor

- **BasisFn_base** (const int &np)

  constructor: takes number of parameters

- const int & **num_params** () const

  the number of parameters of this basis function

- virtual **ftype f** (const **HKL** &hkl, const **Cell** &cell, const std::vector< **ftype** > &params) const

  the value of the resolution function

- virtual const **Fderiv** & **fderiv** (const **HKL** &hkl, const **Cell** &cell, const std::vector< **ftype** > &params) const =0

  the value of the resolution function and its first two derivatives

- virtual **FNtype type** () const

  the type of the function: optionally used to improve convergence

- virtual int **num_diagonals** () const

  number of non-zero diagonals in the upper triangle of the curvatures

Protected Member Functions

- **Fderiv** & **result** () const

  provide write access to result for subclasses

- virtual ~**BasisFn_base** ()

  destructor

19.8.1 Detailed Description

abstract base class for resolution function basis functions A basis function must be able to return its value and derivatives for any given **HKL** (p. 261).

Optionally, performance can be improved by returning a flag to indicate if the value of the basis function for a given reflection is linearly dependent on the values of the parameter, and a value indicating whether the curvature matrix takes an N-diagonal form.

NOTE: for performance reasons the derivatives are returned as a reference to an internal object, so if you store a reference to the result (which is also good for performance, it will be overwritten on the next call. If this matters, store a copy rather than a reference.

19.8.2 Member Function Documentation

19.8.2.1 BasisFn_base::FNtype clipper::BasisFn_base::type () const `[virtual]`

the type of the function: optionally used to improve convergence Defaults to GENERAL, which will always work. If the basis function is linearly dependent on the parameters, override this with a function returning LINEAR for improved performance. See the provided basis functions for examples.

Returns:

The function type enumeration.

Reimplemented in **clipper::BasisFn_binner** (p. 122), **clipper::BasisFn_linear** (p. 126), **clipper::BasisFn_spline** (p. 131), **clipper::BasisFn_log_gaussian** (p. 129), and **clipper::BasisFn_log_aniso_gaussian** (p. 127).

Referenced by clipper::ResolutionFn::ResolutionFn().

19.8.2.2 int clipper::BasisFn_base::num_diagonals () const [virtual]

number of non-zero diagonals in the upper triangle of the curvatures Defaults to 0, which will always work. If the basis function has compact support among the parameters, i.e. the value for any **HKL** (p. 261) depends only on a few parameters, then set this to the number of non-zero diagonals in the upper triangle of the matrix, i.e. 1 for a diagonal matrix, 2 for a tri-diagonal matrix etc.

Returns:

The number of non-zero upper diagonals, or zero for full-matrix.

Reimplemented in **clipper::BasisFn_binner** (p. 122), **clipper::BasisFn_linear** (p. 126), and **clipper::BasisFn_spline** (p. 131).

Referenced by clipper::ResolutionFn::calc_derivs().

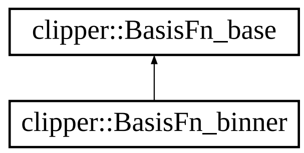
The documentation for this class was generated from the following files:

- resol_fn.h
- resol_fn.cpp

19.9 clipper::BasisFn_binner Class Reference

simple binning basis function

`#include` <`resol_basisfn.h`>Inheritance diagram for clipper::BasisFn_binner::



Public Member Functions

- **BasisFn_binner** (const **HKL_info** &hklinfo, const int &nbins_, const **ftype** power=1.0)

  constructor: include whole reflection list in histogram

- **BasisFn_binner** (const **HKL_data_base** &hkldata, const int &nbins_, const **ftype** power=1.0)

  constructor: include only non-missing reflections in histogram

- **ftype f_s** (const **ftype** &s, const std::vector< **ftype** > &params) const

  the value of the resolution function (override for speed)

- const **BasisFn_base::Fderiv** & **fderiv_s** (const **ftype** &s, const std::vector< **ftype** > &params) const

  the derivative of the resolution function w.r.t. the parameters

- **FNtype type** () const

  the type of the function: optionally used to improve convergence

- int **num_diagonals** () const

  number of non-zero diagonals in the upper triangle of the curvatures

- **ftype f** (const **HKL** &hkl, const **Cell** &cell, const std::vector< **ftype** > &params) const

  the value of the resolution function (override for speed)

- const **BasisFn_base::Fderiv** & **fderiv** (const **HKL** &hkl, const **Cell** &cell, const std::vector< **ftype** > &params) const

  the derivatives of the resolution function w.r.t. the parameters

19.9.1 Detailed Description

simple binning basis function This class bins reflections on the basis of resolution, i.e. it generates a resolution function from spherical shells.

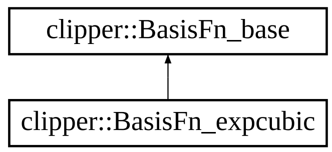
The documentation for this class was generated from the following files:

- resol_basisfn.h
- resol_basisfn.cpp

19.10 clipper::BasisFn_expcubic Class Reference

simple Expcubic basis function

`#include <resol_basisfn.h>`Inheritance diagram for clipper::BasisFn_expcubic::



Public Member Functions

- **BasisFn_expcubic** ()

  constructor

- const **BasisFn_base::Fderiv** & **fderiv_s** (const **ftype** &s, const std::vector< **ftype** > &params) const

  the derivatives of the resolution function w.r.t. the parameters

- const **BasisFn_base::Fderiv** & **fderiv** (const **HKL** &hkl, const **Cell** &cell, const std::vector< **ftype** > &params) const

  the value of the resolution function (override for speed)

19.10.1 Detailed Description

simple Expcubic basis function This class provides a Expcubic basis function.

19.10.2 Member Function Documentation

19.10.2.1 const BasisFn_base::Fderiv& clipper::BasisFn_expcubic::fderiv (const HKL & *hkl*, const Cell & *cell*, const std::vector< ftype > & *params*) const `[inline, virtual]`

the value of the resolution function (override for speed) the derivatives of the resolution function w.r.t. the parameters

Implements **clipper::BasisFn_base** (p. 120).

References fderiv_s(), and clipper::HKL::invresolsq().

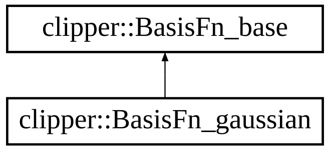
The documentation for this class was generated from the following files:

- resol_basisfn.h
- resol_basisfn.cpp

19.11 clipper::BasisFn_gaussian Class Reference

simple Gaussian basis function

`#include <resol_basisfn.h>`Inheritance diagram for clipper::BasisFn_gaussian::



Public Member Functions

- **BasisFn_gaussian** ()

  constructor:

- const **BasisFn_base::Fderiv** & **fderiv_s** (const **ftype** &s, const std::vector< **ftype** > &params) const

  the value of the resolution function

- const **BasisFn_base::Fderiv** & **fderiv** (const **HKL** &hkl, const **Cell** &cell, const std::vector< **ftype** > &params) const

  the value of the resolution function (override for speed)

- **ftype scale** (const std::vector< **ftype** > &params) const

  return the scale factor corresponding to the Gaussian parameters

- **ftype u_iso** (const std::vector< **ftype** > &params) const

  return the isotropic U corresponding to the Gaussian parameters

19.11.1 Detailed Description

simple Gaussian basis function This class provides a Gaussian basis function.

19.11.2 Member Function Documentation

19.11.2.1 const BasisFn_base::Fderiv & clipper::BasisFn_gaussian::fderiv_s (const ftype & *s*, const std::vector< ftype > & *params*) const

the value of the resolution function the derivatives of the resolution function w.r.t. the parameters

References clipper::BasisFn_base::Fderiv::df, clipper::BasisFn_base::Fderiv::df2, clipper::BasisFn_base::Fderiv::f, clipper::BasisFn_base::f(), and clipper::BasisFn_base::result().

Referenced by fderiv().

19.11.2.2 const BasisFn_base::Fderiv& clipper::BasisFn_gaussian::fderiv (const HKL & *hkl*, const Cell & *cell*, const std::vector< ftype > & *params*) const `[inline, virtual]`

the value of the resolution function (override for speed) the derivatives of the resolution function w.r.t. the parameters

Implements **clipper::BasisFn_base** (p. 120).

References fderiv_s(), and clipper::HKL::invresolsq().

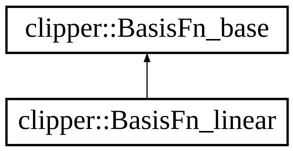
The documentation for this class was generated from the following files:

- resol_basisfn.h
- resol_basisfn.cpp

19.12 clipper::BasisFn_linear Class Reference

simple linear basis function

#include <resol_basisfn.h>Inheritance diagram for clipper::BasisFn_linear::



Public Member Functions

- **BasisFn_linear** (const **HKL_info** &hklinfo, const int &nbins_, const **ftype** power=1.0)
  constructor: include whole reflection list in histogram
- **BasisFn_linear** (const **HKL_data_base** &hkldata, const int &nbins_, const **ftype** power=1.0)
  constructor: include only non-missing reflections in histogram
- **ftype f_s** (const **ftype** &s, const std::vector< **ftype** > &params) const
  the value of the resolution function (override for speed)
- const **BasisFn_base::Fderiv** & **fderiv_s** (const **ftype** &s, const std::vector< **ftype** > &params) const
  the derivative of the resolution function w.r.t. the parameters
- **FNtype type** () const
  the type of the function: optionally used to improve convergence
- int **num_diagonals** () const
  number of non-zero diagonals in the upper triangle of the curvatures
- **ftype f** (const **HKL** &hkl, const **Cell** &cell, const std::vector< **ftype** > &params) const
  the value of the resolution function (override for speed)
- const **BasisFn_base::Fderiv** & **fderiv** (const **HKL** &hkl, const **Cell** &cell, const std::vector< **ftype** > &params) const
  the derivatives of the resolution function w.r.t. the parameters

19.12.1 Detailed Description

simple linear basis function This class fits a piecewise linear function through reflections on the basis of resolution.

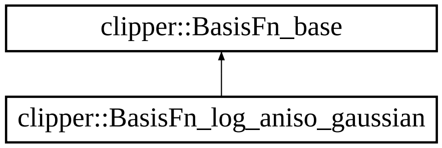
The documentation for this class was generated from the following files:

- resol_basisfn.h
- resol_basisfn.cpp

19.13 clipper::BasisFn_log_aniso_gaussian Class Reference

simple anisotropic Gaussian basis function

#include <resol_basisfn.h>Inheritance diagram for clipper::BasisFn_log_aniso_gaussian::



Public Member Functions

- **BasisFn_log_aniso_gaussian** ()

  constructor:

- const **BasisFn_base::Fderiv** & **fderiv_coord** (const **Coord_reci_orth** &xs, const std::vector< **ftype** > &params) const

  the derivatives of the resolution function w.r.t. the parameters

- const **BasisFn_base::Fderiv** & **fderiv** (const **HKL** &hkl, const **Cell** &cell, const std::vector< **ftype** > &params) const

  the value of the resolution function (override for speed)

- **FNtype type** () const

  the type of the function: optionally used to improve convergence

- **ftype scale** (const std::vector< **ftype** > &params) const

  return the scale factor corresponding to the Gaussian parameters

- **U_aniso_orth u_aniso_orth** (const std::vector< **ftype** > &params) const

  return the anisotropic U corresponding to the Gaussian parameters

19.13.1 Detailed Description

simple anisotropic Gaussian basis function This class provides a anisotropic Gaussian basis function. i.e. a general quadratic function of resolution. Use this in conjunction with a Log-target function to get a fast estimate to a Gaussian fit.

19.13.2 Member Function Documentation

19.13.2.1 const BasisFn_base::Fderiv& clipper::BasisFn_log_aniso_gaussian::fderiv (const HKL & *hkl*, const Cell & *cell*, const std::vector< ftype > & *params*) const `[inline, virtual]`

the value of the resolution function (override for speed) the derivatives of the resolution function w.r.t. the parameters

Implements **clipper::BasisFn_base** (p. 120).

References clipper::HKL::coord_reci_orth(), and fderiv_coord().

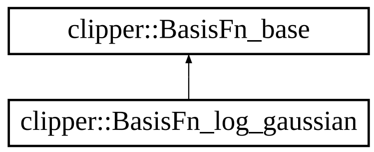
The documentation for this class was generated from the following files:

- resol_basisfn.h
- resol_basisfn.cpp

19.14 clipper::BasisFn_log_gaussian Class Reference

simple log Gaussian basis function

`#include` <`resol_basisfn.h`>Inheritance diagram for clipper::BasisFn_log_gaussian::



Public Member Functions

- **BasisFn_log_gaussian** ()

  constructor:

- const **BasisFn_base::Fderiv** & **fderiv_s** (const **ftype** &s, const std::vector< **ftype** > &params) const

  the value of the resolution function

- const **BasisFn_base::Fderiv** & **fderiv** (const **HKL** &hkl, const **Cell** &cell, const std::vector< **ftype** > &params) const

  the value of the resolution function (override for speed)

- **FNtype type** () const

  the type of the function: optionally used to improve convergence

- **ftype scale** (const std::vector< **ftype** > &params) const

  return the scale factor corresponding to the Gaussian parameters

- **ftype u_iso** (const std::vector< **ftype** > &params) const

  return the isotropic U corresponding to the Gaussian parameters

19.14.1 Detailed Description

simple log Gaussian basis function This class provides a Log Gaussian basis function. i.e. a quadratic function of resolution. Use this in conjunction with a Log-target function to get a fast estimate to a Gaussian fit.

19.14.2 Member Function Documentation

19.14.2.1 const BasisFn_base::Fderiv & clipper::BasisFn_log_gaussian::fderiv_s (const ftype & *s*, const std::vector< ftype > & *params*) const

the value of the resolution function the derivatives of the resolution function w.r.t. the parameters

References clipper::BasisFn_base::Fderiv::df, clipper::BasisFn_base::Fderiv::f, clipper::BasisFn_-base::f(), and clipper::BasisFn_base::result().

Referenced by fderiv().

19.14.2.2 const BasisFn_base::Fderiv& clipper::BasisFn_log_gaussian::fderiv (const HKL & *hkl*, const Cell & *cell*, const std::vector< ftype > & *params*) const [inline, virtual]

the value of the resolution function (override for speed) the derivatives of the resolution function w.r.t. the parameters

Implements **clipper::BasisFn_base** (p. 120).

References fderiv_s(), and clipper::HKL::invresolsq().

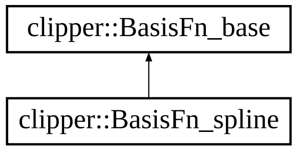
The documentation for this class was generated from the following files:

- resol_basisfn.h
- resol_basisfn.cpp

19.15 clipper::BasisFn_spline Class Reference

simple smooth basis function

#include <resol_basisfn.h>Inheritance diagram for clipper::BasisFn_spline::



Public Member Functions

- **BasisFn_spline** (const **HKL_info** &hklinfo, const int &nbins_, const **ftype** power=1.0)

  constructor: include whole reflection list in histogram

- **BasisFn_spline** (const **HKL_data_base** &hkldata, const int &nbins_, const **ftype** power=1.0)

  constructor: include only non-missing reflections in histogram

- **ftype f_s** (const **ftype** &s, const std::vector< **ftype** > &params) const

  the value of the resolution function (override for speed)

- const **BasisFn_base::Fderiv** & **fderiv_s** (const **ftype** &s, const std::vector< **ftype** > &params) const

  the derivative of the resolution function w.r.t. the parameters

- **FNtype type** () const

  the type of the function: optionally used to improve convergence

- int **num_diagonals** () const

  number of non-zero diagonals in the upper triangle of the curvatures

- **ftype f** (const **HKL** &hkl, const **Cell** &cell, const std::vector< **ftype** > &params) const

  the value of the resolution function (override for speed)

- const **BasisFn_base::Fderiv** & **fderiv** (const **HKL** &hkl, const **Cell** &cell, const std::vector< **ftype** > &params) const

  the derivatives of the resolution function w.r.t. the parameters

19.15.1 Detailed Description

simple smooth basis function This class fits a Bspline through reflections on the basis of resolution.

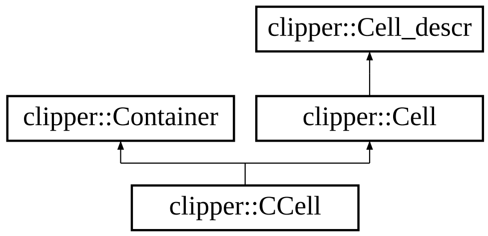
The documentation for this class was generated from the following files:

- resol_basisfn.h
- resol_basisfn.cpp

19.16 clipper::CCell Class Reference

CCell (p. 132) container.

`#include <container_types.h>`Inheritance diagram for clipper::CCell::



Public Member Functions

- **CCell** (const **String** name="", const **Cell** &cell=NullCell)

  constructor: make null object or top object in tree

- **CCell** (**Container** &parent, const **String** name="", const **Cell** &cell=NullCell)

  constructor: make child object

- void **init** (const **Cell** &cell_)

  initialiser: from ***Cell*** *(*p. *133)*

19.16.1 Detailed Description

CCell (p. 132) container. **CCell** (p. 132): This has a name and a cell. It overrides the base cell for any objects below it.

19.16.2 Member Function Documentation

19.16.2.1 void clipper::CCell::init (const Cell & *cell_*)

initialiser: from **Cell** (p. 133) The object is initialised, and children are updated.

Parameters:

cell_ The value to give to the contained object.

References clipper::Container::update().

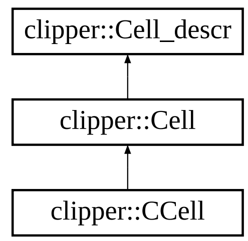
The documentation for this class was generated from the following files:

- container_types.h
- container_types.cpp

19.17 clipper::Cell Class Reference

Cell (p. 133) object.

`#include <cell.h>`Inheritance diagram for clipper::Cell::



Public Member Functions

- **Cell** ()

  null constructor: must initialise later

- **Cell** (const **Cell_descr** &cell_)

  *constructor: takes a **Cell** (p. 133) descriptor*

- void **init** (const **Cell_descr** &cell_)

  initialiser

- bool **is_null** () const

  test if object has been initialised

- **ftype a_star** () const

  *get a**

- **ftype b_star** () const

  *get b**

- **ftype c_star** () const

  *get c**

- **ftype alpha_star** () const

  *get alpha**

- **ftype beta_star** () const

  *get beta**

- **ftype gamma_star** () const

  *get gamma**

- const **ftype** & **a** () const

  get a

- const **ftype** & **b** () const
  get b
- const **ftype** & **c** () const
  get c
- const **ftype** & **alpha** () const
  get alpha
- const **ftype** & **beta** () const
  get beta
- const **ftype** & **gamma** () const
  get gamma
- **ftype alpha_deg** () const
  get alpha in degrees
- **ftype beta_deg** () const
  get alpha in degrees
- **ftype gamma_deg** () const
  get gamma in degrees
- **String format** () const
  return formatted ***String*** *(p. 426) representation*
- const **Cell_descr** & **descr** () const
  return cell dimensions
- const **ftype** & **volume** () const
  return cell volume
- bool **equals** (const **Cell** &other, const **ftype** tol=1.0) const
  test equality with another cell
- const **Mat33** & **matrix_orth** () const
  return orthogonalisation matrix
- const **Mat33** & **matrix_frac** () const
  return fractionalisation matrix
- const **Metric_tensor** & **metric_real** () const
  return real space metric tensor
- const **Metric_tensor** & **metric_reci** () const
  return reciprocal space metric tensor
- void **debug** () const

19.17.1 Detailed Description

Cell (p. 133) object. The **Cell** (p. 133) class is the fully functional description of the unit cell. In addition to the cell parameters, it stores derived information including the cell volume, orthogonalising and fractionalising matrices, and the metric tensors.

19.17.2 Member Function Documentation

19.17.2.1 void clipper::Cell::init (const Cell_descr & *cell_*)

initialiser Initialise the **Cell** (p. 133) object from a cell description.

Parameters:

*cell_* The cell descirption.

References a(), clipper::Cell_descr::a(), a_star(), alpha(), clipper::Cell_descr::alpha(), alpha_-star(), b(), clipper::Cell_descr::b(), b_star(), beta(), clipper::Cell_descr::beta(), beta_star(), c(), clipper::Cell_descr::c(), c_star(), gamma(), clipper::Cell_descr::gamma(), gamma_star(), clipper::Mat33< T >::inverse(), and is_null().

Referenced by Cell().

19.17.2.2 bool clipper::Cell::is_null () const

test if object has been initialised

Returns:

true if the object has not been initalised.

Referenced by equals(), clipper::HKL_info::init(), clipper::HKL_data_base::init(), clipper::CGrid_sampling::init(), clipper::CNXmap_operator< T >::init(), clipper::CXmap< T >::init(), clipper::CHKL_data< T >::init(), clipper::CHKL_info::init(), init(), clipper::Xmap_-base::is_null(), clipper::HKL_info::is_null(), and clipper::HKL_data_base::is_null().

19.17.2.3 ftype clipper::Cell::alpha_deg () const

get alpha in degrees

Returns:

The cell angle in degrees

Reimplemented from **clipper::Cell_descr** (p. 138).

19.17.2.4 ftype clipper::Cell::beta_deg () const

get alpha in degrees

Returns:

The cell angle in degrees

Reimplemented from **clipper::Cell_descr** (p. 139).

19.17.2.5 ftype clipper::Cell::gamma_deg () const

get gamma in degrees

Returns:

The cell angle in degrees

Reimplemented from **clipper::Cell_descr** (p. 139).

19.17.2.6 String clipper::Cell::format () const

return formatted **String** (p. 426) representation

Returns:

A string describing the cell

Reimplemented from **clipper::Cell_descr** (p. 139).

19.17.2.7 bool clipper::Cell::equals (const Cell & *other*, const ftype *tol* = 1.0) const

test equality with another cell Two cells disagree if the difference in their orthogonalisation matrices is sufficient to map a reflection from one cell onto a different reflection in the other cell at the given tolerance, which is the resolution of the reflection in Angstroms.

Parameters:

other The other cell to compare.

tol The tolerance, in Angstroms.

References is_null().

Referenced by clipper::HKL_data_base::init().

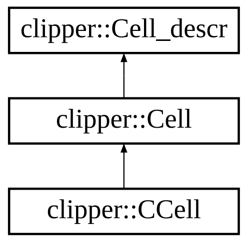
The documentation for this class was generated from the following files:

- cell.h
- cell.cpp

19.18 clipper::Cell_descr Class Reference

cell description (automatically converts to radians)

#include <cell.h>Inheritance diagram for clipper::Cell_descr::



Public Member Functions

- **Cell_descr** ()

  null constructor

- **Cell_descr** (const **ftype** &a, const **ftype** &b, const **ftype** &c, const **ftype** &alpha=90.0f, const ftype &beta=90.0f, const ftype &gamma=90.0f)

  constructor: from cell parameters

- const **ftype** & **a** () const

  get a

- const **ftype** & **b** () const

  get b

- const **ftype** & **c** () const

  get c

- const **ftype** & **alpha** () const

  get alpha

- const **ftype** & **beta** () const

  get beta

- const **ftype** & **gamma** () const

  get gamma

- **ftype alpha_deg** () const

  get alpha in degrees

- **ftype beta_deg** () const

  get alpha in degrees

- **ftype gamma_deg** () const

  get gamma in degrees

- **String format** () const
  return formatted ***String*** *(*p. *426) representation*

Protected Attributes

- **ftype a_**
- **ftype b_**
- **ftype c_**
- **ftype alpha_**
- **ftype beta_**
- **ftype gamma_**

19.18.1 Detailed Description

cell description (automatically converts to radians) The cell description is a compact description of a cell, containing just the cell parameters. It is usually used to construct a full **Cell** (p. 133) object, which provides the expected functionality.

19.18.2 Constructor & Destructor Documentation

19.18.2.1 clipper::Cell_descr::Cell_descr (const ftype & *a*, const ftype & *b*, const ftype & *c*, const ftype & *alpha* = `90.0f`, const ftype & *beta* = `90.0f`, const ftype & *gamma* = `90.0f`)

constructor: from cell parameters

Parameters:

a A axis in Angstroms.
b B axis in Angstroms.
c C axis in Angstroms.
alpha Angle between B and C axes in radians or degrees, default=90
beta Angle between A and C axes in radians or degrees, default=90
gamma Angle between A and C axes in radians or degrees, default=90

References clipper::Util::d2rad().

19.18.3 Member Function Documentation

19.18.3.1 ftype clipper::Cell_descr::alpha_deg () const

get alpha in degrees

Returns:

The cell angle in degrees

Reimplemented in **clipper::Cell** (p. 135).

References clipper::Util::rad2d().

Referenced by format().

19.18.3.2 ftype clipper::Cell_descr::beta_deg () const

get alpha in degrees

Returns:

The cell angle in degrees

Reimplemented in **clipper::Cell** (p. 135).

References clipper::Util::rad2d().

Referenced by format().

19.18.3.3 ftype clipper::Cell_descr::gamma_deg () const

get gamma in degrees

Returns:

The cell angle in degrees

Reimplemented in **clipper::Cell** (p. 136).

References clipper::Util::rad2d().

Referenced by format().

19.18.3.4 String clipper::Cell_descr::format () const

return formatted **String** (p. 426) representation

Returns:

A string describing the cell

Reimplemented in **clipper::Cell** (p. 136).

References a(), alpha_deg(), b(), beta_deg(), c(), and gamma_deg().

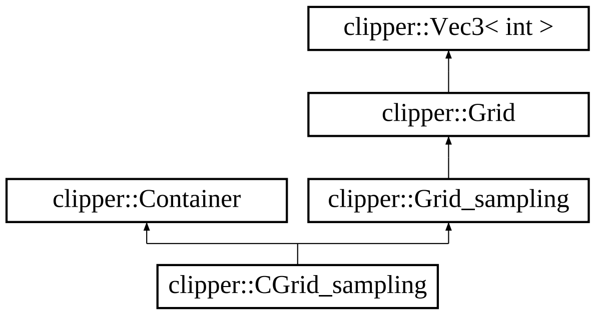
The documentation for this class was generated from the following files:

- cell.h
- cell.cpp

19.19 clipper::CGrid_sampling Class Reference

CGrid_sampling (p. 140) container.

`#include <container_types.h>`Inheritance diagram for clipper::CGrid_sampling::



Public Member Functions

- **CGrid_sampling** (const **String** name="", const **Grid_sampling** &grid=NullGrid_sampling)

  constructor: make null object or top object in tree

- **CGrid_sampling** (**Container** &parent, const **String** name="", const **ftype** rate=1.5)

  constructor: make child object

- **CGrid_sampling** (**Container** &parent, const **String** name, const **Grid_sampling** &grid)

  constructor: make child object with explicit value

- void **init** (const **Spacegroup** &spacegroup, const **Cell** &cell, const **Resolution** &resolution, const **ftype** rate)

  initialiser: from sampling rate

- void **init** (const **Grid_sampling** &grid_sampling_)

  initialiser: from ***Grid_sampling*** *(p. 255)*

- void **update** ()

  hierarchical update

19.19.1 Detailed Description

CGrid_sampling (p. 140) container. **CGrid_sampling** (p. 140): This has a name and a grid sampling It overrides the grid sampling for any objects below it.

19.19.2 Constructor & Destructor Documentation

19.19.2.1 clipper::CGrid_sampling::CGrid_sampling (const String *name* = "", const Grid_sampling & *grid* = `NullGrid_sampling`)

constructor: make null object or top object in tree The top object in a tree is initialised from a known grid.

Parameters:

name The object name.

grid The grid sampling.

19.19.2.2 clipper::CGrid_sampling::CGrid_sampling (Container & *parent*, const String *name* = "", const ftype *rate* = `1.5`)

constructor: make child object The normal form for a child object - spacegroup and cell inherited.

Parameters:

parent The objects parent.

name The object name.

rate The Shannon rate (default 1.5).

References init().

19.19.2.3 clipper::CGrid_sampling::CGrid_sampling (Container & *parent*, const String *name*, const Grid_sampling & *grid*)

constructor: make child object with explicit value This is still a child object but is initialised directly.

Parameters:

parent The objects parent.

name The object name.

grid The grid sampling.

19.19.3 Member Function Documentation

19.19.3.1 void clipper::CGrid_sampling::init (const Spacegroup & *spacegroup*, const Cell & *cell*, const Resolution & *resolution*, const ftype *rate*)

initialiser: from sampling rate The object is initialised if the appropriate parent objects are available, and children are updated.

Parameters:

spacegroup The spacegroup.

cell The cell.

resolution The resolution.

rate_ The Shannon rate (If <1 previous value is used, default 1.5).

Reimplemented from **clipper::Grid_sampling** (p. 257).

References clipper::Grid_sampling::init(), clipper::Resolution::is_null(), clipper::Cell::is_null(), clipper::Spacegroup::is_null(), and update().

Referenced by CGrid_sampling(), and update().

19.19.3.2 void clipper::CGrid_sampling::init (const Grid_sampling & *grid_sampling_*)

initialiser: from **Grid_sampling** (p. 255) The object is initialised, and children are updated.

Parameters:

grid_sampling_ The value to give to the contained object.

References update().

19.19.3.3 void clipper::CGrid_sampling::update () `[virtual]`

hierarchical update Hierarchical update. If this object is uninitialised, an attempt is made to initialise the object using information from its parents in the hierarchy. The childen of the object are then updated.

Reimplemented from **clipper::Container** (p. 169).

References init(), and clipper::Grid_sampling::is_null().

Referenced by init().

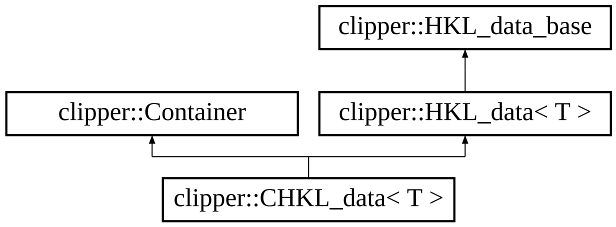
The documentation for this class was generated from the following files:

- container_types.h
- container_types.cpp

19.20 clipper::CHKL_data< T > Class Template Reference

Reflection data list container.

`#include` <`container_hkl.h`>Inheritance diagram for clipper::CHKL_data< T >::



Public Member Functions

- **CHKL_data** ()

  null constructor

- **CHKL_data** (**Container** &parent, const **String** name="")

  constructor: inherit datalist and cell

- void **init** (const **HKL_info** &hkl_info, const **Cell** &cell)

  initialiser: supply or inherit hkl list, and cell

- void **init** (const **Spacegroup** &spacegroup, const **Cell** &cell, const **HKL_sampling** &hkl_sampling)

  *initialiser: from spacegroup, cell, and **HKL_sampling** (p. 298)*

- void **update** ()

  hierarchical update

- **HKL_data**< T > & **operator=** (const **HKL_data**< T > &other)

  assignment operator: copies the data from another list

- **HKL_data**< T > & **operator=** (const T &value)

  assignment operator: assigns a single value to the whole list

19.20.1 Detailed Description

template<class T> class clipper::CHKL_data< T >

Reflection data list container. **CHKL_data** (p. 143): This is the list object containing the actual data. It must be indexed by a parent **HKL** (p. 261) list.

19.20.2 Constructor & Destructor Documentation

19.20.2.1 template<class T > clipper::CHKL_data< T >::CHKL_data (Container & *parent*, const String *name* = "") [inline]

constructor: inherit datalist and cell The object is constructed at the given location in the hierarchy. An attempt is made to initialise the object using information from its parents in the hierarchy.

Parameters:

parent An object in the hierarchy (usually the parent of the new object).

name The path from `parent` to the new object (usually just the name of the new object).

References clipper::CHKL_data< T >::init().

19.20.3 Member Function Documentation

19.20.3.1 template<class T > void clipper::CHKL_data< T >::init (const HKL_info & *hkl_info*, const Cell & *cell*) [inline, virtual]

initialiser: supply or inherit hkl list, and cell An attempt is made to initialise the object using information from the supplied parameters, or if they are Null, from its parents in the hierarchy.

Parameters:

hkl_info The reflection list object for this datalist.

cell The cell object for this datalist.

Reimplemented from **clipper::HKL_data< T >** (p. 271).

References clipper::HKL_info::cell(), clipper::Cell::is_null(), clipper::HKL_info::is_null(), and clipper::CHKL_data< T >::update().

Referenced by clipper::CHKL_data< T >::CHKL_data(), and clipper::CHKL_data< T >::update().

19.20.3.2 template<class T > void clipper::CHKL_data< T >::update () [inline, virtual]

hierarchical update Hierarchical update. If this object is uninitialised, an attempt is made to initialise the object using information from its parents in the hierarchy. The childen of the object are then updated.

The data list is also synchronized with the parent reflection list.

Reimplemented from **clipper::Container** (p. 169).

References clipper::CHKL_data< T >::init(), and clipper::HKL_data_base::is_null().

Referenced by clipper::CHKL_data< T >::init().

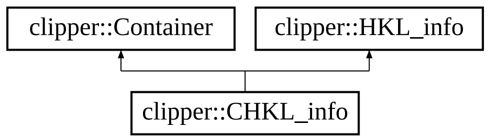
The documentation for this class was generated from the following file:

- container_hkl.h

19.21 clipper::CHKL_info Class Reference

HKL (p. 261) list and indexing object container.

`#include <container_hkl.h>`Inheritance diagram for clipper::CHKL_info::



Public Member Functions

- **CHKL_info** (const **String** name="", const **Spacegroup** &spacegroup=NullSpacegroup, const **Cell** &cell=NullCell, const **Resolution** &resolution=NullResolution, const bool &generate=false)

  constructor: make null object or top object in tree

- **CHKL_info** (**Container** &parent, const **String** name="", const bool &generate=false)

  constructor: inherit spacegroup, cell and resolution

- void **init** (const **Spacegroup** &spacegroup, const **Cell** &cell, const **Resolution** &resolution, const bool &generate=false)

  initialiser: supply or inherit spacegroup, cell and resolution

- void **generate_hkl_list** ()

  synthesize hkl list and update children

- void **update** ()

  hierarchical update

19.21.1 Detailed Description

HKL (p. 261) list and indexing object container. **CHKL_info** (p. 145): This is the reflection list object for reflection data objects to reference in order to identify their data entries.

19.21.2 Constructor & Destructor Documentation

19.21.2.1 clipper::CHKL_info::CHKL_info (const String *name* = `""`, const Spacegroup & *spacegroup* = `NullSpacegroup`, const Cell & *cell* = `NullCell`, const Resolution & *resolution* = `NullResolution`, const bool & *generate* = `false`)

constructor: make null object or top object in tree Construct and initialise as the top object in a tree.

Parameters:

spacegroup The spacegroup.

cell The cell.

resolution The resolution.

generate Generate reflection list if true.

19.21.2.2 clipper::CHKL_info::CHKL_info (Container & *parent*, const String *name* = "", const bool & *generate* = `false`)

constructor: inherit spacegroup, cell and resolution The object is constructed at the given location in the hierarchy. An attempt is made to initialise the object using information from its parents in the hierarchy.

Parameters:

parent An object in the hierarchy (usually the parent of the new object).

name The path from `parent` to the new object (usually just the name of the new object).

References init().

19.21.3 Member Function Documentation

19.21.3.1 void clipper::CHKL_info::init (const Spacegroup & *spacegroup*, const Cell & *cell*, const Resolution & *resolution*, const bool & *generate* = `false`)

initialiser: supply or inherit spacegroup, cell and resolution An attempt is made to initialise the object using information from the supplied parameters, or if they are Null, from its parents in the hierarchy.

Parameters:

spacegroup The spacegroup.

cell The cell.

resolution The resolution.

generate Generate reflection list if true.

Reimplemented from **clipper::HKL_info** (p. 287).

References clipper::HKL_info::init(), clipper::Resolution::is_null(), clipper::Cell::is_null(), clipper::Spacegroup::is_null(), and update().

Referenced by CHKL_info(), and update().

19.21.3.2 void clipper::CHKL_info::generate_hkl_list ()

synthesize hkl list and update children The reflection list is sythesized to match the given spacegroup, cell, and resolution, and a hierarchical update is triggered to update the sizes of the reflection lists for all dependent **HKL_data** (p. 267) objects.

Reimplemented from **clipper::HKL_info** (p. 288).

References update().

19.21.3.3 void clipper::CHKL_info::update () [virtual]

hierarchical update Hierarchical update. If this object is uninitialised, an attempt is made to initialise the object using information from its parents in the hierarchy. The childen of the object are then updated.

Reimplemented from **clipper::Container** (p. 169).

References init(), and clipper::HKL_info::is_null().

Referenced by generate_hkl_list(), and init().

The documentation for this class was generated from the following files:

- container_hkl.h
- container_hkl.cpp

19.22 clipper::ClipperInstance Class Reference

Public Member Functions

- const **Util** & **util** () const
- **ObjectCache**< **Spgr_cacheobj** > & **spacegroup_cache** ()
- **ObjectCache**< **Xmap_cacheobj** > & **xmap_cache** ()
- **ObjectCache**< **HKL_data_cacheobj** > & **hkl_data_cache** ()
- void **destroy** ()

    VERY DANGEROUS, DO NOT USE.

The documentation for this class was generated from the following files:

- clipper_instance.h
- clipper_instance.cpp

19.23 clipper::ClipperInstantiator Class Reference

Static Public Member Functions

- static **ClipperInstance** & **instance** ()

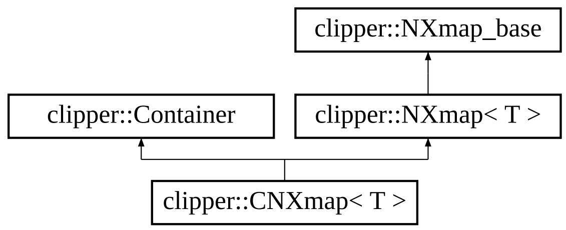
The documentation for this class was generated from the following files:

- clipper_instance.h
- clipper_instance.cpp

19.24 clipper::CNXmap< T > Class Template Reference

Non-Crystallographic map container.

`#include <container_map.h>`Inheritance diagram for clipper::CNXmap< T >::



Public Member Functions

- **CNXmap** ()

  null constructor

- **CNXmap** (**Container** &parent, const **String** name="")

  constructor:

19.24.1 Detailed Description

template<class T> class clipper::CNXmap< T >

Non-Crystallographic map container. **CNXmap** (p. 150): This is a non-crystallographic map. Since it does not exist in a crystallographic frame, it does not inherit anything.

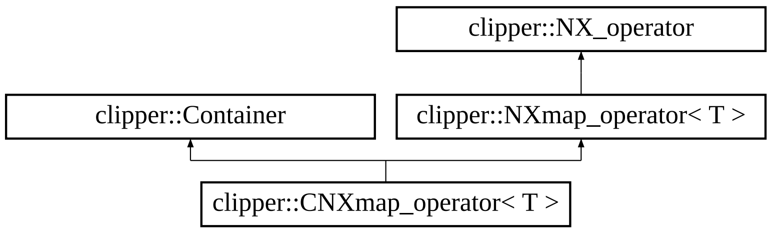
The documentation for this class was generated from the following file:

- container_map.h

19.25 clipper::CNXmap_operator< T > Class Template Reference

Non-Crystallographic map operator container.

`#include <container_map.h>`Inheritance diagram for clipper::CNXmap_operator< T >::



Public Member Functions

- **CNXmap_operator** ()
  null constructor
- **CNXmap_operator** (**Container** &parent, const **String** name="")
  constructor: do not initialise
- **CNXmap_operator** (**Container** &parent, const **String** name, const **NXmap**< T > &nxmap, const **RTop_orth** &nxop=**RTop_orth**(**RTop**<>::null()))
  constructor: inherit cell and grid
- void **init** (const **Cell** &cell, const **Grid_sampling** &grid, const **NXmap**< T > &nxmap, const **RTop_orth** &nxop)
  initialier: supply or inherit cell, grid, ***NXmap*** *(p. 357),* ***RTop_orth*** *(p. 407)*
- void **update** ()
  hierarchical update

19.25.1 Detailed Description

template<class T> class clipper::CNXmap_operator< T >

Non-Crystallographic map operator container. **CNXmap** (p. 150): This is an operator relating a non-crystallographic map into a crystallgraphic frame. It can inherit the crystallographic cell and grid sampling.

19.25.2 Constructor & Destructor Documentation

19.25.2.1 template<class T > clipper::CNXmap_operator< T >::CNXmap_operator (Container & *parent*, const String *name* = "") `[inline]`

constructor: do not initialise The object is not initialised.

Parameters:

parent The objects parent.

name The object name.

19.25.2.2 template<class T > clipper::CNXmap_operator< T >::CNXmap_operator (Container & *parent*, const String *name*, const NXmap< T > & *nxmap*, const RTop_orth & *nxop* = RTop_orth(RTop<>::null())) `[inline]`

constructor: inherit cell and grid The object is initialised if the appropriate parent objects are available, and children are updated.

Parameters:

parent The objects parent.

name The object name.

nxmap The non-crystal map object.

nxop The orth. operator mapping the **NXmap** (p. 357) into the crystal frame.

References clipper::CNXmap_operator< T >::init(), and clipper::RTop_orth::null().

19.25.3 Member Function Documentation

19.25.3.1 template<class T > void clipper::CNXmap_operator< T >::init (const Cell & *cell*, const Grid_sampling & *grid_sampling*, const NXmap< T > & *nxmap*, const RTop_orth & *nxop*) `[inline]`

initialier: supply or inherit cell, grid, **NXmap** (p. 357), **RTop_orth** (p. 407) An attempt is made to initialise the object using information from the supplied parameters, or if they are Null, from its parents in the hierarchy.

Parameters:

cell The unit cell for the crystallographic frame.

grid The grid sampling for the crystallographic frame.

nxmap The non-crystal map object.

nxop The orth. operator mapping the **NXmap** (p. 357) into the crystal frame.

Reimplemented from **clipper::NXmap_operator< T >** (p. 368).

References clipper::RTop< T >::is_null(), clipper::NXmap_base::is_null(), clipper::Grid_sampling::is_null(), clipper::Cell::is_null(), and clipper::CNXmap_operator< T >::update().

Referenced by clipper::CNXmap_operator< T >::CNXmap_operator(), and clipper::CNXmap_operator< T >::update().

19.25.3.2 template<class T > void clipper::CNXmap_operator< T >::update () `[inline, virtual]`

hierarchical update Hierarchical update. If this object is uninitialised, an attempt is made to initialise the object using information from its parents in the hierarchy. The childen of the object are then updated.

Reimplemented from **clipper::Container** (p. 169).

References clipper::CNXmap_operator< T >::init(), clipper::NX_operator::is_null(), and clipper::RTop_orth::null().

Referenced by clipper::CNXmap_operator< T >::init().

The documentation for this class was generated from the following file:

- container_map.h

19.26 clipper::Compare_grid Struct Reference

Public Member Functions

- bool **operator()** (const **Vec3**<> &c1, const **Vec3**<> &c2) const

The documentation for this struct was generated from the following file:

- spacegroup.cpp

19.27 clipper::datatypes::Compute_abcd_from_phifom< dtype > Class Template Reference

Compute from **Phi_fom** (p. 372) to **ABCD** (p. 103) (C = D = 0).

`#include <hkl_compute.h>`

Public Member Functions

- const **ABCD**< dtype > **operator()** (const **HKL_info::HKL_reference_index** &ih, const **Phi_fom**< dtype > &phifom) const

  conv op

19.27.1 Detailed Description

template<class dtype> class clipper::datatypes::Compute_abcd_from_phifom< dtype >

Compute from **Phi_fom** (p. 372) to **ABCD** (p. 103) (C = D = 0).

The documentation for this class was generated from the following files:

- hkl_compute.h
- hkl_compute.cpp

19.28 clipper::datatypes::Compute_add_abcd< dtype > Class Template Reference

Add two **ABCD** (p. 103) datalists.

`#include <hkl_compute.h>`

Public Member Functions

- const **ABCD**< dtype > **operator()** (const **HKL_info::HKL_reference_index** &ih, const **ABCD**< dtype > &abcd1, const **ABCD**< dtype > &abcd2) const

  conv op

19.28.1 Detailed Description

template<class dtype> class clipper::datatypes::Compute_add_abcd< dtype >

Add two **ABCD** (p. 103) datalists.

The documentation for this class was generated from the following files:

- hkl_compute.h
- hkl_compute.cpp

19.29 clipper::datatypes::Compute_add_fphi< dtype > Class Template Reference

Add two **F_phi** (p. 208) datalists.

`#include <hkl_compute.h>`

Public Member Functions

- const **F_phi**< dtype > **operator()** (const **HKL_info::HKL_reference_index** &ih, const **F_phi**< dtype > &fphi1, const **F_phi**< dtype > &fphi2) const

    conv op

19.29.1 Detailed Description

template<class dtype> class clipper::datatypes::Compute_add_fphi< dtype >

Add two **F_phi** (p. 208) datalists.

The documentation for this class was generated from the following files:

- hkl_compute.h
- hkl_compute.cpp

19.30 clipper::datatypes::Compute_diff_fsigf_from_-fsigfano< dtype > Class Template Reference

Compute from F_sigF_anom to **F_sigF** (p. 210) (difference structure factor).

`#include <hkl_compute.h>`

Public Member Functions

- const **F_sigF**< dtype > **operator()** (const **HKL_info::HKL_reference_index** &ih, const **F_sigF_ano**< dtype > &fsigfano) const

  conv op

19.30.1 Detailed Description

template<class dtype> class clipper::datatypes::Compute_diff_fsigf_from_-fsigfano< dtype >

Compute from F_sigF_anom to **F_sigF** (p. 210) (difference structure factor).

The documentation for this class was generated from the following files:

- hkl_compute.h
- hkl_compute.cpp

19.31 clipper::datatypes::Compute_EsigE_from_FsigF< dtype > Class Template Reference

Compute from **F_sigF** (p. 210) to **E_sigE** (p. 202).

`#include <hkl_compute.h>`

Public Member Functions

- const **E_sigE**< dtype > **operator()** (const **HKL_info::HKL_reference_index** &ih, const **F_sigF**< dtype > &fsigf) const

  conv op

19.31.1 Detailed Description

template<class dtype> class clipper::datatypes::Compute_EsigE_from_FsigF< dtype >

Compute from **F_sigF** (p. 210) to **E_sigE** (p. 202).

The documentation for this class was generated from the following files:

- hkl_compute.h
- hkl_compute.cpp

19.32 clipper::datatypes::Compute_fphi_from_fsigf_-phifom< dtype > Class Template Reference

Compute from F_sigF+Phi_fom to **F_phi** (p. 208).

```
#include <hkl_compute.h>
```

Public Member Functions

- const **F_phi**< dtype > **operator()** (const **HKL_info::HKL_reference_index** &ih, const **F_sigF**< dtype > &fsigf, const **Phi_fom**< dtype > &phifom) const

  conv op

19.32.1 Detailed Description

template<class dtype> class clipper::datatypes::Compute_fphi_from_fsigf_-phifom< dtype >

Compute from F_sigF+Phi_fom to **F_phi** (p. 208).

The documentation for this class was generated from the following files:

- hkl_compute.h
- hkl_compute.cpp

19.33 clipper::datatypes::Compute_FsigF< dtype, T > Class Template Reference

Compute from **F_sigF** (p. 210) to **F_sigF** (p. 210).

`#include <hkl_compute.h>`

Public Member Functions

- const **F_sigF**< dtype > **operator()** (const **HKL_info::HKL_reference_index** &ih, const T &fsigf) const

  conv op

19.33.1 Detailed Description

template<class dtype, class T> class clipper::datatypes::Compute_FsigF< dtype, T >

Compute from **F_sigF** (p. 210) to **F_sigF** (p. 210). Use this to get **F_sigF** (p. 210) from **F_sigF** (p. 210) of a different precision or from **F_sigF_ano** (p. 212).

The documentation for this class was generated from the following files:

- hkl_compute.h
- hkl_compute.cpp

19.34 clipper::datatypes::Compute_mean_fsigf_from_-fsigfano< dtype > Class Template Reference

Compute from F_sigF_anom to **F_sigF** (p. 210) (mean structure factor).

`#include <hkl_compute.h>`

Public Member Functions

- const **F_sigF**< dtype > **operator()** (const **HKL_info::HKL_reference_index** &ih, const **F_sigF_ano**< dtype > &fsigfano) const

  conv op

19.34.1 Detailed Description

template<class dtype> class clipper::datatypes::Compute_mean_fsigf_from_-fsigfano< dtype >

Compute from F_sigF_anom to **F_sigF** (p. 210) (mean structure factor).

The documentation for this class was generated from the following files:

- hkl_compute.h
- hkl_compute.cpp

19.35 clipper::datatypes::Compute_neg_fphi< dtype > Class Template Reference

Negate **F_phi** (p. 208) (i.e. advance phase by pi).

`#include <hkl_compute.h>`

Public Member Functions

- const **F_phi**< dtype > **operator()** (const **HKL_info::HKL_reference_index** &ih, const **F_phi**< dtype > &fphi1) const

    conv op

19.35.1 Detailed Description

template<class dtype> class clipper::datatypes::Compute_neg_fphi< dtype >

Negate **F_phi** (p. 208) (i.e. advance phase by pi).

The documentation for this class was generated from the following files:

- hkl_compute.h
- hkl_compute.cpp

19.36 clipper::datatypes::Compute_phifom_from_abcd< dtype > Class Template Reference

Compute from **ABCD** (p. 103) to **Phi_fom** (p. 372) by phase integration (loses bimodality).

`#include <hkl_compute.h>`

Public Member Functions

- const **Phi_fom**< dtype > **operator()** (const **HKL_info::HKL_reference_index** &ih, const **ABCD**< dtype > &abcd) const

  conv op

19.36.1 Detailed Description

template<class dtype> class clipper::datatypes::Compute_phifom_from_abcd< dtype >

Compute from **ABCD** (p. 103) to **Phi_fom** (p. 372) by phase integration (loses bimodality).

The documentation for this class was generated from the following files:

- hkl_compute.h
- hkl_compute.cpp

19.37 clipper::datatypes::Compute_scale_u< T > Class Template Reference

`#include <hkl_compute.h>`

Public Member Functions

- **Compute_scale_u** (const **ftype** &s, const **ftype** &u)
  constructor: takes scale, U value
- const T **operator()** (const **HKL_info::HKL_reference_index** &ih, T data) const
  conv op

19.37.1 Detailed Description

template<class T> class clipper::datatypes::Compute_scale_u< T >

Deprecated

Apply scale and U to any scalable datatype

19.37.2 Constructor & Destructor Documentation

19.37.2.1 template<class T > clipper::datatypes::Compute_scale_u< T >::Compute_scale_u (const ftype & *s*, const ftype & *u*) [inline]

constructor: takes scale, U value DEPRECATED: This operator applies the scale factor against intensities and the U value against magnitudes, which is counterintuitive. **Compute_scale_u_iso** (p. 167) is more intutive.

Construct conversion operator to scale list according to the formula I_new = s.exp(b.|h|^2/2) I_old or F_new^2 = s.exp(b.|h|^2/2) F_old^2 where |h| = invresolsq.

Parameters:

s The intensity scale factor.

u The temperature factor (U-value).

References clipper::Util::twopi2().

The documentation for this class was generated from the following files:

- hkl_compute.h
- hkl_compute.cpp

19.38 clipper::datatypes::Compute_scale_u_aniso< T > Class Template Reference

Apply scale and U to any scalable datatype.

`#include <hkl_compute.h>`

Public Member Functions

- **Compute_scale_u_aniso** (const **ftype** &s, const **U_aniso_orth** &u)
  constructor: takes scale, U value
- const T **operator()** (const **HKL_info::HKL_reference_index** &ih, T data) const
  conv op

19.38.1 Detailed Description

template<class T> class clipper::datatypes::Compute_scale_u_aniso< T >

Apply scale and U to any scalable datatype.

19.38.2 Constructor & Destructor Documentation

19.38.2.1 template<class T > clipper::datatypes::Compute_scale_u_aniso< T >::Compute_scale_u_aniso (const ftype & *s*, const U_aniso_orth & *u*) `[inline]`

constructor: takes scale, U value Construct conversion operator to scale list according to the formula I_new = s^2 .exp(4 ^2 h^T U h) I_old or F_new = s.exp(2 ^2 h^T U h) F_old

Parameters:

s The scale factor.

u The temperature factor (U-value).

References clipper::Util::twopi2().

The documentation for this class was generated from the following files:

- hkl_compute.h
- hkl_compute.cpp

19.39 clipper::datatypes::Compute_scale_u_iso< T > Class Template Reference

Apply scale and U to any scalable datatype.

`#include <hkl_compute.h>`

Public Member Functions

- **Compute_scale_u_iso** (const **ftype** &s, const **ftype** &u)
  constructor: takes scale, U value
- const T **operator()** (const **HKL_info::HKL_reference_index** &ih, T data) const
  conv op

19.39.1 Detailed Description

template<class T> class clipper::datatypes::Compute_scale_u_iso< T >

Apply scale and U to any scalable datatype.

19.39.2 Constructor & Destructor Documentation

19.39.2.1 template<class T > clipper::datatypes::Compute_scale_u_iso< T >::Compute_scale_u_iso (const ftype & *s*, const ftype & *u*) [inline]

constructor: takes scale, U value Construct conversion operator to scale list according to the formula I_new = s^2 .exp(4 ^2 u.|h|^2) I_old or F_new = s.exp(2 ^2 u.|h|^2) F_old where |h|^2 = invresolsq.

Parameters:

s The scale factor.

u The temperature factor (U-value).

References clipper::Util::twopi2().

The documentation for this class was generated from the following files:

- hkl_compute.h
- hkl_compute.cpp

19.40 clipper::datatypes::Compute_sub_fphi< dtype > Class Template Reference

Subtract two **F_phi** (p. 208) datalists.

`#include <hkl_compute.h>`

Public Member Functions

- const **F_phi**< dtype > **operator()** (const **HKL_info::HKL_reference_index** &ih, const **F_phi**< dtype > &fphi1, const **F_phi**< dtype > &fphi2) const

  conv op

19.40.1 Detailed Description

template<class dtype> class clipper::datatypes::Compute_sub_fphi< dtype >

Subtract two **F_phi** (p. 208) datalists.

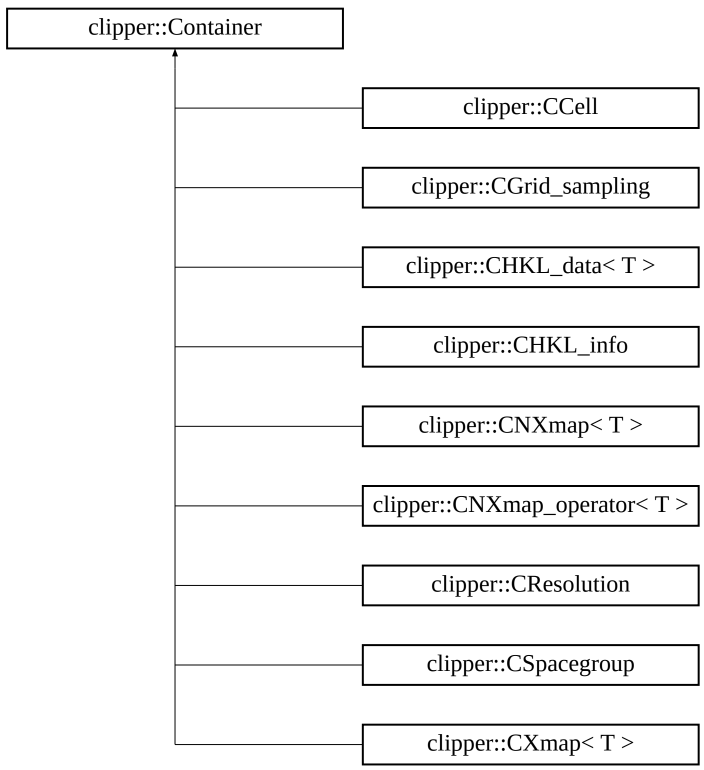
The documentation for this class was generated from the following files:

- hkl_compute.h
- hkl_compute.cpp

19.41 clipper::Container Class Reference

Definition for a generic container Object.

`#include <container.h>`Inheritance diagram for clipper::Container::



Public Member Functions

- **Container** (const **String** name="")

  constructor: make null object or top object in a tree

- **Container** (**Container** &parent, const **String** &path)

  constructor: from any other member and a relative path

- virtual void **update** ()

  update: hierarchical content update function

- **String path** () const

  get the path of this tree object

- **String name** () const

  get the name of this tree object

- void **set_name** (const **String** &name)

  set the name of this tree object

- bool **is_destroyed_with_parent** () const

  is this object to be destroyed when parent is destroyed?

- void **set_destroyed_with_parent** (const bool d=true)

  set this object to be destroyed when parent is destroyed

- void **move** (const **String** &path)

  'move' method moves this object to somewhere else in the hierarchy

- bool **has_parent** () const

  test if this object has a parent

- const **Container** & **parent** () const

  get the parent of this object

- **Container** & **parent** ()

  get the parent of this object

- int **num_children** () const

  return number of children

- const **Container** & **child** (const int &i) const

  get the i'th child of this object

- **Container** & **child** (const int &i)

  get the i'th child of this object

- const **Container** & **ultimate_parent** () const

  get the ultimate parent of this object - the top of the tree

- **Container** & **ultimate_parent** ()

  get the ultimate parent of this object - the top of the tree

- **Container** * **parent_ptr** ()

  get the parent of this object (NULL on fail)

- template<class T >
  T * **parent_of_type_ptr** ()

  search up the tree for a parent of the specified type (NULL on fail)

- **Container** * **find_path_ptr** (const **String** &path)

  find an object using a directory-like path (NULL on fail)

- virtual ~**Container** ()

  destructor: virtual

- void **debug** ()

19.41.1 Detailed Description

Definition for a generic container Object. **Container** (p. 169) is a definition for a generic container object with a name, parents, and children. Any object that wants to be part of the tree simply subclasses this class. The class also implements search and move objects. The tree is navigate using unix-like pathnames. A recursive update method can be overridden to update content after altering the hierarchy.

The top container in a tree is created by passing **Container()** (p. 169) as its parent.

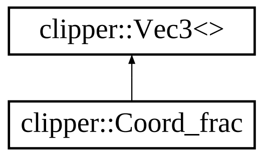
The documentation for this class was generated from the following files:

- container.h
- container.cpp

19.42 clipper::Coord_frac Class Reference

fractional (cell) coordinates

`#include <coords.h>`Inheritance diagram for clipper::Coord_frac::



Public Member Functions

- **Coord_frac** ()
  null constructor
- **Coord_frac** (const **Vec3**<> &v)
  constructor: copy/convert
- **Coord_frac** (const **ftype** &u, const **ftype** &v, const **ftype** &w)
  constructor: from u,v,w
- const **ftype** & **u** () const
  get u
- const **ftype** & **v** () const
  get v
- const **ftype** & **w** () const
  get w
- **ftype lengthsq** (const **Cell** &cell) const
  return square of length of vector in Angstroms
- **Coord_orth coord_orth** (const **Cell** &cell) const
  fractional-orthogonal coordinate conversion
- **Coord_map coord_map** (const **Grid** &g) const
  fractional-grid coordinate conversion
- **Coord_grid coord_grid** (const **Grid** &g) const
  fractional-grid coordinate conversion
- **Coord_frac transform** (const **RTop_frac** &op) const
  return transformed coordinate
- **Coord_frac lattice_copy_zero** () const
  return lattice copy nearest origin

- **Coord_frac lattice_copy_unit** () const
  return lattice copy in unit box (0...1,0...1,0...1)
- **Coord_frac lattice_copy_near** (const **Coord_frac** &n) const
  return lattice copy near the specified coordinate
- **Coord_frac symmetry_copy_near** (const **Spacegroup** &spgr, const **Cell** &cell, const **Coord_frac** &n) const
  return symmetry copy near the specified coordinate
- **String format** () const
  *return formatted **String** (p. 426) representation*

Friends

- **Coord_frac operator-** (const **Coord_frac** &u1)
- **Coord_frac operator+** (const **Coord_frac** &u1, const **Coord_frac** &u2)
- **Coord_frac operator-** (const **Coord_frac** &u1, const **Coord_frac** &u2)
- **Coord_frac operator*** (const **ftype** &s, const **Coord_frac** &u1)
- **Coord_frac operator*** (const **RTop_frac** &op, const **Coord_frac** &x1)

19.42.1 Detailed Description

fractional (cell) coordinates

19.42.2 Member Function Documentation

19.42.2.1 ftype clipper::Coord_frac::lengthsq (const Cell & *cell*) const `[inline]`

return square of length of vector in Angstroms

Returns:

The squared length in Angstroms squared

References clipper::Metric_tensor::lengthsq(), and clipper::Cell::metric_real().

Referenced by symmetry_copy_near().

19.42.2.2 Coord_orth clipper::Coord_frac::coord_orth (const Cell & *cell*) const `[inline]`

fractional-orthogonal coordinate conversion

Parameters:

cell The cell concerned

Returns:

The transformed coordinate.

References clipper::Cell::matrix_orth().

19.42.2.3 Coord_map clipper::Coord_frac::coord_map (const Grid & *g*) const [inline]

fractional-grid coordinate conversion

Parameters:

g The grid concerned

Returns:

The transformed coordinate.

References clipper::Grid::nu(), clipper::Grid::nv(), clipper::Grid::nw(), u(), v(), and w().

Referenced by clipper::Xmap< T >::interp(), clipper::Xmap< T >::interp_curv(), and clipper::Xmap< T >::interp_grad().

19.42.2.4 Coord_grid clipper::Coord_frac::coord_grid (const Grid & *g*) const [inline]

fractional-grid coordinate conversion

Parameters:

g The grid concerned

Returns:

The transformed coordinate.

References clipper::Util::intr(), clipper::Grid::nu(), clipper::Grid::nv(), clipper::Grid::nw(), u(), v(), and w().

19.42.2.5 String clipper::Coord_frac::format () const

return formatted **String** (p. 426) representation

Returns:

The formatted text string

Reimplemented from **clipper::Vec3**<> (p. 461).

References u(), v(), and w().

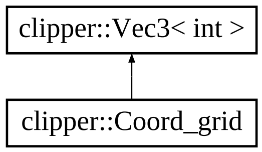
The documentation for this class was generated from the following files:

- coords.h
- coords.cpp

19.43 clipper::Coord_grid Class Reference

Grid (p. 250) coordinate.

`#include <coords.h>`Inheritance diagram for clipper::Coord_grid::



Public Member Functions

- **Coord_grid** ()
  null constructor
- **Coord_grid** (const **Vec3**< int > v)
  constructor: copy/convert
- **Coord_grid** (const int &u, const int &v, const int &w)
  constructor: from u,v,w
- **Coord_grid** (const **Grid** &g, const int &index)
  constructor: from a grid and an index in that grid
- const int & **u** () const
  get u
- const int & **v** () const
  get v
- const int & **w** () const
  get w
- int & **u** ()
  set u
- int & **v** ()
  set v
- int & **w** ()
  set w
- **Coord_map coord_map** () const
  *convert to **Coord_map** (p. 180)*
- **Coord_frac coord_frac** (const **Grid_sampling** &g) const
  *convert to **Coord_frac** (p. 172) using given **Grid_sampling** (p. 255)*

- **Coord_grid transform** (const **Isymop** &op) const
  return transformed coordinate
- **Coord_grid unit** (const **Grid_sampling** &g) const
  reduce to unit box: (0..nu-1, 0..nv-1, 0..nw-1)
- const **Coord_grid** & **next** (const **Grid** &g)
  increment in storage order (see ***index()*** *(p. 179))*
- const **Coord_grid** & **next** (const **Grid_range** &g)
  increment in storage order (see ***index()*** *(p. 179))*
- bool **last** (const **Grid** &g) const
  test if done in storage order (see ***index()*** *(p. 179))*
- bool **last** (const **Grid_range** &g) const
  test if done in storage order (see ***index()*** *(p. 179))*
- int **index** (const **Grid** &g) const
  grid indexing operator
- void **deindex** (const **Grid** &g, const int &index)
  grid deindexing operator
- **String format** () const
  return formatted ***String*** *(p. 426) representation*

Friends

- **Coord_grid operator-** (const **Coord_grid** &r1)
  Vector negation operator.
- **Coord_grid operator+** (const **Coord_grid** &r1, const **Coord_grid** &r2)
  Vector addition operator.
- **Coord_grid operator-** (const **Coord_grid** &r1, const **Coord_grid** &r2)
  Vector subtraction operator.
- **Coord_grid operator∗** (const int &s, const **Coord_grid** &r1)
  Vector scaling operator.
- int **operator==** (const **Coord_grid** &r1, const **Coord_grid** &r2)
  Vector equality (for floating point types see ***equals()*** *(p. 461)).*
- int **operator!=** (const **Coord_grid** &r1, const **Coord_grid** &r2)
  Vector inequality (for floating point types see ***equals()*** *(p. 461)).*
- **Coord_grid operator∗** (const **Isymop** &op, const **Coord_grid** &r1)

19.43.1 Detailed Description

Grid (p. 250) coordinate.

19.43.2 Member Function Documentation

19.43.2.1 Coord_map clipper::Coord_grid::coord_map () const [inline]

convert to **Coord_map** (p. 180)

Returns:

The non-integer coordinate.

Referenced by clipper::Xmap_base::Map_reference_coord::coord_orth(), clipper::Xmap_-base::Map_reference_index::coord_orth(), clipper::NX_operator::nxmap_data(), and clipper::NX_operator::xmap_data().

19.43.2.2 Coord_frac clipper::Coord_grid::coord_frac (const Grid_sampling & *g*) const [inline]

convert to **Coord_frac** (p. 172) using given **Grid_sampling** (p. 255) Fractional coordinate is not normalised onto range 0..1

Parameters:

g The grid concerned

Returns:

The fractional coordinate

References clipper::Grid_sampling::nu(), clipper::Grid_sampling::nv(), clipper::Grid_-sampling::nw(), u(), v(), and w().

19.43.2.3 Coord_grid clipper::Coord_grid::unit (const Grid_sampling & *g*) const [inline]

reduce to unit box: (0..nu-1, 0..nv-1, 0..nw-1)

Parameters:

g The grid concerned

Returns:

The transformed coordinate.

References Coord_grid(), clipper::Util::mod(), clipper::Grid_sampling::nu(), clipper::Grid_-sampling::nv(), clipper::Grid_sampling::nw(), u(), v(), and w().

Referenced by clipper::FFTmap::get_real_data(), clipper::Xmap_base::multiplicity(), clipper::FFTmap_sparse_p1_hx::set_hkl(), clipper::FFTmap_p1::set_hkl(), clipper::FFTmap::set_real_data(), clipper::Xmap_base::to_map_unit(), and clipper::Xmap_-cacheobj::Xmap_cacheobj().

19.43.2.4 const Coord_grid & clipper::Coord_grid::next (const Grid & *g*) [inline]

increment in storage order (see **index()** (p. 179)) guaranteed to increment index(g) by 1

The grid coordinate is incremented efficiently in a manner which is exaclty equivalent to increasing **index()** (p. 179) by 1 in a zero based grid.

Parameters:

g The grid with which this increment is synchronised.

References clipper::Grid::nv(), clipper::Grid::nw(), u(), v(), and w().

Referenced by clipper::Xmap_cacheobj::Xmap_cacheobj().

19.43.2.5 const Coord_grid & clipper::Coord_grid::next (const Grid_range & *g*) [inline]

increment in storage order (see **index()** (p. 179)) guaranteed to increment index(g) by 1

The grid coordinate is incremented efficiently in a manner which is exaclty equivalent to increasing **index()** (p. 179) by 1 in a non-zero based grid.

Parameters:

g The grid with which this increment is synchronised.

References clipper::Grid_range::max(), clipper::Grid_range::min(), u(), v(), and w().

19.43.2.6 bool clipper::Coord_grid::last (const Grid & *g*) const [inline]

test if done in storage order (see **index()** (p. 179)) Test whether this coordinate has been incremented using **next()** (p. 178) beyond the end of the specified zero based grid.

Parameters:

g The grid concerned.

References clipper::Grid::nu(), and u().

Referenced by clipper::Xmap_cacheobj::Xmap_cacheobj().

19.43.2.7 bool clipper::Coord_grid::last (const Grid_range & *g*) const [inline]

test if done in storage order (see **index()** (p. 179)) Test whether this coordinate has been incremented using **next()** (p. 178) beyond the end of the specified non-zero based grid.

Parameters:

g The grid concerned.

References clipper::Grid_range::max(), and u().

19.43.2.8 int clipper::Coord_grid::index (const Grid & *g*) const [inline]

grid indexing operator Return the index in a 1-d array corresponding to this coordinate for a zero based grid.

Parameters:

g The grid concerned.

Returns:

The corresponding index.

References clipper::Grid::nv(), clipper::Grid::nw(), u(), v(), and w().

Referenced by clipper::Grid::index().

19.43.2.9 void clipper::Coord_grid::deindex (const Grid & *g*, const int & *index*) [inline]

grid deindexing operator Return the coordinate corresponding to a given index in a zero based grid.

Parameters:

g The grid concerned.

Returns:

The corresponding coordinate.

References clipper::Grid::nv(), clipper::Grid::nw(), u(), v(), and w().

Referenced by Coord_grid().

19.43.2.10 String clipper::Coord_grid::format () const

return formatted **String** (p. 426) representation

Returns:

The formatted text string

Reimplemented from **clipper::Vec3< int >** (p. 461).

References u(), v(), and w().

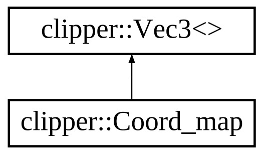
The documentation for this class was generated from the following files:

- coords.h
- coords.cpp

19.44 clipper::Coord_map Class Reference

map coordinate: this is like **Coord_grid** (p. 175), but non-integer

`#include <coords.h>`Inheritance diagram for clipper::Coord_map::



Public Member Functions

- **Coord_map** ()
  null constructor
- **Coord_map** (const **Vec3**<> &v)
  constructor: copy/convert
- **Coord_map** (const **Coord_grid** &c)
  constructor: from ***Coord_grid*** *(p. 175)*
- **Coord_map** (const **ftype** &u, const **ftype** &v, const **ftype** &w)
  constructor: from u,v,w
- **Coord_frac coord_frac** (const **Grid** &g) const
  grid-fractional coordinate conversion
- **Coord_grid coord_grid** () const
  return integer ***Coord_grid*** *(p. 175) nearest this coordinate*
- **Coord_grid floor** () const
  return integer ***Coord_grid*** *(p. 175) below this coordinate*
- **Coord_grid ceil** () const
  return integer ***Coord_grid*** *(p. 175) above this coordinate*
- const **ftype** & **u** () const
  get u
- const **ftype** & **v** () const
  get v
- const **ftype** & **w** () const
  get w
- **String format** () const
  return formatted ***String*** *(p. 426) representation*

Friends

- **Coord_map operator-** (const **Coord_map** &u1)
- **Coord_map operator+** (const **Coord_map** &u1, const **Coord_map** &u2)
- **Coord_map operator-** (const **Coord_map** &u1, const **Coord_map** &u2)
- **Coord_map operator*** (const **ftype** &s, const **Coord_map** &u1)

19.44.1 Detailed Description

map coordinate: this is like **Coord_grid** (p. 175), but non-integer

19.44.2 Member Function Documentation

19.44.2.1 Coord_frac clipper::Coord_map::coord_frac (const Grid & *g*) const [inline]

grid-fractional coordinate conversion

Parameters:

g The grid concerned

Returns:

The transformed coordinate.

References clipper::Grid::nu(), clipper::Grid::nv(), clipper::Grid::nw(), u(), v(), and w().

19.44.2.2 String clipper::Coord_map::format () const

return formatted **String** (p. 426) representation

Returns:

The formatted text string

Reimplemented from **clipper::Vec3<>** (p. 461).

References u(), v(), and w().

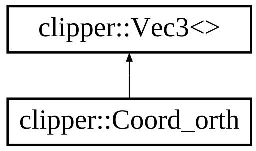
The documentation for this class was generated from the following files:

- coords.h
- coords.cpp

19.45 clipper::Coord_orth Class Reference

orthogonal (Angstrom) coordinates

`#include <coords.h>`Inheritance diagram for clipper::Coord_orth::



Public Member Functions

- **Coord_orth** ()
  null constructor
- **Coord_orth** (const **Vec3**<> &v)
  constructor: copy/convert
- **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** (p. 426) representation*

Static Public Member Functions

- static **ftype length** (const **Coord_orth** &x1, const **Coord_orth** &x2)

    Return length of vector between two coord orths.

- static **ftype angle** (const **Coord_orth** &x1, const **Coord_orth** &x2, const **Coord_orth** &x3)

    Return angle between three coord orths.

- static **ftype torsion** (const **Coord_orth** &x1, const **Coord_orth** &x2, const **Coord_orth** &x3, const **Coord_orth** &x4)

    Return torsion between four coord orths.

Friends

- **Coord_orth operator-** (const **Coord_orth** &x1)
- **Coord_orth operator+** (const **Coord_orth** &x1, const **Coord_orth** &x2)
- **Coord_orth operator-** (const **Coord_orth** &x1, const **Coord_orth** &x2)
- **Coord_orth operator*** (const **ftype** &s, const **Coord_orth** &x1)
- **Coord_orth operator*** (const **RTop_orth** &op, const **Coord_orth** &x1)

19.45.1 Detailed Description

orthogonal (Angstrom) coordinates

19.45.2 Constructor & Destructor Documentation

19.45.2.1 clipper::Coord_orth::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 The coordinate is calculated which extends the sequence of coordinates x1, x2, x3 with the specified distance to x3, angle to x2,x3, and torsion to x1,x2,x3.

Parameters:

x1 First coordinate.

x2 Second coordinate.

x3 Third coordinate.

length x3-new bond length in Angstroms.

angle x2-x3-new opening angle in Radians.

torsion x1-x2-x3-new torsion angle in Radians.

References clipper::Vec3<>::unit().

19.45.3 Member Function Documentation

19.45.3.1 ftype clipper::Coord_orth::lengthsq () const [inline]

return square of length of vector in Angstroms

Returns:

The squared length in Angstroms squared

References x(), y(), and z().

Referenced by length(), clipper::AtomShapeFn::rho(), clipper::AtomShapeFn::rho_curv(), and clipper::AtomShapeFn::rho_grad().

19.45.3.2 Coord_frac clipper::Coord_orth::coord_frac (const Cell & *cell*) const [inline]

orthogonal-fractional coordinate conversion

Parameters:

cell The cell concerned

Returns:

The transformed coordinate.

References clipper::Cell::matrix_frac().

19.45.3.3 String clipper::Coord_orth::format () const

return formatted **String** (p. 426) representation

Returns:

The formatted text string

Reimplemented from **clipper::Vec3**<> (p. 461).

References x(), y(), and z().

19.45.3.4 ftype clipper::Coord_orth::length (const Coord_orth & *x1*, const Coord_orth & *x2*) [static]

Return length of vector between two coord orths.

Returns:

The bond length x1-x2 in Angstroms.

References lengthsq().

19.45.3.5 ftype clipper::Coord_orth::angle (const Coord_orth & *x1*, const Coord_orth & *x2*, const Coord_orth & *x3*) `[static]`

Return angle between three coord orths.

Returns:

The bond angle x1-x2-x3 in Radians.

References clipper::Vec3<>::unit().

19.45.3.6 ftype clipper::Coord_orth::torsion (const Coord_orth & *x1*, const Coord_orth & *x2*, const Coord_orth & *x3*, const Coord_orth & *x4*) `[static]`

Return torsion between four coord orths.

Returns:

The bond torsion x1-x2-x3-x4 in Radians.

References clipper::Vec3<>::unit().

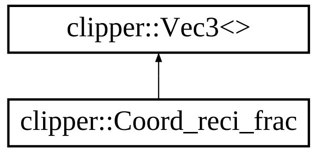
The documentation for this class was generated from the following files:

- coords.h
- coords.cpp

19.46 clipper::Coord_reci_frac Class Reference

fractional reciprocal coordinate (i.e. non-integer hkl)

`#include <coords.h>`Inheritance diagram for clipper::Coord_reci_frac::



Public Member Functions

- **Coord_reci_frac** ()
  null constructor
- **Coord_reci_frac** (const **Vec3**<> &v)
  constructor: copy/convert
- **Coord_reci_frac** (const **ftype** &us, const **ftype** &vs, const **ftype** &ws)
  constructor: from u,v,w
- **Coord_reci_frac** (const **HKL** &hkl)
  *constructor: from **HKL** (p. 261)*
- **HKL hkl** () const
  *round to **HKL** (p. 261)*
- **ftype invresolsq** (const **Cell** &cell) const
  return inverse resolution squared for this reflection in given cell
- const **ftype** & **us** () const
  *get u**
- const **ftype** & **vs** () const
  *get v**
- const **ftype** & **ws** () const
  *get w**
- **Coord_reci_orth coord_reci_orth** (const **Cell** &cell) const
  fractional-orthogonal reciprocal space coordinate conversion
- **Coord_reci_frac transform** (const **RTop_frac** &op) const
  return transformed coordinate
- **String format** () const
  *return formatted **String** (p. 426) representation*

19.46.1 Detailed Description

fractional reciprocal coordinate (i.e. non-integer hkl)

19.46.2 Member Function Documentation

19.46.2.1 ftype clipper::Coord_reci_frac::invresolsq (const Cell & *cell*) const `[inline]`

return inverse resolution squared for this reflection in given cell

Parameters:

cell The cell concerned

Returns:

The inverse resolution squared.

References clipper::Metric_tensor::lengthsq(), and clipper::Cell::metric_reci().

19.46.2.2 Coord_reci_orth clipper::Coord_reci_frac::coord_reci_orth (const Cell & *cell*) const `[inline]`

fractional-orthogonal reciprocal space coordinate conversion

Parameters:

cell The cell concerned

Returns:

The transformed coordinate.

References clipper::Cell::matrix_frac().

Referenced by clipper::HKL::coord_reci_orth().

19.46.2.3 String clipper::Coord_reci_frac::format () const

return formatted **String** (p. 426) representation

Returns:

The formatted text string

Reimplemented from **clipper::Vec3<>** (p. 461).

References us(), vs(), and ws().

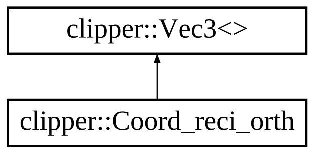
The documentation for this class was generated from the following files:

- coords.h
- coords.cpp

19.47 clipper::Coord_reci_orth Class Reference

orthogonal reciprocal coordinate (length of which is invresolsq)

`#include <coords.h>`Inheritance diagram for clipper::Coord_reci_orth::



Public Member Functions

- **Coord_reci_orth** ()
  null constructor
- **Coord_reci_orth** (const **Vec3**<> &v)
  constructor: copy/convert
- **Coord_reci_orth** (const **ftype** &xs, const **ftype** &ys, const **ftype** &zs)
  constructor: from x,y*,z**
- const **ftype** & **xs** () const
  *get x**
- const **ftype** & **ys** () const
  *get y**
- const **ftype** & **zs** () const
  *get z**
- **ftype invresolsq** () const
  return inverse resolution squared for this coord
- **Coord_reci_frac coord_reci_frac** (const **Cell** &cell) const
  orthogonal-fractional reciprocal space coordinate conversion
- **Coord_reci_orth transform** (const **RTop_orth** &op) const
  return transformed coordinate
- **String format** () const
  *return formatted **String** (p. 426) representation*

19.47.1 Detailed Description

orthogonal reciprocal coordinate (length of which is invresolsq)

19.47.2 Member Function Documentation

19.47.2.1 ftype clipper::Coord_reci_orth::invresolsq () const `[inline]`

return inverse resolution squared for this coord

Returns:

The inverse resolution squared.

References xs(), ys(), and zs().

Referenced by clipper::AtomShapeFn::f().

19.47.2.2 Coord_reci_frac clipper::Coord_reci_orth::coord_reci_frac (const Cell & *cell*) const `[inline]`

orthogonal-fractional reciprocal space coordinate conversion

Parameters:

cell The cell concerned

Returns:

The transformed coordinate.

References clipper::Cell::matrix_orth().

19.47.2.3 String clipper::Coord_reci_orth::format () const

return formatted **String** (p. 426) representation

Returns:

The formatted text string

Reimplemented from **clipper::Vec3**<> (p. 461).

References xs(), ys(), and zs().

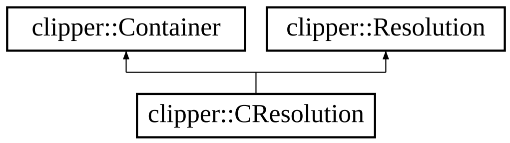
The documentation for this class was generated from the following files:

- coords.h
- coords.cpp

19.48 clipper::CResolution Class Reference

Resolution (p. 389) container.

`#include <container_types.h>`Inheritance diagram for clipper::CResolution::



Public Member Functions

- **CResolution** (const **String** name="", const **Resolution** &resol=NullResolution)

  constructor: make null object or top object in tree

- **CResolution** (**Container** &parent, const **String** name="", const **Resolution** &resol=NullResolution)

  constructor: make child object

- void **init** (const **Resolution** &resolution_)

  *initialiser: from **Resolution** (p. 389)*

19.48.1 Detailed Description

Resolution (p. 389) container. **CResolution** (p. 190): This has a name and a resolution. It overrides the base resolution for any objects below it.

19.48.2 Member Function Documentation

19.48.2.1 void clipper::CResolution::init (const Resolution & *resolution_*)

initialiser: from **Resolution** (p. 389) The object is initialised, and children are updated.

Parameters:

resolution_ The value to give to the contained object.

References clipper::Container::update().

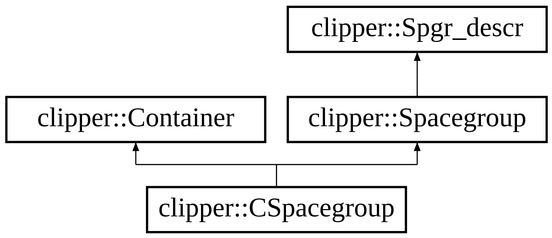
The documentation for this class was generated from the following files:

- container_types.h
- container_types.cpp

19.49 clipper::CSpacegroup Class Reference

Spacegroup (p. 413) container.

`#include <container_types.h>`Inheritance diagram for clipper::CSpacegroup::



Public Member Functions

- **CSpacegroup** (const **String** name="", const **Spacegroup** &spgr=NullSpacegroup)

  constructor: make null object or top object in tree

- **CSpacegroup** (**Container** &parent, const **String** name="", const **Spacegroup** &spgr=NullSpacegroup)

  constructor: make child object

- void **init** (const **Spacegroup** &spacegroup_)

  initialiser: from ***Spacegroup*** *(p. 413)*

19.49.1 Detailed Description

Spacegroup (p. 413) container. **CSpacegroup** (p. 191): This has a name and a spacegroup. It overrides the base spacegroup for any objects below it.

19.49.2 Member Function Documentation

19.49.2.1 void clipper::CSpacegroup::init (const Spacegroup & *spacegroup_*)

initialiser: from **Spacegroup** (p. 413) The object is initialised, and children are updated.

Parameters:

spacegroup_ The value to give to the contained object.

References clipper::Container::update().

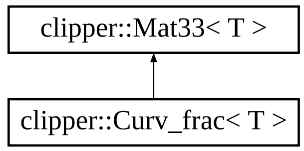
The documentation for this class was generated from the following files:

- container_types.h
- container_types.cpp

19.50 clipper::Curv_frac< T > Class Template Reference

fractional (cell) curvatures, with respect to fractional u,v,w

`#include <derivs.h>`Inheritance diagram for clipper::Curv_frac< T >::



Public Member Functions

- **Curv_frac** ()
  null constructor
- **Curv_frac** (const **Mat33**< T > &m)
  constructor: copy/convert
- **Curv_orth**< T > **curv_orth** (const **Cell** &cell) const
  fractional-orthogonal derivative conversion
- **Curv_map**< T > **curv_map** (const **Grid** &g) const
  fractional-grid derivative conversion

19.50.1 Detailed Description

template<class T> class clipper::Curv_frac< T >

fractional (cell) curvatures, with respect to fractional u,v,w

19.50.2 Member Function Documentation

19.50.2.1 template<class T > Curv_orth< T > clipper::Curv_frac< T >::curv_orth (const Cell & *cell*) const [inline]

fractional-orthogonal derivative conversion

Parameters:

cell The cell concerned

Returns:

The transformed derivative.

References clipper::Cell::matrix_frac().

19.50.2.2 template<class T > Curv_map< T > clipper::Curv_frac< T >::curv_map (const Grid & *g*) const [inline]

fractional-grid derivative conversion

Parameters:

g The grid concerned

Returns:

The transformed derivative.

References clipper::Grid::nu(), clipper::Grid::nv(), and clipper::Grid::nw().

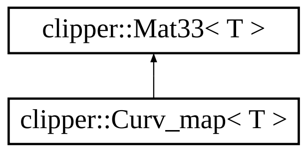
The documentation for this class was generated from the following file:

- derivs.h

19.51 clipper::Curv_map< T > Class Template Reference

map coordinate curvatures, with respect to grid u,v,w

`#include <derivs.h>`Inheritance diagram for clipper::Curv_map< T >::



Public Member Functions

- **Curv_map** ()
  null constructor
- **Curv_map** (const **Mat33**< T > &m)
  constructor: copy/convert
- **Curv_frac**< T > **curv_frac** (const **Grid** &g) const
  grid-fractional derivative conversion

19.51.1 Detailed Description

template<class T> class clipper::Curv_map< T >

map coordinate curvatures, with respect to grid u,v,w

19.51.2 Member Function Documentation

19.51.2.1 template<class T > Curv_frac< T > clipper::Curv_map< T >::curv_frac (const Grid & *g*) const `[inline]`

grid-fractional derivative conversion

Parameters:

g The grid concerned

Returns:

The transformed derivative.

References clipper::Grid::nu(), clipper::Grid::nv(), and clipper::Grid::nw().

Referenced by clipper::Xmap< T >::interp_curv().

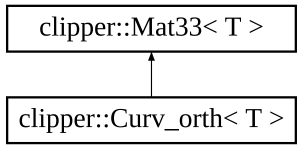
The documentation for this class was generated from the following file:

- derivs.h

19.52 clipper::Curv_orth< T > Class Template Reference

orthogonal (Angstom) curvatures, with respect to orthogonal x,y,z

`#include <derivs.h>`Inheritance diagram for clipper::Curv_orth< T >::



Public Member Functions

- **Curv_orth** ()
  null constructor
- **Curv_orth** (const **Mat33**< T > &m)
  constructor: copy/convert
- **Curv_frac**< T > **curv_frac** (const **Cell** &cell) const
  orthogonal-fractional derivative conversion

19.52.1 Detailed Description

template<class T> class clipper::Curv_orth< T >

orthogonal (Angstom) curvatures, with respect to orthogonal x,y,z

19.52.2 Member Function Documentation

19.52.2.1 template<class T > Curv_frac< T > clipper::Curv_orth< T >::curv_frac (const Cell & *cell*) const `[inline]`

orthogonal-fractional derivative conversion

Parameters:

cell The cell concerned

Returns:

The transformed derivative.

References clipper::Cell::matrix_orth().

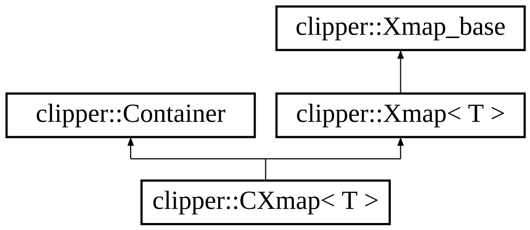
The documentation for this class was generated from the following file:

- derivs.h

19.53 clipper::CXmap< T > Class Template Reference

Crystallographic map container.

#include <container_map.h>Inheritance diagram for clipper::CXmap< T >::



Public Member Functions

- **CXmap** ()
  null constructor
- **CXmap** (**Container** &parent, const **String** name="")
  constructor: inherit spacegroup, cell and grid
- void **init** (const **Spacegroup** &spacegroup, const **Cell** &cell, const **Grid_sampling** &grid_sampling)
  initialiser: supply or inherit spacegroup, cell and grid
- void **update** ()
  hierarchical update

19.53.1 Detailed Description

template<class T> class clipper::CXmap< T >

Crystallographic map container. **CXmap** (p. 196): This is a crystallographic map.

19.53.2 Constructor & Destructor Documentation

19.53.2.1 template<class T > clipper::CXmap< T >::CXmap (Container & *parent*, const String *name* = "") `[inline]`

constructor: inherit spacegroup, cell and grid The object is constructed at the given location in the hierarchy. An attempt is made to initialise the object using information from its parents in the hierarchy.

Parameters:

parent An object in the hierarchy (usually the parent of the new object).

name The path from `parent` to the new object (usually just the name of the new object).

References clipper::CXmap< T >::init().

19.53.3 Member Function Documentation

19.53.3.1 template<class T > void clipper::CXmap< T >::init (const Spacegroup & *spacegroup*, const Cell & *cell*, const Grid_sampling & *grid_sampling*) `[inline]`

initialiser: supply or inherit spacegroup, cell and grid An attempt is made to initialise the object using information from the supplied parameters, or if they are Null, from its parents in the hierarchy.

Parameters:

spacegroup The spacegroup for the map.

name The cell for the map.

grid The grid for the map.

Reimplemented from **clipper::Xmap< T >** (p. 463).

References clipper::Grid_sampling::is_null(), clipper::Cell::is_null(), clipper::Spacegroup::is_-null(), and clipper::CXmap< T >::update().

Referenced by clipper::CXmap< T >::CXmap(), and clipper::CXmap< T >::update().

19.53.3.2 template<class T > void clipper::CXmap< T >::update () `[inline, virtual]`

hierarchical update Hierarchical update. If this object is uninitialised, an attempt is made to initialise the object using information from its parents in the hierarchy. The childen of the object are then updated.

Reimplemented from **clipper::Container** (p. 169).

References clipper::CXmap< T >::init(), and clipper::Xmap_base::is_null().

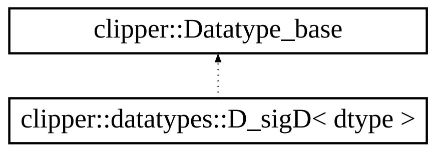
Referenced by clipper::CXmap< T >::init().

The documentation for this class was generated from the following file:

- container_map.h

19.54 clipper::datatypes::D_sigD< dtype > Class Template Reference

`#include` <`hkl_datatypes.h`>Inheritance diagram for clipper::datatypes::D_sigD< dtype >::



Public Member Functions

- **D_sigD** (const dtype &d, const dtype &sigd)
- void **set_null** ()

  initialise to 'missing' (all elements are set to null)
- void **friedel** ()

  apply Friedel transformation to the data
- void **shift_phase** (const **ftype** &dphi)

  apply phase shift to the data
- bool **missing** () const

  return true if data is missing
- void **data_export** (**xtype** array[ ]) const

  conversion to array (for I/O)
- void **data_import** (const **xtype** array[ ])

  conversion from array (for I/O)
- void **scale** (const **ftype** &s)

  this type is scalable - apply magnitude scale factor
- const dtype & **d** () const
- const dtype & **sigd** () const
- dtype & **d** ()
- dtype & **sigd** ()

Static Public Member Functions

- static **String type** ()

  return the name of this data type
- static int **data_size** ()

  return number of data elements in this type

- static **String data_names** ()
  return names of data elements in this type

19.54.1 Detailed Description

template<class dtype> class clipper::datatypes::D_sigD< dtype >

Deprecated

Anomalous difference data type: D + sigD

Provided for i/o compatibbility with legacy code only. Do not use.

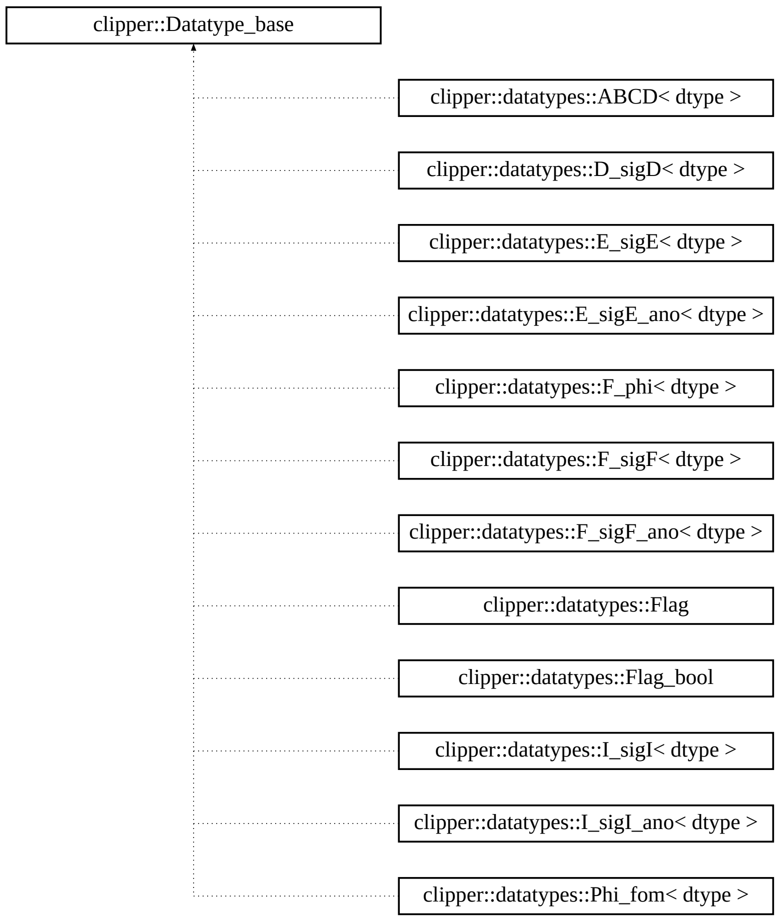
The documentation for this class was generated from the following file:

- hkl_datatypes.h

19.55 clipper::Datatype_base Class Reference

Reflection data type objects.

`#include <hkl_data.h>`Inheritance diagram for clipper::Datatype_base::



Protected Member Functions

- **Datatype_base** ()
  initialise to 'missing'
- void **set_null** ()
  initialise to 'missing' (all elements are set to null)
- void **friedel** ()
  apply Friedel transformation to the data
- void **shift_phase** (const **ftype** &dphi)
  apply phase shift to the data
- bool **missing** () const
  return true if data is missing

- void **data_export** (**xtype** array[ ]) const
  conversion to array (for I/O)

- void **data_import** (const **xtype** array[ ])
  conversion from array (for I/O)

Static Protected Member Functions

- static **String type** ()
  return the name of this data type

- static int **data_size** ()
  return number of data elements in this type

- static **String data_names** ()
  return names of data elements in this type

19.55.1 Detailed Description

Reflection data type objects. A class from which data type objects are usually derived

To define a new type for use in an **HKL_data** (p. 267) structure, subclass this class and override the following methods:

- constructor - initialises to NAN
- **type()** (p. 201) - returns type name, which is a list of the contained data
- **friedel()** (p. 200) - applies Fridel transformation
- **shift_phase()** (p. 200) - applies phase shift transformation
- **missing()** (p. 200) - checks if data is present

- **data_size()** (p. 201) - number of data elements in this type
- **data_names()** (p. 201) - names of data elements in this type
- **data_export()** (p. 201) - conversion to array (for I/O)
- **data_import()** (p. 201) - conversion from array (for I/O)

- scale() - (OPTIONAL) apply magnitude scale factor to data

Note:

polymorphism is NOT used here because virtual tables would be to expensive for every individual reflection, both in terms of space and cpu cycles.

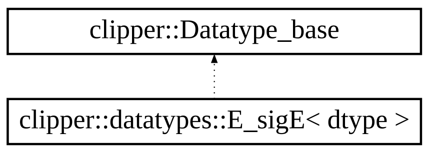
The documentation for this class was generated from the following file:

- hkl_data.h

19.56 clipper::datatypes::E_sigE< dtype > Class Template Reference

Reflection data type: E + sigE.

`#include` <`hkl_datatypes.h`>Inheritance diagram for clipper::datatypes::E_sigE< dtype >::



Public Member Functions

- **E_sigE** (const dtype &E, const dtype &sigE)
- void **set_null** ()

  initialise to 'missing' (all elements are set to null)

- void **friedel** ()

  apply Friedel transformation to the data

- void **shift_phase** (const **ftype** &dphi)

  apply phase shift to the data

- bool **missing** () const

  return true if data is missing

- void **data_export** (**xtype** array[ ]) const

  conversion to array (for I/O)

- void **data_import** (const **xtype** array[ ])

  conversion from array (for I/O)

- void **scale** (const **ftype** &s)

  this type is scalable - apply magnitude scale factor

- const dtype & **E** () const
- const dtype & **sigE** () const
- dtype & **E** ()
- dtype & **sigE** ()
- const dtype & **E_pl** () const
- const dtype & **sigE_pl** () const
- const dtype & **E_mi** () const
- const dtype & **sigE_mi** () const
- dtype **cov** () const

Static Public Member Functions

- static **String type** ()
  return the name of this data type
- static int **data_size** ()
  return number of data elements in this type
- static **String data_names** ()
  return names of data elements in this type

19.56.1 Detailed Description

template<class dtype> class clipper::datatypes::E_sigE< dtype >

Reflection data type: E + sigE. This is not strictly a type for storing E values, but rather a type for storing any sturcture factor magnitude-like quantity which has already had a symmetry enhancement factor (epsilon) removed from it. E's are most commonly stored in this form, wheras F's and U's are not. You can compute corrected F's from uncorrected F's using:

```
clipper::HKL_data<clipper::data32::F_sigF> fsigf;
clipper::HKL_data<clipper::data32::E_sigE> esige;
esige.compute( fsigf, clipper::data32::Compute_EsigE_from_FsigF() );
```

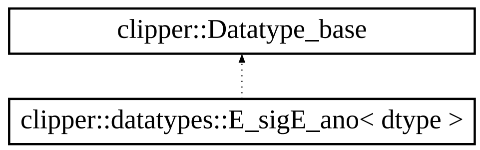
The documentation for this class was generated from the following file:

- hkl_datatypes.h

19.57 clipper::datatypes::E_sigE_ano< dtype > Class Template Reference

Reflection data type: E(+) E(+) sigE(+) sigE(-) cov+-.

#include <hkl_datatypes.h>Inheritance diagram for clipper::datatypes::E_sigE_ano< dtype >::



Public Member Functions

- void **set_null** ()

  initialise to 'missing' (all elements are set to null)

- void **friedel** ()

  apply Friedel transformation to the data

- void **shift_phase** (const **ftype** &dphi)

  apply phase shift to the data

- bool **missing** () const

  return true if data is missing

- void **data_export** (**xtype** a[ ]) const

  conversion to array (for I/O)

- void **data_import** (const **xtype** a[ ])

  conversion from array (for I/O)

- void **scale** (const **ftype** &s)

  this type is scalable - apply magnitude scale factor

- const dtype & **E_pl** () const
- const dtype & **sigE_pl** () const
- const dtype & **E_mi** () const
- const dtype & **sigE_mi** () const
- const dtype & **cov** () const
- dtype & **E_pl** ()
- dtype & **sigE_pl** ()
- dtype & **E_mi** ()
- dtype & **sigE_mi** ()
- dtype & **cov** ()
- dtype **E** () const
- dtype **sigE** () const

Static Public Member Functions

- static **String type** ()
  return the name of this data type

- static int **data_size** ()
  return number of data elements in this type

- static **String data_names** ()
  return names of data elements in this type

19.57.1 Detailed Description

template<class dtype> class clipper::datatypes::E_sigE_ano< dtype >

Reflection data type: E(+) E(+) sigE(+) sigE(-) cov+-. see **datatypes::E_sigE** (p. 202)

The documentation for this class was generated from the following file:

- hkl_datatypes.h

19.58 clipper::Euler< T > Class Template Reference

Euler (p. 206) angle class.

`#include <rotation.h>`

Public Member Functions

- **Euler** ()
  constructor: null
- **Euler** (const **ftype** &alpha, const **ftype** &beta, const **ftype** &gamma)
  constructor: from specified angles
- **Euler** (const **Rotation** &rot)
  constructor: from rotation
- **Rotation rotation** () const
  return rotation
- const **ftype** & **alpha** () const
  return alpha
- const **ftype** & **beta** () const
  return beta
- const **ftype** & **gamma** () const
  return gamma
- **String format** () const
  *return formatted **String** (p. 426) representation*

19.58.1 Detailed Description

template<int T> class clipper::Euler< T >

Euler (p. 206) angle class.

The documentation for this class was generated from the following files:

- rotation.h
- rotation.cpp

19.59 clipper::Euler_ccp4 Class Reference

Euler_ccp4 (p. 207) angle class.

`#include <rotation.h>`

Public Member Functions

- **Euler_ccp4** ()
  constructor: null
- **Euler_ccp4** (const **ftype** &alpha, const **ftype** &beta, const **ftype** &gamma)
  constructor: from specified angles
- const **ftype** & **alpha** () const
  return alpha
- const **ftype** & **beta** () const
  return beta
- const **ftype** & **gamma** () const
  return gamma
- **String format** () const
  *return formatted **String** (p. 426) representation*

19.59.1 Detailed Description

Euler_ccp4 (p. 207) angle class.

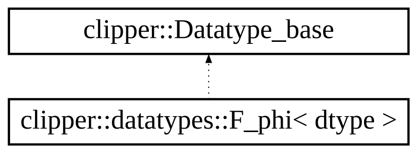
The documentation for this class was generated from the following files:

- rotation.h
- rotation.cpp

19.60 clipper::datatypes::F_phi< dtype > Class Template Reference

Reflection data type: F + phi model or map coeff (e.g. Fcalc, Fbest).

#include <hkl_datatypes.h>Inheritance diagram for clipper::datatypes::F_phi< dtype >::



Public Member Functions

- **F_phi** (const dtype &f, const dtype &phi)
- void **set_null** ()

  initialise to 'missing' (all elements are set to null)
- void **friedel** ()

  apply Friedel transformation to the data
- void **shift_phase** (const **ftype** &dphi)

  apply phase shift to the data
- bool **missing** () const

  return true if data is missing
- void **data_export** (**xtype** array[ ]) const

  conversion to array (for I/O)
- void **data_import** (const **xtype** array[ ])

  conversion from array (for I/O)
- void **scale** (const **ftype** &s)

  this type is scalable - apply magnitude scale factor
- const dtype & **f** () const
- const dtype & **phi** () const
- dtype & **f** ()
- dtype & **phi** ()
- dtype **a** () const

  read real part
- dtype **b** () const

  read imag part
- **F_phi** (const std::complex< dtype > c)

  convert from complex

- **operator std::complex< dtype >** () const

  convert to complex

- dtype **resolve** (const dtype phi)

  resolve along phase direction

- const **F_phi**< dtype > & **norm** ()

  tidy up so that real part is positive and phase 0...twopi

Static Public Member Functions

- static **String type** ()

  return the name of this data type

- static int **data_size** ()

  return number of data elements in this type

- static **String data_names** ()

  return names of data elements in this type

19.60.1 Detailed Description

template<class dtype> class clipper::datatypes::F_phi< dtype >

Reflection data type: F + phi model or map coeff (e.g. Fcalc, Fbest).

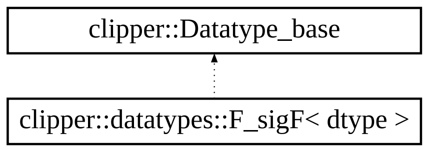
The documentation for this class was generated from the following file:

- hkl_datatypes.h

19.61 clipper::datatypes::F_sigF< dtype > Class Template Reference

Reflection data type: F + sigF.

`#include` <`hkl_datatypes.h`>Inheritance diagram for clipper::datatypes::F_sigF< dtype >::



Public Member Functions

- **F_sigF** (const dtype &f, const dtype &sigf)
- void **set_null** ()

  initialise to 'missing' (all elements are set to null)
- void **friedel** ()

  apply Friedel transformation to the data
- void **shift_phase** (const **ftype** &dphi)

  apply phase shift to the data
- bool **missing** () const

  return true if data is missing
- void **data_export** (**xtype** array[ ]) const

  conversion to array (for I/O)
- void **data_import** (const **xtype** array[ ])

  conversion from array (for I/O)
- void **scale** (const **ftype** &s)

  this type is scalable - apply magnitude scale factor
- const dtype & **f** () const
- const dtype & **sigf** () const
- dtype & **f** ()
- dtype & **sigf** ()
- const dtype & **f_pl** () const
- const dtype & **sigf_pl** () const
- const dtype & **f_mi** () const
- const dtype & **sigf_mi** () const
- dtype **cov** () const

Static Public Member Functions

- static **String type** ()
  return the name of this data type
- static int **data_size** ()
  return number of data elements in this type
- static **String data_names** ()
  return names of data elements in this type

19.61.1 Detailed Description

template<class dtype> class clipper::datatypes::F_sigF< dtype >

Reflection data type: F + sigF. Note that **F_sigF** (p. 210) also has methods for returning f_pl(), sigf_pl(), f_mi, sigf_mi(), so you can use this type in any template type where you would use **F_sigF_ano** (p. 212).

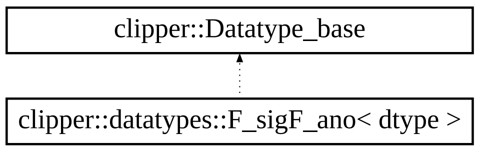
The documentation for this class was generated from the following file:

- hkl_datatypes.h

19.62 clipper::datatypes::F_sigF_ano< dtype > Class Template Reference

Reflection data type: F(+) F(+) sigF(+) sigF(-) cov+-.

`#include <hkl_datatypes.h>`Inheritance diagram for clipper::datatypes::F_sigF_ano< dtype >::



Public Member Functions

- void **set_null** ()

  initialise to 'missing' (all elements are set to null)

- void **friedel** ()

  apply Friedel transformation to the data

- void **shift_phase** (const **ftype** &dphi)

  apply phase shift to the data

- bool **missing** () const

  return true if data is missing

- void **data_export** (**xtype** a[ ]) const

  conversion to array (for I/O)

- void **data_import** (const **xtype** a[ ])

  conversion from array (for I/O)

- void **scale** (const **ftype** &s)

  this type is scalable - apply magnitude scale factor

- const dtype & **f_pl** () const
- const dtype & **sigf_pl** () const
- const dtype & **f_mi** () const
- const dtype & **sigf_mi** () const
- const dtype & **cov** () const
- dtype & **f_pl** ()
- dtype & **sigf_pl** ()
- dtype & **f_mi** ()
- dtype & **sigf_mi** ()
- dtype & **cov** ()
- dtype **f** () const
- dtype **sigf** () const

Static Public Member Functions

- static **String type** ()
  return the name of this data type
- static int **data_size** ()
  return number of data elements in this type
- static **String data_names** ()
  return names of data elements in this type

19.62.1 Detailed Description

template<class dtype> class clipper::datatypes::F_sigF_ano< dtype >

Reflection data type: F(+) F(+) sigF(+) sigF(-) cov+-. Note that **F_sigF_ano** (p. 212) also has methods for returning f(), sigf(), so you can use this type in any template type where you would use **F_sigF** (p. 210).

The documentation for this class was generated from the following file:

- hkl_datatypes.h

19.63 clipper::BasisFn_base::Fderiv Class Reference

object holding the basis function and its first two derivatives

`#include <resol_fn.h>`

Public Member Functions

- **Fderiv** ()
  null constructor
- **Fderiv** (const int &np)

Public Attributes

- **ftype f**
  value of function
- std::vector< **ftype** > **df**
  first derivative vector w.r.t params
- **Matrix df2**
  second derivative matrix w.r.t params

19.63.1 Detailed Description

object holding the basis function and its first two derivatives

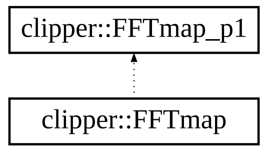
The documentation for this class was generated from the following file:

- resol_fn.h

19.64 clipper::FFTmap Class Reference

FFTmap (p. 215): P1 map with symmetry used for calculating FFTs.

`#include <fftmap.h>`Inheritance diagram for clipper::FFTmap::



Public Member Functions

- **FFTmap** ()

  Null constructor.

- **FFTmap** (const **Spacegroup** &spacegroup, const **Cell** &cell, const **Grid_sampling** grid_sam, const FFTtype type=Default)

  Constructor: takes spacegroup, cell, grid.

- void **init** (const **Spacegroup** &spacegroup, const **Cell** &cell, const **Grid_sampling** grid_-sam, const FFTtype type=Default)

  initialiser

- void **reset** ()

  Reset.

- const **Cell** & **cell** () const

  get the cell

- const **Spacegroup** & **spacegroup** () const

  get the spacegroup

- const **Grid_sampling** & **grid_sampling** () const

  get the cell grid

- void **fft_h_to_x** ()

  Transform to real space.

- void **fft_x_to_h** ()

  Transform to reciprocal space.

- template<class T >
  void **get_recip_data** (const **HKL** &rfl, **datatypes::F_phi**< T > &fphi) const

  get reciprocal space data

- template<class T >
  void **set_recip_data** (const **HKL** &rfl, const **datatypes::F_phi**< T > &fphi)

  set reciprocal space data

- template< class T >
  void **get_real_data** (const **Coord_grid** &c, T &datum) const

  get real space data

- template< class T >
  void **set_real_data** (const **Coord_grid** &c, const T &datum)

  set real space data

- **datatypes::F_phi**< ffttype > **get_recip_data** (const **HKL** &rfl) const

  get reciprocal space data (No error checking)

- const ffttype & **get_real_data** (const **Coord_grid** &c) const

  get real space data (No error checking)

- template< class H , class X >
  void **fft_rfl_to_map** (const H &h, X &x)

  calculate map-like object from reflection-like object

- template< class H , class X >
  void **fft_map_to_rfl** (const X &x, H &h)

  calculate reflection-like object from map-like object

- void **debug** () const

Protected Attributes

- **Cell cell_**

  unit cell

- **Spacegroup spacegroup_**

  spacegroup

- std::vector< **Isymop** > **isymop**

  Integerised symops.

19.64.1 Detailed Description

FFTmap (p. 215): P1 map with symmetry used for calculating FFTs. The **FFTmap** (p. 215) is represented in P1 in memory. However, it also has a spacegroup, and the contained data remains consistent with this spacegroup at all times. It has three states - unassigned, real-space, and reciprocal space. In real space it contains real map data. In reciprocal space it holds a hemisphere of complex structure factors, with the Friedels duplicated on the zero section.

The user should be able to ignore all the issues of spacegroup symmetry, Friedel opposites, and storage order.

19.64.2 Constructor & Destructor Documentation

19.64.2.1 clipper::FFTmap::FFTmap ()

Null constructor. For later initialisation: see **init()** (p. 217)

References clipper::Message::message().

19.64.2.2 clipper::FFTmap::FFTmap (const Spacegroup & *spacegroup*, const Cell & *cell*, const Grid_sampling *grid_sam*, const FFTtype *type* = `Default`)

Constructor: takes spacegroup, cell, grid. Construct an **FFTmap** (p. 215) for a given spacegroup, cell, and grid. The map values are initialised to zero.

The **FFTmap** (p. 215) is initially in neither real nor reciprocal spce, however as soon as one of the 'set' methods is called, it will be defined as in either real or reciprocal space until the next fft.

Parameters:

spacegroup The spacegroup.

cell The cell, used for scaling.

grid_sam The grid sampling of the unit cell.

precalc Perform slow precalculation to get faster FFT. (default: no)

References init(), and clipper::Message::message().

19.64.3 Member Function Documentation

19.64.3.1 void clipper::FFTmap::init (const Spacegroup & *spacegroup*, const Cell & *cell*, const Grid_sampling *grid_sam*, const FFTtype *type* = `Default`)

initialiser Initialise an **FFTmap** (p. 215) for a given spacegroup, cell, and grid. The map values are initialised to zero.

The **FFTmap** (p. 215) is initially in neither real nor reciprocal spce, however as soon as one of the 'set' methods is called, it will be defined as in either real or reciprocal space until the next fft.

Parameters:

spacegroup The spacegroup.

cell The cell, used for scaling.

grid_sam The grid sampling of the unit cell.

precalc Perform slow precalculation to get faster FFT. This adds a penalty of about 4s on Linux for the first FFT of any grid and direction. Subsequent FFTs will be faster. Set to true for programs which will use many FFTs. default: false.

References cell_, isymop, clipper::Spacegroup::num_symops(), spacegroup_, and clipper::Spacegroup::symop().

Referenced by FFTmap().

19.64.3.2 void clipper::FFTmap::reset ()

Reset. Reset the space and zero all the data, if necessary.

Reimplemented from **clipper::FFTmap_p1** (p. 225).

References clipper::FFTmap_p1::datavec, and clipper::FFTmap_p1::mode.

Referenced by fft_map_to_rfl(), and fft_rfl_to_map().

19.64.3.3 void clipper::FFTmap::fft_h_to_x ()

Transform to real space. The data is transformed from recirocal to real space. A scale factor of 1/v (where v is the cell volume) is applied. If the **FFTmap** (p. 215) is already in real space, no action is taken.

References cell(), and clipper::FFTmap_p1::mode.

Referenced by fft_rfl_to_map().

19.64.3.4 void clipper::FFTmap::fft_x_to_h ()

Transform to reciprocal space. The data is transformed from real to recirocal space. A scale factor of v/n (where v is the cell volume and n the number of grid points) is applied. If the **FFTmap** (p. 215) is already in reciproal space, no action is taken.

References cell(), and clipper::FFTmap_p1::mode.

Referenced by fft_map_to_rfl().

19.64.3.5 template<class T > void clipper::FFTmap::get_recip_data (const HKL & *rfl*, datatypes::F_phi< T > & *fphi*) const `[inline]`

get reciprocal space data The data value for the given **HKL** (p. 261), or the conjugate of its Friedel opposite if required, is returned. The symmetry related copies of the data are ignored.

Parameters:

rfl The **HKL** (p. 261) of the data to be returned.

fphi The value, as a magnitude and phase of type `ffttype`

References clipper::FFTmap_p1::get_hkl(), clipper::Message::message(), and clipper::FFTmap_-p1::mode.

Referenced by fft_map_to_rfl().

19.64.3.6 template<class T > void clipper::FFTmap::set_recip_data (const HKL & *rfl*, const datatypes::F_phi< T > & *fphi*) `[inline]`

set reciprocal space data The data value for the given **HKL** (p. 261), or the conjugate of its Friedel opposite if required, is set. All the symmetry related copies of the data, and any Friedel copies in the zero section, are also set.

Parameters:

rfl The **HKL** (p. 261) of the data to be set.

fphi The value, as a magnitude and phase of type `ffttype`

References isymop, clipper::Message::message(), clipper::FFTmap_p1::mode, clipper::Spacegroup::num_primops(), clipper::FFTmap_p1::set_hkl(), spacegroup_-, clipper::HKL::sym_phase_shift(), clipper::Spacegroup::symop(), and clipper::HKL::transform().

Referenced by fft_rfl_to_map().

19.64.3.7 template<class T > void clipper::FFTmap::get_real_data (const Coord_grid & *c*, T & *datum*) const [inline]

get real space data The data value for the given grid coordinate is returned. Symmetry related copies are ignored.

Parameters:

c The coordinate of the data to be returned.

datum The value of the data.

References clipper::FFTmap_p1::grid_real(), clipper::Message::message(), clipper::FFTmap_-p1::mode, clipper::FFTmap_p1::real_data(), and clipper::Coord_grid::unit().

Referenced by fft_rfl_to_map().

19.64.3.8 template<class T > void clipper::FFTmap::set_real_data (const Coord_grid & *c*, const T & *datum*) [inline]

set real space data The data value for the given grid coordinate is set. All the symmetry related copies of the data are also set.

Parameters:

c The coordinate of the data to be set.

datum The value of the data.

References clipper::FFTmap_p1::grid_sam_, isymop, clipper::Message::message(), clipper::FFTmap_p1::mode, clipper::FFTmap_p1::real_data(), clipper::Coord_-grid::transform(), and clipper::Coord_grid::unit().

Referenced by fft_map_to_rfl().

19.64.3.9 datatypes::F_phi< ffttype > clipper::FFTmap::get_recip_data (const HKL & *rfl*) const

get reciprocal space data (No error checking) No error is produced if the space is wrong.

Parameters:

rfl The **HKL** (p. 261) of the data to be returned.

Returns:

The value, as magnitude and phase of type `ffttype`

References clipper::FFTmap_p1::get_hkl().

19.64.3.10 const ffttype & clipper::FFTmap::get_real_data (const Coord_grid & *c*) const [inline]

get real space data (No error checking) No error is produced if the space is wrong.

Parameters:

c The grid coordinate of the data to be returned.

Returns:

The value, as type `ffttype`

References clipper::FFTmap_p1::grid_real(), clipper::FFTmap_p1::real_data(), and clipper::Coord_grid::unit().

19.64.3.11 template<class H , class X > void clipper::FFTmap::fft_rfl_to_map (const H & *h*, X & *x*) [inline]

calculate map-like object from reflection-like object Fill this **FFTmap** (p. 215) object from a reflection object, transform it, and fill the given map object from the **FFTmap** (p. 215). This will work for any reflection data object which implements a HKL_reference_index, and every map data object which implements a Map_reference_index.

For the results to be sensible, the spacegroup, cell and grids should match. (The map will be zeroed if necessary).

Parameters:

h The source reflection data object.

x The target map object.

References fft_h_to_x(), get_real_data(), reset(), and set_recip_data().

19.64.3.12 template<class H , class X > void clipper::FFTmap::fft_map_to_rfl (const X & *x*, H & *h*) [inline]

calculate reflection-like object from map-like object Fill this **FFTmap** (p. 215) object from a map object, transform it, and fill the given reflection object from the **FFTmap** (p. 215). This will work for any reflection data object which implements a HKL_reference_index, and every map data object which implements a Map_reference_index.

For the results to be sensible, the spacegroup, cell and grids should match. (The map will be zeroed if necessary).

Parameters:

x The source map object.

h The target reflection data object.

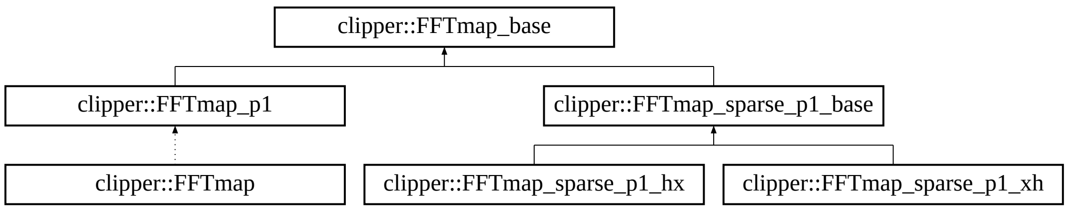
References fft_x_to_h(), get_recip_data(), reset(), and set_real_data().

The documentation for this class was generated from the following files:

- fftmap.h
- fftmap.cpp

19.65 clipper::FFTmap_base Class Reference

Inheritance diagram for clipper::FFTmap_base::



Public Types

- enum **FFTtype** { **Default**, **Measure**, **Estimate** }

Static Protected Attributes

- static **Mutex mutex** = **Mutex**()

  Thread safety.

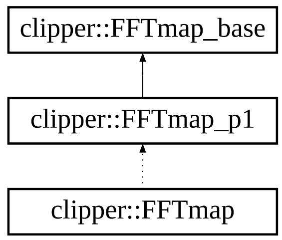
The documentation for this class was generated from the following files:

- fftmap.h
- fftmap.cpp

19.66 clipper::FFTmap_p1 Class Reference

FFTmap_p1 (p. 222): low level P1 map used for calculating FFTs.

`#include <fftmap.h>`Inheritance diagram for clipper::FFTmap_p1::



Public Member Functions

- **FFTmap_p1** ()
  Null constructor.
- **FFTmap_p1** (const **FFTmap_p1** &other)
  Copy constructor.
- **FFTmap_p1** (const **Grid_sampling** &grid_sam, const FFTtype type=Default)
  Constructor: takes grid.
- const **FFTmap_p1** & **operator=** (const **FFTmap_p1** &other)
  Assignment operator.
- void **init** (const **Grid_sampling** &grid_sam, const FFTtype type=Default)
  initialiser: takes grid
- void **reset** ()
  Reset.
- const **Grid_sampling** & **grid_real** () const
  Return real space grid.
- const **Grid** & **grid_reci** () const
  Return reciprocal space grid (i.e. half real grid + 1 section).
- bool **uniq_reci** (const **Coord_grid** &c) const
  Test whether a coordinate is in the valid part of the recip. grid.
- void **fft_h_to_x** (const **ftype** &scale)
  Transform to real space.
- void **fft_x_to_h** (const **ftype** &scale)
  Transform to reciprocal space.

- std::complex< ffttype > **get_hkl** (const **HKL** &hkl) const
  get reciprocal space data: slow form with hemisphere check
- void **set_hkl** (const **HKL** &hkl, const std::complex< ffttype > &f)
  set reciprocal space data: slow form with hemisphere check
- const std::complex< ffttype > & **cplx_data** (const **Coord_grid** &c) const
  get reciprocal space data
- std::complex< ffttype > & **cplx_data** (const **Coord_grid** &c)
  set reciprocal space data
- const ffttype & **real_data** (const **Coord_grid** &c) const
  get real space data
- ffttype & **real_data** (const **Coord_grid** &c)
  set real space data
- void **debug** () const

Static Public Member Functions

- static FFTtype & **default_type** ()
  set/get default optimisation type

Protected Types

- enum **FFTmode** { **NONE**, **RECI**, **REAL**, **OTHER** }

Protected Member Functions

- const **FFTmap_p1** & **copy** (const **FFTmap_p1** &other)
  copy function

Protected Attributes

- FFTmode **mode**
  real or reciprocal space?
- FFTtype **type_**
  optimisation options
- **Grid_sampling grid_sam_**
  unit cell grid
- **Grid grid_reci_**

reciprocal space grid

- **Grid grid_real_**

  real space grid

- **Grid grid_half_**

  half grid (for marking unique)

- **Matrix**< char > **req_kl**
- **Matrix**< char > **req_uv**

  reci section lookup

- std::vector< char > **req_l**
- std::vector< char > **req_u**

  real section lookup

- std::vector< ffttype > **datavec**

  vector for the data

- ffttype * **data_r**

  pointer to real data

- std::complex< ffttype > * **data_c**

  pointer to complex data

Static Protected Attributes

- static FFTtype **default_type_** = FFTmap_base::Estimate

  default optimisation options

19.66.1 Detailed Description

FFTmap_p1 (p. 222): low level P1 map used for calculating FFTs. This is a pure real P1 map, with an extra section in reciprocal space to allow generation of the full set of resiprocal space magnitudes. Access is by **Coord_grid** (p. 175) in both spaces, and indices must be non-negative and in range. The first and last sections along the half-length direction only have half the elements stored, the contents of the other half is ignored.

19.66.2 Constructor & Destructor Documentation

19.66.2.1 clipper::FFTmap_p1::FFTmap_p1 ()

Null constructor. For later initialisation: see **init()** (p. 225)

19.66.2.2 clipper::FFTmap_p1::FFTmap_p1 (const Grid_sampling & *grid_sam*, const FFTtype *type* = `Default`)

Constructor: takes grid. Construct an **FFTmap_p1** (p. 222) for a given spacegroup, cell, and grid. The map values are initialised to zero.

The **FFTmap_p1** (p. 222) is initially in neither real nor reciprocal spce, however as soon as one of the 'set' methods is called, it will be defined as in either real or reciprocal space until the next fft.

Parameters:

grid_sam The grid sampling of the unit cell.

type Can be FFTmap_p1::Measure, FFTmap_p1::Estimate. Measure performs slow precalculation (first time only) to get faster FFT.

References init().

19.66.3 Member Function Documentation

19.66.3.1 void clipper::FFTmap_p1::init (const Grid_sampling & *grid_sam*, const FFTtype *type* = `Default`)

initialiser: takes grid Initialise an **FFTmap_p1** (p. 222) for a given spacegroup, cell, and grid. The map values are initialised to zero.

The **FFTmap_p1** (p. 222) is initially in neither real nor reciprocal spce, however as soon as one of the 'set' methods is called, it will be defined as in either real or reciprocal space until the next fft.

Parameters:

grid_sam The grid sampling of the unit cell.

type Can be FFTmap_p1::Measure, FFTmap_p1::Estimate. Measure performs slow precalculation (first time only) to get faster FFT.

References data_c, data_r, datavec, default_type(), grid_half_, grid_real_, grid_reci_-, grid_sam_, mode, clipper::Grid::nu(), clipper::Grid_sampling::nu(), clipper::Grid::nv(), clipper::Grid_sampling::nv(), clipper::Grid::nw(), clipper::Grid_sampling::nw(), clipper::Grid::size(), and type_.

Referenced by FFTmap_p1().

19.66.3.2 void clipper::FFTmap_p1::reset ()

Reset. Reset the space and zero all the data, if necessary.

Reimplemented in **clipper::FFTmap** (p. 218).

References datavec, and mode.

19.66.3.3 const Grid_sampling & clipper::FFTmap_p1::grid_real () const `[inline]`

Return real space grid.

Returns:

The grid sampling of the real space grid.

References grid_sam_.

Referenced by clipper::FFTmap::get_real_data(), and clipper::FFTmap::grid_sampling().

19.66.3.4 const Grid & clipper::FFTmap_p1::grid_reci () const [inline]

Return reciprocal space grid (i.e. half real grid + 1 section). The reciprocal grid is half-length, plus one section, in the w direction. The remainder of the grid may be generated by Hermitian symmetry. When accessing data with reci_data, the coordinate should always be in this grid. Some points in this grid are redundent, see **FFTmap_p1::uniq_reci()** (p. 226).

Returns:

The reciprocal grid.

References grid_reci_.

19.66.3.5 bool clipper::FFTmap_p1::uniq_reci (const Coord_grid & *c*) const [inline]

Test whether a coordinate is in the valid part of the recip. grid. The w=0 and w=nw/2 sections contain some duplicated points related by a cetre of symmetry. On of these is considered to be significant, and the other redundent. This function returns 'true' for the significant point.

Note:

For some calculations it may be quicker to set the whole grid than call this function for every coordinate.

Parameters:

c The coordinate to test. Must be in **grid_reci()** (p. 226).

Returns:

true if the coordinate is for a significant point.

References grid_half_, clipper::Grid::nu(), clipper::Grid::nv(), clipper::Grid::nw(), clipper::Coord_grid::u(), clipper::Coord_grid::v(), and clipper::Coord_grid::w().

19.66.3.6 void clipper::FFTmap_p1::fft_h_to_x (const ftype & *scale*)

Transform to real space. The data is transformed from recirocal to real space. If the **FFTmap_-p1** (p. 222) is already in real space, no action is taken.

Parameters:

Scale factor to apply (normally 1/cell_volume).

References data_c, grid_reci_, grid_sam_, clipper::Mutex::lock(), mode, clipper::FFTmap_-base::mutex, clipper::Grid_sampling::nu(), clipper::Grid_sampling::nv(), clipper::Grid_-sampling::nw(), clipper::Grid::size(), type_, and clipper::Mutex::unlock().

Referenced by clipper::Xmap< T >::fft_from().

19.66.3.7 void clipper::FFTmap_p1::fft_x_to_h (const ftype & *scale*)

Transform to reciprocal space. The data is transformed from real to recirocal space. If the **FFTmap_p1** (p. 222) is already in reciproal space, no action is taken.

Parameters:

Scale factor to apply (in addition to 1/N_grid factor) (normally cell_volume).

References data_c, data_r, grid_reci_, grid_sam_, clipper::Mutex::lock(), mode, clipper::FFTmap_base::mutex, clipper::Grid_sampling::nu(), clipper::Grid_sampling::nv(), clipper::Grid_sampling::nw(), clipper::Grid::size(), clipper::Grid_sampling::size(), type_, and clipper::Mutex::unlock().

Referenced by clipper::Xmap< T >::fft_to().

19.66.3.8 std::complex< ffttype > clipper::FFTmap_p1::get_hkl (const HKL & *hkl*) const

get reciprocal space data: slow form with hemisphere check This form returns the data for an **HKL** (p. 261). The **HKL** (p. 261) is converted into a grid reference, and the data, or if necessary the conjugate of the opposite, is returned.

Parameters:

hkl The **HKL** (p. 261) of the data.

References cplx_data(), grid_reci_, grid_sam_, clipper::Grid_sampling::nu(), clipper::Grid_-sampling::nv(), clipper::Grid_sampling::nw(), clipper::Grid::nw(), clipper::Coord_grid::u(), clipper::Coord_grid::v(), and clipper::Coord_grid::w().

Referenced by clipper::Xmap< T >::fft_to(), and clipper::FFTmap::get_recip_data().

19.66.3.9 void clipper::FFTmap_p1::set_hkl (const HKL & *hkl*, const std::complex< ffttype > & *f*)

set reciprocal space data: slow form with hemisphere check This form returns the data for an **HKL** (p. 261). The **HKL** (p. 261) is converted into a grid reference, and the data, and if necessary the conjugate of the opposite, is set.

Parameters:

hkl The **HKL** (p. 261) of the data.

References cplx_data(), grid_reci_, grid_sam_, clipper::Grid::nw(), clipper::Coord_grid::unit(), and clipper::Coord_grid::w().

Referenced by clipper::Xmap< T >::fft_from(), and clipper::FFTmap::set_recip_data().

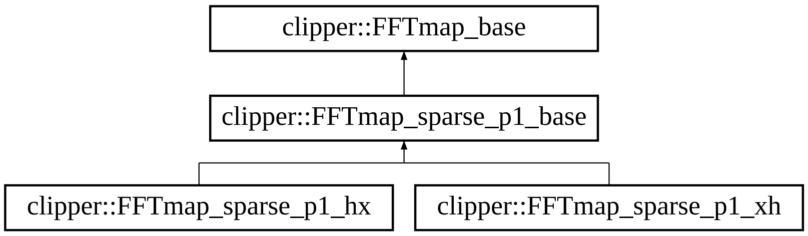
The documentation for this class was generated from the following files:

- fftmap.h
- fftmap.cpp

19.67 clipper::FFTmap_sparse_p1_base Class Reference

base type for sparse P1 fft maps

`#include <fftmap_sparse.h>`Inheritance diagram for clipper::FFTmap_sparse_p1_base::



Public Member Functions

- void **init** (const **Grid_sampling** &grid_sam, const FFTtype type=Default)
  initialiser: takes grid
- ~**FFTmap_sparse_p1_base** ()
  Destructor.
- const **Grid_sampling** & **grid_real** () const
  get real grid sampling
- const **Grid** & **grid_reci** () const
  get reciprocal grid

Static Public Member Functions

- static FFTtype & **default_type** ()
  set/get default optimisation type

Protected Member Functions

- ffttype * **map_uv** (const int &u, const int &v)
  return/create row
- std::complex< ffttype > * **map_kl** (const int &k, const int &l)
  return/create row

Protected Attributes

- **Grid_sampling grid_real_**
  real space grid

- **Grid grid_reci_**
  reciprocal space grid
- FFTtype **type_**
  optimisation options
- **Array2d**< std::complex< ffttype > * > **row_kl**
  section map
- **Array2d**< ffttype * > **row_uv**
  section map

Static Protected Attributes

- static FFTtype **default_type_** = FFTmap_base::Estimate
  default optimisation options

19.67.1 Detailed Description

base type for sparse P1 fft maps

19.67.2 Member Function Documentation

19.67.2.1 void clipper::FFTmap_sparse_p1_base::init (const Grid_sampling & *grid_sam*, const FFTtype *type* = `Default`)

initialiser: takes grid Initialise an **FFTmap_sparse_p1_base** (p. 228) for a grid.

Parameters:

grid_sam The grid sampling of the unit cell.

type Can be FFTmap_sparse_base::Measure, Estimate. Measure performs slow precalculation (first time only) to get faster FFT.

Reimplemented in **clipper::FFTmap_sparse_p1_hx** (p. 231), and **clipper::FFTmap_sparse_p1_xh** (p. 235).

References default_type(), grid_real_, grid_reci_, clipper::Grid_sampling::nu(), clipper::Grid::nv(), clipper::Grid_sampling::nv(), clipper::Grid::nw(), clipper::Grid_sampling::nw(), clipper::Array2d< T >::resize(), row_kl, row_uv, and type_.

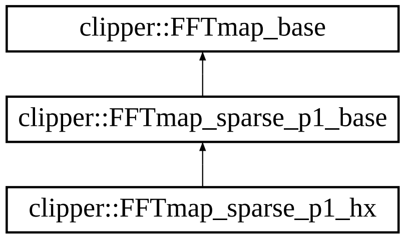
The documentation for this class was generated from the following files:

- fftmap_sparse.h
- fftmap_sparse.cpp

19.68 clipper::FFTmap_sparse_p1_hx Class Reference

FFTmap_sparse_p1_hx (p. 230): low level sparse P1 map used for calculating FFTs.

#include <fftmap_sparse.h>Inheritance diagram for clipper::FFTmap_sparse_p1_hx::



Public Member Functions

- **FFTmap_sparse_p1_hx** ()

  Null constuctor.

- **FFTmap_sparse_p1_hx** (const **Grid_sampling** &grid_sam, const FFTtype type=Default)

  Constructor: takes grid.

- void **init** (const **Grid_sampling** &grid_sam, const FFTtype type=Default)

  initialiser: takes grid

- const **Grid_sampling** & **grid_real** () const

  get real grid sampling

- const **Grid** & **grid_reci** () const

  get reciprocal grid

- void **set_hkl** (const **HKL** &hkl, const std::complex< ffttype > &f)

  set reciprocal space data by hkl

- std::complex< ffttype > & **cplx_data** (const **Coord_grid** &uvw)

  set reciprocal space data (internal use)

- void **require_real_data** (const **Coord_grid** &uvw)

  express need for real space data

- const ffttype & **real_data** (const **Coord_grid** &uvw) const

  *get real space data (uvw must be in **grid_real()** (p. 230))*

- void **fft_h_to_x** (const **ftype** &scale)

  Transform to real space.

19.68.1 Detailed Description

FFTmap_sparse_p1_hx (p. 230): low level sparse P1 map used for calculating FFTs. This version computes sparse Hermititan...real FFTs.

By specifying what parts of the map are needed in advance, it is possible to perform highly optimised FFTs, including some of the benefits of symmetry.

19.68.2 Constructor & Destructor Documentation

19.68.2.1 clipper::FFTmap_sparse_p1_hx::FFTmap_sparse_p1_hx ()

Null constuctor. For later initialisation: see **init()** (p. 231)

19.68.2.2 clipper::FFTmap_sparse_p1_hx::FFTmap_sparse_p1_hx (const Grid_sampling & *grid_sam*, const FFTtype *type* = `Default`)

Constructor: takes grid. Construct an **FFTmap_sparse_p1_hx** (p. 230) for a given grid.

Parameters:

grid_sam The grid sampling of the unit cell.

type Can be FFTmap_sparse_base::Measure, Estimate. Measure performs slow precalculation (first time only) to get faster FFT.

References init().

19.68.3 Member Function Documentation

19.68.3.1 void clipper::FFTmap_sparse_p1_hx::init (const Grid_sampling & *grid_sam*, const FFTtype *type* = `Default`)

initialiser: takes grid Initialise an **FFTmap_sparse_p1_base** (p. 228) for a grid.

Parameters:

grid_sam The grid sampling of the unit cell.

type Can be FFTmap_sparse_base::Measure, Estimate. Measure performs slow precalculation (first time only) to get faster FFT.

Reimplemented from **clipper::FFTmap_sparse_p1_base** (p. 229).

Referenced by FFTmap_sparse_p1_hx().

19.68.3.2 void clipper::FFTmap_sparse_p1_hx::set_hkl (const HKL & *hkl*, const std::complex< ffttype > & *f*)

set reciprocal space data by hkl Friedel opposites are handled correctly

Parameters:

hkl The **HKL** (p. 261) to set.

f The complex value to set.

References cplx_data(), clipper::FFTmap_sparse_p1_base::grid_real_, clipper::FFTmap_-sparse_p1_base::grid_reci_, clipper::Grid::nw(), clipper::Coord_grid::unit(), and clipper::Coord_grid::w().

Referenced by clipper::Xmap< T >::fft_from().

19.68.3.3 void clipper::FFTmap_sparse_p1_hx::require_real_data (const Coord_grid & *uvw*) [inline]

express need for real space data The given **Coord_grid** (p. 175) will be required in the final map. (uvw must be in grid_sampling())

Parameters:

uvw The coordinate to require.

References clipper::FFTmap_sparse_p1_base::map_uv(), clipper::Coord_grid::u(), and clipper::Coord_grid::v().

Referenced by clipper::Xmap< T >::fft_from().

19.68.3.4 const ffttype & clipper::FFTmap_sparse_p1_hx::real_data (const Coord_grid & *uvw*) const [inline]

get real space data (uvw must be in **grid_real()** (p. 230)) (uvw must be in grid_sampling(), and have been requested)

Parameters:

uvw The coordinate to get.

Returns:

The real value at that coordinate.

References clipper::FFTmap_sparse_p1_base::row_uv, clipper::Coord_grid::u(), clipper::Coord_grid::v(), and clipper::Coord_grid::w().

Referenced by clipper::Xmap< T >::fft_from().

19.68.3.5 void clipper::FFTmap_sparse_p1_hx::fft_h_to_x (const ftype & *scale*)

Transform to real space. The 'require' functions must have been called first to mark the required data in the target space. (Source space requirements are inferred automatically).

References clipper::FFTmap_sparse_p1_base::grid_real_, clipper::FFTmap_sparse_p1_-base::grid_reci_, clipper::Mutex::lock(), clipper::FFTmap_base::mutex, clipper::Grid::nu(), clipper::Grid_sampling::nu(), clipper::Grid_sampling::nv(), clipper::Grid::nv(), clipper::Grid::nw(), clipper::Grid_sampling::nw(), clipper::FFTmap_sparse_p1_base::row_kl, clipper::FFTmap_sparse_p1_base::row_uv, clipper::FFTmap_sparse_p1_base::type_, and clipper::Mutex::unlock().

Referenced by clipper::Xmap< T >::fft_from().

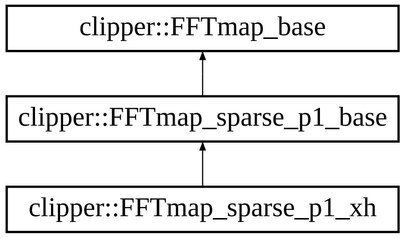
The documentation for this class was generated from the following files:

- fftmap_sparse.h
- fftmap_sparse.cpp

19.69 clipper::FFTmap_sparse_p1_xh Class Reference

FFTmap_sparse_p1_xh (p. 234): low level sparse P1 map used for calculating FFTs.

#include <fftmap_sparse.h>Inheritance diagram for clipper::FFTmap_sparse_p1_xh::



Public Member Functions

- **FFTmap_sparse_p1_xh** ()

  Null constuctor.

- **FFTmap_sparse_p1_xh** (const **Grid_sampling** &grid_sam, const FFTtype type=Default)

  Constructor: takes grid.

- void **init** (const **Grid_sampling** &grid_sam, const FFTtype type=Default)

  initialiser: takes grid

- const **Grid_sampling** & **grid_real** () const

  get real grid sampling

- const **Grid** & **grid_reci** () const

  get reciprocal grid

- ffttype & **real_data** (const **Coord_grid** &uvw)

  *set real space data (uvw must be in **grid_real()** (p. 234))*

- void **require_hkl** (const **HKL** &hkl)

  express need for reciprocal space data by hkl

- const std::complex< ffttype > **get_hkl** (const **HKL** &hkl) const

  get reciprocal space data by hkl

- void **require_cplx_data** (const **Coord_grid** &hkl)

  express need for reciprocal space data (internal use)

- const std::complex< ffttype > & **cplx_data** (const **Coord_grid** &hkl) const

  get reciprocal space data (internal use)

- void **fft_x_to_h** (const **ftype** &scale)

  Transform to real space.

19.69.1 Detailed Description

FFTmap_sparse_p1_xh (p. 234): low level sparse P1 map used for calculating FFTs. This version computes sparse real...Hermititan FFTs.

By specifying what parts of the map are needed in advance, it is possible to perform highly optimised FFTs, including some of the benefits of symmetry.

19.69.2 Constructor & Destructor Documentation

19.69.2.1 clipper::FFTmap_sparse_p1_xh::FFTmap_sparse_p1_xh ()

Null constuctor. For later initialisation: see **init()** (p. 235)

19.69.2.2 clipper::FFTmap_sparse_p1_xh::FFTmap_sparse_p1_xh (const Grid_sampling & *grid_sam*, const FFTtype *type* = `Default`)

Constructor: takes grid. Construct an **FFTmap_sparse_p1_xh** (p. 234) for a given grid.

Parameters:

grid_sam The grid sampling of the unit cell.

type Can be FFTmap_sparse_base::Measure, Estimate. Measure performs slow precalculation (first time only) to get faster FFT.

References init().

19.69.3 Member Function Documentation

19.69.3.1 void clipper::FFTmap_sparse_p1_xh::init (const Grid_sampling & *grid_sam*, const FFTtype *type* = `Default`)

initialiser: takes grid Initialise an **FFTmap_sparse_p1_base** (p. 228) for a grid.

Parameters:

grid_sam The grid sampling of the unit cell.

type Can be FFTmap_sparse_base::Measure, Estimate. Measure performs slow precalculation (first time only) to get faster FFT.

Reimplemented from **clipper::FFTmap_sparse_p1_base** (p. 229).

Referenced by FFTmap_sparse_p1_xh().

19.69.3.2 ffttype & clipper::FFTmap_sparse_p1_xh::real_data (const Coord_grid & *uvw*) `[inline]`

set real space data (uvw must be in **grid_real()** (p. 234)) (uvw must be in **grid_real()** (p. 234))

References clipper::FFTmap_sparse_p1_base::map_uv(), clipper::Coord_grid::u(), clipper::Coord_grid::v(), and clipper::Coord_grid::w().

Referenced by clipper::Xmap< T >::fft_to().

19.69.3.3 void clipper::FFTmap_sparse_p1_xh::require_hkl (const HKL & *hkl*)

express need for reciprocal space data by hkl Friedel opposites are handled correctly

Parameters:

hkl The **HKL** (p. 261) required.

References clipper::FFTmap_sparse_p1_base::grid_real_, clipper::FFTmap_sparse_p1_-base::grid_reci_, clipper::FFTmap_sparse_p1_base::map_kl(), clipper::Grid_sampling::nv(), clipper::Grid_sampling::nw(), clipper::Grid::nw(), clipper::Coord_grid::v(), and clipper::Coord_-grid::w().

Referenced by clipper::Xmap< T >::fft_to().

19.69.3.4 const std::complex< ffttype > clipper::FFTmap_sparse_p1_xh::get_hkl (const HKL & *hkl*) const

get reciprocal space data by hkl Friedel opposites are handled correctly

Parameters:

hkl The required.

References clipper::FFTmap_sparse_p1_base::grid_real_, clipper::FFTmap_sparse_-p1_base::grid_reci_, clipper::Grid_sampling::nu(), clipper::Grid_sampling::nv(), clipper::Grid_sampling::nw(), clipper::Grid::nw(), clipper::FFTmap_sparse_p1_base::row_kl, clipper::Coord_grid::u(), clipper::Coord_grid::v(), and clipper::Coord_grid::w().

Referenced by clipper::Xmap< T >::fft_to().

19.69.3.5 void clipper::FFTmap_sparse_p1_xh::require_cplx_data (const Coord_grid & *hkl*) `[inline]`

express need for reciprocal space data (internal use) The given **Coord_grid** (p. 175) will be required in the final reflections. (uvw must be in **grid_reci()** (p. 234))

Parameters:

uvw The coordinate to require.

References clipper::FFTmap_sparse_p1_base::map_kl(), clipper::Coord_grid::v(), and clipper::Coord_grid::w().

19.69.3.6 const std::complex< ffttype > & clipper::FFTmap_-sparse_p1_xh::cplx_data (const Coord_grid & *hkl*) const `[inline]`

get reciprocal space data (internal use) (hkl must be in **grid_reci()** (p. 234), and have been requested)

Parameters:

uvw The coordinate to get.

Returns:

The complex value at that coordinate.

References clipper::FFTmap_sparse_p1_base::row_kl, clipper::Coord_grid::u(), clipper::Coord_grid::v(), and clipper::Coord_grid::w().

19.69.3.7 void clipper::FFTmap_sparse_p1_xh::fft_x_to_h (const ftype & *scale*)

Transform to real space. The 'require' functions must have been called first to mark the required data in the target space. (Source space requirements are inferred automatically).

References clipper::FFTmap_sparse_p1_base::grid_real_, clipper::FFTmap_sparse_p1_-base::grid_reci_, clipper::Mutex::lock(), clipper::FFTmap_base::mutex, clipper::Grid::nu(), clipper::Grid_sampling::nu(), clipper::Grid_sampling::nv(), clipper::Grid::nv(), clipper::Grid::nw(), clipper::Grid_sampling::nw(), clipper::FFTmap_sparse_p1_base::row_kl, clipper::FFTmap_sparse_p1_base::row_uv, clipper::Grid_sampling::size(), clipper::FFTmap_-sparse_p1_base::type_, and clipper::Mutex::unlock().

Referenced by clipper::Xmap< T >::fft_to().

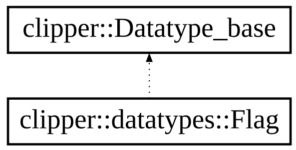
The documentation for this class was generated from the following files:

- fftmap_sparse.h
- fftmap_sparse.cpp

19.70 clipper::datatypes::Flag Class Reference

Reflection data type: Free-R flag.

`#include <hkl_datatypes.h>`Inheritance diagram for clipper::datatypes::Flag::



Public Member Functions

- **Flag** (const int &flag)
- void **set_null** ()

  initialise to 'missing' (all elements are set to null)

- void **friedel** ()

  apply Friedel transformation to the data

- void **shift_phase** (const **ftype** &dphi)

  apply phase shift to the data

- bool **missing** () const

  return true if data is missing

- void **data_export** (**xtype** array[ ]) const

  conversion to array (for I/O)

- void **data_import** (const **xtype** array[ ])

  conversion from array (for I/O)

- const int & **flag** () const
- int & **flag** ()

Static Public Member Functions

- static **String** **type** ()

  return the name of this data type

- static int **data_size** ()

  return number of data elements in this type

- static **String** **data_names** ()

  return names of data elements in this type

19.70.1 Detailed Description

Reflection data type: Free-R flag.

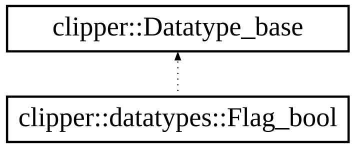
The documentation for this class was generated from the following file:

- hkl_datatypes.h

19.71 clipper::datatypes::Flag_bool Class Reference

Reflection data type: boolean (false = missing).

`#include <hkl_datatypes.h>`Inheritance diagram for clipper::datatypes::Flag_bool::



Public Member Functions

- void **set_null** ()

  initialise to 'missing' (all elements are set to null)

- void **friedel** ()

  apply Friedel transformation to the data

- void **shift_phase** (const **ftype** &dphi)

  apply phase shift to the data

- bool **missing** () const

  return true if data is missing

- void **data_export** (**xtype** array[ ]) const

  conversion to array (for I/O)

- void **data_import** (const **xtype** array[ ])

  conversion from array (for I/O)

- const bool & **flag** () const
- bool & **flag** ()

Static Public Member Functions

- static **String type** ()

  return the name of this data type

- static int **data_size** ()

  return number of data elements in this type

- static **String data_names** ()

  return names of data elements in this type

19.71.1 Detailed Description

Reflection data type: boolean (false = missing).

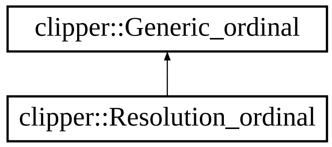
The documentation for this class was generated from the following file:

- hkl_datatypes.h

19.72 clipper::Generic_ordinal Class Reference

Generic ordinal gernerator.

`#include <clipper_stats.h>`Inheritance diagram for clipper::Generic_ordinal::



Public Member Functions

- **Generic_ordinal** ()
  null constructor
- **Generic_ordinal** (const **Range**< **ftype** > &range, const int &n)
  constructor: from range and sampling
- void **init** (const **Range**< **ftype** > &range, const int num_ranges=1000)
  initialiser: takes the source range and sampling
- void **init** (const std::vector< **ftype** > &values, const int num_ranges=1000)
  initialiser: takes the source distibution and a number of bins
- **ftype ordinal** (const **ftype** &value) const
  return reflection ordinal
- void **accumulate** (const **ftype** &value)
  accumulate values to build the distribution
- void **accumulate** (const **ftype** &value, const **ftype** &weight)
  accumulate values to build the distribution
- void **prep_ordinal** ()
  generate the ordinal histogram
- void **invert** ()
  invert distribution to get value from ordinal
- void **init** (const int num_ranges=1000)
  DEPRECATED: initialiser: takes a number of bins for histogram.
- void **add_pass_1** (const **ftype** &value)
  DEPRECATED: add a value to the distribution (pass 1 of 2).
- void **add_pass_2** (const **ftype** &value)
  DEPRECATED: add a value to the distribution (pass 2 of 2).

Protected Attributes

- **ftype nranges**
  number of ranges
- **Range< ftype > range_**
  resolution range of data
- std::vector< **ftype** > **hist**
  histogram of reflections vs resolution

19.72.1 Detailed Description

Generic ordinal gernerator. This is a generic fast ordinal calculator. It is supplied with a list of values, from which it prepares a cumulative distribution function. This may the used to return the approximate fracitonal ordinal (in the range 0...1) for any given value from the distibution.

The distibution may be initialised by providing a vector of values from the distribution, or by adding the values and calling **prep_ordinal()** (p. 242).

This distribution may also be inverted. Generation of a value from an ordinal may be used for generating random values from a given distribution, or for histogram matching.

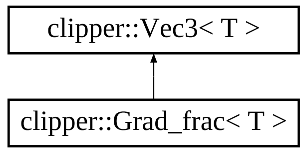
The documentation for this class was generated from the following files:

- clipper_stats.h
- clipper_stats.cpp

19.73 clipper::Grad_frac< T > Class Template Reference

fractional (cell) gradient, with respect to fractional u,v,w

`#include <derivs.h>`Inheritance diagram for clipper::Grad_frac< T >::



Public Member Functions

- **Grad_frac** ()
  null constructor
- **Grad_frac** (const **Vec3**< T > &v)
  constructor: copy/convert
- **Grad_frac** (const T &du, const T &dv, const T &dw)
  constructor: from d/du,d/dv,d/dw
- const T & **du** () const
  get d/du
- const T & **dv** () const
  get d/dv
- const T & **dw** () const
  get d/dw
- **Grad_orth**< T > **grad_orth** (const **Cell** &cell) const
  fractional-orthogonal derivative conversion
- **Grad_map**< T > **grad_map** (const **Grid** &g) const
  fractional-grid derivative conversion
- **String format** () const
  *return formatted **String** (p. 426) representation*

19.73.1 Detailed Description

template<class T> class clipper::Grad_frac< T >

fractional (cell) gradient, with respect to fractional u,v,w

19.73.2 Member Function Documentation

19.73.2.1 template<class T > Grad_orth< T > clipper::Grad_frac< T >::grad_orth (const Cell & *cell*) const [inline]

fractional-orthogonal derivative conversion

Parameters:

cell The cell concerned

Returns:

The transformed derivative.

References clipper::Cell::matrix_frac().

19.73.2.2 template<class T > Grad_map< T > clipper::Grad_frac< T >::grad_map (const Grid & *g*) const [inline]

fractional-grid derivative conversion

Parameters:

g The grid concerned

Returns:

The transformed derivative.

References clipper::Grad_frac< T >::du(), clipper::Grad_frac< T >::dv(), clipper::Grad_frac< T >::dw(), clipper::Grid::nu(), clipper::Grid::nv(), and clipper::Grid::nw().

19.73.2.3 template<class T > String clipper::Grad_frac< T >::format () const [inline]

return formatted **String** (p. 426) representation

Returns:

The formatted text string

Reimplemented from **clipper::Vec3< T >** (p. 461).

References clipper::Grad_frac< T >::du(), clipper::Grad_frac< T >::dv(), and clipper::Grad_frac< T >::dw().

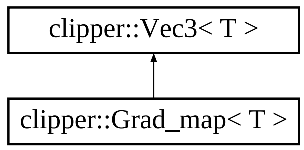
The documentation for this class was generated from the following file:

- derivs.h

19.74 clipper::Grad_map< T > Class Template Reference

map coordinate gradient, with respect to grid u,v,w

`#include <derivs.h>`Inheritance diagram for clipper::Grad_map< T >::



Public Member Functions

- **Grad_map** ()
  null constructor
- **Grad_map** (const **Vec3**< T > &v)
  constructor: copy/convert
- **Grad_map** (const T &du, const T &dv, const T &dw)
  constructor: from d/du,d/dv,d/dw
- const T & **du** () const
  get d/du
- const T & **dv** () const
  get d/dv
- const T & **dw** () const
  get d/dw
- **Grad_frac**< T > **grad_frac** (const **Grid** &g) const
  grid-fractional derivative conversion
- **String format** () const
  return formatted ***String*** *(p. 426) representation*

19.74.1 Detailed Description

template<class T> class clipper::Grad_map< T >

map coordinate gradient, with respect to grid u,v,w

19.74.2 Member Function Documentation

19.74.2.1 template<class T > Grad_frac< T > clipper::Grad_map< T >::grad_frac (const Grid & *g*) const `[inline]`

grid-fractional derivative conversion

Parameters:

g The grid concerned

Returns:

The transformed derivative.

References clipper::Grad_map< T >::du(), clipper::Grad_map< T >::dv(), clipper::Grad_map< T >::dw(), clipper::Grid::nu(), clipper::Grid::nv(), and clipper::Grid::nw().

Referenced by clipper::Xmap< T >::interp_grad().

19.74.2.2 template<class T > String clipper::Grad_map< T >::format () const `[inline]`

return formatted **String** (p. 426) representation

Returns:

The formatted text string

Reimplemented from **clipper::Vec3< T >** (p. 461).

References clipper::Grad_map< T >::du(), clipper::Grad_map< T >::dv(), and clipper::Grad_-map< T >::dw().

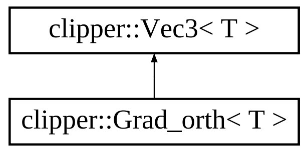
The documentation for this class was generated from the following file:

- derivs.h

19.75 clipper::Grad_orth< T > Class Template Reference

orthogonal (Angstom) gradient, with respect to orthogonal x,y,z

`#include <derivs.h>`Inheritance diagram for clipper::Grad_orth< T >::



Public Member Functions

- **Grad_orth** ()

  null constructor

- **Grad_orth** (const **Vec3**< T > &v)

  constructor: copy/convert

- **Grad_orth** (const T &dx, const T &dy, const T &dz)

  constructor: from d/dx,d/dy,d/dz

- const T & **dx** () const

  get d/dx

- const T & **dy** () const

  get d/dy

- const T & **dz** () const

  get d/dz

- **Grad_frac**< T > **grad_frac** (const **Cell** &cell) const

  orthogonal-fractional derivative conversion

- **String format** () const

  *return formatted **String** (p. 426) representation*

19.75.1 Detailed Description

template<class T> class clipper::Grad_orth< T >

orthogonal (Angstom) gradient, with respect to orthogonal x,y,z

19.75.2 Member Function Documentation

19.75.2.1 template<class T > Grad_frac< T > clipper::Grad_orth< T >::grad_frac (const Cell & *cell*) const `[inline]`

orthogonal-fractional derivative conversion

Parameters:

cell The cell concerned

Returns:

The transformed derivative.

References clipper::Cell::matrix_orth().

19.75.2.2 template<class T > String clipper::Grad_orth< T >::format () const `[inline]`

return formatted **String** (p. 426) representation The result is an RT operator. This is a redudent representation, but is handy for assembling compound operators.

Returns:

The operator
The formatted text string

Reimplemented from **clipper::Vec3< T >** (p. 461).

References clipper::Grad_orth< T >::dx(), clipper::Grad_orth< T >::dy(), and clipper::Grad_orth< T >::dz().

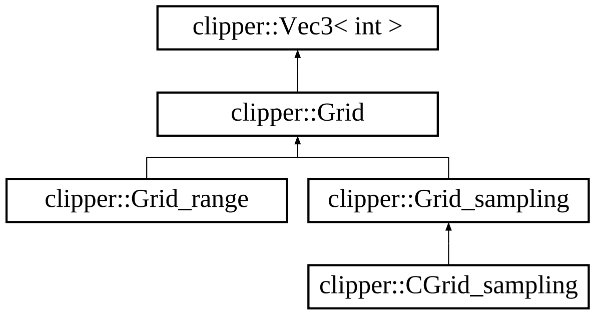
The documentation for this class was generated from the following file:

- derivs.h

19.76 clipper::Grid Class Reference

generic grid

`#include <coords.h>`Inheritance diagram for clipper::Grid::



Public Member Functions

- **Grid** ()

  null constructor

- **Grid** (const int &nu, const int &nv, const int &nw)

  constructor: from nu,nv,nw

- const int & **nu** () const

  get nu

- const int & **nv** () const

  get nv

- const int & **nw** () const

  get nw

- int **size** () const

  return size of grid array

- bool **in_grid** (**Coord_grid** g) const

  determine if a point is in the grid

- int **index** (const **Coord_grid** &c) const

  grid indexing operator

- **Coord_grid deindex** (const int &index) const

  grid deindexing operator

- **String format** () const

  return formatted ***String*** *(p. 426) representation*

- void **debug** () const

19.76.1 Detailed Description

generic grid This holds the dimensions of a 3D array, indexed from 0 along each dimension.

19.76.2 Member Function Documentation

19.76.2.1 String clipper::Grid::format () const

return formatted **String** (p. 426) representation

Returns:

The formatted text string

Reimplemented from **clipper::Vec3**< **int** > (p. 461).

Reimplemented in **clipper::Grid_sampling** (p. 258).

References nu(), nv(), and nw().

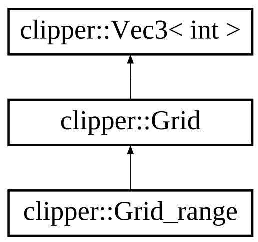
The documentation for this class was generated from the following files:

- coords.h
- coords.cpp

19.77 clipper::Grid_range Class Reference

Grid (p. 250) range class: defines array limits for a grid.

`#include <coords.h>`Inheritance diagram for clipper::Grid_range::



Public Member Functions

- **Grid_range** ()
  null constructor
- **Grid_range** (const **Coord_grid** &min, const **Coord_grid** &max)
  constructor: takes grid limits
- **Grid_range** (const **Grid** &grid, const **Coord_frac** &min, const **Coord_frac** &max)
  constructor: takes cell grid and fractional limits
- **Grid_range** (const **Cell** &cell, const **Grid** &grid, const **ftype** &radius)
  constructor: make grid to hold a sphere from cell, grid, radius
- const **Coord_grid** & **min** () const
  access grid limits
- const **Coord_grid** & **max** () const
  access grid limits
- void **add_border** (const int b)
  border: increase grid to include given border
- bool **in_grid** (**Coord_grid** g) const
  determine if a point is in the grid
- int **index** (const **Coord_grid** &c) const
  grid indexing operator
- **Coord_grid deindex** (const int &index) const
  grid deindexing operator

19.77.1 Detailed Description

Grid (p. 250) range class: defines array limits for a grid. This class is used for describing 3D grids covering an arbitrary part of the 3D space, i.e. which do not start from (0,0,0).

19.77.2 Constructor & Destructor Documentation

19.77.2.1 clipper::Grid_range::Grid_range (const Coord_grid & *min*, const Coord_grid & *max*)

constructor: takes grid limits Make a map grid with an oblong bounded by the coordinates min and max.

Parameters:

- ***min*** The lower bound coordinate in u,v,w.
- ***max*** The upper bound coordinate in u,v,w.

References clipper::Coord_grid::u(), clipper::Coord_grid::v(), and clipper::Coord_grid::w().

19.77.2.2 clipper::Grid_range::Grid_range (const Grid & *g*, const Coord_frac & *min*, const Coord_frac & *max*)

constructor: takes cell grid and fractional limits Make a map grid with an oblong bounded by the fractional coordinates min and max, when the sampling of the cell is g

Parameters:

- ***g*** The grid sampling of the whole unit cell.
- ***min*** The lower bound coordinate in u,v,w.
- ***max*** The upper bound coordinate in u,v,w.

References clipper::Util::intc(), clipper::Util::intf(), clipper::Grid::nu(), clipper::Grid::nv(), clipper::Grid::nw(), clipper::Coord_grid::u(), clipper::Coord_frac::u(), clipper::Coord_grid::v(), clipper::Coord_frac::v(), clipper::Coord_grid::w(), and clipper::Coord_frac::w().

19.77.2.3 clipper::Grid_range::Grid_range (const Cell & *cell*, const Grid & *grid*, const ftype & *radius*)

constructor: make grid to hold a sphere from cell, grid, radius Make a map grid large enough to fully enclose a sphere about the origin of a given radius with a given cell and grid sampling.

Parameters:

- ***cell*** The cell parameters.
- ***grid*** The grid sampling of the whole cell.
- ***radius*** The radius of the sphere in Angstroms.

References clipper::Cell::a_star(), clipper::Cell::b_star(), clipper::Cell::c_star(), clipper::Util::intc(), clipper::Grid::nu(), clipper::Grid::nv(), clipper::Grid::nw(), clipper::Coord_grid::u(), clipper::Coord_grid::v(), and clipper::Coord_grid::w().

19.77.3 Member Function Documentation

19.77.3.1 void clipper::Grid_range::add_border (const int *b*)

border: increase grid to include given border Enlarge the grid by adding `b` cells in every direction. Will shrink the grid if `b` is negative.

Parameters:

b The number of cells by which to enlarge/shrink.

References clipper::Coord_grid::u(), clipper::Coord_grid::v(), and clipper::Coord_grid::w().

Referenced by clipper::Xmap_cacheobj::Xmap_cacheobj().

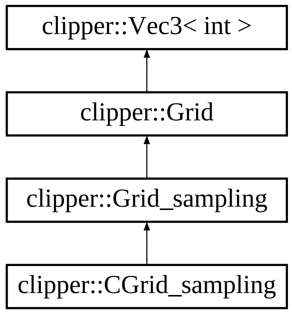
The documentation for this class was generated from the following files:

- coords.h
- coords.cpp

19.78 clipper::Grid_sampling Class Reference

Grid (p. 250) sampling of a unit cell.

`#include <coords.h>`Inheritance diagram for clipper::Grid_sampling::



Public Member Functions

- **Grid_sampling** ()

  null constructor
- **Grid_sampling** (const int &nu, const int &nv, const int &nw)

  constructor: from nu, nv, nw
- **Grid_sampling** (const **Spacegroup** &spacegroup, const **Cell** &cell, const **Resolution** &resol, const **ftype** rate=1.5)

  constructor: from ***Spacegroup*** *(*p. *413),* ***Cell*** *(*p. *133),* ***Resolution*** *(*p. *389), Shannon rate*
- void **init** (const **Spacegroup** &spacegroup, const **Cell** &cell, const **Resolution** &resol, const **ftype** rate=1.5)

  initialiser: from ***Spacegroup*** *(*p. *413),* ***Cell*** *(*p. *133),* ***Resolution*** *(*p. *389), Shannon rate*
- **Mat33 matrix_grid_frac** () const

  return matrix which converts grid to fractional coordinates
- **Mat33 matrix_frac_grid** () const

  return matrix which converts fractional to grid coordinates
- bool **is_null** () const

  test if object has been initialised
- const int & **nu** () const

  get nu
- const int & **nv** () const

  get nv
- const int & **nw** () const

  get nw

- int **size** () const

  return size of grid array

- int **index** (const **Coord_grid** &c) const

  grid indexing operator

- **Coord_grid deindex** (const int &index) const

  grid deindexing operator

- const **String format** () const

  return formatted ***String*** *(p. 426) representation*

19.78.1 Detailed Description

Grid (p. 250) sampling of a unit cell. This class represents the grid sampling of a unit cell. It is otherwise identical to its parent, clipper::Grid_cell, but has an additional constructor which takes a spacegroup, cell and resolution and produces an appropriate grid obeying all of the symmetry constraints, and using efficient factors for the calculation of FFTs.

Note:

The following methods are inherited from **Grid** (p. 250) and Grid_cell but are documented here for convenience: **nu()** (p. 255), **nv()** (p. 255), **nw()** (p. 255), **size()** (p. 256), **index()** (p. 256), **deindex()** (p. 256), **format()** (p. 258), coord_frac(), coord_grid(), to_unit().

19.78.2 Constructor & Destructor Documentation

19.78.2.1 clipper::Grid_sampling::Grid_sampling (const Spacegroup & *spacegroup*, const Cell & *cell*, const Resolution & *resol*, const ftype *rate* = `1.5`)

constructor: from **Spacegroup** (p. 413), **Cell** (p. 133), **Resolution** (p. 389), Shannon rate A grid is chosen to represent the specified cell at the given resolution, obeying any restrictions imposed by the spacegroup. A slightly finer grid may be chosen if doing so is liable to significantly increase the speed of FFTs on that grid.

Parameters:

spacegroup The spacegroup which the grid must obey.

cell The cell which the grid must contain.

resol The resolution to which the grid must sample.

rate The linear Shannon rate (oversampling) required. If rate = 1, the grid spaceing will be half the resolution (the the minimum required). For a grid spaceing of resol/3, use the default rate=1.5.

References init().

19.78.3 Member Function Documentation

19.78.3.1 void clipper::Grid_sampling::init (const Spacegroup & *spacegroup*, const Cell & *cell*, const Resolution & *resol*, const ftype *rate* = 1.5)

initialiser: from **Spacegroup** (p. 413), **Cell** (p. 133), **Resolution** (p. 389), Shannon rate A grid is chosen to represent the specified cell at the given resolution, obeying any restrictions imposed by the spacegroup. A slightly finer grid may be chosen if doing so is liable to significantly increase the speed of FFTs on that grid.

Parameters:

spacegroup The spacegroup which the grid must obey.

cell The cell which the grid must contain.

resol The resolution to which the grid must sample.

rate The linear Shannon rate (oversampling) required. If rate = 1, the grid spaceing will be half the resolution (the the minimum required). For a grid spaceing of resol/3, use the default rate=1.5.

Reimplemented in **clipper::CGrid_sampling** (p. 141).

References clipper::Cell_descr::a(), clipper::Cell_descr::b(), clipper::Cell_descr::c(), clipper::Cell::descr(), clipper::Util::intc(), clipper::Resolution::limit(), clipper::Util::max(), clipper::Util::mod(), clipper::Spacegroup::num_symops(), and clipper::Spacegroup::symop().

Referenced by Grid_sampling(), and clipper::CGrid_sampling::init().

19.78.3.2 Mat33 clipper::Grid_sampling::matrix_grid_frac () const

return matrix which converts grid to fractional coordinates The result is an RT operator. This is a redudent representation, but is handy for assembling compound operators.

Returns:

The operator

References nu(), nv(), and nw().

Referenced by clipper::Xmap_base::init().

19.78.3.3 Mat33 clipper::Grid_sampling::matrix_frac_grid () const

return matrix which converts fractional to grid coordinates The result is an RT operator. This is a redudent representation, but is handy for assembling compound operators.

Returns:

The operator

References nu(), nv(), and nw().

Referenced by clipper::NX_operator::init(), and clipper::NXmap_base::init().

19.78.3.4 bool clipper::Grid_sampling::is_null () const

test if object has been initialised

Returns:

true if the object has not been initalised.

Reimplemented from **clipper::Vec3< int >** (p. 461).

References size().

Referenced by clipper::CNXmap_operator< T >::init(), clipper::CXmap< T >::init(), clipper::Xmap_base::is_null(), and clipper::CGrid_sampling::update().

19.78.3.5 const String clipper::Grid_sampling::format () const

return formatted **String** (p. 426) representation

Returns:

The formatted text string

Reimplemented from **clipper::Grid** (p. 251).

Referenced by clipper::Xmap_cacheobj::format().

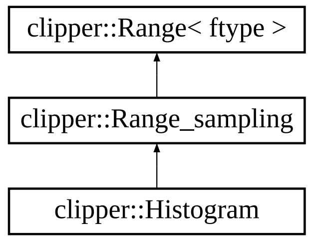
The documentation for this class was generated from the following files:

- coords.h
- coords.cpp

19.79 clipper::Histogram Class Reference

General histogram class.

`#include <clipper_stats.h>`Inheritance diagram for clipper::Histogram::



Public Member Functions

- **Histogram** ()
  null constructor
- **Histogram** (const **Range**< **ftype** > &range, const int &n)
  constructor: from range and sampling
- void **accumulate** (const **ftype** &x)
  add value to histogram (if it is in range)
- void **accumulate** (const **ftype** &x, const **ftype** &w)
  add specified value to histogram (if it is in range)
- **ftype sum** () const
  return sum of whole histogram
- const **ftype** & **y** (const int &i) const
  return value at index in histogram (Note: no bound check on i)
- **ftype y** (const **ftype** &x) const
  return value at interpolated position in histogram
- const **Histogram** & **operator**+= (const **Histogram** &h)
  add the contents of two histograms (size must match)
- **ftype x** (const int &i) const
  return x-value corresponding to centre of i'th range
- **ftype x_min** (const int &i) const
  return x-value corresponding to bottom of i'th range
- **ftype x_max** (const int &i) const
  return x-value corresponding to top of i'th range

- int **size** () const

    return number of samplings in range

19.79.1 Detailed Description

General histogram class. This class is used to accumulate and access a histogram of values spread over a specified range. On storing data or retrieving by interpolation the range is checked.

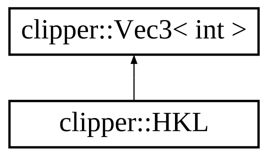
The documentation for this class was generated from the following files:

- clipper_stats.h
- clipper_stats.cpp

19.80 clipper::HKL Class Reference

reflection 'Miller' index

`#include <coords.h>`Inheritance diagram for clipper::HKL::



Public Member Functions

- **HKL** ()
  null constructor
- **HKL** (const **Vec3**< int > &v)
  constructor: copy/convert
- **HKL** (const int &h, const int &k, const int &l)
  constructor: from H,K,L
- const int & **h** () const
  get h
- const int & **k** () const
  get k
- const int & **l** () const
  get l
- int & **h** ()
  set h
- int & **k** ()
  set k
- int & **l** ()
  set l
- **ftype invresolsq** (const **Cell** &cell) const
  return inverse resolution squared for this reflection in given cell
- **Coord_reci_frac coord_reci_frac** () const
  *return fractional reciprocal coordinate (i.e. non-integer **HKL** (p. 261))*
- **Coord_reci_orth coord_reci_orth** (const **Cell** &cell) const
  orthogonal-fractional reciprocal space coordinate conversion

- **HKL transform** (const **Symop** &op) const
  return transformed hkl

- **HKL transform** (const **Isymop** &op) const
  return transformed hkl

- **ftype sym_phase_shift** (const **Symop** &op) const
  *return symmetry phase shift for this **HKL** (p. 261) under op*

- **String format** () const
  *return formatted **String** (p. 426) representation*

Friends

- **HKL operator-** (const **HKL** &h1)
  Vector negation operator.

- **HKL operator+** (const **HKL** &h1, const **HKL** &h2)
  Vector addition operator.

- **HKL operator-** (const **HKL** &h1, const **HKL** &h2)
  Vector subtraction operator.

- **HKL operator*** (const int &s, const **HKL** &h1)
  Vector scaling operator.

- **HKL operator*** (const **Isymop** &op, const **HKL** &h1)

19.80.1 Detailed Description

reflection 'Miller' index

19.80.2 Member Function Documentation

19.80.2.1 ftype clipper::HKL::invresolsq (const Cell & *cell*) const `[inline]`

return inverse resolution squared for this reflection in given cell

Note:

Normally you would get a value through **clipper::HKL_info** (p. 284), unless you specifically want a value for a different cell.

References clipper::Metric_tensor::lengthsq(), and clipper::Cell::metric_reci().

Referenced by clipper::BasisFn_spline::f(), clipper::BasisFn_linear::f(), clipper::BasisFn_-binner::f(), clipper::BasisFn_expcubic::fderiv(), clipper::BasisFn_log_gaussian::fderiv(), clipper::BasisFn_gaussian::fderiv(), clipper::BasisFn_spline::fderiv(), clipper::BasisFn_-linear::fderiv(), clipper::BasisFn_binner::fderiv(), clipper::HKL_info::generate_hkl_list(), clipper::Resolution_ordinal::init(), clipper::HKL_data_base::invresolsq(), clipper::Test_-core::operator()(), clipper::HKL_sampling::resolution(), and clipper::HKL_info::update_hkl_-list().

19.80.2.2 Coord_reci_frac clipper::HKL::coord_reci_frac () const `[inline]`

return fractional reciprocal coordinate (i.e. non-integer **HKL** (p. 261))

Returns:

The non-integer coordinate.

Referenced by coord_reci_orth().

19.80.2.3 Coord_reci_orth clipper::HKL::coord_reci_orth (const Cell & *cell*) const `[inline]`

orthogonal-fractional reciprocal space coordinate conversion

Parameters:

cell The cell concerned

Returns:

The transformed coordinate.

References coord_reci_frac(), and clipper::Coord_reci_frac::coord_reci_orth().

Referenced by clipper::BasisFn_log_aniso_gaussian::fderiv(), clipper::BasisFn_aniso_-gaussian::fderiv(), and clipper::datatypes::Compute_scale_u_aniso< T >::operator()().

19.80.2.4 HKL clipper::HKL::transform (const Symop & *op*) const `[inline]`

return transformed hkl Requires integer->ftype->integer transformation.

Parameters:

op The symmetry operator

Returns:

The transformed coordinate

References transform().

Referenced by clipper::Xmap< T >::fft_from(), clipper::HKL_info::find_sym(), clipper::HKL_-class::HKL_class(), clipper::Test_core::operator()(), clipper::HKL_info::HKL_reference_-coord::set_hkl(), clipper::FFTmap::set_recip_data(), and transform().

19.80.2.5 HKL clipper::HKL::transform (const Isymop & *op*) const `[inline]`

return transformed hkl Optimal version.

Parameters:

op The symmetry operator

Returns:

The transformed coordinate

19.80.2.6 ftype clipper::HKL::sym_phase_shift (const Symop & *op*) const `[inline]`

return symmetry phase shift for this **HKL** (p. 261) under op Get the symmetry phase shift incurred when transforming a reflection by this operator.

Parameters:

hkl The reflection **HKL** (p. 261) to transform.

Returns:

The phase shift.

References clipper::RTop< T >::trn(), and clipper::Util::twopi().

Referenced by clipper::Xmap< T >::fft_from(), clipper::HKL_data< T >::get_data(), clipper::HKL_class::HKL_class(), clipper::HKL_data< T >::operator[](), clipper::HKL_data< T >::set_data(), and clipper::FFTmap::set_recip_data().

19.80.2.7 String clipper::HKL::format () const

return formatted **String** (p. 426) representation

Returns:

The formatted text string

Reimplemented from **clipper::Vec3**< **int** > (p. 461).

References h(), k(), and l().

The documentation for this class was generated from the following files:

- coords.h
- coords.cpp

19.81 clipper::HKL_class Class Reference

reflection class

`#include <coords.h>`

Public Member Functions

- **HKL_class** ()
  null constructor
- **HKL_class** (const **Spacegroup** &spgr, const **HKL** &hkl)
  *constructor - from spacegroup and **HKL*** *(p. 261)*
- **ftype epsilon** () const
  get epsilon
- **ftype epsilonc** () const
  get epsilon for acentric, 2x epsilon for centric
- **ftype allowed** () const
  get allowed phase
- bool **centric** () const
  is centric?
- bool **sys_abs** () const
  is sys abs?

19.81.1 Detailed Description

reflection class This describes the type of a reflection in a given spacegroup, including centricity, systematic absence, phase restriction, and multiplicity.

19.81.2 Constructor & Destructor Documentation

19.81.2.1 clipper::HKL_class::HKL_class (const Spacegroup & *spgr*, const HKL & *hkl*)

constructor - from spacegroup and **HKL** (p. 261) Determine the class of a reflection for a give spacegroup.

Parameters:

spgr The spacegroup.

hkl The reflection **HKL** (p. 261)

References clipper::HKL::h(), clipper::Util::intr(), clipper::HKL::k(), clipper::HKL::l(), clipper::Util::mod(), clipper::Spacegroup::num_symops(), clipper::Util::pi(), clipper::HKL::sym_phase_shift(), clipper::Spacegroup::symop(), and clipper::HKL::transform().

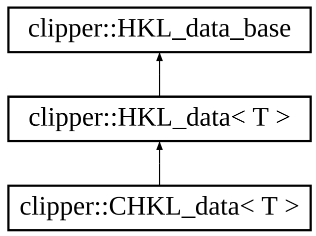
The documentation for this class was generated from the following files:

- coords.h
- coords.cpp

19.82 clipper::HKL_data< T > Class Template Reference

HKL_data<>.

`#include` <**hkl_data.h**>Inheritance diagram for clipper::HKL_data< T >::



Public Member Functions

- **HKL_data** ()

  null constructor

- **HKL_data** (const **HKL_info** &hkl_info)

  constructor: from parent hkl_info

- **HKL_data** (const **HKL_info** &hkl_info, const **Cell** &cell)

  constructor: from parent hkl_info and cell

- **HKL_data** (const **Spacegroup** &spacegroup, const **Cell** &cell, const **HKL_sampling** &hkl_sampling)

  [CLIPPER2] constructor: from spacegroup, cell and hkl_sampling

- **HKL_data** (const **HKL_data_base** &hkl_data)

  *[CLIPPER2] constructor: from another **HKL_data** (p. 267) object*

- void **init** (const **HKL_info** &hkl_info, const **Cell** &cell)

  initialiser: from parent hkl_info and cell

- void **init** (const **Spacegroup** &spacegroup, const **Cell** &cell, const **HKL_sampling** &hkl_sampling)

  *[CLIPPER2] initialiser: from spacegroup, cell, and **HKL_sampling** (p. 298)*

- void **init** (const **HKL_data_base** &hkl_data)

  *[CLIPPER2] initialiser: from another **HKL_data** (p. 267) object*

- void **update** ()

  *update: synchornize info with parent **HKL_info** (p. 284)*

- **String type** () const

  get data type (a list of names corresponding to the im/export values)

- bool **missing** (const int &index) const

  check if a data entry in the list is marked as 'missing'

- void **set_null** (const int &index)
  set data entry in the list to its null value
- int **data_size** () const
  return number of data elements in this type
- **String data_names** () const
  return names of data elements in this type
- void **data_export** (const **HKL** &hkl, **xtype** array[ ]) const
  conversion to array (for I/O)
- void **data_import** (const **HKL** &hkl, const **xtype** array[ ])
  conversion from array (for I/O)
- void **mask** (const **HKL_data_base** &mask)
- const T & **operator[ ]** (const **HKL_info::HKL_reference_index** &i) const
  get data by reflection HKL_reference_index
- T & **operator[ ]** (const **HKL_info::HKL_reference_index** &i)
  set data by reflection HKL_reference_index
- T **operator[ ]** (const **HKL_info::HKL_reference_coord** &ih) const
  get data by HKL_reference_coord
- bool **get_data** (const **HKL_info::HKL_reference_coord** &ih, T &data) const
  get data by HKL_reference_coord (returns false if no equivalent hkl)
- bool **set_data** (const **HKL_info::HKL_reference_coord** &ih, const T &data)
  set data by HKL_reference_coord (returns false if no equivalent hkl)
- const T & **operator[ ]** (const int &index) const
  get data by reflection index
- T & **operator[ ]** (const int &index)
  set data by reflection index
- T **operator[ ]** (const **HKL** &hkl) const
  get data by hkl (returns missing if no equivalent hkl)
- bool **get_data** (const **HKL** &hkl, T &data) const
  get data by hkl (returns false if no equivalent hkl)
- bool **set_data** (const **HKL** &hkl, const T &data)
  set data by hkl (returns false if no equivalent hkl)
- template<class C >
  void **compute** (const C &op)
  Basic computation: fill this data list by function call.

- template<class S , class C >
  void **compute** (const **HKL_data**< S > &src, const C &op)

  Unary computation: fill this data list by computation from another.

- template<class S1 , class S2 , class C >
  void **compute** (const **HKL_data**< S1 > &src1, const **HKL_data**< S2 > &src2, const C &op)

  Binary computation: fill this data list by computation from another.

- const **HKL_info** & **base_hkl_info** () const

  get the parent ***HKL_info*** *(*p. *284) object*

- const **Cell** & **base_cell** () const

  get the parent cell

- const **ftype** **invresolsq** (const int &index) const

  get resolution by reflection index (based on true cell)

- const **Range**< **ftype** > **invresolsq_range** () const

  get resolution limits of the list (based on true cell and missing data)

- const int **num_obs** () const

  get number of observations in this list (based on missing data)

- **HKL_reference_index** **first** () const

  return HKL_reference_index pointing to first reflection

- **HKL_reference_index** **first_data** () const

  return HKL_reference_index pointing to first non-missing data

- **HKL_reference_index** & **next_data** (**HKL_reference_index** &ih) const

  increment HKL_reference_index to next non-missing data

- **HKL_data**< T > & **operator=** (const **HKL_data**< T > &other)

  assignment operator: copies the data from another list

- **HKL_data**< T > & **operator=** (const T &value)

  assignment operator: assigns a single value to the whole list

- void **debug** () const

Protected Attributes

- std::vector< T > **list**

19.82.1 Detailed Description

template<class T> class clipper::HKL_data< T >

HKL_data<>. An actual hkl_data object, containing actual data of type T. This implements the generic interface, and in addition provides type-specific access functions.

Note:

The following methods are inherited from **HKL_data_base** (p. 277) but are documented here for convenience: **base_hkl_info()** (p. 269), **base_cell()** (p. 269), **invresolsq()** (p. 274), **invresolsq_range()** (p. 275), **num_obs()** (p. 275), **first()** (p. 275), **first_data()** (p. 275), **next_data()** (p. 275).

19.82.2 Constructor & Destructor Documentation

19.82.2.1 template<class T > clipper::HKL_data< T >::HKL_data (const HKL_info & *hkl_info*) `[inline, explicit]`

constructor: from parent hkl_info Construct the object using a given reflection list and cell.

Parameters:

hkl_info The reflection list object.

References clipper::HKL_info::cell(), and clipper::HKL_data< T >::init().

19.82.2.2 template<class T > clipper::HKL_data< T >::HKL_data (const HKL_info & *hkl_info*, const Cell & *cell*) `[inline]`

constructor: from parent hkl_info and cell Construct the object using a given reflection list and cell.

Parameters:

hkl_info The reflection list object.

cell The unit cell for this datalist.

References clipper::HKL_data< T >::init().

19.82.2.3 template<class T > clipper::HKL_data< T >::HKL_data (const Spacegroup & *spacegroup*, const Cell & *cell*, const HKL_sampling & *hkl_sampling*) `[inline]`

[CLIPPER2] constructor: from spacegroup, cell and hkl_sampling Construct the object using a given spacegroup, cell, and sampling.

Parameters:

spacegroup The spacegroup for this datalist.

cell The unit cell for this datalist.

hkl_sampling The reflection list description.

References clipper::HKL_data< T >::init().

19.82.2.4 template<class T > clipper::HKL_data< T >::HKL_data (const HKL_data_base & *hkl_data*) [inline, explicit]

[CLIPPER2] constructor: from another **HKL_data** (p. 267) object Construct the object using a given **HKL_data** (p. 267) object. The properties of the object (spacegroup, cell, sampling) are the copied, but the actual data is not.

Parameters:

hkl_data The **HKL_data** (p. 267) object to provide the data.

References clipper::HKL_data< T >::init().

19.82.3 Member Function Documentation

19.82.3.1 template<class T > void clipper::HKL_data< T >::init (const HKL_info & *hkl_info*, const Cell & *cell*) [inline, virtual]

initialiser: from parent hkl_info and cell Initialise the object using a given reflection list and cell.

Parameters:

hkl_info The reflection list object.

cell The unit cell for this datalist.

Reimplemented from **clipper::HKL_data_base** (p. 279).

Reimplemented in **clipper::CHKL_data< T >** (p. 144).

References clipper::HKL_data< T >::update().

Referenced by clipper::HKL_data< T >::HKL_data(), clipper::HKL_data< T >::init(), clipper::HKL_data< T >::operator=(), and clipper::data::Test_data::Test_data().

19.82.3.2 template<class T > void clipper::HKL_data< T >::init (const Spacegroup & *spacegroup*, const Cell & *cell*, const HKL_sampling & *hkl_sampling*) [inline, virtual]

[CLIPPER2] initialiser: from spacegroup, cell, and **HKL_sampling** (p. 298) Initialise the object using a given spacegroup, cell, and sampling.

Parameters:

spacegroup The spacegroup for this datalist.

cell The unit cell for this datalist.

hkl_sampling The reflection list description.

Reimplemented from **clipper::HKL_data_base** (p. 280).

Reimplemented in **clipper::CHKL_data< T >** (p. 143).

References clipper::HKL_data< T >::init(), and clipper::HKL_data< T >::update().

19.82.3.3 template<class T > void clipper::HKL_data< T >::init (const HKL_data_base & *hkl_data*) [inline]

[CLIPPER2] initialiser: from another **HKL_data** (p. 267) object Initialise the object using a given **HKL_data** (p. 267) object. The properties of the object (spacegroup, cell, sampling) are the copied, but the actual data is not.

Parameters:

hkl_data The **HKL_data** (p. 267) object to provide the data.

References clipper::HKL_data< T >::init(), and clipper::HKL_data< T >::update().

19.82.3.4 template<class T > void clipper::HKL_data< T >::update () [inline, virtual]

update: synchornize info with parent **HKL_info** (p. 284) The datalist is resized if necessary to match the parent.

Implements **clipper::HKL_data_base** (p. 278).

Reimplemented in **clipper::CHKL_data< T >** (p. 144).

References clipper::HKL_info::num_reflections().

Referenced by clipper::HKL_data< T >::init().

19.82.3.5 template<class T > void clipper::HKL_data< T >::mask (const HKL_data_base & *mask*) [inline]

For each data element, if the corresponding element in `mask` is missing, then that element in this list is also set to missing.

Parameters:

mask The list to provide the mask.

References clipper::HKL_data_base::missing().

19.82.3.6 template<class T > T clipper::HKL_data< T >::operator[] (const HKL_info::HKL_reference_coord & *ih*) const [inline]

get data by HKL_reference_coord If a symmetry mate of the requested **HKL** (p. 261) exists in the list, then the correct symmetry transformations are applied and the data is returned, otherwise the value of 'missing' for the datatype is returned.

Parameters:

ih The reference to the **HKL** (p. 261).

Returns:

The data, or 'missing'.

References clipper::HKL_info::HKL_reference_coord::friedel(), clipper::HKL_info::HKL_-reference_coord::hkl(), clipper::HKL_info::HKL_reference_coord::index(), clipper::HKL_-info::spacegroup(), clipper::HKL_info::HKL_reference_coord::sym(), clipper::HKL::sym_-phase_shift(), and clipper::Spacegroup::symop().

19.82.3.7 template<class T> bool clipper::HKL_data< T >::get_data (const HKL_info::HKL_reference_coord & *ih*, T & *data*) const [inline]

get data by HKL_reference_coord (returns false if no equivalent hkl) If a symmetry mate of the requested **HKL** (p. 261) exists in the list, then the correct symmetry transformations are applied and the data is returned, otherwise the value of 'missing' for the datatype is returned.

Parameters:

ih The reference to the **HKL** (p. 261).

data Returned with the value of the data.

Returns:

true if the data was returned.

References clipper::HKL_info::HKL_reference_coord::friedel(), clipper::HKL_info::HKL_-reference_coord::hkl(), clipper::HKL_info::HKL_reference_coord::index(), clipper::HKL_-info::spacegroup(), clipper::HKL_info::HKL_reference_coord::sym(), clipper::HKL::sym_-phase_shift(), and clipper::Spacegroup::symop().

Referenced by clipper::HKL_data< datatypes::ABCD< float > >::data_export().

19.82.3.8 template<class T> bool clipper::HKL_data< T >::set_data (const HKL_info::HKL_reference_coord & *ih*, const T & *data*) [inline]

set data by HKL_reference_coord (returns false if no equivalent hkl) If a symmetry mate of the requested **HKL** (p. 261) exists in the list, then the correct symmetry transformations are applied and data is set to the supplied values, otherwise the function returns false.

Parameters:

ih The reference to the **HKL** (p. 261).

data Value of the data to set.

Returns:

true if the data was set.

References clipper::HKL_info::HKL_reference_coord::friedel(), clipper::HKL_info::HKL_-reference_coord::hkl(), clipper::HKL_info::HKL_reference_coord::index(), clipper::HKL_-info::spacegroup(), clipper::HKL_info::HKL_reference_coord::sym(), clipper::HKL::sym_-phase_shift(), and clipper::Spacegroup::symop().

Referenced by clipper::HKL_data< datatypes::ABCD< float > >::data_import(), and clipper::data::Test_data::Test_data().

19.82.3.9 template<class T > T clipper::HKL_data< T >::operator[] (const HKL & *hkl*) const [inline]

get data by hkl (returns missing if no equivalent hkl) If a symmetry mate of the requested **HKL** (p. 261) exists in the list, then the correct symmetry transformations are applied and the data is returned, otherwise the value of 'missing' for the datatype is returned.

Parameters:

hkl The reflection **HKL** (p. 261).

Returns:

The data, or 'missing'.

References clipper::HKL_info::index_of(), clipper::HKL_info::spacegroup(), clipper::HKL::sym_phase_shift(), and clipper::Spacegroup::symop().

19.82.3.10 template<class T> bool clipper::HKL_data< T >::get_data (const HKL & *hkl*, T & *data*) const [inline]

get data by hkl (returns false if no equivalent hkl) If a symmetry mate of the requested **HKL** (p. 261) exists in the list, then the correct symmetry transformations are applied and the supplied datatype is set, otherwise the function returns false.

Parameters:

hkl The reflection **HKL** (p. 261).

data Returned with the value of the data.

Returns:

true if the data was returned.

References clipper::HKL_info::index_of(), clipper::HKL_info::spacegroup(), clipper::HKL::sym_phase_shift(), and clipper::Spacegroup::symop().

19.82.3.11 template<class T> bool clipper::HKL_data< T >::set_data (const HKL & *hkl*, const T & *data_*) [inline]

set data by hkl (returns false if no equivalent hkl) If a symmetry mate of the requested **HKL** (p. 261) exists in the list, then the correct symmetry transformations are applied and data is set to the supplied values, otherwise the function returns false.

Parameters:

hkl The reflection **HKL** (p. 261).

data Value of the data to set.

Returns:

true if the data was set.

References clipper::HKL_info::index_of(), clipper::HKL_info::spacegroup(), clipper::HKL::sym_phase_shift(), and clipper::Spacegroup::symop().

19.82.3.12 template<class T> const ftype clipper::HKL_data< T >::invresolsq (const int & *index*) const

get resolution by reflection index (based on true cell) Return the resolution of a particular reflection. If the cell of this list closely matches (to within 0.5A) the cell of the parent list, this is a simple lookup, otherwise a metric calculation is required.

Reimplemented from **clipper::HKL_data_base** (p. 281).

19.82.3.13 template<class T> const Range<ftype> clipper::HKL_data< T >::invresolsq_range () const

get resolution limits of the list (based on true cell and missing data)

Returns:

The high and low resolution limits of the non-missing data.

Reimplemented from **clipper::HKL_data_base** (p. 281).

19.82.3.14 template<class T> const int clipper::HKL_data< T >::num_obs () const

get number of observations in this list (based on missing data)

Returns:

The number of non-missing data in the object.

Reimplemented from **clipper::HKL_data_base** (p. 281).

19.82.3.15 template<class T> HKL_reference_index clipper::HKL_data< T >::first () const

return HKL_reference_index pointing to first reflection

Returns:

HKL (p. 261) reference to the first data in this object.

Reimplemented from **clipper::HKL_data_base** (p. 281).

Referenced by clipper::Test_core::operator()().

19.82.3.16 template<class T> HKL_reference_index clipper::HKL_data< T >::first_data () const

return HKL_reference_index pointing to first non-missing data

Returns:

HKL (p. 261) reference to the first non-missing data in this object.

Reimplemented from **clipper::HKL_data_base** (p. 282).

19.82.3.17 template<class T> HKL_reference_index& clipper::HKL_data< T >::next_data (HKL_reference_index & *ih*) const

increment HKL_reference_index to next non-missing data

Parameters:

ih The **HKL** (p. 261) reference to increment.

Returns:

HKL (p. 261) reference to the next non-missing data in this object.

Reimplemented from **clipper::HKL_data_base** (p. 282).

19.82.3.18 template<class T> HKL_data< T > & clipper::HKL_data< T >::operator= (const HKL_data< T > & *other*) [inline]

assignment operator: copies the data from another list The data list is copied from the assignment source to the target. If the target does not have a defined **HKL_info** (p. 284), then that and the **Cell** (p. 133) are copied as well. If however the target does have a defined **HKL_info** (p. 284) the **HKL_info** (p. 284) objects are compared, and if they do not match an exception is thrown.

Parameters:

other The datalist to copy.

Returns:

This list.

References clipper::HKL_data< T >::init(), and clipper::Message::message().

19.82.3.19 template<class T> HKL_data< T > & clipper::HKL_data< T >::operator= (const T & *value*) [inline]

assignment operator: assigns a single value to the whole list All values, including missing values, are overwritten by the value.

Parameters:

value The value to which the list is to be set.

Reimplemented in **clipper::CHKL_data< T >** (p. 143).

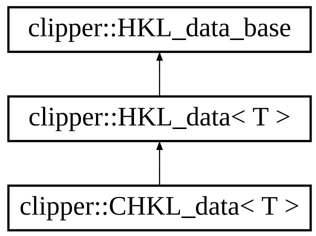
The documentation for this class was generated from the following file:

- hkl_data.h

19.83 clipper::HKL_data_base Class Reference

HKL_data_base (p. 277).

`#include <hkl_data.h>`Inheritance diagram for clipper::HKL_data_base::



Public Types

- typedef **HKL_info::HKL_reference_index HKL_reference_index**

  *Basic HKL_reference_index: see **HKL_info** (p. 284).*

- typedef **HKL_info::HKL_reference_coord HKL_reference_coord**

  ***HKL** (p. 261) HKL_reference_index: see **HKL_info** (p. 284).*

Public Member Functions

- virtual void **init** (const **HKL_info** &hkl_info, const **Cell** &cell)

  initialiser: from parent hkl_info and cell

- virtual void **init** (const **HKL_data_base** &hkl_data)

  initialiser: from another hkl_data

- virtual void **init** (const **Spacegroup** &spacegroup, const **Cell** &cell, const **HKL_sampling** &hkl_sampling)

  *[CLIPPER2] initialiser: from spacegroup, cell, and **HKL_sampling** (p. 298)*

- bool **is_null** () const

  test if object has been initialised

- const **HKL_info** & **base_hkl_info** () const

  *get the parent **HKL_info** (p. 284) object*

- const **Cell** & **base_cell** () const

  get the parent cell

- const **Spacegroup** & **spacegroup** () const

  [CLIPPER2] get spacegroup

- const **Cell** & **cell** () const

  [CLIPPER2] get cell

- const **Resolution** & **resolution** () const
  [CLIPPER2] get resolution
- const **HKL_sampling** & **hkl_sampling** () const
  *[CLIPPER2] get **HKL_sampling** (p. 298)*
- const **HKL_info** & **hkl_info** () const
  *[CLIPPER2] get **HKL_info** (p. 284) object*
- **ftype invresolsq** (const int &index) const
  get resolution by reflection index (based on true cell)
- **Range**< **ftype** > **invresolsq_range** () const
  get resolution limits of the list (based on true cell and missing data)
- int **num_obs** () const
  get number of observations in this list (based on missing data)
- virtual void **update** ()=0
  *update: synchornize info with parent **HKL_info** (p. 284)*
- virtual **String type** () const =0
  get data type (a list of names corresponding to the im/export values)
- virtual bool **missing** (const int &index) const =0
  check if a data entry in the list is marked as 'missing'
- virtual void **set_null** (const int &index)=0
  set data entry in the list to its null value
- virtual int **data_size** () const =0
  return number of data elements in this type
- virtual **String data_names** () const =0
  return names of data elements in this type
- virtual void **data_export** (const **HKL** &hkl, **xtype** array[ ]) const =0
  conversion to array (for I/O)
- virtual void **data_import** (const **HKL** &hkl, const **xtype** array[ ])=0
  conversion from array (for I/O)
- virtual void **mask** (const **HKL_data_base** &mask)=0
  mask the data by marking any data missing in 'mask' as missing
- **HKL_reference_index first** () const
  return HKL_reference_index pointing to first reflection
- **HKL_reference_index first_data** () const

return HKL_reference_index pointing to first non-missing data

- **HKL_reference_index** & **next_data** (**HKL_reference_index** &ih) const
  increment HKL_reference_index to next non-missing data
- void **debug** () const

Protected Member Functions

- **HKL_data_base** ()
  null constructor
- virtual ~**HKL_data_base** ()
  destructor

Protected Attributes

- const **HKL_info** ∗ **parent_hkl_info**
- const **Cell** ∗ **parent_cell**
- bool **cell_matches_parent**
- **ObjectCache**< **HKL_data_cacheobj** >::Reference **cacheref**
  object cache ref
- **Spacegroup spacegroup_**
- **Cell cell_**
- **HKL_sampling hkl_sampling_**
- **Resolution resolution_**

19.83.1 Detailed Description

HKL_data_base (p. 277). This is the virtual base for the typed hkl_data objects. It exists to guarantee and interface by which data can be managed without knowledge of the specific data type.

19.83.2 Constructor & Destructor Documentation

19.83.2.1 clipper::HKL_data_base::HKL_data_base () `[protected]`

null constructor For later initialisation: see **init()** (p. 279)

References clipper::Message::message().

19.83.3 Member Function Documentation

19.83.3.1 void clipper::HKL_data_base::init (const HKL_info & *hkl_info*, const Cell & *cell*) `[virtual]`

initialiser: from parent hkl_info and cell Initialise the object using a given reflection list and cell.

Parameters:

hkl_info The reflection list object.

cell The unit cell for this datalist.

Reimplemented in **clipper::CHKL_data< T >** (p. 144), **clipper::HKL_data< T >** (p. 271), **clipper::HKL_data< T1 >** (p. 271), **clipper::HKL_data< T2 >** (p. 271), **clipper::HKL_data< datatypes::F_sigF< float > >** (p. 271), and **clipper::HKL_data< datatypes::ABCD< float > >** (p. 271).

References clipper::HKL_info::cell(), and clipper::Cell::equals().

Referenced by init().

19.83.3.2 void clipper::HKL_data_base::init (const HKL_data_base & *hkl_data*) [virtual]

initialiser: from another hkl_data Initialise the object using a given reflection list and cell.

Parameters:

hkl_data Object from which to inherit spacegrpoup, cell, sampling.

19.83.3.3 void clipper::HKL_data_base::init (const Spacegroup & *spacegroup*, const Cell & *cell*, const HKL_sampling & *hkl_sampling*) [virtual]

[CLIPPER2] initialiser: from spacegroup, cell, and **HKL_sampling** (p. 298) Initialise the object using a given spacegroup, cell, and sampling.

Parameters:

spacegroup The spacegroup for this datalist.

cell The unit cell for this datalist.

hkl_sampling The reflection list description.

Reimplemented in **clipper::CHKL_data< T >** (p. 143), **clipper::HKL_data< T >** (p. 271), **clipper::HKL_data< T1 >** (p. 271), **clipper::HKL_data< T2 >** (p. 271), **clipper::HKL_data< datatypes::F_sigF< float > >** (p. 271), and **clipper::HKL_data< datatypes::ABCD< float > >** (p. 271).

References clipper::ObjectCache< T >::cache(), cacheref, init(), clipper::HKL_sampling::is_null(), clipper::Cell::is_null(), clipper::Spacegroup::is_null(), and clipper::HKL_sampling::resolution().

19.83.3.4 bool clipper::HKL_data_base::is_null () const

test if object has been initialised

Returns:

true if the object has not been initalised.

References clipper::Cell::is_null(), and clipper::HKL_info::is_null().

Referenced by clipper::CHKL_data< T >::update().

19.83.3.5 ftype clipper::HKL_data_base::invresolsq (const int & *index*) const

get resolution by reflection index (based on true cell) Return the resolution of a particular reflection. If the cell of this list closely matches (to within 0.5A) the cell of the parent list, this is a simple lookup, otherwise a metric calculation is required.

Reimplemented in **clipper::HKL_data< T >** (p. 274), **clipper::HKL_data< T1 >** (p. 274), **clipper::HKL_data< T2 >** (p. 274), **clipper::HKL_data< datatypes::F_sigF< float > >** (p. 274), and **clipper::HKL_data< datatypes::ABCD< float > >** (p. 274).

References base_cell(), base_hkl_info(), clipper::HKL_info::hkl_of(), clipper::HKL::invresolsq(), and clipper::HKL_info::invresolsq().

Referenced by clipper::HKL_info::HKL_reference_base::invresolsq(), and invresolsq_range().

19.83.3.6 Range< ftype > clipper::HKL_data_base::invresolsq_range () const

get resolution limits of the list (based on true cell and missing data)

Returns:

The high and low resolution limits of the non-missing data.

Reimplemented in **clipper::HKL_data< T >** (p. 275), **clipper::HKL_data< T1 >** (p. 275), **clipper::HKL_data< T2 >** (p. 275), **clipper::HKL_data< datatypes::F_sigF< float > >** (p. 275), and **clipper::HKL_data< datatypes::ABCD< float > >** (p. 275).

References first_data(), clipper::Range< T >::include(), clipper::HKL_info::HKL_reference_-index::index(), invresolsq(), clipper::HKL_info::HKL_reference_index::last(), and next_data().

19.83.3.7 int clipper::HKL_data_base::num_obs () const

get number of observations in this list (based on missing data)

Returns:

The number of non-missing data in the object.

Reimplemented in **clipper::HKL_data< T >** (p. 275), **clipper::HKL_data< T1 >** (p. 275), **clipper::HKL_data< T2 >** (p. 275), **clipper::HKL_data< datatypes::F_sigF< float > >** (p. 275), and **clipper::HKL_data< datatypes::ABCD< float > >** (p. 275).

References first_data(), clipper::HKL_info::HKL_reference_index::last(), and next_data().

19.83.3.8 HKL_info::HKL_reference_index clipper::HKL_data_base::first () const

return HKL_reference_index pointing to first reflection

Returns:

HKL (p. 261) reference to the first data in this object.

Reimplemented in **clipper::HKL_data< T >** (p. 275), **clipper::HKL_data< T1 >** (p. 275), **clipper::HKL_data< T2 >** (p. 275), **clipper::HKL_data< datatypes::F_sigF< float > >** (p. 275), and **clipper::HKL_data< datatypes::ABCD< float > >** (p. 275).

19.83.3.9 HKL_info::HKL_reference_index clipper::HKL_data_base::first_data () const

return HKL_reference_index pointing to first non-missing data

Returns:

HKL (p. 261) reference to the first non-missing data in this object.

Reimplemented in **clipper::HKL_data< T >** (p. 275), **clipper::HKL_data< T1 >** (p. 275), **clipper::HKL_data< T2 >** (p. 275), **clipper::HKL_data< datatypes::F_sigF< float > >** (p. 275), and **clipper::HKL_data< datatypes::ABCD< float > >** (p. 275).

References next_data().

Referenced by clipper::Resolution_ordinal::init(), invresolsq_range(), and num_obs().

19.83.3.10 HKL_info::HKL_reference_index & clipper::HKL_data_base::next_data (HKL_info::HKL_reference_index & *ih*) const

increment HKL_reference_index to next non-missing data

Parameters:

ih The **HKL** (p. 261) reference to increment.

Returns:

HKL (p. 261) reference to the next non-missing data in this object.

Reimplemented in **clipper::HKL_data< T >** (p. 275), **clipper::HKL_data< T1 >** (p. 275), **clipper::HKL_data< T2 >** (p. 275), **clipper::HKL_data< datatypes::F_sigF< float > >** (p. 275), and **clipper::HKL_data< datatypes::ABCD< float > >** (p. 275).

References clipper::HKL_info::HKL_reference_index::index(), clipper::HKL_info::HKL_reference_index::last(), missing(), and clipper::HKL_info::HKL_reference_index::next().

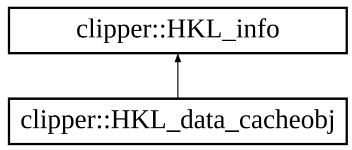
Referenced by first_data(), clipper::Resolution_ordinal::init(), invresolsq_range(), and num_obs().

The documentation for this class was generated from the following files:

- hkl_data.h
- hkl_data.cpp

19.84 clipper::HKL_data_cacheobj Class Reference

Inheritance diagram for clipper::HKL_data_cacheobj::



Classes

- class **Key**

Public Member Functions

- **HKL_data_cacheobj** (const **Key** &hkl_data_cachekey)
- bool **matches** (const **Key** &hkl_data_cachekey) const
- **String format** () const

Static Public Attributes

- static **Mutex mutex** = **Mutex**()

  thread safety

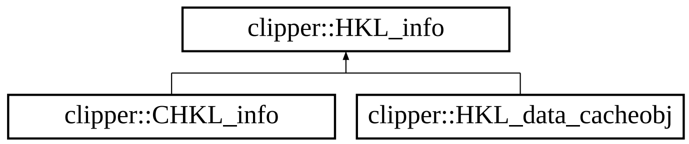
The documentation for this class was generated from the following files:

- hkl_data.h
- hkl_data.cpp

19.85 clipper::HKL_info Class Reference

HKL (p. 261) list container and tree root.

`#include <hkl_info.h>`Inheritance diagram for clipper::HKL_info::



Classes

- class **HKL_reference_base**

  HKL *(p. 261) reference base class.*

- class **HKL_reference_coord**

  HKL *(p. 261) reference with coord-like behaviour.*

- class **HKL_reference_index**

  HKL *(p. 261) reference with index-like behaviour.*

Public Member Functions

- **HKL_info** ()

  null constructor

- **HKL_info** (const **Spacegroup** &spacegroup, const **Cell** &cell, const **Resolution** &resolution, const bool &generate=false)

  constructor: Takes spacegroup, cell, and resolution

- void **init** (const **Spacegroup** &spacegroup, const **Cell** &cell, const **Resolution** &resolution, const bool &generate=false)

  initialiser: Takes spacegroup, cell, and resolution

- void **init** (const **Spacegroup** &spacegroup, const **Cell** &cell, const **HKL_sampling** &hkl_sampling, const bool &generate=true)

  initialiser: Takes spacegroup, cell, and ***HKL_sampling*** *(p. 298)*

- bool **is_null** () const

  test if object has been initialised

- const **Cell** & **cell** () const

  get the cell

- const **Spacegroup** & **spacegroup** () const

  get the spacegroup

- const **HKL_sampling** & **hkl_sampling** () const
  [CLIPPER2] get ***HKL_sampling*** *(*p. *298)*
- const **Resolution** & **resolution** () const
  get the resolution
- void **generate_hkl_list** ()
  synthesize hkl list
- void **add_hkl_list** (const std::vector< **HKL** > &add)
  add new reflections to the list
- int **num_reflections** () const
  get number of reflections in the object
- const **HKL** & **hkl_of** (const int &index) const
  reflection hkl from index
- int **index_of** (const **HKL** &rfl) const
  reflection index from hkl
- const ftype32 & **invresolsq** (const int &index) const
  get reflection resolution using lookup
- const **Range**< **ftype** > & **invresolsq_range** () const
  get resolution limits of the list
- const **HKL_class** & **hkl_class** (const int &index) const
  get reflection class using lookup
- **HKL find_sym** (const **HKL** &rfl, int &sym, bool &friedel) const
  find symop no and friedel to bring an ***HKL*** *(*p. *261) into ASU*
- **HKL_reference_index first** () const
  return ***HKL_reference_index*** *(*p. *296) pointing to first reflection*
- void **debug** () const

Protected Member Functions

- void **update_hkl_list** ()
  finalise reflection list

Protected Attributes

- **Spacegroup spacegroup_**
  spacegroup

- **Cell cell_**
  unit cell
- **HKL_sampling hkl_sampling_**
  hkl sampling
- **Resolution resolution_**
  resolution limit
- std::vector< **Isymop** > **isymop**
  integer symops
- std::vector< **HKL** > **hkl**
  the reflection list
- std::vector< **HKL_class** > **hkl_class_lookup**
  fast epsilon/centricity lookup table
- std::vector< ftype32 > **invresolsq_lookup**
  fast resolution lookup table
- **HKL_lookup lookup**
  fast reflection lookup table
- **Range< ftype > invresolsq_range_**
  resolution limit of the current reflection list

Friends

- class **HKL_info::HKL_reference_base**
- class **HKL_info::HKL_reference_index**
- class **HKL_info::HKL_reference_coord**

19.85.1 Detailed Description

HKL (p. 261) list container and tree root. This object contains contains a reflection list, and all the properties on which such a list depends, i.e. spacegroup, cell, resolution. It also keeps a fast reflection lookup list and lookup lists for resolutions and reflection classes.

19.85.2 Constructor & Destructor Documentation

19.85.2.1 clipper::HKL_info::HKL_info (const Spacegroup & *spacegroup*, const Cell & *cell*, const Resolution & *resolution*, const bool & *generate* = `false`)

constructor: Takes spacegroup, cell, and resolution Construct and initialise **HKL_info** (p. 284) object. This updates the spacegroup and cell and clears the reflection list. The resolution is used as a rejection criterion for reflections - no **HKL** (p. 261) will be stored beyond the given limit. Initially there are no reflections in the reflection list: see **generate_hkl_list()** (p. 288).

If any of the parameters have null values, the existing values will be unchanged. The object will only be fully initialised once all parameters are available.

Parameters:

spacegroup The spacegroup.

cell The unit cell.

resolution The resolution limit.

References init(), and clipper::Message::message().

19.85.3 Member Function Documentation

19.85.3.1 void clipper::HKL_info::init (const Spacegroup & *spacegroup*, const Cell & *cell*, const Resolution & *resolution*, const bool & *generate* = `false`)

initialiser: Takes spacegroup, cell, and resolution Initialise the **HKL_info** (p. 284) object. This updates the spacegroup and cell and clears the reflection list. The resolution is used as a rejection criterion for reflections - no **HKL** (p. 261) will be stored beyond the given limit. Initially there are no reflections in the reflection list: see **generate_hkl_list()** (p. 288).

If any of the parameters have null values, the existing values will be unchanged. The object will only be fully initialised once all parameters are available.

Parameters:

spacegroup The spacegroup.

cell The unit cell.

resolution The resolution limit.

generate If true, a reflection list will be generated for an ASU.

Reimplemented in **clipper::CHKL_info** (p. 146).

References cell_, generate_hkl_list(), hkl, is_null(), isymop, clipper::Spacegroup::num_-symops(), resolution_, spacegroup_, clipper::Spacegroup::symop(), and update_hkl_list().

Referenced by HKL_info(), clipper::CHKL_info::init(), and clipper::data::Test_data::Test_-data().

19.85.3.2 void clipper::HKL_info::init (const Spacegroup & *spacegroup*, const Cell & *cell*, const HKL_sampling & *hkl_sampling*, const bool & *generate* = `true`)

initialiser: Takes spacegroup, cell, and **HKL_sampling** (p. 298) Initialise the **HKL_info** (p. 284) object. This updates the spacegroup and cell and clears the reflection list. The **HKL_-sampling** (p. 298) determines the reflection list.

If any of the parameters have null values, the existing values will be unchanged. The object will only be fully initialised once all parameters are available.

Parameters:

spacegroup The spacegroup.

cell The unit cell.

hkl_sampling The resolution limit.

generate If true, a reflection list will be generated for an ASU.

References add_hkl_list(), cell_, clipper::HKL::h(), hkl, clipper::Spacegroup::hkl_class(), clipper::HKL_sampling::hkl_limit(), hkl_sampling_, clipper::HKL_sampling::in_resolution(), clipper::HKL_sampling::is_null(), clipper::Cell::is_null(), clipper::Spacegroup::is_null(), isymop, clipper::HKL::k(), clipper::HKL::l(), clipper::Spacegroup::num_symops(), clipper::Spacegroup::recip_asu(), clipper::HKL_sampling::resolution(), resolution_, spacegroup_, clipper::Spacegroup::symop(), clipper::HKL_class::sys_abs(), and update_hkl_list().

19.85.3.3 bool clipper::HKL_info::is_null () const

test if object has been initialised

Returns:

true if the object has not been initalised.

References cell_, clipper::Resolution::is_null(), clipper::Cell::is_null(), clipper::Spacegroup::is_null(), resolution_, and spacegroup_.

Referenced by init(), clipper::CHKL_data< T >::init(), clipper::HKL_data_base::is_null(), and clipper::CHKL_info::update().

19.85.3.4 void clipper::HKL_info::generate_hkl_list ()

synthesize hkl list Using current cell, spacegroup, resolution.

Reimplemented in **clipper::CHKL_info** (p. 146).

References clipper::Cell_descr::a(), add_hkl_list(), clipper::Cell_descr::b(), clipper::Cell_descr::c(), cell_, clipper::Cell::descr(), clipper::HKL::h(), hkl, clipper::Spacegroup::hkl_class(), clipper::HKL::invresolsq(), clipper::Resolution::invresolsq_limit(), clipper::HKL::k(), clipper::HKL::l(), clipper::Resolution::limit(), clipper::Spacegroup::recip_asu(), resolution_, spacegroup_, and clipper::HKL_class::sys_abs().

Referenced by init().

19.85.3.5 void clipper::HKL_info::add_hkl_list (const std::vector< HKL > & *add*)

add new reflections to the list The new HKLs are transformed to the default reciprocal ASU, and added to the reflection list. Duplicates and reflections outside the resoluution limit are ignored. Then the fast lookup tables for **HKL** (p. 261), invresolsq, and reflection class are rebuilt.

Parameters:

add The list of new reflections to add.

References cell_, find_sym(), hkl, clipper::HKL_lookup::index_of(), invresolsq(), clipper::Resolution::invresolsq_limit(), lookup, resolution_, and update_hkl_list().

Referenced by generate_hkl_list(), and init().

19.85.3.6 const HKL& clipper::HKL_info::hkl_of (const int & *index*) const [inline]

reflection hkl from index

Parameters:

index The index.

Returns:

The corresponding **HKL** (p. 261).

References hkl.

Referenced by clipper::HKL_info::HKL_reference_index::hkl(), clipper::HKL_data_-base::invresolsq(), clipper::HKL_info::HKL_reference_coord::next(), and update_hkl_list().

19.85.3.7 int clipper::HKL_info::index_of (const HKL & *rfl*) const [inline]

reflection index from hkl This does not check symmetry equivalents (see find_sym).

Parameters:

rfl The **HKL** (p. 261).

Returns:

The index, or -1 if it does not exist.

References clipper::HKL_lookup::index_of(), and lookup.

Referenced by clipper::HKL_data< T >::get_data(), clipper::HKL_info::HKL_reference_-coord::HKL_reference_coord(), clipper::HKL_data< T >::operator[](), clipper::HKL_data< T >::set_data(), and clipper::HKL_info::HKL_reference_coord::set_hkl().

19.85.3.8 HKL clipper::HKL_info::find_sym (const HKL & *rfl*, int & *sym*, bool & *friedel*) const

find symop no and friedel to bring an **HKL** (p. 261) into ASU Returns the index of the reflection, the sym no. and Friedel flag.

References isymop, clipper::Message::message(), clipper::Spacegroup::num_primops(), clipper::Spacegroup::recip_asu(), spacegroup_, and clipper::HKL::transform().

Referenced by add_hkl_list(), clipper::HKL_info::HKL_reference_coord::HKL_reference_-coord(), and clipper::HKL_info::HKL_reference_coord::set_hkl().

19.85.3.9 void clipper::HKL_info::update_hkl_list () [protected]

finalise reflection list Update all the lookup tables to be consistent with the modified reflection list

References cell_, hkl, clipper::Spacegroup::hkl_class(), hkl_class_lookup, hkl_of(), clipper::Range< T >::include(), clipper::HKL_lookup::init(), clipper::HKL::invresolsq(), invresolsq_lookup, invresolsq_range_, lookup, num_reflections(), and spacegroup_.

Referenced by add_hkl_list(), and init().

The documentation for this class was generated from the following files:

- hkl_info.h
- hkl_info.cpp

19.86 clipper::HKL_lookup Class Reference

Fast reflection lookup object.

`#include <hkl_lookup.h>`

Classes

- struct **hlookup**
- struct **klookup**
- struct **llookup**

Public Member Functions

- void **init** (const std::vector< **HKL** > &hkl)

  initialise: make a reflection index for a list of HKLs

- int **index_of** (const **HKL** &rfl) const

  lookup function

- void **debug** ()

19.86.1 Detailed Description

Fast reflection lookup object. This version uses a ragged array index to get reflection indices from Miller indices.

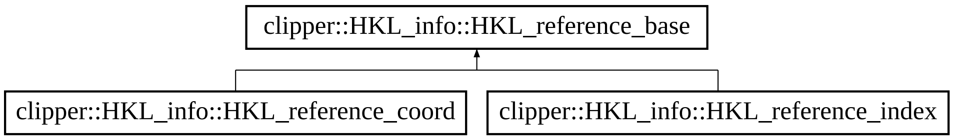
The documentation for this class was generated from the following files:

- hkl_lookup.h
- hkl_lookup.cpp

19.87 clipper::HKL_info::HKL_reference_base Class Reference

HKL (p. 261) reference base class.

`#include <hkl_info.h>`Inheritance diagram for clipper::HKL_info::HKL_reference_base::



Public Member Functions

- const **HKL_info** & **base_hkl_info** () const
  return the parent ***HKL_info*** *(p. 284)*
- const int & **index** () const
  return the current index (-1 if invalid)
- **ftype invresolsq** (const **HKL_data_base** &hkldata) const
  return the inv resol sq for the reflection (assumes index valid)
- **ftype invresolsq** () const
  return the inv resol sq for the reflection (assumes index valid)
- bool **last** () const
  test if index has gone past last reflection

Protected Attributes

- const **HKL_info** ∗ **hklinfo**
- int **index_**

19.87.1 Detailed Description

HKL (p. 261) reference base class. This is a reference to an **HKL** (p. 261). It forms a base class for index-like and coordinate-like **HKL** (p. 261) references. If you write a method which will work with either, then specify this instead of either of the derived classed.

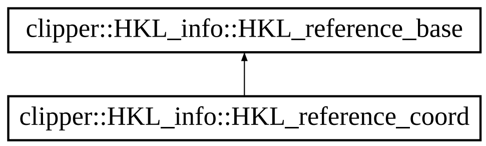
The documentation for this class was generated from the following files:

- hkl_info.h
- hkl_data.h

19.88 clipper::HKL_info::HKL_reference_coord Class Reference

HKL (p. 261) reference with coord-like behaviour.

`#include <hkl_info.h>`Inheritance diagram for clipper::HKL_info::HKL_reference_coord::



Public Member Functions

- **HKL_reference_coord** ()
  Null constructor.
- **HKL_reference_coord** (const **HKL_info** &hklinfo_, const **HKL** &hkl)
  *Constructor: takes parent **HKL_info** (p. 284) and initial **HKL** (p. 261).*
- const **HKL** & **hkl** () const
  *return the current **HKL** (p. 261)*
- const int & **sym** () const
  get current symop number
- const bool & **friedel** () const
  get current friedel flag
- **HKL_reference_coord** & **set_hkl** (const **HKL** &hkl__)
  *assign from **HKL** (p. 261)*
- **HKL_reference_coord** & **next** ()
  increment to next reflection
- **HKL_reference_coord** & **next_h** ()
  increment h
- **HKL_reference_coord** & **next_k** ()
  increment k
- **HKL_reference_coord** & **next_l** ()
  increment l
- **HKL_reference_coord** & **prev_h** ()
  decrement h
- **HKL_reference_coord** & **prev_k** ()

decrement k

- **HKL_reference_coord** & **prev_l** ()
  decrement l
- **HKL_reference_coord** & **operator=** (const **HKL** &hkl__)
  operator assign from ***HKL*** *(p. 261)*
- const **HKL_info** & **base_hkl_info** () const
  return the parent ***HKL_info*** *(p. 284)*
- const int & **index** () const
  return the current index (-1 if invalid)
- const **ftype invresolsq** () const
  return the inv resol sq for the reflection (assumes index valid)
- bool **last** () const
  test if index has gone past last reflection

Protected Attributes

- **HKL hkl_**
- int **sym_**
- bool **friedel_**

19.88.1 Detailed Description

HKL (p. 261) reference with coord-like behaviour. This is a reference to an **HKL** (p. 261). It behaves like an **HKL** (p. 261), but also stores the index of the corresponding reflection in the reflection list, if it exists, and the symmetry and friedel operators required to get there.

Note:

The following methods are inherited from **HKL_reference_base** (p. 292) but are documented here for convenience: **base_hkl_info()** (p. 294), **index()** (p. 294), **invresolsq()** (p. 294), **last()** (p. 294).

19.88.2 Member Function Documentation

19.88.2.1 HKL_reference_coord& clipper::HKL_info::HKL_reference_coord::set_hkl (const HKL & *hkl__*) [inline]

assign from **HKL** (p. 261) The new **HKL** (p. 261) must exist in the reflection list. The calculation is optimised for the case when the new **HKL** (p. 261) is near the old one.

References clipper::HKL_info::find_sym(), clipper::HKL_info::index_of(), clipper::HKL_info::isymop, and clipper::HKL::transform().

Referenced by next_h(), next_k(), next_l(), operator=(), prev_h(), prev_k(), and prev_l().

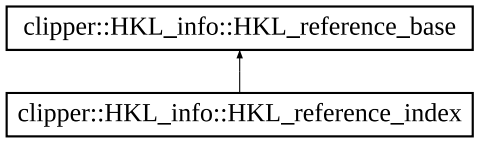
The documentation for this class was generated from the following file:

- hkl_info.h

19.89 clipper::HKL_info::HKL_reference_index Class Reference

HKL (p. 261) reference with index-like behaviour.

`#include <hkl_info.h>`Inheritance diagram for clipper::HKL_info::HKL_reference_index::



Public Member Functions

- **HKL_reference_index** ()

  Null constructor.

- **HKL_reference_index** (const **HKL_info** &hklinfo_, const int &index)

  *Constructor: takes parent **HKL_info** (p. 284) and initial index.*

- const **HKL** & **hkl** () const

  *return the current **HKL** (p. 261)*

- const **HKL_class** & **hkl_class** () const

  return the reflection class for the reflection

- **HKL_reference_index** & **next** ()

  increment to next reflection

- const **HKL_info** & **base_hkl_info** () const

  *return the parent **HKL_info** (p. 284)*

- const int & **index** () const

  return the current index (-1 if invalid)

- const **ftype invresolsq** () const

  return the inv resol sq for the reflection (assumes index valid)

- bool **last** () const

  test if index has gone past last reflection

19.89.1 Detailed Description

HKL (p. 261) reference with index-like behaviour. This is a reference to an **HKL** (p. 261). It behaves like a simple index into the reflection list, but can be easily converted into an **HKL** (p. 261) as and when required. It also implements methods for iterating through a reflection list.

Note:

The following methods are inherited from **HKL_reference_base** (p. 292) but are documented here for convenience: **base_hkl_info()** (p. 296), **index()** (p. 296), **invresolsq()** (p. 296), **last()** (p. 296).

The documentation for this class was generated from the following file:

- hkl_info.h

19.90 clipper::HKL_sampling Class Reference

HKL (p. 261) sampling of reciprocal space.

`#include <coords.h>`

Public Member Functions

- **HKL_sampling** ()
  null constructor
- **HKL_sampling** (const **Cell** &cell, const **Resolution** &resolution)
  constructor: takes parameters of normal or inverse cell
- **HKL hkl_limit** () const
  return limiting values of H, K, L
- **Resolution resolution** (const **Cell** &cell) const
  return approximate resolution given cell
- bool **in_resolution** (const **HKL** &h) const
  test if a reflection is within the resolution limit
- bool **is_null** () const
  test if object has been initialised
- **String format** () const
  *return formatted **String** (p. 426) representation*

Friends

- int **operator==** (const **HKL_sampling** &h1, const **HKL_sampling** &h2)

19.90.1 Detailed Description

HKL (p. 261) sampling of reciprocal space. The **HKL_sampling** (p. 298) class uniquely describes a P0 reflection list bounded by some resolution limit in reciprocal space. It is described in terms of large integers, and so immune from rounding errors once the object is constructed.

19.90.2 Constructor & Destructor Documentation

19.90.2.1 clipper::HKL_sampling::HKL_sampling ()

null constructor Null constructor

Initialise to 'null'

19.90.2.2 clipper::HKL_sampling::HKL_sampling (const Cell & *cell*, const Resolution & *resolution*)

constructor: takes parameters of normal or inverse cell Initialise using cell and resolution.

References clipper::Cell::a_star(), clipper::Cell::alpha_star(), clipper::Cell::b_star(), clipper::Cell::beta_star(), clipper::Cell::c_star(), clipper::Cell::gamma_star(), and clipper::Resolution::invresolsq_limit().

19.90.3 Member Function Documentation

19.90.3.1 HKL clipper::HKL_sampling::hkl_limit () const

return limiting values of H, K, L Returned **HKL** (p. 261) contains maximum possible values of H, K, L respectively.

Returns:

Limiting h,k,l.

References clipper::Util::isqrt().

Referenced by clipper::HKL_info::init(), and resolution().

19.90.3.2 Resolution clipper::HKL_sampling::resolution (const Cell & *cell*) const

return approximate resolution given cell Returned resolution is an estimate based on highest reflection in list.

Returns:

The resolution.

References clipper::HKL::h(), hkl_limit(), in_resolution(), clipper::HKL::invresolsq(), clipper::HKL::k(), clipper::HKL::l(), and clipper::Util::max().

Referenced by clipper::HKL_info::init(), and clipper::HKL_data_base::init().

The documentation for this class was generated from the following files:

- coords.h
- coords.cpp

19.91 clipper::HKL_lookup::hlookup Struct Reference

Public Attributes

- int **min**
- int **max**
- std::vector< **klookup** > **p**

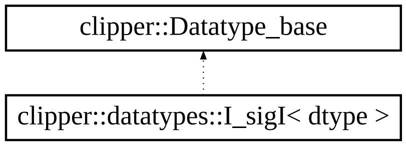
The documentation for this struct was generated from the following file:

- hkl_lookup.h

19.92 clipper::datatypes::I_sigI< dtype > Class Template Reference

Reflection data type: I + sigI.

`#include <hkl_datatypes.h>`Inheritance diagram for clipper::datatypes::I_sigI< dtype >::



Public Member Functions

- **I_sigI** (const dtype &I, const dtype &sigI)
- void **set_null** ()

  initialise to 'missing' (all elements are set to null)

- void **friedel** ()

  apply Friedel transformation to the data

- void **shift_phase** (const **ftype** &dphi)

  apply phase shift to the data

- bool **missing** () const

  return true if data is missing

- void **data_export** (**xtype** array[ ]) const

  conversion to array (for I/O)

- void **data_import** (const **xtype** array[ ])

  conversion from array (for I/O)

- void **scale** (const **ftype** &s)

  this type is scalable - apply magnitude scale factor

- const dtype & **I** () const
- const dtype & **sigI** () const
- dtype & **I** ()
- dtype & **sigI** ()
- const dtype & **I_pl** () const
- const dtype & **sigI_pl** () const
- const dtype & **I_mi** () const
- const dtype & **sigI_mi** () const
- dtype **cov** () const

Static Public Member Functions

- static **String type** ()
  return the name of this data type
- static int **data_size** ()
  return number of data elements in this type
- static **String data_names** ()
  return names of data elements in this type

19.92.1 Detailed Description

template<class dtype> class clipper::datatypes::I_sigI< dtype >

Reflection data type: I + sigI. Note that **I_sigI** (p. 301) also has methods for returning I_pl(), sigI_pl(), I_mi, sigI_mi(), so you can use this type in any template type where you would use **I_sigI_ano** (p. 303).

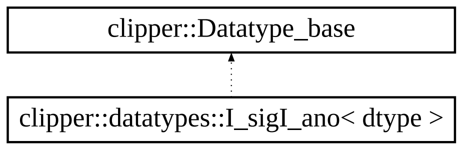
The documentation for this class was generated from the following file:

- hkl_datatypes.h

19.93 clipper::datatypes::I_sigI_ano< dtype > Class Template Reference

Reflection data type: I(+) I(+) sigI(+) sigI(-) cov+-.

#include <hkl_datatypes.h>Inheritance diagram for clipper::datatypes::I_sigI_ano< dtype >::



Public Member Functions

- void **set_null** ()
  initialise to 'missing' (all elements are set to null)

- void **friedel** ()
  apply Friedel transformation to the data

- void **shift_phase** (const **ftype** &dphi)
  apply phase shift to the data

- bool **missing** () const
  return true if data is missing

- void **data_export** (**xtype** a[ ]) const
  conversion to array (for I/O)

- void **data_import** (const **xtype** a[ ])
  conversion from array (for I/O)

- void **scale** (const **ftype** &s)
  this type is scalable - apply magnitude scale factor

- const dtype & **I_pl** () const
- const dtype & **sigI_pl** () const
- const dtype & **I_mi** () const
- const dtype & **sigI_mi** () const
- const dtype & **cov** () const
- dtype & **I_pl** ()
- dtype & **sigI_pl** ()
- dtype & **I_mi** ()
- dtype & **sigI_mi** ()
- dtype & **cov** ()
- dtype **I** () const
- dtype **sigI** () const

Static Public Member Functions

- static **String type** ()
  return the name of this data type
- static int **data_size** ()
  return number of data elements in this type
- static **String data_names** ()
  return names of data elements in this type

19.93.1 Detailed Description

template<class dtype> class clipper::datatypes::I_sigI_ano< dtype >

Reflection data type: I(+) I(+) sigI(+) sigI(-) cov+-. Note that **I_sigI_ano** (p. 303) also has methods for returning I(), sigI(), so you can use this type in any template type where you would use **I_sigI** (p. 301).

The documentation for this class was generated from the following file:

- hkl_datatypes.h

19.94 clipper::Interp_cubic Class Reference

Wrapper class for third-order (cubic) interpolation fns.

`#include <map_interp.h>`

Static Public Member Functions

- template<class M >
  static bool **can_interp** (const M &map, const **Coord_map** &pos)
  Test if we can interpolate in map M at coord.
- template<class T , class M >
  static void **interp** (const M &map, const **Coord_map** &pos, T &val)
  Interpolate map M using type T at coord.
- template<class T , class M >
  static void **interp_grad** (const M &map, const **Coord_map** &pos, T &val, **Grad_map**< T > &grad)
- template<class T , class M >
  static void **interp_curv** (const M &map, const **Coord_map** &pos, T &val, **Grad_map**< T > &grad, **Curv_map**< T > &curv)
- static int **order** ()
  Order of interpolant.

19.94.1 Detailed Description

Wrapper class for third-order (cubic) interpolation fns. These can be used through the built-in methods in Xmap/NXmap, or passed to methods to allow a choice of interpolation methods, or directly by providing the map as an argument. For example:

```
NXmap<float> nxmap;
Coord_map c;
float x1, x2;
...
Interp_cubic::interp( nxmap, c, x1 );
x2 = nxmap.interp<Interp_cubic>( c );
```

19.94.2 Member Function Documentation

19.94.2.1 template<class M > bool clipper::Interp_cubic::can_interp (const M & *map*, const Coord_map & *pos*) [inline, static]

Test if we can interpolate in map M at coord. The map is queried to see if interpolation is possible at the given coord. For a crystallographic map, this is always true. For a non-crystallographic map, this depends if the point and enough neighbours are in the grid.

Parameters:

map The map on which to perform the calculation.

pos The map coord at which the density is to be calcuated.

References clipper::Coord_map::floor(), and order().

19.94.2.2 template<class T , class M > void clipper::Interp_cubic::interp (const M & *map*, const Coord_map & *pos*, T & *val*) [inline, static]

Interpolate map M using type T at coord. The value of the map at the supplied map coordinate is calculated by third order (cubic) interpolation based on the surrounding 64 points.

Parameters:

pos The fractional coord at which the density is to be calcuated.

Returns:

The value of the density at that point.

References clipper::Coord_map::u(), clipper::Coord_map::v(), and clipper::Coord_map::w().

19.94.2.3 template<class T , class M > void clipper::Interp_cubic::interp_grad (const M & *map*, const Coord_map & *pos*, T & *val*, Grad_map< T > & *grad*) [inline, static]

The value of the map at the supplied map coordinate and its gradient are calculated by third order (cubic) interpolation based on the surrounding 64 points.

Parameters:

pos The fractional coord at which the density is to be calcuated.

val The value of the density at that point.

grad The interpolated value as a gradient vector with respect to the fractional coordinates (see Cell::coord_orth).

References clipper::Coord_map::u(), clipper::Coord_map::v(), and clipper::Coord_map::w().

19.94.2.4 template<class T , class M > void clipper::Interp_cubic::interp_curv (const M & *map*, const Coord_map & *pos*, T & *val*, Grad_map< T > & *grad*, Curv_map< T > & *curv*) [inline, static]

The value of the map at the supplied map coordinate and its gradient are calculated by third order (cubic) interpolation based on the surrounding 64 points.

Parameters:

pos The fractional coord at which the density is to be calcuated.

val The value of the density at that point.

grad The interpolated value as a gradient vector with respect to the fractional coordinates (see Cell::coord_orth).

References clipper::Coord_map::u(), clipper::Coord_map::v(), and clipper::Coord_map::w().

The documentation for this class was generated from the following file:

- map_interp.h

19.95 clipper::Interp_linear Class Reference

Wrapper class for first-order (linear) interpolation fns.

`#include <map_interp.h>`

Static Public Member Functions

- template<class M >
  static bool **can_interp** (const M &map, const **Coord_map** &pos)
  Test if we can interpolate in map M at coord.
- template<class T , class M >
  static void **interp** (const M &map, const **Coord_map** &pos, T &val)
  Interpolate map M using type T at coord.
- static int **order** ()
  Order of interpolant.

19.95.1 Detailed Description

Wrapper class for first-order (linear) interpolation fns. These can be used through the built-in methods in Xmap/NXmap, or passed to methods to allow a choice of interpolation methods, or directly by providing the map as an argument. For example:

```
NXmap<float> nxmap;
Coord_map c;
float x1, x2;
...
Interp_linear<float>::interp( nxmap, c, x1 );
x2 = nxmap.interp<Interp_linear>( c );
```

19.95.2 Member Function Documentation

19.95.2.1 template<class M > bool clipper::Interp_linear::can_interp (const M & *map*, const Coord_map & *pos*) `[inline, static]`

Test if we can interpolate in map M at coord. The map is queried to see if interpolation is possible at the given coord. For a crystallographic map, this is always true. For a non-crystallographic map, this depends if the point and enough neighbours are in the grid.

Parameters:

map The map on which to perform the calculation.
pos The map coord at which the density is to be calcuated.

References clipper::Coord_map::floor(), and order().

19.95.2.2 template<class T , class M > void clipper::Interp_linear::interp (const M & *map*, const Coord_map & *pos*, T & *val*) `[inline, static]`

Interpolate map M using type T at coord. The value of the map at the supplied map coordinate is calculated by first order (linear) interpolation based on 8 neighbouring points.

Parameters:

map The map on which to perform the calculation.

pos The map coord at which the density is to be calcuated.

Returns:

The value of the density at that point.

References clipper::Coord_map::u(), clipper::Coord_map::v(), and clipper::Coord_map::w().

The documentation for this class was generated from the following file:

- map_interp.h

19.96 clipper::Interp_nearest Class Reference

Wrapper class for zeroth-order (nearest neighbour) interpolation fns.

`#include <map_interp.h>`

Static Public Member Functions

- template<class M >
  static bool **can_interp** (const M &map, const **Coord_map** &pos)

  Test if we can interpolate in map M at coord.

- template<class T , class M >
  static void **interp** (const M &map, const **Coord_map** &pos, T &val)

  Interpolate map M using type T at coord.

- static int **order** ()

  Order of interpolant.

19.96.1 Detailed Description

Wrapper class for zeroth-order (nearest neighbour) interpolation fns. These can be used through the built-in methods in Xmap/NXmap, or passed to methods to allow a choice of interpolation methods, or directly by providing the map as an argument. For example:

```
NXmap<float> nxmap;
Coord_map c;
...
x1 = Interp_nearest<float>::interp( nxmap, c );
x2 = nxmap.interp<Interp_nearest>( c );
```

19.96.2 Member Function Documentation

19.96.2.1 template<class M > bool clipper::Interp_nearest::can_interp (const M & *map*, const Coord_map & *pos*) `[inline, static]`

Test if we can interpolate in map M at coord. The map is queried to see if interpolation is possible at the given coord. For a crystallographic map, this is always true. For a non-crystallographic map, this depends if the point and enough neighbours are in the grid.

Parameters:

map The map on which to perform the calculation.

pos The map coord at which the density is to be calcuated.

References clipper::Coord_map::coord_grid().

19.96.2.2 template<class T , class M > void clipper::Interp_nearest::interp (const M & *map*, const Coord_map & *pos*, T & *val*) `[inline, static]`

Interpolate map M using type T at coord. The value of the map at the supplied map coordinate is calculated by zeroth order (nearest neighbour) interpolation based on 1 point.

Parameters:

map The map on which to perform the calculation.

pos The map coord at which the density is to be calcuated.

Returns:

The value of the density at that point.

References clipper::Coord_map::coord_grid().

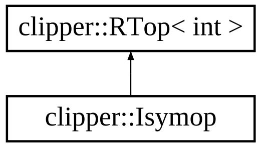
The documentation for this class was generated from the following file:

- map_interp.h

19.97 clipper::Isymop Class Reference

Integerised symmetry matrix.

`#include` <`symop.h`>Inheritance diagram for clipper::Isymop::



Public Member Functions

- **Isymop** ()
  null constructor
- **Isymop** (const **RTop**< int > &rt)
  constructor: ***RTop*** *(p. 402)*
- **Isymop** (const **Symop** &symop, const **Grid** &grid)
  constructor

19.97.1 Detailed Description

Integerised symmetry matrix. This is used for optimised calculations in real and reciprocal space

19.97.2 Constructor & Destructor Documentation

19.97.2.1 clipper::Isymop::Isymop (const Symop & *symop*, const Grid & *grid*)

constructor Integerised symops are more efficient when handling integer coordinate types, e.g. **HKL** (p. 261), **Coord_grid** (p. 175). The rotation parts of the integerised symop are general and can be used for any recirpocal space data. The translation part is specific to an individual grid.

Parameters:

symop The conventional symop.

grid The specific grid.

References clipper::Util::intr(), clipper::Grid::nu(), clipper::Grid::nv(), clipper::Grid::nw(), clipper::RTop< T >::rot(), clipper::RTop< int >::rot(), clipper::RTop< T >::trn(), and clipper::RTop< int >::trn().

The documentation for this class was generated from the following files:

- symop.h
- symop.cpp

19.98 clipper::Xmap_cacheobj::Key Class Reference

Public Member Functions

- **Key** (const **Spgr_descr** &spgr_descr, const **Grid_sampling** &grid)
- const **Spgr_descr** & **spgr_descr** () const
- const **Grid_sampling** & **grid_sampling** () const

The documentation for this class was generated from the following file:

- xmap.h

19.99 clipper::HKL_data_cacheobj::Key Class Reference

Public Member Functions

- **Key** (const **Spgr_descr** &spgr_descr, const **Cell** &cell_descr, const **HKL_sampling** &hkl_sam)
- const **Spgr_descr** & **spgr_descr** () const
- const **Cell_descr** & **cell_descr** () const
- const **HKL_sampling** & **hkl_sampling** () const

The documentation for this class was generated from the following file:

- hkl_data.h

19.100 clipper::HKL_lookup::klookup Struct Reference

Public Attributes

- int **min**
- int **max**
- std::vector< **llookup** > **p**

The documentation for this struct was generated from the following file:

- hkl_lookup.h

19.101 clipper::data::LGdata Struct Reference

Public Attributes

- int **lghash**
- ASUfn **asufn**
- const char * **lgname**

The documentation for this struct was generated from the following file:

- spacegroup_data.h

19.102 clipper::HKL_lookup::llookup Struct Reference

Public Attributes

- int **min**
- int **max**
- std::vector< int > **p**

The documentation for this struct was generated from the following file:

- hkl_lookup.h

19.103 clipper::LogPhaseProb< N > Class Template Reference

Log phase probability distribution object.

#include <hkl_operators.h>

Public Member Functions

- **LogPhaseProb** (const **HKL_class** &hkl_class)

  constructor: from ***HKL*** *(*p. *261)* *class*

- template<class dtype >
  void **set_abcd** (const **datatypes::ABCD**< dtype > &abcd)

  set HL coeffs

- template<class dtype >
  void **get_abcd** (**datatypes::ABCD**< dtype > &abcd) const

  get HL coeffs

- template<class dtype >
  void **set_phi_fom** (const **datatypes::Phi_fom**< dtype > &phifom)

  set phi/fom

- template<class dtype >
  void **get_phi_fom** (**datatypes::Phi_fom**< dtype > &phifom) const

  get phi/fom

- const **ftype** & **operator[]** (const int &p) const

  get log probability

- **ftype** & **operator[]** (const int &p)

  set log probability

- **ftype phase** (const int &p) const

  return phase associated with index

- int **size** () const

  return num. of phases

Static Public Member Functions

- static int **sampling** ()

  return phase sampling

19.103.1 Detailed Description

template<int N> class clipper::LogPhaseProb< N >

Log phase probability distribution object. This object is used to store and manipulate phase log-probability distributions. Centrics are handled by two values on the phase circle, acentrics by a list of values. The values can be indexed like and array. The **phase()** (p. 317) function returns the phase corresponding to the given array index. Conversion to and from Hendrickson-Lattman coefficients is provided. The object is templatised on the sampling of the phase circle.

The documentation for this class was generated from the following files:

- hkl_operators.h
- hkl_operators.cpp

19.104 clipper::Map_index_sort Class Reference

Generic map sorting class.

`#include <map_utils.h>`

Classes

- class **Compare_density**

  Internal helper class used for sorting.

Static Public Member Functions

- template<class M >
  static void **sort_increasing** (const M &map, std::vector< int > &index)

  Sort a list into increasing order.

- template<class M >
  static void **sort_decreasing** (const M &map, std::vector< int > &index)

  Sort a list into decreasing order.

19.104.1 Detailed Description

Generic map sorting class. This class is used to sort a vector of integer indices into a map. This includes sorting the whole map to get highest or lowest density first, or sorting some subset, e.g. a list of peak indices. Integer indices are used because they are the most compact way of referencing a unique map location. e.g.

```
clipper::Xmap<float>::Map_reference_index ix;
std::vector<int> index;
for ( ix = xmap.first(); !ix.last(); ix.next() )
  index.push_back( ix.index() );
Map_index_sort::sort_decreasing( xmap, index );
```

19.104.2 Member Function Documentation

19.104.2.1 template<class M > void clipper::Map_index_sort::sort_increasing (const M & *map*, std::vector< int > & *index*) `[inline, static]`

Sort a list into increasing order. The index is sorted in place map The map to be sorted. index The list of indices to sort.

19.104.2.2 template<class M > void clipper::Map_index_sort::sort_decreasing (const M & *map*, std::vector< int > & *index*) `[inline, static]`

Sort a list into decreasing order. The index is sorted in place map The map to be sorted. index The list of indices to sort.

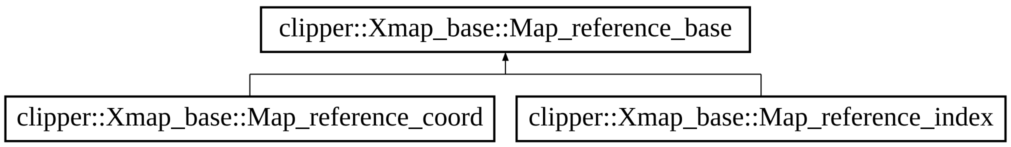
The documentation for this class was generated from the following files:

- map_utils.h
- map_utils.cpp

19.105 clipper::Xmap_base::Map_reference_base Class Reference

Map reference base class.

#include <xmap.h>Inheritance diagram for clipper::Xmap_base::Map_reference_base::



Public Member Functions

- const **Xmap_base** & **base_xmap** () const

  return the parent ***Xmap*** *(*p. *463)*

- const int & **index** () const

  Get the index into the map data array.

- bool **last** () const

  Check for end of map.

Protected Attributes

- const **Xmap_base** ∗ **map_**

  pointer to map for which this ***Map_reference_index*** *(*p. *328) is defined*

- int **index_**

  integer index_ into map data array

19.105.1 Detailed Description

Map reference base class. This is a reference to an Map. It forms a base class for index-like and coordinate-like Map references. If you write a method which will work with either, then specify this instead of either of the derived classed.

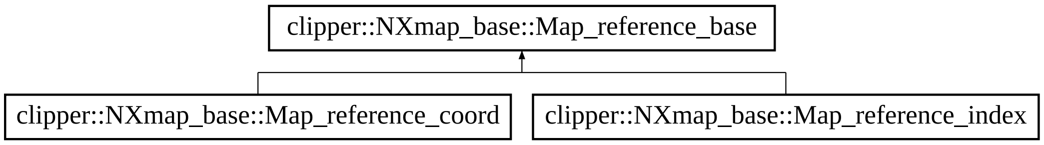
The documentation for this class was generated from the following file:

- xmap.h

19.106 clipper::NXmap_base::Map_reference_base Class Reference

Map reference base class.

`#include <nxmap.h>`Inheritance diagram for clipper::NXmap_base::Map_reference_base::



Public Member Functions

- const **NXmap_base** & **base_nxmap** () const
  return the parent ***NXmap*** *(*p. *357)*
- const int & **index** () const
  Get the index into the map data array.
- bool **last** () const
  Check for end of map.

Protected Attributes

- const **NXmap_base** ∗ **map_**
  pointer to map for which this ***Map_reference_index*** *(*p. *330) is defined*
- int **index_**
  integer index into map data array

19.106.1 Detailed Description

Map reference base class. This is a reference to an Map. It forms a base class for index-like and coordinate-like Map references. If you write a method which will work with either, then specify this instead of either of the derived classed.

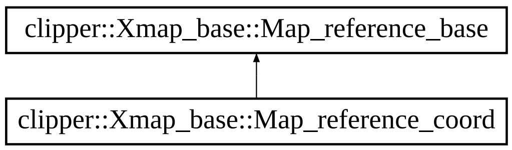
The documentation for this class was generated from the following file:

- nxmap.h

19.107 clipper::Xmap_base::Map_reference_coord Class Reference

Map reference with coordinate-like behaviour.

#include <xmap.h>Inheritance diagram for clipper::Xmap_base::Map_reference_coord::



Public Member Functions

- **Map_reference_coord** ()
  Null constructor.
- **Map_reference_coord** (const **Xmap_base** &map)
  Constructor: takes parent map.
- **Map_reference_coord** (const **Xmap_base** &map, const **Coord_grid** &pos)
  Constructor: takes parent map and coord.
- const **Coord_grid** & **coord** () const
  Get current value of coordinate.
- const **Coord_orth coord_orth** () const
  Get current value of orthogonal coordinate.
- const int & **sym** () const
  Get current symmetry operator.
- **Map_reference_coord** & **set_coord** (const **Coord_grid** &pos)
  Set current value of coordinate - optimised for nearby coords.
- **Map_reference_coord** & **next** ()
  Simple increment.
- **Map_reference_coord** & **next_u** ()
  increment u
- **Map_reference_coord** & **next_v** ()
  increment v
- **Map_reference_coord** & **next_w** ()
  increment w
- **Map_reference_coord** & **prev_u** ()

increment u

- **Map_reference_coord** & **prev_v** ()

  decrement v

- **Map_reference_coord** & **prev_w** ()

  decrement w

- **Map_reference_coord** & **operator=** (const **Coord_grid** &pos)

  Assignment operator from a coord.

- const **Xmap_base** & **base_xmap** () const

  *return the parent **Xmap** (p. 463)*

- const int & **index** () const

  Get the index into the map data array.

- bool **last** () const

  Check for end of map.

Protected Member Functions

- void **edge** ()

  Reset index for a different symop when we hit the map border.

Protected Attributes

- int **sym_**

  Current symop.

- **Coord_grid pos_**

  Current coord.

19.107.1 Detailed Description

Map reference with coordinate-like behaviour. This is a reference to a map coordinate. It behaves like a coordinate, but also stores the index of the corresponding point in the map, and the symmetry operator required to get there. It also implements methods for iterating through the a map. Since the current coordinate and symmetry are stored, **coord()** (p. 323) is fast. However, it requires 1 pointer and 5 words of storage.

Note:

The following methods are inherited from **Map_reference_base** (p. 321) but are documented here for convenience: **base_xmap()** (p. 324), **index()** (p. 324), **last()** (p. 324).

19.107.2 Member Function Documentation

19.107.2.1 Map_reference_coord& clipper::Xmap_base::Map_reference_coord::next () [inline]

Simple increment. Use of this function resets the stored coordinate and sym

References clipper::Xmap_base::asu, clipper::Grid_range::deindex(), clipper::Xmap_base::Map_reference_base::index_, last(), clipper::Xmap_base::Map_reference_base::map_, clipper::Xmap_base::map_grid, pos_, and sym_.

Referenced by Map_reference_coord().

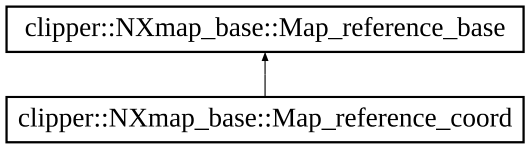
The documentation for this class was generated from the following files:

- xmap.h
- xmap.cpp

19.108 clipper::NXmap_base::Map_reference_coord Class Reference

Map reference with coordinate-like behaviour.

`#include <nxmap.h>`Inheritance diagram for clipper::NXmap_base::Map_reference_coord::



Public Member Functions

- **Map_reference_coord** ()
  Null constructor.
- **Map_reference_coord** (const **NXmap_base** &map)
  Constructor: need parent map.
- **Map_reference_coord** (const **NXmap_base** &map, const **Coord_grid** &pos)
  Constructor: need parent map and coord.
- **Coord_grid coord** () const
  Get current value of coordinate.
- const **Coord_orth coord_orth** () const
  Get current value of orthogonal coordinate.
- **Map_reference_coord** & **set_coord** (const **Coord_grid** &pos)
  Set current value of coordinate - optimised for nearby coords.
- **Map_reference_coord** & **next** ()
  Simple increment.
- **Map_reference_coord** & **next_u** ()
  increment u
- **Map_reference_coord** & **next_v** ()
  increment v
- **Map_reference_coord** & **next_w** ()
  increment w
- **Map_reference_coord** & **prev_u** ()
  decrement u
- **Map_reference_coord** & **prev_v** ()

decrement v

- **Map_reference_coord** & **prev_w** ()

  decrement w

- **Map_reference_coord** & **operator=** (const **Coord_grid** &pos)

  Assignment operator from a coord.

- const **NXmap_base** & **base_nxmap** () const

  *return the parent **NXmap** (p. 357)*

- const int & **index** () const

  Get the index into the map data array.

- bool **last** () const

  Check for end of map.

Protected Attributes

- **Coord_grid pos_**

  Current coord.

19.108.1 Detailed Description

Map reference with coordinate-like behaviour. This is a reference to a map coordinate. It behaves like a coordinate, but also stores the index of the corresponding point in the map. It also implements methods for iterating through the a map. Since the current coordinate is stored, **coord()** (p. 326) is fast. However it required 5 words of storage.

Note:

The following methods are inherited from **Map_reference_base** (p. 322) but are documented here for convenience: **base_nxmap()** (p. 327), **index()** (p. 327), **last()** (p. 327).

19.108.2 Member Function Documentation

19.108.2.1 Map_reference_coord& clipper::NXmap_base::Map_reference_coord::next () `[inline]`

Simple increment. Use of this function resets the stored coordinate and sym

References clipper::Grid::deindex(), clipper::NXmap_base::grid_, clipper::NXmap_base::Map_reference_base::index_, clipper::NXmap_base::Map_reference_base::map_, and pos_.

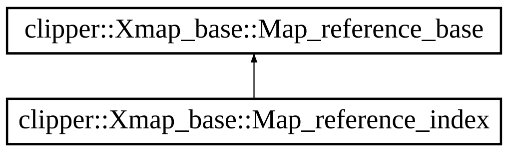
The documentation for this class was generated from the following file:

- nxmap.h

19.109 clipper::Xmap_base::Map_reference_index Class Reference

Map reference with index-like behaviour.

#include <xmap.h>Inheritance diagram for clipper::Xmap_base::Map_reference_index::



Public Member Functions

- **Map_reference_index** ()
  Null constructor.
- **Map_reference_index** (const **Xmap_base** &map)
  Constructor: takes parent map.
- **Map_reference_index** (const **Xmap_base** &map, const **Coord_grid** &pos)
  Constructor: takes parent map and coord.
- **Coord_grid coord** () const
  Get current grid coordinate.
- const **Coord_orth coord_orth** () const
  Get current value of orthogonal coordinate.
- **Map_reference_index** & **set_coord** (const **Coord_grid** &pos)
  Set current value of coordinate - optimised for nearby coords.
- **Map_reference_index** & **next** ()
  Simple increment.
- int **index_offset** (const int &**du**, const int &**dv**, const int &**dw**) const
  Index of neighbouring point.
- const **Xmap_base** & **base_xmap** () const
  return the parent ***Xmap*** *(p. 463)*
- const int & **index** () const
  Get the index into the map data array.
- bool **last** () const
  Check for end of map.

19.109.1 Detailed Description

Map reference with index-like behaviour. This is a reference to a map coordinate. It behaves like a simple index into the map, but can be easily converted into a coordinate as and when required. It also implements methods for iterating through the unique portion of a map. It is very compact, but **coord()** (p. 328) involves some overhead and loses any information concerning symmetry equivelents.

Note:

The following methods are inherited from **Map_reference_base** (p. 321) but are documented here for convenience: **base_xmap()** (p. 328), **index()** (p. 328), **last()** (p. 328).

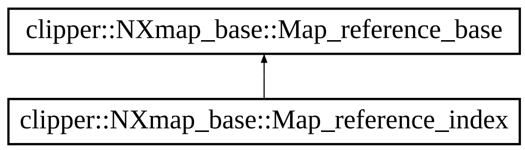
The documentation for this class was generated from the following file:

- xmap.h

19.110 clipper::NXmap_base::Map_reference_index Class Reference

Map reference with index-like behaviour.

`#include <nxmap.h>`Inheritance diagram for clipper::NXmap_base::Map_reference_index::



Public Member Functions

- **Map_reference_index** ()

  Null constructor.

- **Map_reference_index** (const **NXmap_base** &map)

  Constructor: need parent map.

- **Map_reference_index** (const **NXmap_base** &map, const **Coord_grid** &pos)

  Constructor: need parent map and coord.

- **Coord_grid coord** () const

  Get current grid coordinate.

- const **Coord_orth coord_orth** () const

  Get current value of orthogonal coordinate.

- **Map_reference_index** & **set_coord** (const **Coord_grid** &pos)

  Set current value of coordinate - optimised for nearby coords.

- **Map_reference_index** & **next** ()

  Simple increment.

- int **index_offset** (const int &du, const int &dv, const int &**dw**) const

  Index of neighbouring point.

- const **NXmap_base** & **base_nxmap** () const

  *return the parent **NXmap** (p. 357)*

- const int & **index** () const

  Get the index into the map data array.

- bool **last** () const

  Check for end of map.

19.110.1 Detailed Description

Map reference with index-like behaviour. This is a reference to a map coordinate. It behaves like a simple index into the map, but can be easily converted into a coordinate as and when required. It also implements methods for iterating through the map. It is very compact, but **coord()** (p. 330) involves some overhead.

Note:

The following methods are inherited from **Map_reference_base** (p. 322) but are documented here for convenience: **base_nxmap()** (p. 330), **index()** (p. 330), **last()** (p. 330).

The documentation for this class was generated from the following file:

- nxmap.h

19.111 clipper::Map_stats Class Reference

Generic map statistics class.

`#include <map_utils.h>`

Public Member Functions

- **Map_stats** ()
  null constructor
- template<class M >
  Map_stats (const M &map)
  *Constructor: from **Xmap** (p. 463).*
- const **ftype** & **mean** () const
  Mean of map.
- const **ftype** & **std_dev** () const
  Std deviation of map.
- const **ftype** & **min** () const
  Minimum of map.
- const **ftype** & **max** () const
  Maximum of map.
- const **Range range** () const
  ***Range** (p. 384).*

19.111.1 Detailed Description

Generic map statistics class. This class is used to calculate and store the mean and standard deviation of a generic map object of scalar types (e.g. **Xmap** (p. 463), **NXmap** (p. 357)). If the map contains NaN values, those points are excluded for the calculation. In the case of an **Xmap** (p. 463), the proper multiplicty corrections are applied to give statistics for a whole unit cell

19.111.2 Constructor & Destructor Documentation

19.111.2.1 template<class M > clipper::Map_stats::Map_stats (const M & *map*) `[inline]`

Constructor: from **Xmap** (p. 463). For float and double maps map The map for which moments are to be calculated.

References clipper::Util::is_nan().

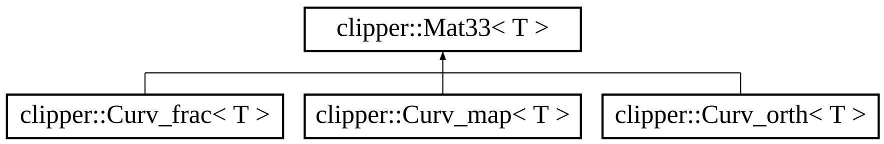
The documentation for this class was generated from the following file:

- map_utils.h

19.112 clipper::Mat33< T > Class Template Reference

3x3-matrix class

`#include` <`clipper_types.h`>Inheritance diagram for clipper::Mat33< T >::



Public Member Functions

- **Mat33** ()

  null constructor

- **Mat33** (const T &m00, const T &m01, const T &m02, const T &m10, const T &m11, const T &m12, const T &m20, const T &m21, const T &m22)

  constructor

- template<class TT >
  Mat33 (const **Mat33**< TT > &m)

  constructor: copy/convert

- template<class TT >
  Mat33 (const **Mat33sym**< TT > &m)

  constructor: copy/convert from symmetric matrix

- T **det** () const

  determinant

- **Mat33**< T > **inverse** () const

  inverse

- **Mat33**< T > **transpose** () const

  transpose

- bool **equals** (const **Mat33**< T > &m, const T &tol) const

  test equality

- const T & **operator()** (const int &i, const int &j) const

  get element

- T & **operator()** (const int &i, const int &j)

  set element

- **String format** () const

  *return formatted **String** (p. 426) representation*

- bool **is_null** () const

  test for null matrix (only valid for floating point types)

Static Public Member Functions

- static **Mat33**< T > **identity** ()

  return identity matrix

- static **Mat33**< T > **null** ()

  return null matrix (only valid for floating point types)

Friends

- **Vec3**< T > **operator*** (const **Mat33**< T > &m, const **Vec3**< T > &v)

  Matrix-vector product.

- **Vec3**< T > **operator*** (const **Vec3**< T > &v, const **Mat33**< T > &m)

  Vector-matrix product.

- **Mat33**< T > **operator*** (const **Mat33**< T > &m1, const **Mat33**< T > &m2)

  Matrix-matrix product.

- **Mat33**< T > **operator+** (const **Mat33**< T > &m1, const **Mat33**< T > &m2)

  Matrix *(*p. *338) sum.*

- **Mat33**< T > **operator-** (const **Mat33**< T > &m)

  Unary minus.

19.112.1 Detailed Description

template<class T = ftype> class clipper::Mat33< T >

3x3-matrix class

19.112.2 Friends And Related Function Documentation

19.112.2.1 template<class T = ftype> Vec3<T> operator* (const Mat33< T > & *m*, const Vec3< T > & *v*) [friend]

Matrix-vector product. Assumes a column vector

19.112.2.2 template<class T = ftype> Vec3<T> operator* (const Vec3< T > & *v*, const Mat33< T > & *m*) [friend]

Vector-matrix product. Assumes a row vector, i.e. equivalent to the matrix-vector product with the matrix trasposed

The documentation for this class was generated from the following file:

- clipper_types.h

19.113 clipper::Mat33sym< T > Class Template Reference

Compressed form for 3x3 symmetric matrix class.

`#include <clipper_types.h>`

Public Member Functions

- **Mat33sym** ()
  null constructor
- template<class TT >
  Mat33sym (const **Mat33**< TT > &m)
  *constructor: from **Mat33** (p. 333) (does not check for symmetry)*
- template<class TT >
  Mat33sym (const **Mat33sym**< TT > &m)
  *constructor: from **Mat33sym** (p. 336)*
- **Mat33sym** (const T &c00, const T &c11, const T &c22, const T &c01, const T &c02, const T &c12)
  constructor: from coefficients
- **String format** () const
  *return formatted **String** (p. 426) representation*
- bool **is_null** () const
  test for null matrix (only valid for floating point types)
- T **quad_form** (const **Vec3**< T > &v) const
  return quadratic form with vector
- T **det** () const
  determinant
- **Mat33**< T > **sqrt** () const
  square root
- **Mat33sym**< T > **inverse** () const
  inverse
- const T & **mat00** () const
  element (0,0)
- const T & **mat11** () const
  element (1,1)
- const T & **mat22** () const
  element (2,2)

- const T & **mat01** () const
  element (0,1)
- const T & **mat02** () const
  element (0,2)
- const T & **mat12** () const
  element (1,2)
- const T & **operator()** (const int &i, const int &j) const
  access elements as 3x3 matrix (inefficient)

Static Public Member Functions

- static **Mat33sym**< T > **identity** ()
  return identity matrix
- static **Mat33sym**< T > **null** ()
  return null matrix (only valid for floating point types)

Friends

- **Vec3**< T > **operator*** (const **Mat33sym**< T > &m, const **Vec3**< T > &v)
  Matrix-vector product.
- **Mat33sym**< T > **operator+** (const **Mat33sym**< T > &m1, const **Mat33sym**< T > &m2)
  Matrix *(*p. *338)* *sum.*
- **Mat33sym**< T > **operator-** (const **Mat33sym**< T > &m)
  Unary minus.

19.113.1 Detailed Description

template<class T = ftype> class clipper::Mat33sym< T >

Compressed form for 3x3 symmetric matrix class.

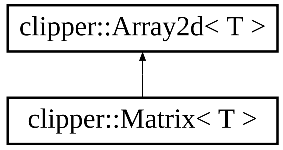
The documentation for this class was generated from the following file:

- clipper_types.h

19.114 clipper::Matrix< T > Class Template Reference

General matrix class: like **Array2d** (p. 105) but with numerical methods.

#include <clipper_types.h>Inheritance diagram for clipper::Matrix< T >::



Public Member Functions

- **Matrix** ()
  null constructor
- **Matrix** (const int &d1, const int &d2)
  constructor
- **Matrix** (const int &d1, const int &d2, T val)
  constructor
- std::vector< T > **solve** (const std::vector< T > &b) const
  equation solver (square matrices only)
- std::vector< T > **eigen** (const bool sort=true)
  eigenvalue calculation (square symmetric matrices only)

Friends

- std::vector< T > **operator*** (const **Matrix**< T > &m, const std::vector< T > &v)
  Matrix-vector product.

19.114.1 Detailed Description

template<class T = ftype> class clipper::Matrix< T >

General matrix class: like **Array2d** (p. 105) but with numerical methods.

19.114.2 Member Function Documentation

19.114.2.1 template<class T> std::vector< T > clipper::Matrix< T >::solve (const std::vector< T > & *b*) const `[inline]`

equation solver (square matrices only) Solve the system of linear equations Ax=b for x Uses elimination. Only suitable for small systems.

References clipper::Message::message(), clipper::Array2d< T >::rows(), and clipper::Util::swap().

Referenced by clipper::ResolutionFn::ResolutionFn(), and clipper::ResolutionFn_nonlinear::ResolutionFn_nonlinear().

19.114.2.2 template<class T > std::vector< T > clipper::Matrix< T >::eigen (const bool *sort* = `true`) `[inline]`

eigenvalue calculation (square symmetric matrices only) Find the Eigenvalues and Eigenvectors of the matrix. Uses the Jacobi method. Only suitable for small systems (dimension<20). The matrix is replaced by the matrix of eigenvectors (as columns).

Parameters:

sort Sort the eigenvalues and vectors, smallest first. (default=true)

Returns:

Vector of eigenvalues.

References clipper::Message::message(), clipper::Array2d< T >::rows(), and clipper::Util::swap().

Referenced by clipper::RTop_orth::RTop_orth().

19.114.3 Friends And Related Function Documentation

19.114.3.1 template<class T = ftype> std::vector<T> operator* (const Matrix< T > & *m*, const std::vector< T > & *v*) `[friend]`

Matrix-vector product. Assumes a column vector

The documentation for this class was generated from the following file:

- clipper_types.h

19.115 clipper::Message Class Reference

Message (p. 340) handler class.

#include <clipper_message.h>

Public Member Functions

- **Message** ()
  null constuctor

Static Public Member Functions

- static std::ostream & **stream** ()
  return the current stream
- static const int & **message_level** ()
  return the current message level
- static const int & **fatal_level** ()
  return the current fatal error level
- static void **set_stream** (std::ostream &stream)
  set the output stream
- static void **set_message_level** (const int &level)
  set the current message level
- static void **set_fatal_level** (const int &level)
  set the current fatal error level
- template<class T >
  static void **message** (const T &message)
  pass a message

19.115.1 Detailed Description

Message (p. 340) handler class. The message handler is a static class which handles messages and errors. It has 3 properties:

- the output stream: to which messages will be directed (default stderr)
- message level: messages with a level >= this will be output (default 5)
- fatal level: messages with a level >= this will be fatal (default 9)

Levels may be in the range 1-9. They are priorotised as follows:

- 1-4 messages, never fatal

- 5-8 warning, may be fatal
- 9: always fatal. The fatal level must be greater than or equal to the message level, and greater than or equal to 5.

A message is any object which implements the following methods:

```
const std::string& text() const;
int level() const;
```

The level method may be static. Messages are usually derived from **Message_base** (p. 342).

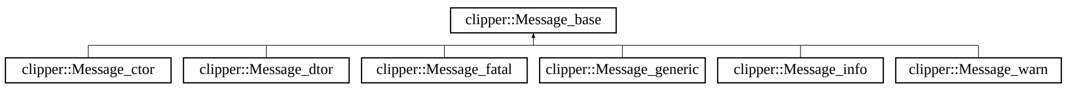
The documentation for this class was generated from the following files:

- clipper_message.h
- clipper_message.cpp

19.116 clipper::Message_base Class Reference

Base type for messages.

`#include <clipper_message.h>`Inheritance diagram for clipper::Message_base::



Public Member Functions

- const std::string & **text** () const
- int **level** () const

19.116.1 Detailed Description

Base type for messages.

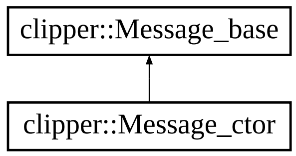
The documentation for this class was generated from the following file:

- clipper_message.h

19.117 clipper::Message_ctor Class Reference

Constructor message (level = 2).

#include <clipper_message.h>Inheritance diagram for clipper::Message_ctor::



Public Member Functions

- **Message_ctor** (const std::string &text)
- const std::string & **text** () const

Static Public Member Functions

- static int **level** ()

19.117.1 Detailed Description

Constructor message (level = 2).

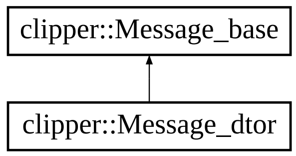
The documentation for this class was generated from the following file:

- clipper_message.h

19.118 clipper::Message_dtor Class Reference

Destructor message (level = 2).

#include <clipper_message.h>Inheritance diagram for clipper::Message_dtor::



Public Member Functions

- **Message_dtor** (const std::string &text)
- const std::string & **text** () const

Static Public Member Functions

- static int **level** ()

19.118.1 Detailed Description

Destructor message (level = 2).

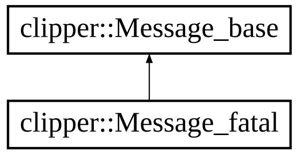
The documentation for this class was generated from the following file:

- clipper_message.h

19.119 clipper::Message_fatal Class Reference

Fatal message (level = 9).

`#include <clipper_message.h>`Inheritance diagram for clipper::Message_fatal::



Public Member Functions

- **Message_fatal** (const std::string &text)
- const std::string & **text** () const

Static Public Member Functions

- static int **level** ()

19.119.1 Detailed Description

Fatal message (level = 9).

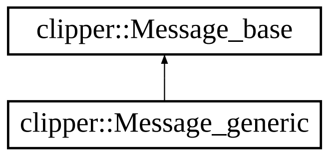
The documentation for this class was generated from the following file:

- clipper_message.h

19.120 clipper::Message_generic Class Reference

Generic message.

`#include <clipper_message.h>`Inheritance diagram for clipper::Message_generic::



Public Member Functions

- **Message_generic** (const int &level, const std::string &text)
- const std::string & **text** () const
- const int & **level** () const

19.120.1 Detailed Description

Generic message.

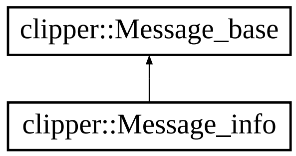
The documentation for this class was generated from the following file:

- clipper_message.h

19.121 clipper::Message_info Class Reference

Info message (level = 1).

`#include <clipper_message.h>`Inheritance diagram for clipper::Message_info::



Public Member Functions

- **Message_info** (const std::string &text)
- const std::string & **text** () const

Static Public Member Functions

- static int **level** ()

19.121.1 Detailed Description

Info message (level = 1).

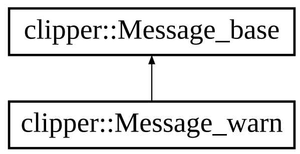
The documentation for this class was generated from the following file:

- clipper_message.h

19.122 clipper::Message_warn Class Reference

Warning message (level = 5).

`#include <clipper_message.h>`Inheritance diagram for clipper::Message_warn::



Public Member Functions

- **Message_warn** (const std::string &text)
- const std::string & **text** () const

Static Public Member Functions

- static int **level** ()

19.122.1 Detailed Description

Warning message (level = 5).

The documentation for this class was generated from the following file:

- clipper_message.h

19.123 clipper::Metric_tensor Class Reference

Metric tensor.

`#include <cell.h>`

Public Member Functions

- **Metric_tensor** ()
  null constructor
- **Metric_tensor** (const **ftype** &a, const **ftype** &b, const **ftype** &c, const **ftype** &alph, const **ftype** &beta, const **ftype** &gamm)
  constructor: takes parameters of normal or inverse cell
- **ftype lengthsq** (const **Vec3**<> &v) const
  apply metric to vector
- **ftype lengthsq** (const **Vec3**< int > &v) const
  apply metric to int vector
- **String format** () const
  *return formatted **String** (p. 426) representation*

19.123.1 Detailed Description

Metric tensor. The metric tensor is used to determine a distance in real or reciprocal space using fraction coordinates or Miller indices. It is symmetrical, so only the upper triangle is stored with the off-diagonal elements doubled.

19.123.2 Constructor & Destructor Documentation

19.123.2.1 clipper::Metric_tensor::Metric_tensor () `[inline]`

null constructor Null constructor

19.123.2.2 clipper::Metric_tensor::Metric_tensor (const ftype & *a*, const ftype & *b*, const ftype & *c*, const ftype & *alph*, const ftype & *beta*, const ftype & *gamm*)

constructor: takes parameters of normal or inverse cell Construct and initialise a metric tensor, given a set of real or reciprocal cell parameters.

Parameters:

a Length of **a** axis in Angstroms or reciprocal Angstroms.

b Length of **b** axis in Angstroms or reciprocal Angstroms.

c Length of **c** axis in Angstroms or reciprocal Angstroms.

alph Angle between **b** and **c** in radians.

beta Angle between **a** and **c** in radians.

gamm Angle between **a** and **b** in radians.

The documentation for this class was generated from the following files:

- cell.h
- cell.cpp

19.124 clipper::Mutex Class Reference

Mutex (p. 351) class: used for locking and unlocking shared resources.

#include <clipper_thread.h>

Public Member Functions

- **Mutex** ()
  constructor: create the mutex
- ~**Mutex** ()
  destructor: destroy the mutex
- void **lock** ()
  lock the mutex
- void **unlock** ()
  unlock the mutex

Protected Attributes

- CLIPPER_MUTEX_TYPE **mutex**

19.124.1 Detailed Description

Mutex (p. 351) class: used for locking and unlocking shared resources. Create a mutex for any sharted resource, i.e. non-stack object used by a multi-threaded program. The lock and unlock methods lock that resource. Recursive locks are not allowed.

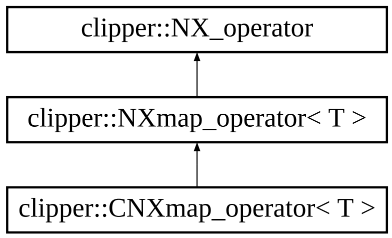
The documentation for this class was generated from the following file:

- clipper_thread.h

19.125 clipper::NX_operator Class Reference

NX_operator (p. 352): non-crystal map operator.

`#include` <`nxmap_operator.h`>Inheritance diagram for clipper::NX_operator::



Public Member Functions

- **NX_operator** ()

  null constructor

- **NX_operator** (const **Xmap_base** &xmap, const **NXmap_base** &nxmap, const **RTop_orth** &rtop)

  *constructor: from **Xmap** (p. 463), **NXmap** (p. 357), and operator*

- **NX_operator** (const **Cell** &cell, const **Grid_sampling** &grid, const **NXmap_base** &nxmap, const **RTop_orth** &rtop)

  *constructor: from cell, grid sampling, **NXmap** (p. 357), and operator*

- void **init** (const **Xmap_base** &xmap, const **NXmap_base** &nxmap, const **RTop_orth** &rtop)

  *initialiser:: from **Xmap** (p. 463), **NXmap** (p. 357), and operator*

- void **init** (const **Cell** &cell, const **Grid_sampling** &grid, const **NXmap_base** &nxmap, const **RTop_orth** &rtop)

  *initialiser:: from cell, grid sampling, **NXmap** (p. 357), and operator*

- **Coord_map coord_map** (const **Coord_frac** &c) const

  convert xtal frac coord to nxmap map coord

- **Coord_frac coord_frac** (const **Coord_map** &c) const

  convert nxmap map coord to xtal frac coord

- template<class I , class T , class M >
  T **nxmap_data** (const M &nxmap, const **Coord_grid** &c) const

  get value of nxmap at xmap grid coord using fastest appropriate method

- template<class I , class T , class M >
  T **xmap_data** (const M &xmap, const **Coord_grid** &c) const

  get value of xmap at nxmap grid coord using fastest appropriate method

- bool **is_null** () const

test if object has been initialised

- void **debug** () const

Protected Attributes

- **RTop xfrac_nxgrid**

  xtal_cell -> nxmap operator

- **RTop nxgrid_xfrac**

  nxmap -> xtal_cell operator

- **RTop xgrid_nxgrid**

  xtal_grid -> nxmap operator

- **RTop nxgrid_xgrid**

  nxmap -> xtal_grid operator

- **RTop**< int > **xgrid_nxgrid_int**

  xtal_grid -> nxmap integer operator

- **RTop**< int > **nxgrid_xgrid_int**

  nxmap -> xtal_grid integer operator

- bool **x_nx_is_int**

  true if int operator exists

- bool **x_nx_is_trn**

  true if int operator exists and is pure transln

- bool **nx_x_is_int**

  true if int operator exists

- bool **nx_x_is_trn**

  true if int operator exists and is pure transln

19.125.1 Detailed Description

NX_operator (p. 352): non-crystal map operator. This class holds a reference to a non-crystal map frame from somewhere within a crystallographic map frame. In the general case, an orthogonal rotation-translation operator is provided which maps the orthogonal frame of the crystal space onto the orthogonal frame of the **NXmap** (p. 357) space.

The object calculates and stores optimised transformations between the crystallgoraphic frame (described either in fractional or grid coordinates), and the **NXmap** (p. 357) grid. Fast paths are generated automatically if the grids are related.

19.125.2 Constructor & Destructor Documentation

19.125.2.1 clipper::NX_operator::NX_operator ()

null constructor The object is not initialised, and will return **is_null()** (p. 352).

References xfrac_nxgrid.

19.125.2.2 clipper::NX_operator::NX_operator (const Xmap_base & *xmap*, const NXmap_base & *nxmap*, const RTop_orth & *rtop*)

constructor: from **Xmap** (p. 463), **NXmap** (p. 357), and operator The operator and inverse operator, together with any possible optimisations, are constructed to relate the give crystallographic and non-crystallographic grid frames, using the supplied orthogonal operator.

Parameters:

xmap An **Xmap** (p. 463) defining the crystal grid frame.

nxmap An **NXmap** (p. 357) defining the non-crystal grid frame.

rtop The operator relating the orthogonal frame of the **NXmap** (p. 357) onto the orthogonal frame of the **Xmap** (p. 463).

References init().

19.125.2.3 clipper::NX_operator::NX_operator (const Cell & *cell*, const Grid_sampling & *grid*, const NXmap_base & *nxmap*, const RTop_orth & *rtop*)

constructor: from cell, grid sampling, **NXmap** (p. 357), and operator The operator and inverse operator, together with any possible optimisations, are constructed to relate the give crystallographic and non-crystallographic grid frames, using the supplied orthogonal operator.

Parameters:

cell The cell defining the crystal grid frame.

grid The grid defining the crystal grid frame.

nxmap An **NXmap** (p. 357) defining the non-crystal grid frame.

rtop The operator relating the orthogonal frame of the **NXmap** (p. 357) onto the orthogonal frame of the **Xmap** (p. 463).

References init().

19.125.3 Member Function Documentation

19.125.3.1 void clipper::NX_operator::init (const Xmap_base & *xmap*, const NXmap_base & *nxmap*, const RTop_orth & *rtop*)

initialiser:: from **Xmap** (p. 463), **NXmap** (p. 357), and operator The operator and inverse operator, together with any possible optimisations, are constructed to relate the give crystallographic and non-crystallographic grid frames, using the supplied orthogonal operator.

Parameters:

xmap An **Xmap** (p. 463) defining the crystal grid frame.

nxmap An **NXmap** (p. 357) defining the non-crystal grid frame.

rtop The operator relating the orthogonal frame of the **NXmap** (p. 357) onto the orthogonal frame of the **Xmap** (p. 463).

References clipper::Xmap_base::cell(), and clipper::Xmap_base::grid_sampling().

Referenced by NX_operator().

19.125.3.2 void clipper::NX_operator::init (const Cell & *cell*, const Grid_sampling & *grid*, const NXmap_base & *nxmap*, const RTop_orth & *rtop*)

initialiser:: from cell, grid sampling, **NXmap** (p. 357), and operator The operator and inverse operator, together with any possible optimisations, are constructed to relate the give crystallographic and non-crystallographic grid frames, using the supplied orthogonal operator.

Parameters:

cell The cell defining the crystal grid frame.

grid The grid defining the crystal grid frame.

nxmap An **NXmap** (p. 357) defining the non-crystal grid frame.

rtop The operator relating the orthogonal frame of the **NXmap** (p. 357) onto the orthogonal frame of the **Xmap** (p. 463).

References clipper::Mat33< T >::equals(), clipper::RTop< T >::equals(), clipper::RTop< T >::inverse(), clipper::Cell::matrix_frac(), clipper::Grid_sampling::matrix_frac_grid(), nx_x_is_int, nx_x_is_trn, nxgrid_xfrac, nxgrid_xgrid, nxgrid_xgrid_int, clipper::NXmap_base::operator_grid_orth(), clipper::RTop< T >::rot(), clipper::RTop< T >::trn(), x_nx_is_int, x_nx_is_trn, xfrac_nxgrid, xgrid_nxgrid, and xgrid_nxgrid_int.

19.125.3.3 template<class I , class T , class M > T clipper::NX_operator::nxmap_data (const M & *nxmap*, const Coord_grid & *c*) const [inline]

get value of nxmap at xmap grid coord using fastest appropriate method The density of the non-crystal map at the position corresponding to a crystallographic map grid coordinate is returned. If the grids match exactly either by pure translation or by rotation+translation, then fast paths are used to return the requested density directly. Otherwise the supplied interpolation template is used. No checking is performed for coordinates outside the **NXmap** (p. 357).

Parameters:

nxmap The non-crystal map (**NXmap** (p. 357)) to be queried.

c The grid coordinate in the crystallographic coordinate frame.

Returns:

The value of the **NXmap** (p. 357) at the requested position.

References clipper::Coord_grid::coord_map(), clipper::RTop< T >::trn(), x_nx_is_int, x_nx_is_trn, xgrid_nxgrid, and xgrid_nxgrid_int.

19.125.3.4 template<class I , class T , class M > T clipper::NX_-operator::xmap_data (const M & *xmap*, const Coord_grid & *c*) const [inline]

get value of xmap at nxmap grid coord using fastest appropriate method The density of the crystal map at the position corresponding to a non-crystallographic map grid coordinate is returned. If the grids match exactly either by pure translation or by rotation+translation, then fast paths are used to return the requested density directly. Otherwise the supplied interpolation template is used.

Parameters:

xmap The crystal map (**Xmap** (p. 463)) to be queried.

c The grid coordinate in the crystallographic coordinate frame.

Returns:

The value of the **Xmap** (p. 463) at the requested position.

References clipper::Coord_grid::coord_map(), nx_x_is_int, nx_x_is_trn, nxgrid_xgrid, nxgrid_xgrid_int, and clipper::RTop< T >::trn().

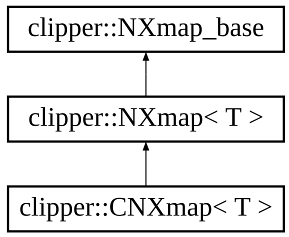
The documentation for this class was generated from the following files:

- nxmap_operator.h
- nxmap_operator.cpp

19.126 clipper::NXmap< T > Class Template Reference

NXmap<T>: actual non-crystallographic map class.

`#include <nxmap.h>`Inheritance diagram for clipper::NXmap< T >::



Public Member Functions

- **NXmap** ()

  Null constructor, for later initialisation.

- **NXmap** (const **Grid** &grid, const **RTop**<> &rt)

  Constructor: takes grid and orthogonal->grid coordinate operator.

- **NXmap** (const **Cell** &cell, const **Grid_sampling** &grid, const **Grid_range** &grid_-extent)

  Constructor: takes grid, cell, and extent.

- void **init** (const **Grid** &grid, const **RTop**<> &rt)

  initialiser: takes grid and orthogonal->grid coordinate operator

- void **init** (const **Cell** &cell, const **Grid_sampling** &grid, const **Grid_range** &grid_-extent)

  initialiser: takes grid, cell, and fraction limits

- const T & **operator[]** (const **NXmap_base::Map_reference_index** i) const

  get data by Map_reference_index

- T & **operator[]** (const **NXmap_base::Map_reference_index** i)

  set data by Map_reference_index

- const T & **operator[]** (const **NXmap_base::Map_reference_coord** i) const

  get data by Map_reference_coord

- T & **operator[]** (const **NXmap_base::Map_reference_coord** i)

  set data by Map_reference_coord

- const T & **get_data** (const **Coord_grid** &pos) const

  get a density value for an arbitrary position

- void **set_data** (const **Coord_grid** &pos, const T &val)

  set a density value for an arbitrary position

- template<class I >
  T **interp** (const **Coord_map** &pos) const

  get map value for map coord using supplied interpolator

- template<class I >
  void **interp_grad** (const **Coord_map** &pos, T &val, **Grad_map**< T > &grad) const

  get map value and grad for map coord using supplied interpolator

- template<class I >
  void **interp_curv** (const **Coord_map** &pos, T &val, **Grad_map**< T > &grad, **Curv_-map**< T > &curv) const

  get map value and curv for map coord using supplied interpolator

- const **Grid** & **grid** () const

  return the grid dimensions for this map

- const **RTop operator_orth_grid** () const

  return the orthogonal-to-grid coordinate operator

- const **RTop operator_grid_orth** () const

  return the grid-to-orthogonal coordinate operator

- const **Coord_orth coord_orth** (const **Coord_map** &cg) const

  convert map coordinate to orthogonal

- const **Coord_map coord_map** (const **Coord_orth** &co) const

  convert orthogonal coordinate to map

- const **Map_reference_index first** ()
- const **Map_reference_coord first_coord** ()
- const T & **operator**= (const T &value)

  assignment operator: assigns a single value to the whole map

- const **NXmap**< T > & **operator**+= (const **NXmap**< T > &other)

  add another map to this one

- const **NXmap**< T > & **operator-**= (const **NXmap**< T > &other)

  subtract another map from this one

19.126.1 Detailed Description

template<class T> class clipper::NXmap< T >

NXmap<T>: actual non-crystallographic map class. The non-crystallographic map class stores a map of arbitrary data type. Unlike an **Xmap** (p. 463) it is finite in extent and has no symmetry. An RT operator provides mapping onto an arbitrary orthogonal coordinate frame. Iterators provide efficient access to data.

This is derived from **NXmap_base** (p. 363), and adds the templatised data itself and the methods which deal with it.

Note:

The following methods are inherited from **NXmap_base** (p. 363) but are documented here for convenience: **grid()** (p. 358), **coord_orth()** (p. 361), coord_grid(), first(), first_coord().

19.126.2 Constructor & Destructor Documentation

19.126.2.1 template<class T > clipper::NXmap< T >::NXmap (const Grid & *grid*, const RTop<> & *rt*) `[inline]`

Constructor: takes grid and orthogonal->grid coordinate operator. Initialise an **NXmap** (p. 357) to some rhomboid chosen from within a crystal coordinate space, specified by the grid and a transformation from orthogonal to grid coordinates.

Parameters:

grid The grid dimensions of the desired map.

rt The rotation/transln op from orthogonal to grid coordinates.

References clipper::NXmap< T >::init().

19.126.2.2 template<class T > clipper::NXmap< T >::NXmap (const Cell & *cell*, const Grid_sampling & *grid*, const Grid_range & *grid_extent*) `[inline]`

Constructor: takes grid, cell, and extent. Initialise an **NXmap** (p. 357) to some rhomboid chosen from within a crystal grid coordinate space, specified by a cell, sampling and box within that grid. This is useful for creating an **NXmap** (p. 357) which exactly matches some subregion of a crystallographic map.

Parameters:

cell Unit cell defining the crystal space.

grid The grid sampling of the given unit cell.

grid_extent The map extent within that cell.

References clipper::NXmap< T >::init().

19.126.3 Member Function Documentation

19.126.3.1 template<class T > void clipper::NXmap< T >::init (const Grid & *grid*, const RTop<> & *rt*) `[inline]`

initialiser: takes grid and orthogonal->grid coordinate operator Initialise an **NXmap** (p. 357) to some rhomboid chosen from within a crystal coordinate space, specified by the grid and a transformation from orthogonal to grid coordinates.

Parameters:

grid The grid dimensions of the desired map.

rt The rotation/transln op from orthogonal to grid coordinates.

Reimplemented from **clipper::NXmap_base** (p. 366).

References clipper::Grid::size().

Referenced by clipper::NXmap< T >::init(), and clipper::NXmap< T >::NXmap().

19.126.3.2 template<class T > void clipper::NXmap< T >::init (const Cell & *cell*, const Grid_sampling & *grid*, const Grid_range & *grid_extent*) [inline]

initialiser: takes grid, cell, and fraction limits Initialise an **NXmap** (p. 357) to some rhomboid chosen from within a crystal grid coordinate space, specified by a cell, sampling and box within that grid. This is useful for creating an **NXmap** (p. 357) which exactly matches some subregion of a crystallographic map.

Parameters:

cell Unit cell defining the crystal space.

grid The grid sampling of the given unit cell.

grid_extent The map extent within that cell.

Reimplemented from **clipper::NXmap_base** (p. 366).

References clipper::NXmap< T >::init(), and clipper::Grid::size().

19.126.3.3 template<class T > template<class I > T clipper::NXmap< T >::interp (const Coord_map & *pos*) const [inline]

get map value for map coord using supplied interpolator The value of the map at the desired non-grid map coordinate are calculated using the supplied interpolator template.

Parameters:

pos The map coord at which the density is to be calcuated.

Returns:

The value of the density at that point. map coordinates (see Cell::coord_orth).

19.126.3.4 template<class T > template<class I > void clipper::NXmap< T >::interp_grad (const Coord_map & *pos*, T & *val*, Grad_map< T > & *grad*) const [inline]

get map value and grad for map coord using supplied interpolator The value of the map at the desired non-grid map coordinate and its gradient are calculated using the supplied interpolator template.

Parameters:

pos The map coord at which the density is to be calcuated.

val The value of the density at that point.

grad The interpolated gradient vector with respect to the map coordinates (see Cell::coord_-orth).

curv The interpolated curvature matrix with respect to the map coordinates (see Cell::coord_orth).

19.126.3.5 template<class T > template<class I > void clipper::NXmap< T >::interp_curv (const Coord_map & *pos*, T & *val*, Grad_map< T > & *grad*, Curv_map< T > & *curv*) const `[inline]`

get map value and curv for map coord using supplied interpolator The value of the map at the desired non-grid map coordinate and its gradient and curvature are calculated using the supplied interpolator template.

Parameters:

pos The map coord at which the density is to be calcuated.

val The value of the density at that point.

grad The interpolated gradient vector with respect to the map coordinates (see Cell::coord_-orth).

curv The interpolated curvature matrix with respect to the map coordinates (see Cell::coord_orth).

19.126.3.6 template<class T> const Coord_orth clipper::NXmap< T >::coord_orth (const Coord_map & *cm*) const

convert map coordinate to orthogonal

Parameters:

cm The grid coordinate to be converted.

Returns:

The equivalent orthogonal coordinate.

Reimplemented from **clipper::NXmap_base** (p. 365).

19.126.3.7 template<class T> const Coord_map clipper::NXmap< T >::coord_map (const Coord_orth & *co*) const

convert orthogonal coordinate to map

Parameters:

co The orthogonal coordinate to be converted.

Returns:

The equivalent grid coordinate.

Reimplemented from **clipper::NXmap_base** (p. 365).

19.126.3.8 template<class T > const T & clipper::NXmap< T >::operator= (const T & *value*) `[inline]`

assignment operator: assigns a single value to the whole map All values, including missing values, are overwritten by the value.

Parameters:

value The value to which the map is to be set.

References clipper::NXmap_base::Map_reference_index::index(), clipper::NXmap_base::Map_-reference_index::last(), and clipper::NXmap_base::Map_reference_index::next().

19.126.3.9 template< class T > const NXmap< T > & clipper::NXmap< T >::operator+= (const NXmap< T > & *other*) [inline]

add another map to this one The map grids must match.

References clipper::NXmap< T >::grid(), clipper::NXmap_base::Map_reference_index::index(), clipper::NXmap_base::Map_reference_index::last(), clipper::Message::message(), and clipper::NXmap_base::Map_reference_index::next().

19.126.3.10 template< class T > const NXmap< T > & clipper::NXmap< T >::operator-= (const NXmap< T > & *other*) [inline]

subtract another map from this one The map grids must match.

References clipper::NXmap< T >::grid(), clipper::NXmap_base::Map_reference_index::index(), clipper::NXmap_base::Map_reference_index::last(), clipper::Message::message(), and clipper::NXmap_base::Map_reference_index::next().

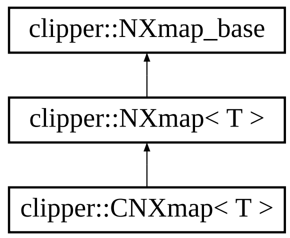
The documentation for this class was generated from the following file:

- nxmap.h

19.127 clipper::NXmap_base Class Reference

NXmap_base (p. 363): base for non-crystallographic map class.

#include <nxmap.h>Inheritance diagram for clipper::NXmap_base::



Classes

- class **Map_reference_base**

  Map reference base class.

- class **Map_reference_coord**

  Map reference with coordinate-like behaviour.

- class **Map_reference_index**

  Map reference with index-like behaviour.

Public Member Functions

- bool **is_null** () const

  test if object has been initialised

- const **Grid** & **grid** () const

  return the grid dimensions for this map

- const **RTop** & **operator_orth_grid** () const

  return the orthogonal-to-grid coordinate operator

- const **RTop** & **operator_grid_orth** () const

  return the grid-to-orthogonal coordinate operator

- **Coord_orth coord_orth** (const **Coord_map** &cm) const

  convert map coordinate to orthogonal

- **Coord_map coord_map** (const **Coord_orth** &co) const

  convert orthogonal coordinate to map

- bool **in_map** (const **Coord_grid** &pos) const

  is the given coord available in the map?

- template<class I >
  bool **in_map** (const **Coord_map** &cm) const

  is the given coord available in the map using the given interpolant?

- int **multiplicity** (const **Coord_grid** &pos) const

  get multiplicity of a map grid point (always 1 for ***NXmap*** *(p. 357))*

- **Map_reference_index first** () const

  return a basic ***Map_reference_index*** *(p. 330) for this map*

- **Map_reference_coord first_coord** () const

  return a coord ***Map_reference_index*** *(p. 330) for this map*

Protected Member Functions

- **NXmap_base** ()

  Null constructor, for later initialisation.

- void **init** (const **Grid** &grid, const **RTop**<> &rt)

  initialiser: takes grid and orthogonal->grid coordinate operator

- void **init** (const **Cell** &cell, const **Grid_sampling** &grid, const **Grid_range** &grid_-extent)

  initialiser: takes grid, cell, and fraction limits

Protected Attributes

- **Grid grid_**

  grid for the map

- **RTop rt_orth_grid**

  orth->grid operator

- **RTop rt_grid_orth**

  grid->orth operator

- int **du**
- int **dv**
- int **dw**

  steps for shifts along u,v,w

Friends

- class **NXmap_base::Map_reference_base**
- class **NXmap_base::Map_reference_index**
- class **NXmap_base::Map_reference_coord**

19.127.1 Detailed Description

NXmap_base (p. 363): base for non-crystallographic map class. The non-crystallographic map class stores a map of arbitrary data type. Unlike an **Xmap** (p. 463) it is finite in extent and has no symmetry. An RT operator provides mapping onto an arbitrary orthogonal coordinate frame. Iterators provide efficient access to data.

This base contains everything except the data, which is templated in the derived type clipper::NXmap<T>.

19.127.2 Constructor & Destructor Documentation

19.127.2.1 clipper::NXmap_base::NXmap_base () `[protected]`

Null constructor, for later initialisation. For later initialisation: see **init()** (p. 366)

References clipper::Message::message(), and rt_orth_grid.

19.127.3 Member Function Documentation

19.127.3.1 bool clipper::NXmap_base::is_null () const

test if object has been initialised

Returns:

true if the object has not been initalised.

References clipper::RTop< T >::is_null(), and rt_orth_grid.

Referenced by clipper::CNXmap_operator< T >::init().

19.127.3.2 Coord_orth clipper::NXmap_base::coord_orth (const Coord_map & *cm*) const `[inline]`

convert map coordinate to orthogonal

Parameters:

cm The grid coordinate to be converted.

Returns:

The equivalent orthogonal coordinate.

Reimplemented in **clipper::NXmap< T >** (p. 361).

References rt_grid_orth.

Referenced by clipper::NXmap_base::Map_reference_coord::coord_orth(), and clipper::NXmap_base::Map_reference_index::coord_orth().

19.127.3.3 Coord_map clipper::NXmap_base::coord_map (const Coord_orth & *co*) const `[inline]`

convert orthogonal coordinate to map

Parameters:

co The orthogonal coordinate to be converted.

Returns:

The equivalent grid coordinate.

Reimplemented in **clipper::NXmap< T >** (p. 361).

References rt_orth_grid.

Referenced by clipper::NXmap_base::Map_reference_coord::coord_orth(), and clipper::NXmap_base::Map_reference_index::coord_orth().

19.127.3.4 template<class I > bool clipper::NXmap_base::in_map (const Coord_map & *cm*) const [inline]

is the given coord available in the map using the given interpolant? Note that the higher the order of the interpolant, the more of the boundary of the map becomes inaccessible.

Parameters:

cm The coord_map to test.

Returns:

true if interpolation can be performed at that coordinate.

19.127.3.5 void clipper::NXmap_base::init (const Grid & *grid*, const RTop<> & *rt*) [protected]

initialiser: takes grid and orthogonal->grid coordinate operator Initialise an **NXmap** (p. 357) to some rhomboid chosen from within a crystal coordinate space, specified by the grid and a transformation from orthogonal to grid coordinates.

Parameters:

grid The grid dimensions of the desired map.

rt The rotation translation op from orthogonal to grid coordinates.

Reimplemented in **clipper::NXmap< T >** (p. 359).

References dw, grid_, clipper::Grid::index(), clipper::RTop< T >::inverse(), rt_grid_orth, and rt_orth_grid.

Referenced by init().

19.127.3.6 void clipper::NXmap_base::init (const Cell & *cell*, const Grid_sampling & *grid*, const Grid_range & *grid_extent*) [protected]

initialiser: takes grid, cell, and fraction limits Initialise an **NXmap** (p. 357) to some rhomboid chosen from within a crystal grid coordinate space, specified by a cell, sampling and box within that grid. This is useful for creating an **NXmap** (p. 357) which exactly matches some subregion of a crystallographic map.

Parameters:

cell Unit cell defining the crystal space.

grid The grid sampling of the given unit cell.

grid_extent The map extent within that cell.

Reimplemented in **clipper::NXmap< T >** (p. 360).

References init(), clipper::Cell::matrix_frac(), clipper::Grid_sampling::matrix_frac_grid(), and clipper::Grid_range::min().

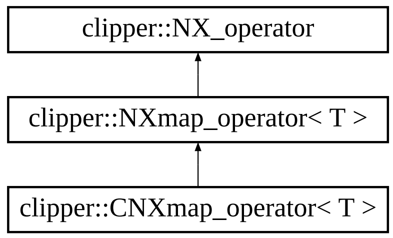
The documentation for this class was generated from the following files:

- nxmap.h
- nxmap.cpp

19.128 clipper::NXmap_operator< T > Class Template Reference

NXmap_operator (p. 368): non-crystal map operator referencing a particular **NXmap** (p. 357).

`#include <nxmap_operator.h>`Inheritance diagram for clipper::NXmap_operator< T >::



Public Member Functions

- **NXmap_operator** ()

  null constructor

- **NXmap_operator** (const **Xmap_base** &xmap, const **NXmap**< T > &nxmap, const **RTop_orth** &rtop)

  constructor: from ***Xmap*** *(p. 463),* ***NXmap*** *(p. 357), and operator*

- **NXmap_operator** (const **Cell** &cell, const **Grid_sampling** &grid, const **NXmap**< T > &nxmap, const **RTop_orth** &rtop)

  constructor: from cell, grid sampling, ***NXmap*** *(p. 357), and operator*

- void **init** (const **Xmap_base** &xmap, const **NXmap**< T > &nxmap, const **RTop_orth** &rtop)

  initialiser:: from ***Xmap*** *(p. 463),* ***NXmap*** *(p. 357), and operator*

- void **init** (const **Cell** &cell, const **Grid_sampling** &grid, const **NXmap**< T > &nxmap, const **RTop_orth** &rtop)

  initialiser:: from cell, grid sampling, ***NXmap*** *(p. 357), and operator*

- template<class I >
  T **nxmap_data** (const **Coord_grid** &c) const

  access ***NXmap*** *(p. 357) directly from xmap grid coord using fastest method*

- const **NXmap**< T > & **nxmap** () const

  get the target ***NXmap*** *(p. 357) of this operator*

19.128.1 Detailed Description

template<class T> class clipper::NXmap_operator< T >

NXmap_operator (p. 368): non-crystal map operator referencing a particular **NXmap** (p. 357). This class holds a reference to a non-crystal map object from somewhere within a crystallographic map frame. In the general case, an orthogonal rotation-translation operator is provided which maps the orthogonal frame of the crystal space onto the orthogonal frame of the **NXmap** (p. 357) space.

The object calculates and stores optimised transformations between the crystallgoraphic frame (described either in fractional or grid coordinates), and the **NXmap** (p. 357) grid. Fast paths are generated automatically if the grids are related.

Note:

This object differes from **NX_operator** (p. 352) in that it keeps a reference to an individual **NXmap** (p. 357), which may be used to access that object directly.

The documentation for this class was generated from the following file:

- nxmap_operator.h

19.129 clipper::ObjectCache< T > Class Template Reference

Object Cache manager.

`#include <clipper_memory.h>`

Classes

- class **Reference**

    ObjectCache *(p. 370) reference class.*

Public Types

- enum **MODE** { **NORMAL**, **MINMEM**, **MAXMEM** }

Public Member Functions

- **ObjectCache** ()

    constructor

- ~**ObjectCache** ()

    destructor, can message on contents

- void **set_mode** (const MODE &mode)

    set garbage collection mode

- void **purge** ()

    purge unreferenced objects from cache

- void **destroy** ()

    VERY DANGEROUS, DO NOT USE.

- void **debug** () const
- **Reference cache** (const typename T::Key &key)

    cache or return data by key

19.129.1 Detailed Description

template<class T> class clipper::ObjectCache< T >

Object Cache manager. The object cache is a tool for storing information which may appear several times. Examples include tables of information for spacegroups or crystallographic maps. When a new object is created, a check is first done to see if such an object already exists in the cache, in which case that copy is used. Otherwise a new copy is added to the cache.

A cached object must implement:

- a constructor from a type T

- a method 'matches(T)' which tests if it constructed from that object
- a 'format()' method, returning a string description of the contents The type T should be a compact unique description of the object.

Referring to the cache returns an ObjectCache<T>::Reference to a cache object. This object performs reference counting, which is used for garbage collection.

To retrieve the actual cached data, use the ObjectCache<T>::Reference::data() method. The data is held at a fixed memory location, therefore pointers to the data may be safely kept, as long as they are discarded as soon as the reference is discarded (at which point garbage collection may occur).

Ideally this would be a class with static members only, but some compilers have trouble with static members of template classes.

Garbage collection modes include:

- ObjectCache<T>::NORMAL : Remove an old object only when a new object is required and an old object is no longer in use. (default)
- ObjectCache<T>::MINMEM : Remove an old object as soon as it is no longer in use.
- ObjectCache<T>::MAXMEM : Never remove old objects. The more memory hungry modes may improve performance for some problems where a new object may be created which was already used and destroyed before.

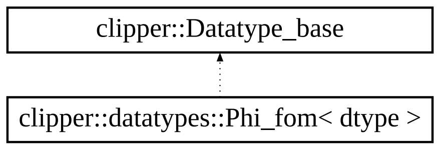
The documentation for this class was generated from the following files:

- clipper_memory.h
- clipper_instance.cpp

19.130 clipper::datatypes::Phi_fom< dtype > Class Template Reference

Reflection data type: best phi + fom.

#include <hkl_datatypes.h>Inheritance diagram for clipper::datatypes::Phi_fom< dtype >::



Public Member Functions

- **Phi_fom** (const dtype &phi, const dtype &fom)
- void **set_null** ()

  initialise to 'missing' (all elements are set to null)

- void **friedel** ()

  apply Friedel transformation to the data

- void **shift_phase** (const **ftype** &dphi)

  apply phase shift to the data

- bool **missing** () const

  return true if data is missing

- void **data_export** (**xtype** array[ ]) const

  conversion to array (for I/O)

- void **data_import** (const **xtype** array[ ])

  conversion from array (for I/O)

- const dtype & **phi** () const
- const dtype & **fom** () const
- dtype & **phi** ()
- dtype & **fom** ()

Static Public Member Functions

- static **String type** ()

  return the name of this data type

- static int **data_size** ()

  return number of data elements in this type

- static **String data_names** ()

  return names of data elements in this type

19.130.1 Detailed Description

template<class dtype> class clipper::datatypes::Phi_fom< dtype >

Reflection data type: best phi + fom.

The documentation for this class was generated from the following file:

- hkl_datatypes.h

19.131 clipper::Polar_ccp4 Class Reference

Polar_ccp4 (p. 374) angle class.

`#include <rotation.h>`

Public Member Functions

- **Polar_ccp4** ()
  null constructor
- **Polar_ccp4** (const **ftype** &omega, const **ftype** &phi, const **ftype** &kappa)
  constructor: from specified angles
- const **ftype** & **psi** () const
  return omega
- const **ftype** & **omega** () const
  return omega
- const **ftype** & **phi** () const
  return phi
- const **ftype** & **kappa** () const
  return kappa
- **String format** () const
  *return formatted **String** (p. 426) representation*

19.131.1 Detailed Description

Polar_ccp4 (p. 374) angle class.

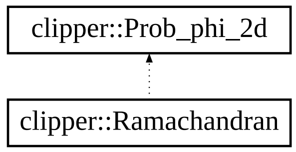
The documentation for this class was generated from the following files:

- rotation.h
- rotation.cpp

19.132 clipper::Prob_phi_2d Class Reference

2-d angular probability distibution class

`#include <ramachandran.h>`Inheritance diagram for clipper::Prob_phi_2d::



Public Member Functions

- void **init** (const int &size)
  initialise: with sampling
- void **accumulate** (const ftype32 table[ ])
  accumulate new table of samples to probability
- void **accumulate** (const **ftype** &phi1, const **ftype** &phi2, **ftype** wgt=1.0)
  accumulate new sample to probability
- void **normalise** ()
  normalise to integrate to 1/(2pi)^2
- **ftype probability** (const **ftype** &phi1, const **ftype** &phi2) const
  get probability for a particular pair of angles
- **String format** () const
  formatted string representation (as C++ code)
- const **ftype** & **data** (const int &i, const int &j) const
  2d read access
- **ftype** & **data** (const int &i, const int &j)
  2d write access

19.132.1 Detailed Description

2-d angular probability distibution class Base for **Ramachandran** (p. 382) class (and other similar classes, such as a pseudo-ramachandran plot or the JPD of two phases).

19.132.2 Member Function Documentation

19.132.2.1 void clipper::Prob_phi_2d::accumulate (const ftype & *phi1*, const ftype & *phi2*, ftype *wgt* = 1.0)

accumulate new sample to probability linear interpolation onto grid

References data(), clipper::Util::intf(), clipper::Util::mod(), and clipper::Util::twopi().

19.132.2.2 void clipper::Prob_phi_2d::normalise ()

normalise to integrate to 1/(2pi)^2 normalise mean value to 1/(2pi)^2

References clipper::Util::twopi().

Referenced by clipper::Ramachandran::init().

19.132.2.3 ftype clipper::Prob_phi_2d::probability (const ftype & *phi1*, const ftype & *phi2*) const

get probability for a particular pair of angles linear interpolation off of grid

Reimplemented in **clipper::Ramachandran** (p. 382).

References data(), clipper::Util::intf(), clipper::Util::mod(), and clipper::Util::twopi().

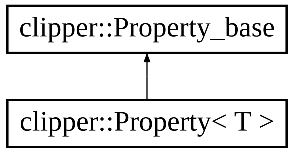
The documentation for this class was generated from the following files:

- **ramachandran.h**
- ramachandran.cpp

19.133 clipper::Property< T > Class Template Reference

Template for a property holding an arbitrary type.

`#include <clipper_memory.h>`Inheritance diagram for clipper::Property< T >::



Public Member Functions

- **Property** (const T &val)

  constructor: takes contents

- **Property_base** ∗ **clone** () const

  factory copy method

- const T & **value** () const

  return value of contents

19.133.1 Detailed Description

template<class T> class clipper::Property< T >

Template for a property holding an arbitrary type.

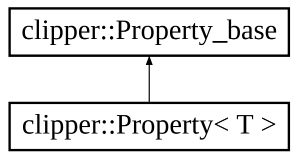
The documentation for this class was generated from the following file:

- clipper_memory.h

19.134 clipper::Property_base Class Reference

Base class for properties of arbitrary types.

`#include <clipper_memory.h>`Inheritance diagram for clipper::Property_base::



Public Member Functions

- virtual **Property_base** ∗ **clone** () const =0

  factory copy method

19.134.1 Detailed Description

Base class for properties of arbitrary types.

The documentation for this class was generated from the following file:

- clipper_memory.h

19.135 clipper::PropertyManager Class Reference

Class for holding a list of labelled properties of arbitrary types.

#include <clipper_memory.h>

Public Member Functions

- **PropertyManager** ()
  null constructor
- **PropertyManager** (const **PropertyManager** &mgr)
  copy constructor
- **PropertyManager** & **operator=** (const **PropertyManager** &mgr)
  assign op
- ~**PropertyManager** ()
  destructor
- **PropertyManager** & **copy** (const **PropertyManager** &mgr)
  copy manager
- bool **set_property** (const std::string &label, const **Property_base** &property)
  add a labelled property to the list
- const **Property_base** & **get_property** (const std::string &label) const
  get a labelled property from the list
- bool **exists_property** (const std::string &label) const
  test for property
- bool **delete_property** (const std::string &label)
  delete property

19.135.1 Detailed Description

Class for holding a list of labelled properties of arbitrary types. To add a property list to an object, derive it from this class, or include a member and mirror the methods. To add a property, simply call insert_property(label,property). Properties must be objects derived from clipper::Propert_-base. Usually, you can just use the template form, clipper::Property<T>.

To read a property which you know exists and is of a particular type, use:

```
const T& obj = dynamic_cast<const Property<T>& >(list.get_property( label )).
  value();
```

If you are unsure if a property is present, use the exists_property(label) method. If you are unsure of a property's type, dynamic cast a pointer and test for null. e.g.

```
if ( !list.exists_property( label ) ) { error("No such property"); }
const Property_base* ptr = &list.get_property( label );
if ( dynamic_cast<const T*>(ptr) == NULL )  { error("Wrong type"); }
const T& obj = *(dynamic_cast<const T*>(ptr));
```

19.135.2 Constructor & Destructor Documentation

19.135.2.1 clipper::PropertyManager::PropertyManager (const PropertyManager & *mgr*)

copy constructor Makes copies of all property objects.

References copy().

19.135.2.2 clipper::PropertyManager::∼PropertyManager ()

destructor Deletes all stored properties.

19.135.3 Member Function Documentation

19.135.3.1 PropertyManager & clipper::PropertyManager::operator= (const PropertyManager & *mgr*)

assign op Clears manager then makes copies of all property objects.

References copy().

19.135.3.2 PropertyManager & clipper::PropertyManager::copy (const PropertyManager & *mgr*)

copy manager This function is used by the copy constructor and assignement operator and is also useful for derived classes.

Referenced by operator=(), and PropertyManager().

19.135.3.3 const Property_base & clipper::PropertyManager::get_property (const std::string & *label*) const

get a labelled property from the list

Parameters:

label The label of the property to be returned.

Returns:

the property object.

References clipper::Message::message().

19.135.3.4 bool clipper::PropertyManager::exists_property (const std::string & *label*) const

test for property

Parameters:

label The label of the property to be tested.

Returns:

true on success.

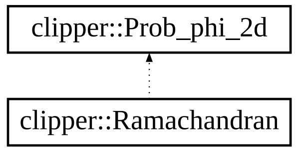
The documentation for this class was generated from the following files:

- clipper_memory.h
- clipper_memory.cpp

19.136 clipper::Ramachandran Class Reference

Ramachandran (p. 382) plot class.

`#include <ramachandran.h>`Inheritance diagram for clipper::Ramachandran::



Public Types

- enum **TYPE** {
  Gly, **Pro**, **NonGlyPro**, **NonGly**,
  All }

  enumeration of built-in ***Ramachandran*** *(p. 382) tables*

Public Member Functions

- **Ramachandran** ()

  null constructor

- **Ramachandran** (**TYPE** type)

  constructor: from standard plot

- void **init** (**TYPE** type)

  initialise: from standard plot

- void **set_thresholds** (**ftype** prob_favored=0.01, **ftype** prob_allowed=0.0005)

  change threshholds to different values

- **ftype probability** (const **ftype** &phi, const **ftype** &psi) const

  get probability for a particular pair of angles

- bool **favored** (const **ftype** &phi, const **ftype** &psi) const

  test if a pair of angles are in the favored region

- bool **allowed** (const **ftype** &phi, const **ftype** &psi) const

  test if a pair of angles are in the allowed region

19.136.1 Detailed Description

Ramachandran (p. 382) plot class. This class provides a reference **Ramachandran** (p. 382) plot for Gly, Pro, other, and combinations of those types of residues. The source data comes from the best residues from the 'top500' best-determined structures list of D. C. and J. S. Richardson, `http://kinemage.biochem.duke.edu/index.html`

The **Ramachandran** (p. 382) plot is normalised in inverse radians squared, so the mean value of a probability is $1/(2\ \mathrm{pi})^2$.

19.136.2 Constructor & Destructor Documentation

19.136.2.1 clipper::Ramachandran::Ramachandran (TYPE *type*)

constructor: from standard plot Construct a **Ramachandran** (p. 382) plot of a given type.

Parameters:

> ***type*** The residue type of the plot. Options include: Ramachandran::Gly, Ramachandran::Pro, Ramachandran::NonGlyPro, Ramachandran::NonGly, Ramachandran::All

References init().

19.136.3 Member Function Documentation

19.136.3.1 void clipper::Ramachandran::init (TYPE *type*)

initialise: from standard plot Construct a **Ramachandran** (p. 382) plot of a given type.

Parameters:

> ***type*** The residue type of the plot. Options include: Ramachandran::Gly, Ramachandran::Pro, Ramachandran::NonGlyPro, Ramachandran::NonGly, Ramachandran::All

References clipper::Prob_phi_2d::accumulate(), clipper::Prob_phi_2d::normalise(), and set_thresholds().

Referenced by Ramachandran().

19.136.3.2 void clipper::Ramachandran::set_thresholds (ftype *prob_favored* = `0.01`, ftype *prob_allowed* = `0.0005`)

change threshholds to different values Set thresholds for favorable and allowed regions of the **Ramachandran** (p. 382) plot. The US spelling is used because it is the same length as 'allowed'. I should get out more. Sorry.

Parameters:

> ***prob_favored*** The probability threshold for the favored region.
>
> ***prob_allowed*** The probability threshold for the allowed region.

Referenced by init().

The documentation for this class was generated from the following files:

- **ramachandran.h**
- ramachandran.cpp

19.137 clipper::Range< T > Class Template Reference

Range (p. 384) - upper and lower bounds of some type.

`#include <clipper_stats.h>`

Public Member Functions

- **Range** ()
  null constructor
- **Range** (const T &min, const T &max)
  constructor
- const T & **min** () const
  minimum value
- const T & **max** () const
  maximum value
- T **range** () const
  range = max - min
- void **include** (const T &datum)
  update limits to include a new datum
- bool **contains** (const T &datum) const
  test if data is within limits (min <= datum <= max)
- T **truncate** (const T &datum) const
  truncate data to be within range

19.137.1 Detailed Description

template<class T = ftype> class clipper::Range< T >

Range (p. 384) - upper and lower bounds of some type.

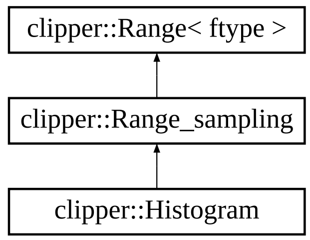
The documentation for this class was generated from the following file:

- clipper_stats.h

19.138 clipper::Range_sampling Class Reference

Range (p. 384) sampling: discrete sampling of a real range.

`#include <clipper_stats.h>`Inheritance diagram for clipper::Range_sampling::



Public Member Functions

- **Range_sampling** ()
  null constructor
- **Range_sampling** (const int &n)
  constructor: from number of samplings
- **Range_sampling** (const **Range**< **ftype** > &range, const int &n)
  constructor: from range and number of samplings
- **ftype indexf** (const **ftype** &x) const
  return fractional posn in counting range from x-value (0..n)
- **ftype x** (const **ftype** &i) const
  return x-value (0..n) from fractional posn in counting range
- int **index** (const **ftype** &x) const
  return nearest index to particular x-value
- int **index_bounded** (const **ftype** &x) const
  return nearest index to particular x-value (bounded 0...n-1)
- **ftype x** (const int &i) const
  return x-value corresponding to centre of i'th range
- **ftype x_min** (const int &i) const
  return x-value corresponding to bottom of i'th range
- **ftype x_max** (const int &i) const
  return x-value corresponding to top of i'th range
- int **size** () const
  return number of samplings in range

19.138.1 Detailed Description

Range (p. 384) sampling: discrete sampling of a real range.

The documentation for this class was generated from the following file:

- clipper_stats.h

19.139 clipper::TargetFn_base::Rderiv Class Reference

object holding the residual function and first two derivatives

`#include <resol_fn.h>`

Public Attributes

- **ftype r**
  the value of the function
- **ftype dr**
  first derivative w.r.t basis fn
- **ftype dr2**
  second derivative w.r.t basis fn

19.139.1 Detailed Description

object holding the residual function and first two derivatives

The documentation for this class was generated from the following file:

- resol_fn.h

19.140 clipper::ObjectCache< T >::Reference Class Reference

ObjectCache (p. 370) reference class.

`#include <clipper_memory.h>`

Public Member Functions

- **Reference** (const **Reference** &other)
- void **operator=** (const **Reference** &other)
- bool **is_null** () const
- const T & **data** () const

Friends

- class **ObjectCache< T >**

19.140.1 Detailed Description

template<class T> class clipper::ObjectCache< T >::Reference

ObjectCache (p. 370) reference class.

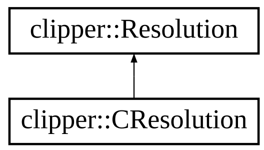
The documentation for this class was generated from the following files:

- clipper_memory.h
- clipper_instance.cpp

19.141 clipper::Resolution Class Reference

Resolution (p. 389) in angstroms.

`#include <coords.h>`Inheritance diagram for clipper::Resolution::



Public Member Functions

- **Resolution** ()
  null constructor
- **Resolution** (const **ftype** &resol_)
  constructor: from ftype
- void **init** (const **ftype** &resol_)
  initialiser: from ftype
- const **ftype** & **limit** () const
  get resolution limit
- **ftype invresolsq_limit** () const
  get invresolsq limit
- bool **is_null** () const
  test if value has been initialised

19.141.1 Detailed Description

Resolution (p. 389) in angstroms. This object represents a resolution limit which will be used for all aspects of a calculation. This is a base for a donor type.

19.141.2 Constructor & Destructor Documentation

19.141.2.1 clipper::Resolution::Resolution (const ftype & *resol_*) `[explicit]`

constructor: from ftype

Parameters:

resol_ The resolution limit in Angstroms.

References init().

19.141.3 Member Function Documentation

19.141.3.1 void clipper::Resolution::init (const ftype & *resol_*)

initialiser: from ftype

Parameters:

*resol_* The resolution limit in Angstroms.

Referenced by Resolution().

19.141.3.2 const ftype & clipper::Resolution::limit () const

get resolution limit

Returns:

The resolution limit in Angstroms.

Referenced by clipper::HKL_info::generate_hkl_list(), and clipper::Grid_sampling::init().

19.141.3.3 ftype clipper::Resolution::invresolsq_limit () const

get invresolsq limit

Returns:

The resolution limit in inverse squared Angstroms.

Referenced by clipper::HKL_info::add_hkl_list(), clipper::HKL_info::generate_hkl_list(), and clipper::HKL_sampling::HKL_sampling().

19.141.3.4 bool clipper::Resolution::is_null () const

test if value has been initialised

Returns:

true if the object has not been initalised.

Referenced by clipper::CGrid_sampling::init(), clipper::CHKL_info::init(), and clipper::HKL_-info::is_null().

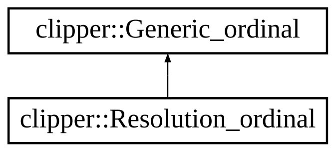
The documentation for this class was generated from the following files:

- coords.h
- coords.cpp

19.142 clipper::Resolution_ordinal Class Reference

Resolution (p. 389) ordinal gernerator.

#include <resol_basisfn.h>Inheritance diagram for clipper::Resolution_ordinal::



Public Member Functions

- void **init** (const **HKL_info** &hklinfo, const **ftype** &power)
  *initialiser: takes an **HKL_info** (p. 284) and uses all reflections.*
- void **init** (const **HKL_data_base** &hkldata, const **ftype** &power)
  *initialiser: takes an **HKL_data** (p. 267) & uses non-missing reflections.*
- void **init** (const **HKL_data_base** &hkldata, const **Cell** &cell, const **ftype** &power)
  *initialiser: takes an **HKL_data** (p. 267) + **Cell** (p. 133) & uses non-missing reflections.*

19.142.1 Detailed Description

Resolution (p. 389) ordinal gernerator. This class is a helper class for functions which need to divide reflections up by resolution whilst guaranteeing a certain distribution of number of reflections per range. It takes a list of reflections, one at a time, and calculates a function to get the approximate ordinal number of a reflection in a list sorted by resolution.

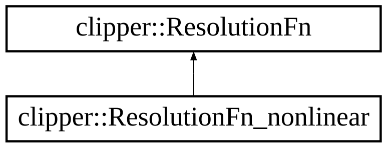
The documentation for this class was generated from the following files:

- resol_basisfn.h
- resol_basisfn.cpp

19.143 clipper::ResolutionFn Class Reference

2nd order resolution function evaluator

`#include <resol_fn.h>`Inheritance diagram for clipper::ResolutionFn::



Public Member Functions

- **ResolutionFn** (const **HKL_info** &hkl_info, const **BasisFn_base** &basisfn, const **TargetFn_base** &targetfn, const std::vector< **ftype** > &params, const **ftype** damp=0.0, const bool debug=false)

  constructor: need reflections, basis fn and target fn.

- **ftype f** (const **HKL_info::HKL_reference_index** &ih) const

  return the value of the basis function with the current paramters

- const std::vector< **ftype** > & **params** () const

  return the values of the parameters

- void **debug** () const

  print the target, gradient, and curvatures with respect to the params

Protected Member Functions

- void **calc_derivs** (const std::vector< **ftype** > &params, **ftype** &r, std::vector< **ftype** > &drdp, **Matrix**<> &drdp2) const

  calculate derivatives of target wrt params

- **ResolutionFn** ()

  null constructor

Protected Attributes

- const **HKL_info** ∗ **hkl_info_**

  reflection list

- const **TargetFn_base** ∗ **targetfn_**

  target function

- const **BasisFn_base** ∗ **basisfn_**

  basis function

- std::vector< **ftype** > **params_**

  basis function parameters

- **Cell cell_**

  cell

19.143.1 Detailed Description

2nd order resolution function evaluator This is an automatic evaluator for arbitrary functions of **HKL** (p. 261), most commonly used for evaluating a function of resolution (such a mean F^2 or sigmaa), although more general tasks including local scaling of reflections and anisotropic functions can also be handled. This form is for target functions which approach zero quadratically, e.g. least-squares targets.

This version implements a naive Newton-Raphson minimiser, which only uses the gradient and curvature of the target function, ignoring its value. It is ideal for quadratic targets with linear basis functions.

To evaluate a resolution function, this class must be provided with two objects:

- The basis function (and gradients), which describes the value of the function for any reflection given a set of paramters.

- The target function (and derivatives), which is used to determine the values of the basis function parameters.

For example, the following code may be used to calculate a smooth scaling function to scale one set of data to another using an anisotropic Gaussian scaling function:

```
make data object
   clipper::HKL_info hkls;
   clipper::HKL_data<clipper::data32::F_sigF> fsig1( hkls );
   clipper::HKL_data<clipper::data32::F_sigF> fsig2( hkls );
INITIALISE THEM HERE!

calculate the scaling function
   std::vector<clipper::ftype> params( 7, 0.0 );  // initial parameters
   clipper::BasisFn_aniso_gaussian basisfn;       // aniso gaussian
   clipper::TargetFn_scaleF1F2<clipper::data32::F_sigF,clipper::data32::F_sigF>t
     argetfn( fsig1, fsig2 );
   clipper::ResolutionFn_nonlinear rfn( hkls, basisfn, targetfn, params );

now scale the data
   clipper::HKL_info::HKL_reference_index ih;
   for ( ih = hkls.first(); !ih.last(); ih.next() )
     fsig1[ih].scale( sqrt( rfn.f(ih) ) );
```

The most useful isotropic resolution function is the **BasisFn_spline** (p. 131), since it is linear and provides a good fit to most data.

19.143.2 Constructor & Destructor Documentation

19.143.2.1 clipper::ResolutionFn::ResolutionFn (const HKL_info & *hkl_info*, const BasisFn_base & *basisfn*, const TargetFn_base & *targetfn*, const std::vector< ftype > & *params*, const ftype *damp* = 0.0, const bool *debug* = false)

constructor: need reflections, basis fn and target fn. The constructor performs the full minimisation calculation.

Parameters:

hkl_info **HKL_info** (p. 284) object which provides the reflection list.

basisfn The basis function used to describe the desired property.

targetfn The target function to be minimised.

params Initial values for the function parameters.

damp_ If > 0.0, shifts are fdamped during early cycles to help convergence with difficult bases/target conbinations.

References basisfn_, calc_derivs(), clipper::HKL_info::cell(), cell_, hkl_info_-, clipper::BasisFn_base::num_params(), params_, clipper::Matrix< T >::solve(), targetfn_, clipper::TargetFn_base::type(), and clipper::BasisFn_base::type().

19.143.3 Member Function Documentation

19.143.3.1 const std::vector< ftype > & clipper::ResolutionFn::params () const

return the values of the parameters

Returns:

The refined basis function parameters

References params_.

19.143.3.2 void clipper::ResolutionFn::calc_derivs (const std::vector< ftype > & *params*, ftype & *r*, std::vector< ftype > & *drdp*, Matrix<> & *drdp2*) const [protected]

calculate derivatives of target wrt params The target and its derivatives are calculated by the chain rule and accumulated over all HKLs.

Parameters:

params Current value of parameters.

r The value of the target function.

drdp The derivative of the target function.

drdp2 The curvature of the target function.

References basisfn_, cell_, clipper::BasisFn_base::Fderiv::df, clipper::BasisFn_base::Fderiv::df2, clipper::TargetFn_base::Rderiv::dr, clipper::TargetFn_base::Rderiv::dr2, clipper::HKL_-class::epsilonc(), f(), clipper::BasisFn_base::fderiv(), clipper::HKL_info::first(), clipper::HKL_-info::HKL_reference_index::hkl(), clipper::HKL_info::HKL_reference_index::hkl_class(), hkl_-info_, clipper::HKL_info::HKL_reference_index::last(), clipper::Util::max(), clipper::Util::min(), clipper::HKL_info::HKL_reference_index::next(), clipper::BasisFn_base::num_diagonals(), clipper::BasisFn_base::num_params(), clipper::TargetFn_base::Rderiv::r, clipper::TargetFn_-base::rderiv(), and targetfn_.

Referenced by ResolutionFn(), and clipper::ResolutionFn_nonlinear::ResolutionFn_nonlinear().

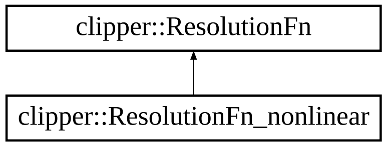
The documentation for this class was generated from the following files:

- resol_fn.h
- resol_fn.cpp

19.144 clipper::ResolutionFn_nonlinear Class Reference

2nd order resolution function evaluator

`#include <resol_fn.h>`Inheritance diagram for clipper::ResolutionFn_nonlinear::



Public Member Functions

- **ResolutionFn_nonlinear** (const **HKL_info** &hkl_info, const **BasisFn_base** &basisfn, const **TargetFn_base** &targetfn, const std::vector< **ftype** > &params, const **ftype** damp=0.0, const bool debug=false)

    constructor: need reflections, basis fn and target fn.

19.144.1 Detailed Description

2nd order resolution function evaluator This is an automatic evaluator for arbitrary functions of **HKL** (p. 261), most commonly used for evaluating a function of resolution (such a mean F^2 or sigmaa), although more general tasks including local scaling of reflections and anisotropic functions can also be handled. This form is for target functions which approach zero quadratically, e.g. least-squares targets.

Note:

This version implements a minimiser which uses both Newton-Raphson and gradient steps depending on the situation. It can be used for non-quadratic targets or non-linear basis functions.

To evaluate a resolution function, this class must be provided with two objects:

- The basis function (and gradients), which describes the value of the function for any reflection given a set of paramters.
- The target function (and derivatives), which is used to determine the values of the basis function parameters.

19.144.2 Constructor & Destructor Documentation

19.144.2.1 clipper::ResolutionFn_nonlinear::ResolutionFn_nonlinear (const HKL_info & *hkl_info*, const BasisFn_base & *basisfn*, const TargetFn_base & *targetfn*, const std::vector< ftype > & *params*, const ftype *damp* = `0.0`, const bool *debug* = `false`)

constructor: need reflections, basis fn and target fn. The constructor performs the full minimisation calculation.

Parameters:

hkl_info **HKL_info** (p. 284) object which provides the reflection list.

basisfn The basis function used to describe the desired property.

targetfn The target function to be minimised.

damp_ If > 0.0, shifts are fdamped during early cycles to help convergence with difficult bases/target conbinations

References clipper::ResolutionFn::basisfn_, clipper::ResolutionFn::calc_derivs(), clipper::HKL_info::cell(), clipper::ResolutionFn::cell_, clipper::HKL_class::epsilonc(), clipper::ResolutionFn::f(), clipper::HKL_info::first(), clipper::HKL_info::HKL_reference_index::hkl_class(), clipper::ResolutionFn::hkl_info_, clipper::Util::is_nan(), clipper::HKL_info::HKL_reference_index::last(), clipper::HKL_info::HKL_reference_index::next(), clipper::BasisFn_base::num_params(), clipper::ResolutionFn::params_, clipper::TargetFn_base::rderiv(), clipper::Matrix< T >::solve(), and clipper::ResolutionFn::targetfn_.

The documentation for this class was generated from the following files:

- resol_fn.h
- resol_fn.cpp

19.145 clipper::Rotation Class Reference

Rotation (p. 398) class.

`#include <rotation.h>`

Public Types

- enum **EULERtype** {
  EulerXYZr, **EulerXYZs**, **EulerXYXr**, **EulerXYXs**,
  EulerXZXr, **EulerXZXs**, **EulerXZYr**, **EulerXZYs**,
  EulerYZXr, **EulerYZXs**, **EulerYZYr**, **EulerYZYs**,
  EulerYXYr, **EulerYXYs**, **EulerYXZr**, **EulerYXZs**,
  EulerZXYr, **EulerZXYs**, **EulerZXZr**, **EulerZXZs**,
  EulerZYZr, **EulerZYZs**, **EulerZYXr**, **EulerZYXs** }

  Enumeration of ***Euler*** *(p. 206) conventions.*

Public Member Functions

- **Rotation** ()

  null constructor

- template<int T>
  Rotation (const **Euler**< T > &euler)

  constructor: from generic ***Euler*** *(p. 206)*

- **Rotation** (const **Euler_ccp4** &euler)

  constructor: from ***Euler_ccp4*** *(p. 207)*

- **Rotation** (const **Polar_ccp4** &polar)

  constructor: from ***Polar_ccp4*** *(p. 374)*

- **Rotation** (const **Mat33**<> &matrix)

  constructor: from ***Matrix*** *(p. 338)*

- **Rotation** (const **ftype** &w, const **ftype** &x, const **ftype** &y, const **ftype** &z)

  constructor: from components

- const **ftype** & **w** () const

  return w component

- const **ftype** & **x** () const

  return x component

- const **ftype** & **y** () const

  return y component

- const **ftype** & **z** () const

  return z component

- template<int T>
  Euler< T > **euler** () const

  *< return **Euler** (*p. *206) angles*

- **Euler_ccp4 euler_ccp4** () const

  *return **Euler_ccp4** (*p. *207) angles*

- **Polar_ccp4 polar_ccp4** () const

  *return **Polar_ccp4** (*p. *374) angles*

- **Mat33 matrix** () const

  return 3x3 matrix

- const **Rotation** & **norm** ()

  normalise this quaternion

- **ftype abs_angle** () const

  return absolute rotation angle

- **Rotation inverse** () const

  return inverse rotation

- bool **is_null** () const

  test for null (uninitialised) rotation

- **String format** () const

  *return formatted **String** (*p. *426) representation*

Static Public Member Functions

- static **Rotation zero** ()

  return zero rotation

- static **Rotation null** ()

  return null rotation

Protected Attributes

- **ftype w_**
- **ftype x_**
- **ftype y_**
- **ftype z_**

Friends

- **Rotation operator*** (const **Rotation** &r1, const **Rotation** &r2)
  combine two rotations

19.145.1 Detailed Description

Rotation (p. 398) class. This class represents a rotation. The internal representation is as a unit quaternion, which is easily combined, inverted, or converted to or from other commonly used forms.

19.145.2 Member Function Documentation

19.145.2.1 Euler_ccp4 clipper::Rotation::euler_ccp4 () const

return **Euler_ccp4** (p. 207) angles If beta ~= 0, then alpha is set to zero.

Returns:

The **Euler_ccp4** (p. 207) angles.

19.145.2.2 Polar_ccp4 clipper::Rotation::polar_ccp4 () const

return **Polar_ccp4** (p. 374) angles If omega ~= 0, then phi is set to zero.

Returns:

The **Polar_ccp4** (p. 374) angles.

Referenced by clipper::RTop_orth::axis_coordinate_near().

19.145.2.3 Mat33 clipper::Rotation::matrix () const

return 3x3 matrix The resulting rotation matrix would commonly be used to construct a **clipper::RTop_orth** (p. 407).

Returns:

The rotation matrix.

Referenced by clipper::RTop_orth::RTop_orth().

19.145.2.4 const Rotation & clipper::Rotation::norm ()

normalise this quaternion The normalisation is performed in-place. If a rotation becomes significantly denormalised, the conversion methods will fail. Therefore it may be safer to call this before a conversion.

Referenced by clipper::RTop_orth::axis_coordinate_near(), and clipper::RTop_orth::RTop_-orth().

The documentation for this class was generated from the following files:

- rotation.h
- rotation.cpp

19.146 clipper::RTop< T > Class Template Reference

Rotation-translation operator.

`#include <clipper_types.h>`

Public Member Functions

- **RTop** ()
  null constructor
- **RTop** (const **Mat33**< T > &r)
  constructor: from rotation
- **RTop** (const **Mat33**< T > &r, const **Vec3**< T > &t)
  constructor: from rotation and translation
- **RTop**< T > **inverse** () const
  inverse
- bool **equals** (const **RTop**< T > &m, const T &tol) const
  test equality with some tolerance
- const **Mat33**< T > & **rot** () const
  get rotation
- const **Vec3**< T > & **trn** () const
  get translation
- **Mat33**< T > & **rot** ()
  set rotation
- **Vec3**< T > & **trn** ()
  set translation
- bool **is_null** () const
  test for null operator
- **String format** () const
  return formatted ***String*** *(p. 426) representation*

Static Public Member Functions

- static **RTop**< T > **identity** ()
  return identity operator
- static **RTop**< T > **null** ()
  return identity operator

Friends

- **Vec3**< T > **operator**∗ (const **RTop**< T > &r, const **Vec3**< T > &v)
  *apply **RTop** (p. 402) to vector*
- **RTop**< T > **operator**∗ (const **RTop**< T > &r1, const **RTop**< T > &r2)
  ***RTop** (p. 402) product.*

19.146.1 Detailed Description

template<class T = ftype> class clipper::RTop< T >

Rotation-translation operator.

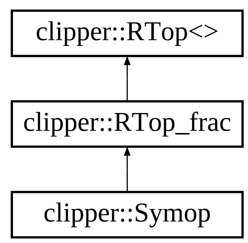
The documentation for this class was generated from the following file:

- clipper_types.h

19.147 clipper::RTop_frac Class Reference

Fractional operator class.

`#include <symop.h>`Inheritance diagram for clipper::RTop_frac::



Public Member Functions

- **RTop_frac** ()

  null constructor

- **RTop_frac** (const **RTop**<> &o)

  constructor: copy/convert

- **RTop_frac** (const **Mat33**<> &r)

  constructor: from rotation

- **RTop_frac** (const **String** &strop)

  constructor: from string description

- **RTop_frac** (const **Mat33**<> &r, const **Vec3**<> &t)

  constructor: from rotation and translation

- **RTop_orth rtop_orth** (const **Cell** &cell) const

  fractional-orthogonal conversion

- **RTop_frac inverse** () const

  inverse operator

Static Public Member Functions

- static **RTop_frac identity** ()

  return identity operator

- static **RTop_frac null** ()

  return null (uninitialised) operator

19.147.1 Detailed Description

Fractional operator class. This class is used for any RT-operator which operates on fractional coordinates. For a full list of methods, see **clipper::RTop** (p. 402)

19.147.2 Constructor & Destructor Documentation

19.147.2.1 clipper::RTop_frac::RTop_frac (const String & *strop*) `[explicit]`

constructor: from string description Construct an RT operator from a string description, e.g. 1/2x,z-y+2/3,x '*' is optional for multiplication, commas are compulsory.

References clipper::Message::message(), clipper::String::rational(), clipper::RTop<>::rot(), clipper::String::split(), and clipper::RTop<>::trn().

19.147.3 Member Function Documentation

19.147.3.1 RTop_orth clipper::RTop_frac::rtop_orth (const Cell & *cell*) const

fractional-orthogonal conversion

Parameters:

cell The cell concerned

Returns:

The transformed coordinate.

References clipper::Cell::matrix_frac(), and clipper::Cell::matrix_orth().

19.147.3.2 RTop_frac clipper::RTop_frac::inverse () const

inverse operator

Returns:

The inverse of the operator.

Reimplemented from **clipper::RTop<>** (p. 402).

References RTop_frac().

Referenced by clipper::Spgr_descr::Symop_codes::init_hall().

19.147.3.3 RTop_frac clipper::RTop_frac::identity () `[static]`

return identity operator

Returns:

The identity operator.

Reimplemented from **clipper::RTop**<> (p. 402).

References RTop_frac().

Referenced by clipper::Spgr_descr::Symop_codes::init_hall().

19.147.3.4 RTop_frac clipper::RTop_frac::null () [static]

return null (uninitialised) operator

Returns:

The null (uninitialised) operator.

Reimplemented from **clipper::RTop**<> (p. 402).

References RTop_frac().

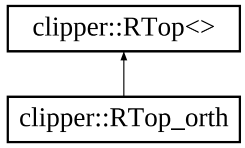
The documentation for this class was generated from the following files:

- symop.h
- symop.cpp

19.148 clipper::RTop_orth Class Reference

Orthogonal operator class.

`#include <coords.h>`Inheritance diagram for clipper::RTop_orth::



Public Member Functions

- **RTop_orth** ()
  null constructor
- **RTop_orth** (const **RTop**<> &o)
  constructor: copy/convert
- **RTop_orth** (const **Mat33**<> &r)
  constructor: from rotation
- **RTop_orth** (const **Mat33**<> &r, const **Vec3**<> &t)
  constructor: from rotation and translation
- **RTop_orth** (const std::vector< **Coord_orth** > &src, const std::vector< **Coord_orth** > &tgt)
  *constructor: from two vectors of **Coord_orth** (p. 182)*
- **RTop_orth** (const std::vector< **Coord_orth** > &src, const std::vector< **Coord_orth** > &tgt, const std::vector< **ftype** > &wgt)
  *constructor: from two vectors of **Coord_orth** (p. 182)*
- template<class T >
  RTop_orth (const T &src, const T &tgt)
  constructor: from two atom-list type objects
- **RTop_frac rtop_frac** (const **Cell** &cell) const
  orthogonal-fractional conversion
- **RTop_orth inverse** () const
  inverse operator
- **Coord_orth axis_coordinate_near** (const **Coord_orth** &centre) const
  return point on axis near the specified coordinate
- **Coord_orth screw_translation** () const
  return screw translation

Static Public Member Functions

- static **RTop_orth identity** ()
  return identity operator
- static **RTop_orth null** ()
  return null (uninitialised) operator

19.148.1 Detailed Description

Orthogonal operator class. This class is used for any RT-operator which operates on orthogonal coordinates. For a full list of methods, see **clipper::RTop** (p. 402)

19.148.2 Constructor & Destructor Documentation

19.148.2.1 clipper::RTop_orth::RTop_orth (const std::vector< Coord_orth > & *src*, const std::vector< Coord_orth > & *tgt*)

constructor: from two vectors of **Coord_orth** (p. 182) Construct the operator which give the least-squares fit of one set of coordinates onto another. The coodinates are stored as STL vectors of **Coord_orth** (p. 182). The lists must be the same size, and each atom in the source list must correspond to the same atom in the target list. The algorithm employed is that of Kearsley, S.K. (1989) 'On the orthogonal transformation used for structural comparisons'. Acta Cryst. A45, 208-210.

Parameters:

src The source list (i.e. the atoms to be transformed).

tgt The target list (i.e. the fixed atoms).

References clipper::Matrix< T >::eigen(), clipper::Rotation::matrix(), clipper::Message::message(), clipper::Rotation::norm(), clipper::RTop<>::rot(), and RTop_orth().

19.148.2.2 clipper::RTop_orth::RTop_orth (const std::vector< Coord_orth > & *src*, const std::vector< Coord_orth > & *tgt*, const std::vector< ftype > & *wgt*)

constructor: from two vectors of **Coord_orth** (p. 182) Construct the operator which give the least-squares fit of one set of coordinates onto another. The coodinates are stored as STL vectors of **Coord_orth** (p. 182). The lists must be the same size, and each atom in the source list must correspond to the same atom in the target list. The algorithm employed is that of Kearsley, S.K. (1989) 'On the orthogonal transformation used for structural comparisons'. Acta Cryst. A45, 208-210.

Parameters:

src The source list (i.e. the atoms to be transformed).

tgt The target list (i.e. the fixed atoms).

wgt The weight to apply to each atom.

References clipper::Matrix< T >::eigen(), clipper::Rotation::matrix(), clipper::Message::message(), clipper::Rotation::norm(), clipper::RTop<>::rot(), and RTop_orth().

19.148.2.3 template<class T > clipper::RTop_orth::RTop_orth (const T & *src*, const T & *tgt*) [inline]

constructor: from two atom-list type objects Construct the operator which relates one atom-list like object onto another. The lists must be the same size, and have the following properties:

- a size() method.
- a [int] operator, with int ranging from 0 to size()-1.
- the object returned by the [] operator must have a coord_orth() method. Suitable objects include a vector of **Atom** (p. 107), or an **Atom_list** (p. 109).

References RTop_orth().

19.148.3 Member Function Documentation

19.148.3.1 RTop_frac clipper::RTop_orth::rtop_frac (const Cell & *cell*) const

orthogonal-fractional conversion

Parameters:

cell The cell concerned

Returns:

The transformed coordinate.

References clipper::Cell::matrix_frac(), and clipper::Cell::matrix_orth().

19.148.3.2 RTop_orth clipper::RTop_orth::inverse () const

inverse operator

Returns:

The inverse of the operator.

Reimplemented from **clipper::RTop**<> (p. 402).

References RTop_orth().

19.148.3.3 Coord_orth clipper::RTop_orth::axis_coordinate_near (const Coord_orth & *centre*) const

return point on axis near the specified coordinate

Parameters:

centre An arbitrary point.

Returns:

point on axis near the specified coordinate, 000 if rotation is zero

References clipper::Rotation::abs_angle(), clipper::Polar_ccp4::kappa(), clipper::Rotation::norm(), clipper::Rotation::polar_ccp4(), clipper::RTop<>::rot(), clipper::RTop<>::trn(), clipper::Rotation::x(), clipper::Rotation::y(), and clipper::Rotation::z().

19.148.3.4 Coord_orth clipper::RTop_orth::screw_translation () const

return screw translation

Returns:

screw translation, 000 if rotation is zero

References clipper::Rotation::abs_angle(), clipper::RTop<>::rot(), clipper::RTop<>::trn(), clipper::Rotation::x(), clipper::Rotation::y(), and clipper::Rotation::z().

19.148.3.5 RTop_orth clipper::RTop_orth::identity () `[static]`

return identity operator

Returns:

The identity operator.

Reimplemented from **clipper::RTop**<> (p. 402).

References RTop_orth().

19.148.3.6 RTop_orth clipper::RTop_orth::null () `[static]`

return null (uninitialised) operator

Returns:

The null (uninitialised) operator.

Reimplemented from **clipper::RTop**<> (p. 402).

References RTop_orth().

Referenced by clipper::CNXmap_operator< T >::CNXmap_operator(), and clipper::CNXmap_operator< T >::update().

The documentation for this class was generated from the following files:

- coords.h
- coords.cpp

19.149 clipper::data::SFdata Struct Reference

Public Attributes

- char **name** [8]
- ftype64 **a** [6]
- ftype64 **b** [6]

The documentation for this struct was generated from the following file:

- atomsf.cpp

19.150 clipper::data::SGdata Struct Reference

Public Attributes

- int **sghash**
- const char * **hall**
- const char * **hm**
- char **ext**
- int **num**

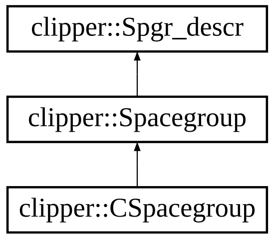
The documentation for this struct was generated from the following file:

- spacegroup_data.h

19.151 clipper::Spacegroup Class Reference

Spacegroup (p. 413) object.

`#include <spacegroup.h>`Inheritance diagram for clipper::Spacegroup::



Public Types

- enum **TYPE** { **Null**, **P1** }

  enumeration for fast construction of Null or P1 spacegroup

- enum **AXIS** { **A** = 0, **B** = 1, **C** = 2 }

  enumeration for cell axes

Public Member Functions

- **Spacegroup** ()

  null constructor

- **Spacegroup** (**TYPE** type)

  constructor: fast constructor for Null or P1 spacegroup

- **Spacegroup** (const **Spgr_descr** &spgr_descr)

  constructor: from spacegroup description

- void **init** (const **Spgr_descr** &spgr_descr)

  initialiser: from spacegroup description

- bool **is_null** () const

  test if object has been initialised

- const **Spgr_descr** & **descr** () const

  get spacegroup description

- const int & **num_symops** () const

  get number of symops

- const int & **num_primops** () const

  *get number of primitive symops (identical to **num_primitive_symops()** (p. 414))*

- const int & **num_primitive_symops** () const
  get number of primitive symops (inc identity and inversion)
- const int & **num_centering_symops** () const
  get number of centering symops (inc identity)
- const int & **num_inversion_symops** () const
  get number of inversion symops (inc identity)
- const int & **num_primitive_noninversion_symops** () const
  get number of primitive non-inversion symops (inc identity)
- const **Symop** & **symop** (const int &sym_no) const
  get n'th symop
- const **Symop** & **primitive_symop** (const int &sym_no) const
  get n'th primitive symop (identical to symop(sym_no))
- const **Symop** & **inversion_symop** (const int &sym_no) const
  get n'th inversion symop (0...1 max)
- const **Symop** & **centering_symop** (const int &sym_no) const
  get n'th centering symop (0...3 max)
- int **order_of_symmetry_about_axis** (const **AXIS** axis) const
  get the order of rotational symmetry about a given axis
- **HKL_class hkl_class** (const **HKL** &hkl) const
  get 'class' of reflection: multiplicity, allowed phase, absence
- bool **recip_asu** (const **HKL** &hkl) const
  test if hkl is in default reciprocal ASU
- int **product_op** (const int &s1, int &s2) const
  get symop number corresponding to the product of two symops
- int **inverse_op** (const int &s) const
  get symop number corresponding to the inverse of a symop
- **Coord_frac asu_max** () const
  get map ASU, upper bound
- **Coord_frac asu_min** () const
  get map ASU, lower bound
- bool **invariant_under_change_of_hand** () const
  test if change of hand preserves spacegroup
- int **spacegroup_number** () const
  return the spacegroup number

- **String symbol_hall** () const

  return the Hall symbol

- **String symbol_hm** () const

  return the H-M symbol

- **String symbol_laue** () const

  return the Laue group symbol

- void **debug** () const

Static Public Member Functions

- static **Spacegroup p1** ()

  Return P1 spacegroup.

- static **Spacegroup null** ()

  Return null spacegroup.

19.151.1 Detailed Description

Spacegroup (p. 413) object. The spacegroup object is a full description of a spacegroup, including all the most regularly used information in an efficient form. It may be initialised from a **clipper::Spgr_descr** (p. 421). This object.

For more details of spacegroup symbols, see Sydney R. Hall & Ralf W. Grosse-Kunstleve 'Concise Space-Group Symbols', `http://www.kristall.ethz.ch/LFK/software/sginfo/hall_symbols.html`

19.151.2 Constructor & Destructor Documentation

19.151.2.1 clipper::Spacegroup::Spacegroup (TYPE *type*) `[explicit]`

constructor: fast constructor for Null or P1 spacegroup Construct null or P1 spacegroup. This is faster than the normal constructor.

Parameters:

type Spacegroup::Null or Spacegroup::P1

References init(), and clipper::Spgr_descr::Spgr_descr().

19.151.2.2 clipper::Spacegroup::Spacegroup (const Spgr_descr & *spgr_descr*) `[explicit]`

constructor: from spacegroup description Construct a spacegroup and initialise with a spacegroup description.

Parameters:

spgr_ descr The spacegroup description.

References init().

19.151.3 Member Function Documentation

19.151.3.1 void clipper::Spacegroup::init (const Spgr_descr & *spgr_ descr*)

initialiser: from spacegroup description Initialise the spacegroup.

Parameters:

spgr_ descr The spacegroup description.

References clipper::ObjectCache< T >::cache(), clipper::Spgr_descr::generator_ops(), clipper::Spgr_descr::generators_, clipper::Spgr_descr::hash(), and clipper::Spgr_descr::hash_.

Referenced by Spacegroup().

19.151.3.2 bool clipper::Spacegroup::is_null () const

test if object has been initialised

Returns:

true if the object has not been initalised.

Referenced by clipper::HKL_info::init(), clipper::HKL_data_base::init(), clipper::CGrid_-sampling::init(), clipper::CXmap< T >::init(), clipper::CHKL_info::init(), clipper::Xmap_-base::is_null(), and clipper::HKL_info::is_null().

19.151.3.3 int clipper::Spacegroup::order_of_symmetry_about_axis (const AXIS *axis*) const

get the order of rotational symmetry about a given axis The number of rotational operators parallel to the specified axis is returned.

Parameters:

axis The axis, A, B or C.

Returns:

The order of the axis.

19.151.3.4 HKL_class clipper::Spacegroup::hkl_class (const HKL & *hkl*) const

get 'class' of reflection: multiplicity, allowed phase, absence The reflection class describes the type of a reflection in a given spacegroup, including centricity, systematic absence, phase restriction, and multiplicity.

This is a shortcut to constructing an **HKL_class** (p. 265) from the spacegroup and **HKL** (p. 261).

Parameters:

hkl The reflection **HKL** (p. 261)

Referenced by clipper::HKL_info::generate_hkl_list(), clipper::HKL_info::init(), and clipper::HKL_info::update_hkl_list().

19.151.3.5 bool clipper::Spacegroup::recip_asu (const HKL & *hkl*) const

test if hkl is in default reciprocal ASU The reciprocal ASU is chosen from one of 47 optimised functions.

Parameters:

hkl The **HKL** (p. 261) to test.

Returns:

true if the **HKL** (p. 261) is in the ASU.

References clipper::HKL::h(), clipper::HKL::k(), and clipper::HKL::l().

Referenced by clipper::HKL_info::find_sym(), clipper::HKL_info::generate_hkl_list(), and clipper::HKL_info::init().

19.151.3.6 Coord_frac clipper::Spacegroup::asu_max () const

get map ASU, upper bound The map ASU is an oblong which contains at least one assymetric unit. It is guaranteed to be contained withing the unit box. The lower limit is always 0,0,0.

Returns:

Fractional coordinate of the upper bound of the ASU.

19.151.3.7 Coord_frac clipper::Spacegroup::asu_min () const

get map ASU, lower bound The map ASU is an oblong which contains at least one assymetric unit. It is guaranteed to be contained withing the unit box. The lower limit is always 0,0,0.

Returns:

Fractional coordinate of the lower bound of the ASU.

19.151.3.8 bool clipper::Spacegroup::invariant_under_change_of_hand () const

test if change of hand preserves spacegroup Test if hand-change is possible.

Returns:

true if a change of hand preserves the spacegroup.

19.151.3.9 int clipper::Spacegroup::spacegroup_number () const

return the spacegroup number The spacegroup number is only available if the spacegroup exists in the internal table, see Hall & Grosse-Kunstleve.

Returns:

The spacegroup number, or 0 if unavailable.

Reimplemented from **clipper::Spgr_descr** (p. 424).

Referenced by clipper::Test_core::operator()().

19.151.3.10 String clipper::Spacegroup::symbol_hall () const

return the Hall symbol The Hall symbol is only available if the spacegroup exists in the internal table, see Hall & Grosse-Kunstleve.

Returns:

The Hall symbol, or "Unknown" if unavailable.

Reimplemented from **clipper::Spgr_descr** (p. 424).

19.151.3.11 String clipper::Spacegroup::symbol_hm () const

return the H-M symbol The H-M symbol is only available if the spacegroup exists in the internal table, see Hall & Grosse-Kunstleve.

Returns:

The H-M symbol, or "Unknown" if unavailable.

Reimplemented from **clipper::Spgr_descr** (p. 424).

19.151.3.12 String clipper::Spacegroup::symbol_laue () const

return the Laue group symbol

Returns:

The Laue group symbol. i.e. one of -1, 2/m, 2/mmm, -3, -3m, 4/m, 4/mmm, 6/m, 6/mmm, m-3, m-3m

The documentation for this class was generated from the following files:

- spacegroup.h
- spacegroup.cpp

19.152 clipper::Spgr_cacheobj Class Reference

Public Types

- typedef **Spgr_descr Key**

Public Member Functions

- **Spgr_cacheobj** (const **Key** &spgr_cachekey)

  construct entry
- bool **matches** (const **Key** &spgr_cachekey) const

  compare entry
- **String format** () const

  string description

Public Attributes

- **Key spgr_cachekey_**

  spacegroup cachekey
- int **nsym**
- int **nsymn**
- int **nsymi**
- int **nsymc**
- int **nsymp**

  number of syms: total, primitive
- int **lgrp**

  Laue group number.
- std::vector< **Symop** > **symops**

  symmetry operators
- std::vector< **Isymop** > **isymops**

  symmetry operators
- **Vec3 asu_min_**
- **Vec3 asu_max_**

  real space ASU

Static Public Attributes

- static **Mutex mutex** = **Mutex**()

  thread safety

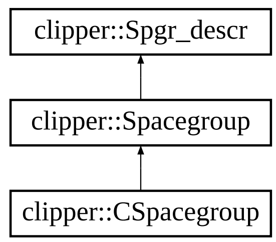
The documentation for this class was generated from the following files:

- spacegroup.h
- spacegroup.cpp

19.153 clipper::Spgr_descr Class Reference

spacegroup description

`#include <spacegroup.h>`Inheritance diagram for clipper::Spgr_descr::



Classes

- class **Symop_codes**

  Vector of symop codes and associated methods.

Public Types

- enum **TYPE** {
  Hall, **HM**, **XHM**, **Symops**,
  Number, **Unknown** }

Public Member Functions

- **Spgr_descr** ()

  null constructor

- **Spgr_descr** (const **String** &symb, TYPE type=Unknown)

  constructor: from symbol or operators.

- **Spgr_descr** (const int &num)

  constructor: from number.

- int **spacegroup_number** () const

  return the spacegroup number

- **String symbol_hall** () const

  return the Hall symbol

- **String symbol_hm** () const

  return the H-M symbol

- **String symbol_xhm** () const

  return the extended H-M symbol

- **String symbol_hm_ext** () const

  return the extension H-M symbol

- **Spgr_descr** (const **Symop_codes** &ops)

  constructor: from symop list.

- const **Symop_codes** & **generator_ops** () const

  return the generators for the spacegroup

- const unsigned int & **hash** () const

  return the hash code for the spacegroup

Static Public Member Functions

- static void **set_preferred** (const char &c)

  set preferred default spacegroup choice

Protected Attributes

- unsigned int **hash_**

  hash code of spacegroup

- **Symop_codes generators_**

  codes for symop generators

Static Protected Attributes

- static char **pref_12** = '1'
- static char **pref_hr** = 'H'

  preferred origin and hex/romb symbols

19.153.1 Detailed Description

spacegroup description The spacegroup description is a compact description of a spacegroup. It may be initialised from Hall or H-M symbols, a string of symops or a number. Internally a hash code is used to refer to the spacegroup, so this object is only 32 bits in size.

For more details of spacegroup symbols, see Sydney R. Hall & Ralf W. Grosse-Kunstleve 'Concise Space-Group Symbols', `http://www.kristall.ethz.ch/LFK/software/sginfo/hall_symbols.html`

19.153.2 Constructor & Destructor Documentation

19.153.2.1 clipper::Spgr_descr::Spgr_descr ()

null constructor Construct a null description spacegroup. The result is initialised to an invalid spacegroup code.

References hash_.

Referenced by clipper::Spacegroup::Spacegroup().

19.153.2.2 clipper::Spgr_descr::Spgr_descr (const String & *symb*, TYPE *type* = `Unknown`) `[explicit]`

constructor: from symbol or operators. Construct a spacegroup description from a text description, i.e. a symbol or operators. This may be one of the following:

- Hall symbol, e.g. " P 2ac 2ab"
- H-M symbol, e.g. "P 21 21 21"
- Number, e.g. "19"
- List of symmetry operators separated by semicolons, e.g. "x,y,z;-x+1/2,-y,z+1/2;x+1/2,-y+1/2,-z;-x,y+1/2,-z+1/2"

It is best to specify the type of symbol being used, however if this parameter is omitted a guess will be made. Unfortunately, Hall and H-M symbols may be ambiguous. Any ambiguity may be removed by appending parentheses "()" to the end of the Hall symbol, otherwise the symbol will be interpreted as an H-M symbol, and a Hall symbol if that fails.

H-M symbols and spacegroup numbers may correspond to 2 different entries in international tables. The choice between 2 origin settings or hexagonal/rhomohedral settings is made using the **set_preferred()** (p. 425) method.

Parameters:

name The spacegroup symbol or operators.

type The type of symbol: Spacegroup::Symops, Spacegroup::Hall, Spacegroup::HM, Spacegroup::XHM, Spacegroup::Number

References clipper::Spgr_descr::Symop_codes::generator_ops(), generators_-, clipper::Spgr_descr::Symop_codes::hash(), hash_, clipper::String::i(), clipper::Spgr_descr::Symop_codes::init_hall(), clipper::Spgr_descr::Symop_codes::init_symops(), clipper::Message::message(), pref_hr, and clipper::String::trim().

19.153.2.3 clipper::Spgr_descr::Spgr_descr (const int & *num*) `[explicit]`

constructor: from number. See previous constuctor.

Parameters:

num The spacegroup number.

References clipper::Spgr_descr::Symop_codes::expand(), clipper::Spgr_descr::Symop_-codes::generator_ops(), generators_, clipper::Spgr_descr::Symop_codes::hash(), hash_, clipper::Spgr_descr::Symop_codes::init_hall(), clipper::Message::message(), and pref_hr.

19.153.2.4 clipper::Spgr_descr::Spgr_descr (const Symop_codes & *ops*) [explicit]

constructor: from symop list. This is not normally used, except in conjunction with Spgr_-desc::generator_ops() to derive one group from another.

References clipper::Spgr_descr::Symop_codes::generator_ops(), generators_, clipper::Spgr_-descr::Symop_codes::hash(), and hash_.

19.153.3 Member Function Documentation

19.153.3.1 int clipper::Spgr_descr::spacegroup_number () const

return the spacegroup number The spacegroup number is only available if the spacegroup exists in the internal table, see Hall & Grosse-Kunstleve.

Returns:

The spacegroup number, or 0 if unavailable.

Reimplemented in **clipper::Spacegroup** (p. 418).

References hash_.

19.153.3.2 String clipper::Spgr_descr::symbol_hall () const

return the Hall symbol The Hall symbol is only available if the spacegroup exists in the internal table, see Hall & Grosse-Kunstleve.

Returns:

The Hall symbol, or "Unknown" if unavailable.

Reimplemented in **clipper::Spacegroup** (p. 418).

References hash_.

Referenced by clipper::Xmap_cacheobj::format(), and clipper::Spgr_cacheobj::format().

19.153.3.3 String clipper::Spgr_descr::symbol_hm () const

return the H-M symbol The H-M symbol is only available if the spacegroup exists in the internal table, see Hall & Grosse-Kunstleve.

Returns:

The H-M symbol, or "Unknown" if unavailable.

Reimplemented in **clipper::Spacegroup** (p. 418).

References hash_.

19.153.3.4 String clipper::Spgr_descr::symbol_xhm () const

return the extended H-M symbol The extended H-M symbol is only available if the spacegroup exists in the internal table, see Hall & Grosse-Kunstleve.

Returns:

The extended H-M symbol, or "Unknown" if unavailable.

References hash_.

19.153.3.5 String clipper::Spgr_descr::symbol_hm_ext () const

return the extension H-M symbol The extension H-M symbol is only available if the spacegroup exists in the internal table, see Hall & Grosse-Kunstleve.

Returns:

The extension H-M symbol, or ""

References hash_.

19.153.3.6 void clipper::Spgr_descr::set_preferred (const char & *c*) `[static]`

set preferred default spacegroup choice Sets the preferred origin or setting for initialising all **Spgr_descr** (p. 421) objects using H-M symbols or **Spacegroup** (p. 413) numbers. cctbx uses origin choice '1' by default, CCP4 uses '2'. Both packages use 'H' in preference to 'R'. Preferred values are stored for both. Defaults are '1' and 'H'.

CCP4 users may wish to add the following before using H-M codes or numbers.

```
Spgr_descr::set_preferred('2');
```

Parameters:

c Either '1' or '2', 'H' or 'R'.

References pref_hr.

The documentation for this class was generated from the following files:

- spacegroup.h
- spacegroup.cpp

19.154 clipper::String Class Reference

String (p. 426) extension with simple parsing methods.

`#include <clipper_types.h>`

Public Member Functions

- **String** ()
  null constructor
- **String** (const std::string str)
  constructor: from string
- **String** (const char *str)
  *constructor: from char**
- **String** (const char *str, const int l)
  *constructor: from char**
- **String** (const int i, const int w=4)
  constructor: from int
- **String** (const long i, const int w=4)
  constructor: from long
- **String** (const float f, const int w=6, const int p=6)
  constructor: from float
- **String** (const double f, const int w=6, const int p=6)
  constructor: from double
- std::vector< **String** > **split** (const **String** sep) const
  ***String** (p. 426) splitter - a very simple parser component.*
- **String trim** () const
  Return copy of string without leading and trailing blanks.
- **String tail** () const
  get trailing path element
- **String head** () const
  remove trailing path element
- **String nohead** () const
  get leading path element
- **String notail** () const
  remove leading path element

- int **i** () const
  convert to int
- long **l** () const
  convert to long
- ftype32 **f32** () const
  convert to float
- ftype64 **f64** () const
  convert to double
- **ftype f** () const
  convert to ftype
- **ftype rational** () const
  convert from rational to ftype

Static Public Member Functions

- static **String rational** (const double f, const int b, const bool sign=false)
  construct string from rational f using base b

19.154.1 Detailed Description

String (p. 426) extension with simple parsing methods. **String** (p. 426) extension with primitive 'split' operation for parsing and pathname processing operations.

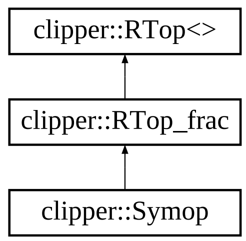
The documentation for this class was generated from the following files:

- clipper_types.h
- clipper_types.cpp

19.155 clipper::Symop Class Reference

Crystallographic symmetry operator.

`#include` <`symop.h`>Inheritance diagram for clipper::Symop::



Public Member Functions

- **Symop** ()

  null constructor

- **Symop** (const **RTop**<> &rt)

  constructor: ***RTop*** *(*p. *402)*

- **Symop** (const **ftype** mat[4][4])

  constructor: from 4x4 matrix

- **String format** () const

  return formatted ***String*** *(*p. *426) representation*

19.155.1 Detailed Description

Crystallographic symmetry operator. This is identical to a fractional **RTop** (p. 402), but has its own class since not all fractional RTops are symops. For a full list of methods, see **clipper::RTop** (p. 402) and **clipper::RTop_frac** (p. 404)

19.155.2 Constructor & Destructor Documentation

19.155.2.1 clipper::Symop::Symop (const RTop<> & *rt*) `[explicit]`

constructor: **RTop** (p. 402) Construct a symmetry operator and initialise it to the supplied **RTop** (p. 402). Translations are rounded to a basis of 48, and put on the range 0..1

Parameters:

mat The **RTop** (p. 402) to use.

References clipper::Util::intr(), clipper::Util::mod(), clipper::RTop< T >::rot(), clipper::RTop<>::rot(), clipper::RTop< T >::trn(), and clipper::RTop<>::trn().

19.155.2.2 clipper::Symop::Symop (const ftype *mat*[4][4]) `[explicit]`

constructor: from 4x4 matrix Construct a symmetry operator and initialise it to the supplied matrix. Translations are rounded to a basis of 48, and put on the range 0..1

Parameters:

mat The 4x4 matrix to use. The [i][3] elements contain the translation.

References clipper::Util::intr(), clipper::Util::mod(), clipper::RTop<>::rot(), and clipper::RTop<>::trn().

19.155.3 Member Function Documentation

19.155.3.1 String clipper::Symop::format () const

return formatted **String** (p. 426) representation Return formatted representation of the symmetry operator.

Returns:

The formatted text string, e.g. -x, -y+1/2, z.

Reimplemented from **clipper::RTop<>** (p. 402).

References clipper::Util::intr(), clipper::String::rational(), clipper::RTop<>::rot(), and clipper::RTop<>::trn().

The documentation for this class was generated from the following files:

- symop.h
- symop.cpp

19.156 clipper::Symop_code Class Reference

Compressed encoded symmetry operator.

`#include <symop.h>`

Public Member Functions

- **Symop_code** ()
  null constructor
- **Symop_code** (const int &code)
  constructor: from int
- **Symop_code** (const **Symop** &op)
  constructor: from ***Symop*** *(p. 428)*
- **Symop_code** (const **Isymop** &op)
  constructor: from ***Isymop*** *(p. 311)*
- void **init** (const **Isymop** &op)
  initialiser: from ***Isymop*** *(p. 311)*
- **Symop_code code_rot** () const
  return code for rotation part
- **Symop_code code_trn** () const
  return code for translation part
- **Symop symop** () const
  convert to symop
- **Isymop isymop** () const
  convert to integerised symop
- **operator int** () const
  convert to integer

Static Public Member Functions

- static **Symop_code identity** ()
  identity code

19.156.1 Detailed Description

Compressed encoded symmetry operator. This is a compresses representation of a crystallographic symmetry operator, stored as a single 32-bit integer. It may be converted to or from a symop or an int and compared, sorted, etc. The following guarantees are made concerning the code:

- The identity operator has a code of zero.
- Operators with non-identity rotations will have higher codes than operators with identity rotations, for the same translation.
- Operators with non-zero translations will have higher codes than operators with zero translations, for the same rotation.

19.156.2 Member Function Documentation

19.156.2.1 Symop clipper::Symop_code::symop () const

convert to symop Construct a symmetry operator and initialise it to the matrix encoded in the given int.

Parameters:

code The integer code.

References isymop(), clipper::RTop< T >::rot(), and clipper::RTop< T >::trn().

19.156.2.2 Isymop clipper::Symop_code::isymop () const

convert to integerised symop Construct an integerised symmetry operator and initialise it to the matrix encoded in the given int, with a grid (base) of (24,24,24).

Parameters:

code The integer code.

References code_rot(), code_trn(), clipper::Util::mod(), clipper::RTop< T >::rot(), and clipper::RTop< T >::trn().

Referenced by symop().

The documentation for this class was generated from the following files:

- symop.h
- symop.cpp

19.157 clipper::Spgr_descr::Symop_codes Class Reference

Vector of symop codes and associated methods.

`#include <spacegroup.h>`

Public Member Functions

- void **init_hall** (const **String** &symb)

    initialise from Hall symbol

- void **init_symops** (const **String** &symb)

    initialise from symops

- **Symop_codes expand** () const

    expand (incomplete) list of symops

- **Symop_codes primitive_noninversion_ops** () const

    return primitive non-inversion ops (by computation)

- **Symop_codes inversion_ops** () const

    return inversion ops (by computation)

- **Symop_codes primitive_ops** () const

    return primitive incl inversion ops (by computation)

- **Symop_codes centering_ops** () const

    return lattice centering ops (by computation)

- **Symop_codes laue_ops** () const

    return Laue ops

- **Symop_codes pgrp_ops** () const

    return point group ops

- **Symop_codes patterson_ops** () const

    return Patterson ops

- **Symop_codes generator_ops** () const

    return minimal list of generator ops

- **Symop_codes product** (const **Symop_codes** &ops2) const

    return product of this (expanded) list by another (expanded) list

- unsigned int **hash** () const

    return hash code of symop list

19.157.1 Detailed Description

Vector of symop codes and associated methods.

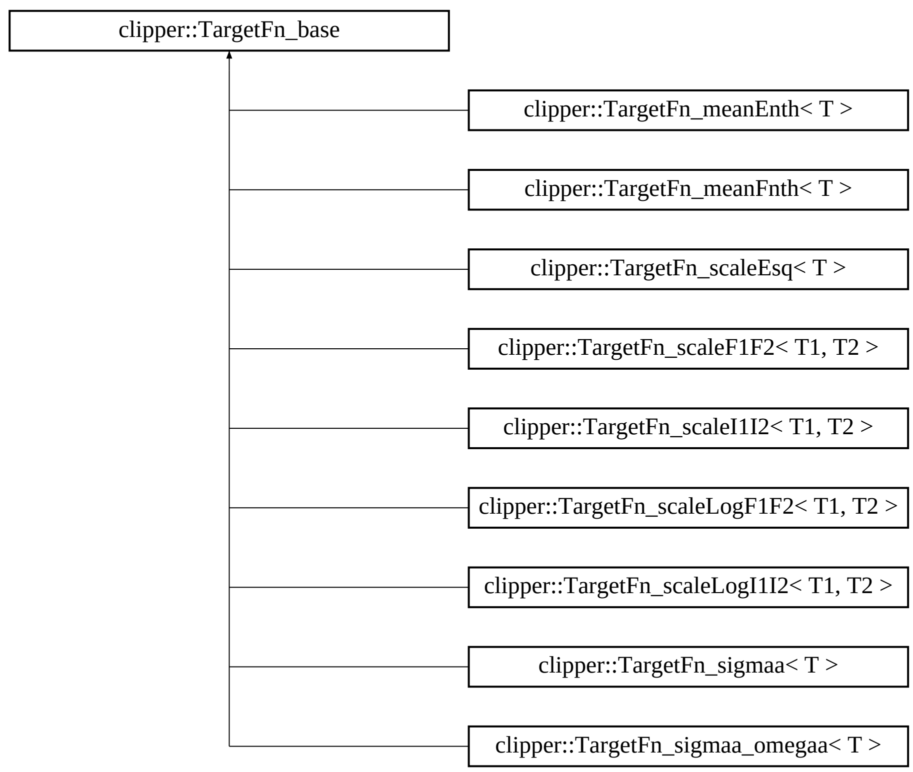
The documentation for this class was generated from the following files:

- spacegroup.h
- spacegroup.cpp

19.158 clipper::TargetFn_base Class Reference

abstract base class for least-squares resolution function target functions

#include <resol_fn.h>Inheritance diagram for clipper::TargetFn_base::



Classes

- class **Rderiv**

  object holding the residual function and first two derivatives

Public Types

- enum **FNtype** { **GENERAL**, **QUADRATIC** }

  enumeration of function types: optionally used to improve convergence

Public Member Functions

- virtual **Rderiv rderiv** (const **HKL_info::HKL_reference_index** &ih, const **ftype** &fh) const =0

  return the value and derivatives of the target function

- virtual **FNtype type** () const

the type of the function: optionally used to improve convergence

- virtual ~**TargetFn_base** ()
  destructor
- void **debug** (const **HKL_info** &hkl_info) const
  test that the residuals, gradients, and curvatures are consistent

19.158.1 Detailed Description

abstract base class for least-squares resolution function target functions A target function must be able to return its value given the value of the basis function for all **HKL** (p. 261), and its derivative with respect the values of the basis function for all **HKL** (p. 261).

Optionally, performance can be improved by returning a flag to indicate if the target function is quadratic.

19.158.2 Member Function Documentation

19.158.2.1 virtual Rderiv clipper::TargetFn_base::rderiv (const HKL_info::HKL_reference_index & *ih*, const ftype & *fh*) const `[pure virtual]`

return the value and derivatives of the target function If the value of f(h) is invalid, rderiv.r should be set to NaN

Implemented in **clipper::TargetFn_meanFnth< T >** (p. 437), **clipper::TargetFn_-meanEnth< T >** (p. 436), **clipper::TargetFn_scaleF1F2< T1, T2 >** (p. 439), **clipper::TargetFn_scaleLogF1F2< T1, T2 >** (p. 441), **clipper::TargetFn_scaleI1I2< T1, T2 >** (p. 440), **clipper::TargetFn_scaleLogI1I2< T1, T2 >** (p. 442), **clipper::TargetFn_scaleEsq< T >** (p. 438), **clipper::TargetFn_sigmaa_omegaa< T >** (p. 444), and **clipper::TargetFn_sigmaa< T >** (p. 443).

Referenced by clipper::ResolutionFn::calc_derivs(), debug(), and clipper::ResolutionFn_-nonlinear::ResolutionFn_nonlinear().

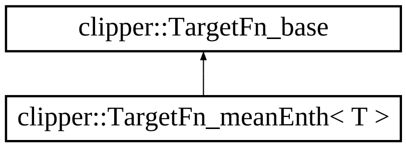
The documentation for this class was generated from the following files:

- resol_fn.h
- resol_fn.cpp

19.159 clipper::TargetFn_meanEnth< T > Class Template Reference

simple mean $|E|^n$ target

#include <resol_targetfn.h>Inheritance diagram for clipper::TargetFn_meanEnth< T >::



Public Member Functions

- **TargetFn_meanEnth** (const **HKL_data**< T > &hkl_data_, const **ftype** &n)

  constructor: takes the datalist against which to calc target, and power

- **Rderiv rderiv** (const **HKL_info::HKL_reference_index** &ih, const **ftype** &fh) const

  return the value and derivatives of the target function

- **FNtype type** () const

  the type of the function: optionally used to improve convergence

19.159.1 Detailed Description

template<class T> class clipper::TargetFn_meanEnth< T >

simple mean $|E|^n$ target This class implements the target function for calculating mean $|E|^n$ as a function of position in reciprocal space. It includes the appropriate multiplicity correction, and so can be applied to any type with an 'E' member with the same dimensions as an |E| (or corrected |F| or |U|).

This function should not be used to scale F's to E's. See **TargetFn_scaleEsq** (p. 438).

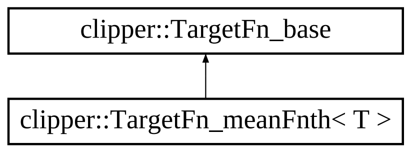
The documentation for this class was generated from the following file:

- resol_targetfn.h

19.160 clipper::TargetFn_meanFnth< T > Class Template Reference

simple mean $|F|^n$ target

`#include <resol_targetfn.h>`Inheritance diagram for clipper::TargetFn_meanFnth< T >::



Public Member Functions

- **TargetFn_meanFnth** (const **HKL_data**< T > &hkl_data_, const **ftype** &n)

  constructor: takes the datalist against which to calc target, and power

- **Rderiv rderiv** (const **HKL_info::HKL_reference_index** &ih, const **ftype** &fh) const

  return the value and derivatives of the target function

- **FNtype type** () const

  the type of the function: optionally used to improve convergence

19.160.1 Detailed Description

template<class T> class clipper::TargetFn_meanFnth< T >

simple mean $|F|^n$ target This class implements the target function for calculating mean $|F|^n$ as a function of position in reciprocal space. It includes the appropriate multiplicity correction, and so can be applied to any type with an 'f' member with the same dimensions as an |F| or |U| (or an uncorrected |E|).

This function should not be used to scale F's to E's. See **TargetFn_scaleEsq** (p. 438).

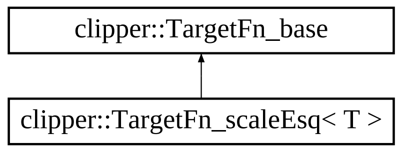
The documentation for this class was generated from the following file:

- resol_targetfn.h

19.161 clipper::TargetFn_scaleEsq< T > Class Template Reference

$|E|^2$ scaling target

`#include <resol_targetfn.h>`Inheritance diagram for clipper::TargetFn_scaleEsq< T >::



Public Member Functions

- **TargetFn_scaleEsq** (const **HKL_data**< T > &hkl_data_)

  constructor: takes the datalist against which to calc target

- **Rderiv rderiv** (const **HKL_info::HKL_reference_index** &ih, const **ftype** &fh) const

  return the value and derivatives of the target function

- **FNtype type** () const

  the type of the function: optionally used to improve convergence

19.161.1 Detailed Description

template<class T> class clipper::TargetFn_scaleEsq< T >

$|E|^2$ scaling target This class implements the target function for calculating the scale factor to normalise to $<|E|^2> = 1$. Note that this is not the same as dividing by $<|E|^2>$, except in a few special cases, e.g. a simple resolution bins calculation. The resulting targen function is the square of the value by which $|E|$ should be multiplied to acheive the correct normalisation.

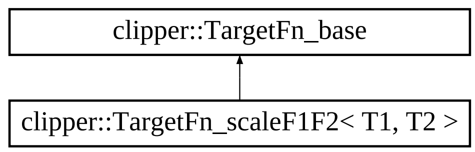
The documentation for this class was generated from the following file:

- resol_targetfn.h

19.162 clipper::TargetFn_scaleF1F2< T1, T2 > Class Template Reference

$|F|^2$ scaling target

`#include <resol_targetfn.h>`Inheritance diagram for clipper::TargetFn_scaleF1F2< T1, T2 >::



Public Member Functions

- **TargetFn_scaleF1F2** (const **HKL_data**< T1 > &hkl_data1_, const **HKL_data**< T2 > &hkl_data2_)

    constructor: takes the datalist against which to calc target

- **Rderiv rderiv** (const **HKL_info::HKL_reference_index** &ih, const **ftype** &fh) const

    return the value and derivatives of the target function

- **FNtype type** () const

    the type of the function: optionally used to improve convergence

19.162.1 Detailed Description

template<class T1, class T2> class clipper::TargetFn_scaleF1F2< T1, T2 >

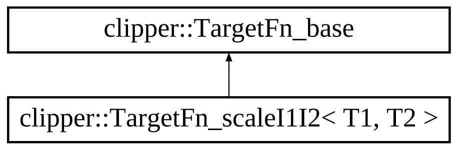
$|F|^2$ scaling target This class implements the target function for calculating the scale factor to scale one set of F's to another. The resulting scale is the square of the factor that scales the first set of data to match the second.

The documentation for this class was generated from the following file:

- resol_targetfn.h

19.163 clipper::TargetFn_scaleI1I2< T1, T2 > Class Template Reference

#include <resol_targetfn.h>Inheritance diagram for clipper::TargetFn_scaleI1I2< T1, T2 >::



Public Member Functions

- **TargetFn_scaleI1I2** (const **HKL_data**< T1 > &hkl_data1_, const **HKL_data**< T2 > &hkl_data2_)

  constructor: takes the datalist against which to calc target
- **Rderiv rderiv** (const **HKL_info::HKL_reference_index** &ih, const **ftype** &fh) const

  return the value and derivatives of the target function
- **FNtype type** () const

  the type of the function: optionally used to improve convergence

19.163.1 Detailed Description

template<class T1, class T2> class clipper::TargetFn_scaleI1I2< T1, T2 >

This class implements the target function for calculating the scale factor to scale one set of I's to another. The resulting scale is the square of the factor that scales the first set of data to match the second.

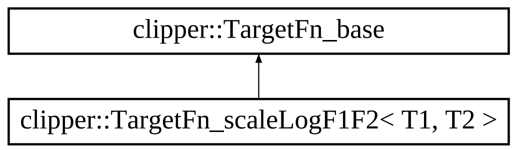
The documentation for this class was generated from the following file:

- resol_targetfn.h

19.164 clipper::TargetFn_scaleLogF1F2< T1, T2 > Class Template Reference

log $|F|^2$ scaling target

#include <resol_targetfn.h>Inheritance diagram for clipper::TargetFn_scaleLogF1F2< T1, T2 >::



Public Member Functions

- **TargetFn_scaleLogF1F2** (const **HKL_data**< T1 > &hkl_data1_, const **HKL_data**< T2 > &hkl_data2_)

  constructor: takes the datalist against which to calc target

- **Rderiv rderiv** (const **HKL_info::HKL_reference_index** &ih, const **ftype** &fh) const

  return the value and derivatives of the target function

- **FNtype type** () const

  the type of the function: optionally used to improve convergence

19.164.1 Detailed Description

template<class T1, class T2> class clipper::TargetFn_scaleLogF1F2< T1, T2 >

log $|F|^2$ scaling target This class implements the target function for calculating the scale factor to scale the weighted log of one set of F's to another. The resulting scale is the square of the factor that scales the first set of data to match the second. The log scaling target is used in conjunction with the log-Gaussian basis functions for a fast and robust approximation to iso/aniso Gaussian scaling.

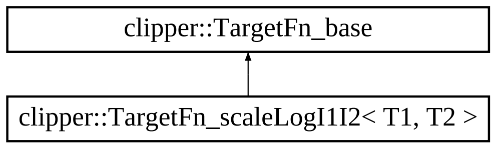
The documentation for this class was generated from the following file:

- resol_targetfn.h

19.165 clipper::TargetFn_scaleLogI1I2< T1, T2 > Class Template Reference

log |I| scaling target

#include <resol_targetfn.h>Inheritance diagram for clipper::TargetFn_scaleLogI1I2< T1, T2 >::



Public Member Functions

- **TargetFn_scaleLogI1I2** (const **HKL_data**< T1 > &hkl_data1_, const **HKL_data**< T2 > &hkl_data2_)

  constructor: takes the datalist against which to calc target

- **Rderiv rderiv** (const **HKL_info::HKL_reference_index** &ih, const **ftype** &fh) const

  return the value and derivatives of the target function

- **FNtype type** () const

  the type of the function: optionally used to improve convergence

19.165.1 Detailed Description

template<class T1, class T2> class clipper::TargetFn_scaleLogI1I2< T1, T2 >

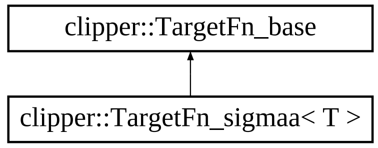
log |I| scaling target This class implements the target function for calculating the scale factor to scale the weighted log of one set of I's to another. The resulting scale is the square of the factor that scales the first set of data to match the second. The log scaling target is used in conjunction with the log-Gaussian basis functions for a fast and robust approximation to iso/aniso Gaussian scaling.

The documentation for this class was generated from the following file:

- resol_targetfn.h

19.166 clipper::TargetFn_sigmaa< T > Class Template Reference

#include <resol_targetfn.h>Inheritance diagram for clipper::TargetFn_sigmaa< T >::



Public Member Functions

- **TargetFn_sigmaa** (const **HKL_data**< T > &eo, const **HKL_data**< T > &ec)
  constructor: takes the datalist against which to calc target
- **Rderiv rderiv** (const **HKL_info::HKL_reference_index** &ih, const **ftype** &sigmaa0) const
  return the value and derivatives of the target function

Static Public Member Functions

- static **ftype sigmaa** (const **ftype** &sigm)
  convert function to sigmaa

19.166.1 Detailed Description

template<class T> class clipper::TargetFn_sigmaa< T >

Deprecated

simple sigma_a target function

Warning: Convergence of this basis-function can be

unreliable under some circumstances. Use **clipper::TargetFn_sigmaa_omegaa** (p. 444) instead, except for development purposes.

This class implements the target function for calculating sigma_a. Required is a datalist containing Eo, Ec.

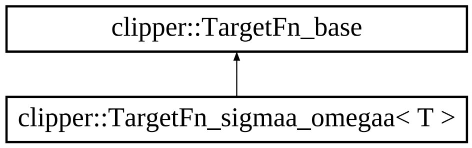
This version simplifies terms in |Eo|^2 and |Ec|^2 which should average out to 1 if the normalisation scheme is consistent with the sigmaa calc.

The documentation for this class was generated from the following file:

- resol_targetfn.h

19.167 clipper::TargetFn_sigmaa_omegaa< T > Class Template Reference

`#include <resol_targetfn.h>`Inheritance diagram for clipper::TargetFn_sigmaa_omegaa< T >::



Public Member Functions

- **TargetFn_sigmaa_omegaa** (const **HKL_data**< T > &eo, const **HKL_data**< T > &ec)

  constructor: takes the datalist against which to calc target

- **Rderiv rderiv** (const **HKL_info::HKL_reference_index** &ih, const **ftype** &omegaa) const

  return the value and derivatives of the target function

Static Public Member Functions

- static **ftype sigmaa** (const **ftype** &omegaa)

  convert omegaa to sigmaa

19.167.1 Detailed Description

template<class T> class clipper::TargetFn_sigmaa_omegaa< T >

Deprecated

simple sigma_a target function

This class implements the target function for calculating sigma_a. Required is a datalist containing Eo, Ec.

It actually refines omegaa = sigmaa/(1-sigmaa^2). This has better proerties for refinement. To get sigmaa use

```
sigmaa = ( sqrt( 4*omegaa^2 + 1 ) - 1 ) / ( 2*omegaa )
```

This is available as a static function:

```
sigmaa = targetfn.sigmaa( omegaa )
```

This version simplifies terms in |Eo|^2 and |Ec|^2 which should average out to 1 if the normalisation scheme is consistent with the sigmaa calc.

Convergence is good for calculations using the 'binner' basis function, however the smooth basis function have convergence problems. This is still under investigation.

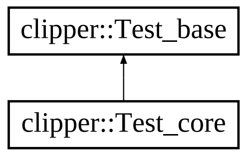
The documentation for this class was generated from the following file:

- resol_targetfn.h

19.168 clipper::Test_base Class Reference

Base class for **clipper** (p. 81) self-test classes.

`#include <clipper_test.h>`Inheritance diagram for clipper::Test_base::



Public Member Functions

- void **set_stream** (std::ostream &stream)

Protected Member Functions

- bool **test** (const **String** &id, const double &value)

  test that the next value matches the stored data

- bool **test** (const **String** &id, const double &value, const double &val, const double &tol)
- bool **test** (const **String** &id, const int &value, const int &val)

Protected Attributes

- std::ostream * **stream_**
- std::vector< float > **data_val**
- std::vector< float > **data_tol**
- int **count**
- int **error_count**

19.168.1 Detailed Description

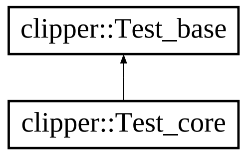
Base class for **clipper** (p. 81) self-test classes. Override this class with test classes for each **clipper** (p. 81) package.

The documentation for this class was generated from the following files:

- clipper_test.h
- clipper_test.cpp

19.169 clipper::Test_core Class Reference

Inheritance diagram for clipper::Test_core::



Public Member Functions

- bool **operator()** ()
  do the self-tests

The documentation for this class was generated from the following files:

- test_core.h
- test_core.cpp

19.170 clipper::data::Test_data Class Reference

Class to return test data.

`#include <test_data.h>`

Public Member Functions

- **Test_data** ()
  Null constructor: fills the arrays.
- const **HKL_data**< **datatypes::F_sigF**< float > > & **hkl_data_f_sigf** () const
  Return ***HKL_data*** *(p. 267) class.*
- const **HKL_data**< **datatypes::ABCD**< float > > & **hkl_data_abcd** () const
  Return ***HKL_data*** *(p. 267) class.*
- const **Atom_list** & **atom_list** () const
  Return atom list.

19.170.1 Detailed Description

Class to return test data.

The documentation for this class was generated from the following files:

- test_data.h
- test_data.cpp

19.171 clipper::data::TESThkldata Struct Reference

Public Attributes

- int **h**
- int **k**
- int **l**
- float **f**
- float **s**
- float **a**
- float **b**
- float **c**
- float **d**

The documentation for this struct was generated from the following file:

- test_data.cpp

19.172 clipper::data::TESTxyzdata Struct Reference

Public Attributes

- const char * **element**
- float **x**
- float **y**
- float **z**
- float **u_iso**
- float **occ**

The documentation for this struct was generated from the following file:

- test_data.cpp

19.173 clipper::Thread_base Class Reference

Thread base class: Override this to create new threads.

`#include <clipper_thread.h>`

Public Member Functions

- bool **run** ()
- bool **join** ()
- int **id** () const

Static Public Member Functions

- static void **lock** ()
- static void **unlock** ()

Protected Member Functions

- virtual void **Run** ()=0

19.173.1 Detailed Description

Thread base class: Override this to create new threads. To create a thread, override this class. Store data as members with accessors to set input and read output. Override the Run() method to do the actual work. e.g. the following class implements a thread which can sum a list of numbers.

```
class Thread_test : public Thread_base {
public:
  class Data : public std::vector<int> {
  public:
    Data() {}
    Data( std::vector<int> v ) : std::vector<int>(v) {}
  };

  void Run() {
    sum = 0;
    while ( 1 ) {
        lock();
        int c = current++;
        unlock();
        if ( c >= data_.size() ) break;
        sum += data_[c];
    }
  }
  static void set_data( Data d ) { data_ = d; }
  static Data data_;
  static int current;
  int sum;
};
```

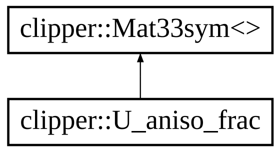
The documentation for this class was generated from the following files:

- clipper_thread.h
- clipper_thread.cpp

19.174 clipper::U_aniso_frac Class Reference

Anisotropic fractional atomic displacement parameters.

`#include <coords.h>`Inheritance diagram for clipper::U_aniso_frac::



Public Member Functions

- **U_aniso_frac** ()

  null constructor

- **U_aniso_frac** (const **Mat33sym**<> &m)

  constructor: from ***Mat33sym*** *(*p. *336)*

- **U_aniso_frac** (const **ftype** &u11, const **ftype** &u22, const **ftype** &u33, const **ftype** &u12, const **ftype** &u13, const **ftype** &u23)

  constructor: from Uij

- **U_aniso_orth u_aniso_orth** (const **Cell** &cell) const

  fractional-orthogonal conversion

- **U_aniso_frac transform** (const **RTop_frac** &op) const

  return transformed U_aniso

Friends

- **U_aniso_frac operator+** (const **U_aniso_frac** &u1, const **U_aniso_frac** &u2)
- **U_aniso_frac operator-** (const **U_aniso_frac** &u)
- **U_aniso_frac operator*** (const **ftype** &s, const **U_aniso_frac** &u)

19.174.1 Detailed Description

Anisotropic fractional atomic displacement parameters. These are defined on fractional atomic coordinates in A^{-2} , i.e. they are anisotropic U values.

19.174.2 Member Function Documentation

19.174.2.1 U_aniso_orth clipper::U_aniso_frac::u_aniso_orth (const Cell & *cell*) const

fractional-orthogonal conversion

Parameters:

cell The cell concerned

Returns:

The transformed coordinate.

References clipper::Cell::matrix_orth(), and clipper::Mat33< T >::transpose().

19.174.2.2 U_aniso_frac clipper::U_aniso_frac::transform (const RTop_frac & *op*) const

return transformed U_aniso The aniso U is transformed by the given RT op.

Parameters:

u The aniso U.

References clipper::Mat33< T >::inverse(), clipper::RTop< T >::rot(), clipper::Mat33< T >::transpose(), and U_aniso_frac().

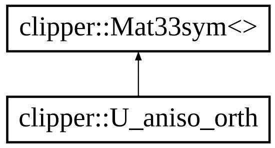
The documentation for this class was generated from the following files:

- coords.h
- coords.cpp

19.175 clipper::U_aniso_orth Class Reference

Anisotropic orthogonal atomic displacement parameters.

`#include` <`coords.h`>Inheritance diagram for clipper::U_aniso_orth::



Public Member Functions

- **U_aniso_orth** ()
  null constructor
- **U_aniso_orth** (const **Mat33sym**<> &m)
  constructor: from ***Mat33sym*** *(*p. *336)*
- **U_aniso_orth** (const **ftype** &u)
  constructor: from isotropic U
- **U_aniso_orth** (const **ftype** &u11, const **ftype** &u22, const **ftype** &u33, const **ftype** &u12, const **ftype** &u13, const **ftype** &u23)
  constructor: from Uij
- **ftype u_iso** () const
  return nearest isotropic U
- **U_aniso_frac u_aniso_frac** (const **Cell** &cell) const
  orthogonal-fractional conversion
- **U_aniso_orth transform** (const **RTop_orth** &op) const
  return transformed U_aniso

Friends

- **U_aniso_orth operator+** (const **U_aniso_orth** &u1, const **U_aniso_orth** &u2)
- **U_aniso_orth operator-** (const **U_aniso_orth** &u)
- **U_aniso_orth operator*** (const **ftype** &s, const **U_aniso_orth** &u)

19.175.1 Detailed Description

Anisotropic orthogonal atomic displacement parameters. These are defined on orthogonal atomic coordinates in A^{-2} , i.e. they are anisotropic U values.

19.175.2 Member Function Documentation

19.175.2.1 ftype clipper::U_aniso_orth::u_iso () const

return nearest isotropic U The best isotropic U is the cube root of the determinant of the matrix of anisotropic coefficients. NOTE: This is not the conventional definition, but the mathematically correct one, and gives a better approximation to the anisotropic U (i.e. lower R-factors).

Returns:

The nearest isotropic U.

References clipper::Mat33sym<>::det().

Referenced by clipper::AtomShapeFn::init().

19.175.2.2 U_aniso_frac clipper::U_aniso_orth::u_aniso_frac (const Cell & *cell*) const

orthogonal-fractional conversion

Parameters:

cell The cell concerned

Returns:

The transformed coordinate.

References clipper::Cell::matrix_frac(), and clipper::Mat33< T >::transpose().

19.175.2.3 U_aniso_orth clipper::U_aniso_orth::transform (const RTop_orth & *op*) const

return transformed U_aniso The aniso U is transformed by the given RT op.

Parameters:

u The aniso U.

References clipper::Mat33< T >::inverse(), clipper::RTop< T >::rot(), clipper::Mat33< T >::transpose(), and U_aniso_orth().

The documentation for this class was generated from the following files:

- coords.h
- coords.cpp

19.176 clipper::Util Class Reference

Utility class.

#include <clipper_util.h>

Public Member Functions

- **Util** ()
  null constructor

Static Public Member Functions

- static const **ftype** & **nan** ()
  *fast **Util::nan()** (p. 456) value*
- static const float & **nanf** ()
  *fast **Util::nan()** (p. 456) value*
- static const double & **nand** ()
  *fast **Util::nan()** (p. 456) value*
- static void **set_null** (ftype32 &f)
  set null floating value - a specific value of NaN used for missings
- static void **set_null** (ftype64 &f)
  set null floating value - a specific value of NaN used for missings
- static bool **is_null** (const ftype32 &f)
  fast test for null floating value - only works if set from Util::null()
- static bool **is_null** (const ftype64 &f)
  fast test for null floating value - only works if set from Util::null()
- static bool **is_nan** (const ftype32 f)
  *fast **Util::nan()** (p. 456) test*
- static bool **is_nan** (const ftype64 f)
  *fast **Util::nan()** (p. 456) test*
- static bool **isnan** (const ftype32 f)
  slow general NaN test for compatibility
- static bool **isnan** (const ftype64 f)
  slow general NaN test for compatibility
- static **ftype sim** (const **ftype** &x)
  Sim function: I1(X)/I0(X).

- static **ftype invsim** (const **ftype** &x)

  Inverse Sim function: I1(X)/I0(X).

- static **ftype sim_integ** (const **ftype** &x)

  Integral of Sim function: log(I0(X)).

- static **ftype sim_deriv** (const **ftype** &x)

  Derivative of Sim function: d/dx(I1(X)/I0(x)).

- static **ftype sim_deriv_recur** (const **ftype** &x)

  Derivative of Sim function using recurrance: -sim(x)/x + (1 - sim(x)^2).

- static **ftype atanh** (const **ftype** &x)

  Arc hyperbolic tangent.

- static **ftype bessel_i0** (const **ftype** &x)

  Modified Bessel function of the first kind.

- static **ftype u2b** (const **ftype** &x)

  Convert isotropic U-value to B-factor.

- static **ftype b2u** (const **ftype** &x)

  Convert isotropic B-factor to U-value.

- template<class T >
  static T **mean** (const T &pl, const T &mi)

  Convert F+/F- to mean F, with NaN checks.

- template<class T >
  static T **sig_mean** (const T &pl, const T &mi, const T &cov)

  Convert sigF+/sigF-/cov to sig F, with NaN checks.

- static int **intf** (const **ftype** &a)

  Truncate-to-integer: int(floor(a)).

- static int **intc** (const **ftype** &a)

  Truncate-to-integer above: int(ceil(a)).

- static int **intr** (const **ftype** &a)

  Round-to-integer: int(round(a)).

- static **ftype mod** (const **ftype** &a, const **ftype** &b)

  Corrected mod.

- static int **mod** (const int &a, const int &b)

  Corrected mod.

- template<class T >
  static T **max** (const T &a, const T &b)

  max

- template<class T >
  static T **min** (const T &a, const T &b)

  min

- template<class T >
  static T **bound** (const T &min, const T &val, const T &max)

  bound a value by limits

- template<class T >
  static void **swap** (T &a, T &b)

  swap the contents of two objects

- template<class T >
  static void **swap** (T &a, T &b, T &c)

  swap the contents of two objects, using third as store (for speed)

- template<class T >
  static T **sqr** (const T &a)

  square

- template<class T >
  static T **isqrt** (const T &n)

  Integer square root (returns floor of sqrt).

- static const **ftype** & **pi** ()

  pi

- static const **ftype** & **twopi** ()

  2 pi

- static const **ftype** & **twopi2** ()

  2 pi squared

- static const **ftype** & **eightpi2** ()

  8 pi squared

- static **ftype d2rad** (const **ftype** &x)

  degree-to-radian conversion

- static **ftype rad2d** (const **ftype** &x)

  degree-to-radian conversion

19.176.1 Detailed Description

Utility class. This class holds a set of useful static functions and values. You should never need to instantiate this class: Refer to members using **Util** (p. 456)::

19.176.2 Member Function Documentation

19.176.2.1 static bool clipper::Util::is_nan (const ftype32 *f*) [inline, static]

fast **Util::nan()** (p. 456) test Used for missing entries: THIS DOES NOT DISTINGUISH BETWEEN NAN & INF

Referenced by clipper::datatypes::Phi_fom< dtype >::friedel(), clipper::datatypes::F_phi< dtype >::friedel(), clipper::AtomShapeFn::init(), clipper::Rotation::is_null(), clipper::Mat33sym<>::is_null(), clipper::Mat33< ftype >::is_null(), clipper::Vec3< ftype >::is_null(), clipper::Map_stats::Map_stats(), mean(), clipper::datatypes::D_sigD< dtype >::missing(), clipper::datatypes::ABCD< dtype >::missing(), clipper::datatypes::Phi_fom< dtype >::missing(), clipper::datatypes::F_phi< dtype >::missing(), clipper::datatypes::E_sigE_ano< dtype >::missing(), clipper::datatypes::E_sigE< dtype >::missing(), clipper::datatypes::F_sigF_ano< dtype >::missing(), clipper::datatypes::F_sigF< dtype >::missing(), clipper::datatypes::I_sigI_ano< dtype >::missing(), clipper::datatypes::I_sigI< dtype >::missing(), clipper::Test_core::operator()(), clipper::datatypes::Compute_diff_fsigf_from_fsigfano< dtype >::operator()(), clipper::datatypes::Compute_mean_fsigf_from_fsigfano< dtype >::operator()(), clipper::ResolutionFn_nonlinear::ResolutionFn_nonlinear(), clipper::datatypes::Phi_fom< dtype >::shift_phase(), clipper::datatypes::F_phi< dtype >::shift_phase(), and sig_mean().

19.176.2.2 static bool clipper::Util::is_nan (const ftype64 *f*) [inline, static]

fast **Util::nan()** (p. 456) test Used for missing entries: THIS DOES NOT DISTINGUISH BETWEEN NAN & INF

19.176.2.3 static bool clipper::Util::isnan (const ftype32 *f*) [inline, static]

slow general NaN test for compatibility Works for all architectures with IEEE arithmetic only

Referenced by clipper::Test_core::operator()().

19.176.2.4 static bool clipper::Util::isnan (const ftype64 *f*) [inline, static]

slow general NaN test for compatibility Works for all architectures with IEEE arithmetic only

19.176.2.5 ftype clipper::Util::sim (const ftype & *x*) [static]

Sim function: I1(X)/I0(X).

Parameters:

x The argument

Returns:

I1(x)/I0(x)

Referenced by clipper::Test_core::operator()(), clipper::TargetFn_sigmaa< T >::rderiv(), clipper::TargetFn_sigmaa_omegaa< T >::rderiv(), and sim_deriv_recur().

19.176.2.6 ftype clipper::Util::invsim (const ftype & *x*) [static]

Inverse Sim function: I1(X)/I0(X).

Parameters:

x I1(y)/I0(y)

Returns:

y

Referenced by clipper::Test_core::operator()(), clipper::datatypes::Compute_abcd_from_phifom< dtype >::operator()(), and clipper::LogPhaseProb< N >::set_phi_fom().

19.176.2.7 ftype clipper::Util::d2rad (const ftype & *x*) [static]

degree-to-radian conversion

Parameters:

x Angle in degrees

Returns:

Angle in radians

Referenced by clipper::Cell_descr::Cell_descr().

19.176.2.8 ftype clipper::Util::rad2d (const ftype & *x*) [static]

degree-to-radian conversion

Parameters:

x Angle in radians

Returns:

Angle in degrees

Referenced by clipper::Cell_descr::alpha_deg(), clipper::Cell_descr::beta_deg(), clipper::Polar_ccp4::format(), clipper::Euler_ccp4::format(), clipper::Euler< T >::format(), clipper::Cell_descr::gamma_deg(), and clipper::datatypes::Compute_phifom_from_abcd< dtype >::operator()().

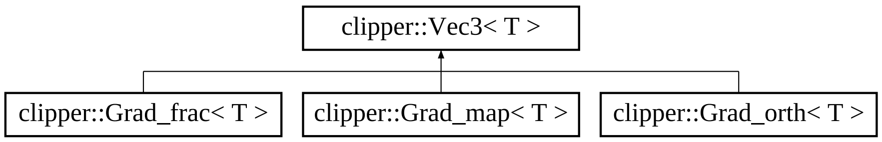
The documentation for this class was generated from the following files:

- clipper_util.h
- clipper_util.cpp

19.177 clipper::Vec3< T > Class Template Reference

3-vector class

`#include <clipper_types.h>`Inheritance diagram for clipper::Vec3< T >::



Public Member Functions

- **Vec3** ()

  null constructor

- **Vec3** (const T &v0, const T &v1, const T &v2)

  constructor: from individual values

- template<class TT >
  Vec3 (const **Vec3**< TT > &v)

  constructor: copy/convert

- bool **equals** (const **Vec3**< T > &v, const T &tol) const

  test equality

- const T & **operator[ ]** (const int &i) const

  get element

- T & **operator[ ]** (const int &i)

  set element

- **Vec3**< T > **unit** () const

  return unit vector with same direction as this vector

- bool **is_null** () const

  test for null vector

- **String format** () const

  *return formatted **String** (p. 426) representation*

- const **Vec3**< T > & **operator**+= (const **Vec3**< T > &v)

  add another vector to this one

- const **Vec3**< T > & **operator-**= (const **Vec3**< T > &v)

  subtract another vector from this one

Static Public Member Functions

- static **Vec3**< T > **zero** ()
  return zero vector
- static **Vec3**< T > **null** ()
  return null vector (only valid for floating point types)
- static T **dot** (const **Vec3**< T > &v1, const **Vec3**< T > &v2)
  *Vector dot product (equivalent to *).*
- static **Vec3**< T > **cross** (const **Vec3**< T > &v1, const **Vec3**< T > &v2)
  Vector cross product.

Friends

- int **operator==** (const **Vec3**< T > &v1, const **Vec3**< T > &v2)
  Vector equality (for floating point types see ***equals()*** *(p. 461)).*
- int **operator!=** (const **Vec3**< T > &v1, const **Vec3**< T > &v2)
  Vector inequality (for floating point types see ***equals()*** *(p. 461)).*
- **Vec3**< T > **operator-** (const **Vec3**< T > &v)
  Vector negation operator.
- **Vec3**< T > **operator+** (const **Vec3**< T > &v1, const **Vec3**< T > &v2)
  Vector addition operator.
- **Vec3**< T > **operator-** (const **Vec3**< T > &v1, const **Vec3**< T > &v2)
  Vector subtraction operator.
- **Vec3**< T > **operator*** (const T &s, const **Vec3**< T > &v1)
  Vector scaling operator.
- **Vec3**< T > **operator*** (const **Vec3**< T > &v1, const T &s)
  Vector scaling operator.
- T **operator*** (const **Vec3**< T > &v1, const **Vec3**< T > &v2)
  Vector dot product.

19.177.1 Detailed Description

template<class T = ftype> class clipper::Vec3< T >

3-vector class

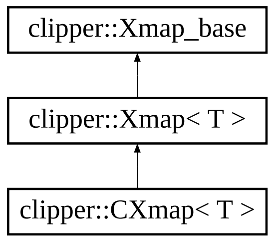
The documentation for this class was generated from the following file:

- clipper_types.h

19.178 clipper::Xmap< T > Class Template Reference

Xmap<T>: actual crystallographic map class.

`#include <xmap.h>`Inheritance diagram for clipper::Xmap< T >::



Public Member Functions

- **Xmap** ()

  Null constructor, for later initialisation.

- **Xmap** (const **Spacegroup** &spacegroup, const **Cell** &cell, const **Grid_sampling** &grid_-sam)

  constructor: from spacegroup, cell, and grid

- void **init** (const **Spacegroup** &spacegroup, const **Cell** &cell, const **Grid_sampling** &grid_sam)

  initialiser: from spacegroup, cell, and grid

- const T & **operator[ ]** (const **Xmap_base::Map_reference_index** &ix) const

  get data by Map_reference_index

- T & **operator[ ]** (const **Xmap_base::Map_reference_index** &ix)

  set data by Map_reference_index

- const T & **operator[ ]** (const **Xmap_base::Map_reference_coord** &ix) const

  get data by Map_reference_coord

- T & **operator[ ]** (const **Xmap_base::Map_reference_coord** &ix)

  set data by Map_reference_coord

- const T & **get_data** (const **Coord_grid** &pos) const

  get a density value for an arbitrary position

- void **set_data** (const **Coord_grid** &pos, const T &val)

  set a density value for an arbitrary position

- const T & **get_data** (const int &index) const

  get data by index (not recommended)

- bool **set_data** (const int &index, const T &val)

  set data by index (not recommended)

- template<class I >
  T **interp** (const **Coord_frac** &pos) const

  get map value for fractional coord using supplied interpolator

- template<class I >
  void **interp_grad** (const **Coord_frac** &pos, T &val, **Grad_frac**< T > &grad) const

  get map value and grad for fractional coord using supplied interpolator

- template<class I >
  void **interp_curv** (const **Coord_frac** &pos, T &val, **Grad_frac**< T > &grad, **Curv_-frac**< T > &curv) const

  get map value and curv for fractional coord using supplied interpolator

- template<class I >
  T **interp** (const **Coord_map** &pos) const

  get map value for map coord using supplied interpolator

- template<class I >
  void **interp_grad** (const **Coord_map** &pos, T &val, **Grad_map**< T > &grad) const

  get map value and grad for map coord using supplied interpolator

- template<class I >
  void **interp_curv** (const **Coord_map** &pos, T &val, **Grad_map**< T > &grad, **Curv_-map**< T > &curv) const

  get map value and curv for map coord using supplied interpolator

- template<class H >
  void **fft_from** (const H &fphidata, const FFTtype type=Default)

  FFT from reflection list to map.

- template<class H >
  void **fft_to** (H &fphidata, const FFTtype type=Default) const

  FFT from map to reflection list.

- const **Cell** & **cell** () const

  get the cell

- const **Spacegroup** & **spacegroup** () const

  get the spacegroup

- const **Grid_sampling** & **grid_sampling** () const

  get the cell grid

- const **Grid_range** & **grid_asu** () const

  get the ASU grid

- **Coord_grid** **coord_of** (const int &index) const

  map coordinate from index

- int **index_of** (const **Coord_grid** &coord) const *map index from coordinate*

- const int **multiplicity** (const **Coord_grid** &pos) const

  get multiplicity of a map grid point

- const **Coord_grid to_map_unit** (const **Coord_grid** &pos) const

  function to pick right cell repeat for any grid coord

- const **Map_reference_index first** () const

  return a Map_reference_index for this map

- const **Map_reference_coord first_coord** () const

  return a Map_reference_coord for this map

- const T & **operator=** (const T &value)

  assignment operator: assigns a single value to the whole map

- const **Xmap**< T > & **operator+=** (const **Xmap**< T > &other)

  add another map to this one

- const **Xmap**< T > & **operator-=** (const **Xmap**< T > &other)

  subtract another map from this one

19.178.1 Detailed Description

template<class T> class clipper::Xmap< T >

Xmap<T>: actual crystallographic map class. The crystallographic map class stores a map of arbitrary data type. Its main difference from a 3-d array is that the data extent appears to be infinite, and yet internally only a unique ASU is stored. Iterators provide efficient access to data.

This is derived from **Xmap_base** (p. 472), and adds the templatised data itself and the methods which deal with it.

Note:

The following methods are inherited from **Xmap_base** (p. 472) but are documented here for convenience: **cell()** (p. 464), **spacegroup()** (p. 464), **grid_sampling()** (p. 464), **grid_asu()** (p. 464), in_asu(), **multiplicity()** (p. 470), **to_map_unit()** (p. 465), **first()** (p. 465), **first_coord()** (p. 465).

19.178.2 Member Function Documentation

19.178.2.1 template<class T > const T & clipper::Xmap< T >::get_data (const Coord_grid & *pos*) const `[inline]`

get a density value for an arbitrary position Accessing the data by coordinate, rather than by Map_reference_index or Map_reference_coord, involves a symmetry lookup and is therefore slow. Avoid using these methods when you can.

19.178.2.2 template<class T > void clipper::Xmap< T >::set_data (const Coord_grid & *pos*, const T & *val*) [inline]

set a density value for an arbitrary position Accessing the data by coordinate, rather than by Map_reference_index or Map_reference_coord, involves a symmetry lookup and is therefore slow. Avoid using these methods when you can.

19.178.2.3 template<class T > const T & clipper::Xmap< T >::get_data (const int & *index*) const [inline]

get data by index (not recommended) Accessing the data by index, rather than by Map_reference_index or Map_reference_coord, is generally to be avoided since the indices do not start at zero and do not increase contiguously. These methods are only useful when a large number of references into a map must be stored, e.g. for sorting into density order.

19.178.2.4 template<class T > bool clipper::Xmap< T >::set_data (const int & *index*, const T & *val*) [inline]

set data by index (not recommended) Accessing the data by index, rather than by Map_reference_index or Map_reference_coord, is generally to be avoided since the indices do not start at zero and do not increase contiguously. These methods are only useful when a large number of references into a map must be stored, e.g. for sorting into density order.

Returns:

true if data was set, i.e. index is valid.

References clipper::Xmap_base::asu.

19.178.2.5 template<class T > template<class I > T clipper::Xmap< T >::interp (const Coord_frac & *pos*) const [inline]

get map value for fractional coord using supplied interpolator The value of the map at the desired non-grid fractional coordinate are calculated using the supplied interpolator template.

```
Coord_frac f( u, v, w );
y = xmap.interp<Interp_cubic>( f );
```

Parameters:

pos The fractional coord at which the density is to be calcuated.

Returns:

The value of the density at that point.

References clipper::Coord_frac::coord_map(), and clipper::Xmap_base::grid_sam_.

Referenced by clipper::Xmap< T >::interp().

19.178.2.6 template<class T > template<class I > void clipper::Xmap< T >::interp_grad (const Coord_frac & *pos*, T & *val*, Grad_frac< T > & *grad*) const [inline]

get map value and grad for fractional coord using supplied interpolator The value of the map at the desired non-grid fractional coordinate and its gradient are calculated using the supplied interpolator template.

Parameters:

pos The fractional coord at which the density is to be calcuated.

val The value of the density at that point.

grad The interpolated gradient vector with respect to the fractional coordinates (see Cell::coord_orth).

References clipper::Coord_frac::coord_map(), clipper::Grad_map< T >::grad_frac(), and clipper::Xmap_base::grid_sam_.

Referenced by clipper::Xmap< T >::interp_grad().

19.178.2.7 template<class T > template<class I > void clipper::Xmap< T >::interp_curv (const Coord_frac & *pos*, T & *val*, Grad_frac< T > & *grad*, Curv_frac< T > & *curv*) const [inline]

get map value and curv for fractional coord using supplied interpolator The value of the map at the desired non-grid fractional coordinate and its gradient and curvature are calculated using the supplied interpolator template. e.g.

Parameters:

pos The fractional coord at which the density is to be calcuated.

val The value of the density at that point.

grad The interpolated gradient vector with respect to the fractional coordinates (see Cell::coord_orth).

curv The interpolated curvature matrix with respect to the fractional coordinates (see Cell::coord_orth).

References clipper::Coord_frac::coord_map(), clipper::Curv_map< T >::curv_frac(), and clipper::Xmap_base::grid_sam_.

Referenced by clipper::Xmap< T >::interp_curv().

19.178.2.8 template<class T > template<class I > T clipper::Xmap< T >::interp (const Coord_map & *pos*) const [inline]

get map value for map coord using supplied interpolator The value of the map at the desired non-grid map coordinate are calculated using the supplied interpolator template.

```
Coord_map m( u, v, w );
y = xmap.interp<Interp_cubic>( m );
```

Parameters:

pos The map coord at which the density is to be calcuated.

Returns:

The value of the density at that point.

References clipper::Xmap< T >::interp().

19.178.2.9 template<class T > template<class I > void clipper::Xmap< T >::interp_grad (const Coord_map & *pos*, T & *val*, Grad_map< T > & *grad*) const [inline]

get map value and grad for map coord using supplied interpolator The value of the map at the desired non-grid map coordinate and its gradient are calculated using the supplied interpolator template.

Parameters:

pos The map coord at which the density is to be calcuated.

val The value of the density at that point.

grad The interpolated gradient vector with respect to the map coordinates (see Cell::coord_-orth).

References clipper::Xmap< T >::interp_grad().

19.178.2.10 template<class T > template<class I > void clipper::Xmap< T >::interp_curv (const Coord_map & *pos*, T & *val*, Grad_map< T > & *grad*, Curv_map< T > & *curv*) const [inline]

get map value and curv for map coord using supplied interpolator The value of the map at the desired non-grid map coordinate and its gradient and curvature are calculated using the supplied interpolator template. e.g.

Parameters:

pos The map coord at which the density is to be calcuated.

val The value of the density at that point.

grad The interpolated gradient vector with respect to the map coordinates (see Cell::coord_-orth).

curv The interpolated curvature matrix with respect to the map coordinates (see Cell::coord_orth).

References clipper::Xmap< T >::interp_curv().

19.178.2.11 template<class T > template<class H > void clipper::Xmap< T >::fft_from (const H & *fphidata*, const FFTtype *type* = Default) [inline]

FFT from reflection list to map. An FFT is calculated using the provided reflection list of F_phi, and used to fill this map. The reflection list is unchanged.

Parameters:

fphidata The reflection data list to use

References clipper::Xmap< T >::cell(), clipper::Xmap_base::default_type(), clipper::FFTmap_-p1::fft_h_to_x(), clipper::FFTmap_sparse_p1_hx::fft_h_to_x(), clipper::Xmap< T >::first(), clipper::Xmap< T >::grid_sampling(), clipper::Xmap_base::isymop, clipper::Xmap_-base::Map_reference_index::last(), clipper::Spacegroup::num_primops(), clipper::FFTmap_-p1::real_data(), clipper::FFTmap_sparse_p1_hx::real_data(), clipper::FFTmap_sparse_-p1_hx::require_real_data(), clipper::FFTmap_p1::set_hkl(), clipper::FFTmap_sparse_-p1_hx::set_hkl(), clipper::Xmap_base::spacegroup_, clipper::HKL::sym_phase_shift(), clipper::Spacegroup::symop(), and clipper::HKL::transform().

Referenced by clipper::Test_core::operator()().

19.178.2.12 template<class T > template<class H > void clipper::Xmap< T >::fft_to (H & *fphidata*, const FFTtype *type* = `Default`) const `[inline]`

FFT from map to reflection list. The Fourier transform of this map is calculated and used to fill a reflection list of F_phi. The map is unchanged.

Arguably this should be part of hkl_data<F_phi<T>>. But that requires writing a specialisation of hkl_data for F_phi. This is simpler and imposes less demands on the compiler.

Parameters:

fphidata The reflection data list to set.

References clipper::Xmap_base::cacheref, clipper::Xmap< T >::cell(), clipper::Xmap_-base::default_type(), clipper::FFTmap_p1::fft_x_to_h(), clipper::FFTmap_sparse_-p1_xh::fft_x_to_h(), clipper::Xmap< T >::first(), clipper::FFTmap_p1::get_hkl(), clipper::FFTmap_sparse_p1_xh::get_hkl(), clipper::Xmap_base::grid_sam_, clipper::Xmap< T >::grid_sampling(), clipper::Xmap_base::isymop, clipper::Xmap_base::Map_reference_-index::last(), clipper::FFTmap_p1::real_data(), clipper::FFTmap_sparse_p1_xh::real_data(), and clipper::FFTmap_sparse_p1_xh::require_hkl().

Referenced by clipper::Test_core::operator()().

19.178.2.13 template<class T> Coord_grid clipper::Xmap< T >::coord_of (const int & *index*) const `[inline]`

map coordinate from index

Parameters:

index The index.

Returns:

The corresponding grid coordinate.

Reimplemented from **clipper::Xmap_base** (p. 475).

19.178.2.14 template<class T> int clipper::Xmap< T >::index_of (const Coord_grid & *coord*) const `[inline]`

map index from coordinate This does not check symmetry equivalents.

Parameters:

coord The coordinate.

Returns:

The index, or -1 if it does not exist.

Reimplemented from **clipper::Xmap_base** (p. 476).

19.178.2.15 template<class T> const int clipper::Xmap< T >::multiplicity (const Coord_grid & *pos*) const

get multiplicity of a map grid point The multiplicity is the number of times the spacegroup operators map a particular grid point onto itself. This is required in order to properly weight map statistics so as to get the same result from just an ASU as using the whole cell.

Parameters:

pos The coordinate of the grid point.

Returns:

The multiplicty of the point.

Reimplemented from **clipper::Xmap_base** (p. 477).

19.178.2.16 template<class T > const T & clipper::Xmap< T >::operator= (const T & *value*) `[inline]`

assignment operator: assigns a single value to the whole map All values, including missing values, are overwritten by the value.

Parameters:

value The value to which the map is to be set.

References clipper::Xmap< T >::first(), and clipper::Xmap_base::Map_reference_index::last().

19.178.2.17 template<class T > const Xmap< T > & clipper::Xmap< T >::operator+= (const Xmap< T > & *other*) `[inline]`

add another map to this one The map grids and spacegroups must match.

References clipper::Xmap< T >::first(), clipper::Xmap< T >::grid_sampling(), clipper::Spgr_-descr::hash(), clipper::Xmap_base::Map_reference_index::last(), clipper::Message::message(), and clipper::Xmap< T >::spacegroup().

19.178.2.18 template<class T > const Xmap< T > & clipper::Xmap< T >::operator-= (const Xmap< T > & *other*) `[inline]`

subtract another map from this one The map grids and spacegroups must match.

References clipper::Xmap< T >::first(), clipper::Xmap< T >::grid_sampling(), clipper::Spgr_-descr::hash(), clipper::Xmap_base::Map_reference_index::last(), clipper::Message::message(), and clipper::Xmap< T >::spacegroup().

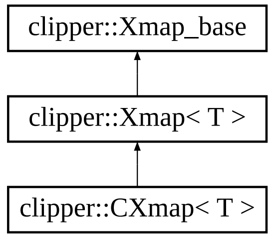
The documentation for this class was generated from the following file:

- xmap.h

19.179 clipper::Xmap_base Class Reference

Xmap_base (p. 472): base for crystallographic map class.

`#include <xmap.h>`Inheritance diagram for clipper::Xmap_base::



Classes

- class **Map_reference_base**
  Map reference base class.
- class **Map_reference_coord**
  Map reference with coordinate-like behaviour.
- class **Map_reference_index**
  Map reference with index-like behaviour.

Public Types

- enum **FFTtype** { **Default**, **Normal**, **Sparse** }

Public Member Functions

- bool **is_null** () const
  test if object has been initialised
- const **Cell** & **cell** () const
  get the cell
- const **Spacegroup** & **spacegroup** () const
  get the spacegroup
- const **Grid_sampling** & **grid_sampling** () const
  get the cell grid
- const **Grid_range** & **grid_asu** () const
  get the ASU grid
- **Coord_grid coord_of** (const int &index) const
  map coordinate from index

- int **index_of** (const **Coord_grid** &coord) const
  map index from coordinate
- **Coord_grid to_map_unit** (const **Coord_grid** &pos) const
  function to pick right cell repeat for any grid coord
- const **RTop** & **operator_orth_grid** () const
  return the orthogonal-to-grid coordinate operator (translation is zero)
- const **RTop** & **operator_grid_orth** () const
  return the grid-to-orthogonal coordinate operator (translation is zero)
- **Coord_orth coord_orth** (const **Coord_map** &cm) const
  convert map coordinate to orthogonal
- **Coord_map coord_map** (const **Coord_orth** &co) const
  convert orthogonal coordinate to map
- bool **in_map** (const **Coord_grid** &pos) const
  *(This method is for compatibility with **NXmap** (p. 357) - it always returns true)*
- template<class I >
  bool **in_map** (const **Coord_map** &cm) const
  *(This method is for compatibility with **NXmap** (p. 357) - it always returns true)*
- int **multiplicity** (const **Coord_grid** &pos) const
  get multiplicity of a map grid point
- **Map_reference_index first** () const
  *return a **Map_reference_index** (p. 328) for this map*
- **Map_reference_coord first_coord** () const
  *return a **Map_reference_coord** (p. 323) for this map*

Static Public Member Functions

- static FFTtype & **default_type** ()
  set/get default backend type

Protected Member Functions

- **Xmap_base** ()
  Null constructor, for later initialisation.
- void **init** (const **Spacegroup** &spacegroup, const **Cell** &cell, const **Grid_sampling** &grid_sam)

initialiser

- void **find_sym** (const **Coord_grid** &base, int &index, int &sym) const
- void **asu_error** (const **Coord_grid** &pos) const

Protected Attributes

- **ObjectCache**< **Xmap_cacheobj** >::Reference **cacheref**

  object cache reference

- const unsigned char * **asu**

  fast access ptr

- const **Isymop** * **isymop**

  fast access ptr

- const int * **du**

  fast access ptr

- const int * **dv**

  fast access ptr

- const int * **dw**

  fast access ptr

- **Grid_range asu_grid**

  fast access copy

- **Grid_range map_grid**

  fast access copy

- int **nsym**

  fast access copy

- **Cell cell_**

  unit cell

- **Spacegroup spacegroup_**

  spacegroup

- **Grid_sampling grid_sam_**

  grid for the whole cell

- **RTop rt_orth_grid**

  orth->grid operator

- **RTop rt_grid_orth**

  grid->orth operator

Static Protected Attributes

- static FFTtype **default_type_** = Xmap_base::Sparse

  default backend type

Friends

- class **Xmap_base::Map_reference_base**
- class **Xmap_base::Map_reference_index**
- class **Xmap_base::Map_reference_coord**

19.179.1 Detailed Description

Xmap_base (p. 472): base for crystallographic map class. The crystallographic map class stores a map of arbitrary data type. Its main difference from a 3-d array is that the data extent appears to be infinite, and yet internally only a unique ASU is stored. Iterators provide efficient access to data.

This base contains everything except the data, which is templated in the derived type Xmap<T>

19.179.2 Constructor & Destructor Documentation

19.179.2.1 clipper::Xmap_base::Xmap_base () `[protected]`

Null constructor, for later initialisation. For later initialisation: see **init()** (p. 477)

References clipper::Message::message().

19.179.3 Member Function Documentation

19.179.3.1 bool clipper::Xmap_base::is_null () const

test if object has been initialised

Returns:

true if the object has not been initalised.

References cell_, grid_sam_, clipper::Grid_sampling::is_null(), clipper::Cell::is_null(), clipper::Spacegroup::is_null(), and spacegroup_.

Referenced by init(), and clipper::CXmap< T >::update().

19.179.3.2 Coord_grid clipper::Xmap_base::coord_of (const int & *index*) const `[inline]`

map coordinate from index

Parameters:

index The index.

Returns:

The corresponding grid coordinate.

Reimplemented in **clipper::Xmap**< **T** > (p. 469).

References cacheref.

19.179.3.3 int clipper::Xmap_base::index_of (const Coord_grid & *coord*) const `[inline]`

map index from coordinate This does not check symmetry equivalents.

Parameters:

coord The coordinate.

Returns:

The index, or -1 if it does not exist.

Reimplemented in **clipper::Xmap**< **T** > (p. 469).

References asu, and cacheref.

19.179.3.4 Coord_orth clipper::Xmap_base::coord_orth (const Coord_map & *cm*) const `[inline]`

convert map coordinate to orthogonal

Parameters:

cm The grid coordinate to be converted.

Returns:

The equivalent orthogonal coordinate.

References clipper::RTop< T >::rot(), and rt_grid_orth.

19.179.3.5 Coord_map clipper::Xmap_base::coord_map (const Coord_orth & *co*) const `[inline]`

convert orthogonal coordinate to map

Parameters:

co The orthogonal coordinate to be converted.

Returns:

The equivalent grid coordinate.

References clipper::RTop< T >::rot(), and rt_orth_grid.

19.179.3.6 int clipper::Xmap_base::multiplicity (const Coord_grid & *pos*) const

get multiplicity of a map grid point The multiplicity is the number of times the spacegroup operators map a particular grid point onto itself. This is required in order to properly weight map statistics so as to get the same result from just an ASU as using the whole cell.

Parameters:

pos The coordinate of the grid point.

Returns:

The multiplicty of the point.

Reimplemented in **clipper::Xmap< T >** (p. 470).

References cacheref, grid_sam_, isymop, clipper::Coord_grid::transform(), and clipper::Coord_grid::unit().

19.179.3.7 void clipper::Xmap_base::init (const Spacegroup & *spacegroup*, const Cell & *cell*, const Grid_sampling & *grid_sam*) [protected]

initialiser The **Xmap** (p. 463) is initialised with a given cell, spacegroup, and grid sampling. A unique assymetric unit (ASU) of grid cells is selected and will be used to store a unique set of data.

If any of the parameters have null values, the existing values will be unchanged. The object will only be fully initialised once all parameters are available.

Parameters:

spacegroup The spacegroup for the map

cell The cell for the map

grid_sam The grid sampling for the map, i.e. the sampling along each axis for one whole cell

Reimplemented in **clipper::CXmap< T >** (p. 197), and **clipper::Xmap< T >** (p. 463).

References asu, asu_grid, clipper::ObjectCache< T >::cache(), cacheref, cell_, du, dv, dw, grid_sam_, clipper::RTop< T >::inverse(), is_null(), isymop, map_grid, clipper::Grid_sampling::matrix_grid_frac(), clipper::Cell::matrix_orth(), nsym, rt_grid_orth, rt_orth_grid, and spacegroup_.

The documentation for this class was generated from the following files:

- xmap.h
- xmap.cpp

19.180 clipper::Xmap_cacheobj Class Reference

Classes

- class **Key**

Public Member Functions

- **Xmap_cacheobj** (const **Key** &xmap_cachekey)
  construct entry
- bool **matches** (const **Key** &xmap_cachekey) const
  compare entry
- **String format** () const
  string description

Public Attributes

- **Key key**
  key
- **Grid_sampling xtl_grid**
  grid for the cell
- **Grid_range asu_grid**
  grid for the ASU
- **Grid_range map_grid**
  grid for the ASU, plus border
- int **nsym**
- std::vector< unsigned char > **asu**
  ASU flag array.
- std::vector< **Isymop** > **isymop**
  Integerised symops.
- std::vector< int > **du**
- std::vector< int > **dv**
- std::vector< int > **dw**
  symmetry grid shifts to index
- **Array2d**< unsigned char > **symperm**
  Perumtation matrix of symops.
- **Mat33 mat_grid_orth**
  for backward compatibility

Static Public Attributes

- static **Mutex mutex** = **Mutex**()
  thread safety

The documentation for this class was generated from the following files:

- xmap.h
- xmap.cpp

Chapter 20

File Documentation

20.1 atomsf.h File Reference

`#include "coords.h"`

Classes

- class **clipper::AtomShapeFn**
  Atomic shape function object.
- class **clipper::AtomSF**
  Atomic scattering factor object.

Namespaces

- namespace **clipper**

20.1.1 Detailed Description

Header file for atomic scattering factors

20.2 ramachandran.h File Reference

```
#include "clipper_types.h"
```

Classes

- class **clipper::Prob_phi_2d**
  2-d angular probability distibution class
- class **clipper::Ramachandran**
  Ramachandran *(*p. *382) plot class.*

Namespaces

- namespace **clipper**

20.2.1 Detailed Description

Ramachandran Plot class

Index