HKL Reference Manual

0.2

Generated by Doxygen 1.3.9.1

Tue Jan 4 14:16:11 2005

Contents

Chapter 1

HKL Hierarchical Index

1.1 HKL Class Hierarchy

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

Chapter 2

HKL Class Index

2.1 HKL Class List

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

Chapter 3

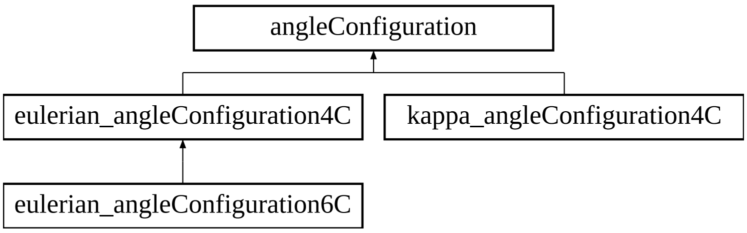
HKL Class Documentation

3.1 angleConfiguration Class Reference

The base class to represent the different kinds of configurations whatever the diffractometer type is.

`#include <angleconfig.h>`

Inheritance diagram for angleConfiguration::


angleConfiguration
eulerian_angleConfiguration4C
kappa_angleConfiguration4C
eulerian_angleConfiguration6C

3.1.1 Detailed Description

The base class to represent the different kinds of configurations whatever the diffractometer type is.

Store the current angle configuration according to the type of diffractometer.

Definition at line 9 of file angleconfig.h.

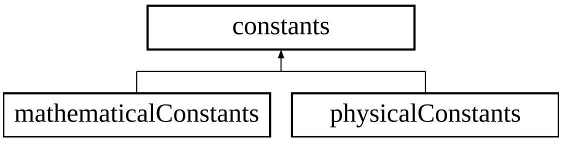
The documentation for this class was generated from the following file:

- D:/DS-sources/HKL/src/angleconfig.h

3.2 constants Class Reference

`#include <constants.h>`

Inheritance diagram for constants::



3.2.1 Detailed Description

This file contains all the constants we use in the HKL project. They are stored as static variables so we do not have to create instances of such objects.

Definition at line 11 of file constants.h.

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

- D:/DS-sources/HKL/src/constants.h

3.3 cristal Class Reference

```
#include <cristal.h>
```

Public Member Functions

- cristal (double alpha1, double alpha2, double alpha3, double beta1, double beta2, double beta3, double a1, double a2, double a3, double b1, double b2, double b3)
- cristal (double alpha1, double alpha2, double alpha3, double a1, double a2, double a3)
- cristal (const cristal &C)
- void set (double alpha1, double alpha2, double alpha3, double a1, double a2, double a3)

  Reset the fields with new values and recompute the reciprocal lattice and B.

- void set (const cristal &C)

  Copy a cristal.

- int check_cristal (const smatrix &B) const

Static Public Member Functions

- int test_cristals ()

3.3.1 Detailed Description

Class cristal to store direct and reciprocal lattice parameters and the matrix to move from the reciprocal lattice to the cristal cartesian system.

References :

William R. Busing and Henri A. Levy "Angle calculation for 3- and 4- Circle X-ray and Neutron Diffractometer" (1967) `Acta Cryst.`, **22**, 457-464.

A.J.C. Wilson "X-Ray Optics, The Diffraction of X-Rays By Finite and Imperfect Crystals" (1962) John Wiley & Sons Inc., 14-17.

Definition at line 21 of file cristal.h.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 cristal::cristal (double *alpha1*, double *alpha2*, double *alpha3*, double *beta1*, double *beta2*, double *beta3*, double *a1*, double *a2*, double *a3*, double *b1*, double *b2*, double *b3*)

Constructor to fill the class with data from both the direct and reciprocal lattice. Length units for a1, a2, a3, b1, b2, b3 have to be consistent with the wave length defined in the class source.

Parameters:

alpha1 The direct space first angle.

alpha2 The direct space second angle.

alpha3 The direct space third angle.

beta1 The reciprocal space first angle.

beta2 The reciprocal space second angle.

beta3 The reciprocal space third angle.

a1 The direct space first length.

a2 The direct space second length.

a3 The direct space third length.

b1 The reciprocal space first length.

b2 The reciprocal space second length.

b3 The reciprocal space third length.

Definition at line 14 of file cristal.cpp.

3.3.2.2 cristal::cristal (double *alpha1*, double *alpha2*, double *alpha3*, double *a1*, double *a2*, double *a3*)

Constructor to fill the class with data from the direct lattice and compute the reciprocal parameters with computeReciprocalLattice(), then call computeB(). Length units for a1, a2, a3 have to be consistent with the wave length defined in the class source.

Parameters:

alpha1 The direct space first angle.

alpha2 The direct space second angle.

alpha3 The direct space third angle.

a1 The direct space first length.

a2 The direct space second length.

a3 The direct space third length.

Definition at line 39 of file cristal.cpp.

3.3.2.3 cristal::cristal (const cristal & *C*)

Copy constructor.

Parameters:

C The crystal we want to copy.

Definition at line 53 of file cristal.cpp.

References m_a1, m_a2, m_a3, m_alpha1, m_alpha2, m_alpha3, m_B, m_b1, m_b2, m_b3, m_beta1, m_beta2, m_beta3, and set().

3.3.3 Member Function Documentation

3.3.3.1 int cristal::check_cristal (const smatrix & *B*) const

Check if the matrices B are the same in both crystals.

Returns:

0 if everything is OK, -1 otherwise.

Definition at line 177 of file cristal.cpp.

Referenced by test_cristals().

3.3.3.2 void cristal::set (double *alpha1*, double *alpha2*, double *alpha3*, double *a1*, double *a2*, double *a3*)

Reset the fields with new values and recompute the reciprocal lattice and B.

Fill the class with data from the direct lattice and compute the reciprocal parameters with computeReciprocalLattice(), then call computeB(). Length units for a1,a2,a3 have to be consistent with the wave length defined in the class source.

Parameters:

alpha1 The direct space first angle.

alpha2 The direct space second angle.

alpha3 The direct space third angle.

a1 The direct space first length.

a2 The direct space second length.

a3 The direct space third length.

Definition at line 119 of file cristal.cpp.

Referenced by cristal(), set(), diffractometer::setCrystal(), and test_cristals().

3.3.3.3 int cristal::test_cristals () `[static]`

Test six different cristals (cubic, orthorombic, hexagonal, triclinic) to make sure the computations are OK.

Returns:

0 if everything's fine, otherwise return the number of the cristal whose reciprocal lattice or matrix is wrong.

Definition at line 213 of file cristal.cpp.

References check_cristal(), and set().

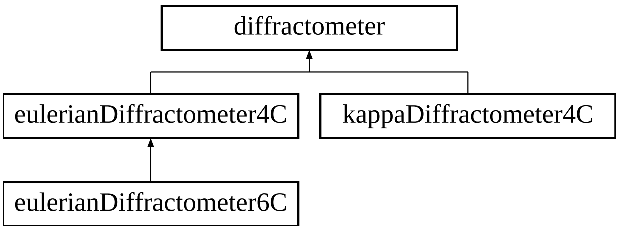
The documentation for this class was generated from the following files:

- D:/DS-sources/HKL/src/cristal.h
- D:/DS-sources/HKL/src/cristal.cpp

3.4 diffractometer Class Reference

```
#include <diffractometer.h>
```

Inheritance diagram for diffractometer::



Public Member Functions

- virtual angleConfiguration ∗ computeAngles (double h, double k, double l)=0

  The main function to get a sample of angles from (h,k,l).

- virtual angleConfiguration ∗ computeAngles_Rafin (double h, double k, double l)=0

  Designed for testing with Rafin algorithm.

- virtual void computeHKL (double &h, double &k, double &l, angleConfiguration ∗ac)=0

  Compute (h,k,l) from a sample of angles.

- virtual smatrix computeR ()=0

  Return the rotation matrix R for the current configuration.

- virtual smatrix computeR (angleConfiguration ∗ac1)=0

  Return the rotation matrix R for the given configuration ac1.

- virtual void setAngleConfiguration (angleConfiguration ∗ac1)=0

  Set the angle configuration and compute the corresponding rotation matrices.

- virtual smatrix computeU (angleConfiguration ∗ac1, double h1, double k1, double l1, angleConfiguration ∗ac2, double h2, double k2, double l2)=0

  Compute the orientation matrix from two basic non-parallel reflections.

- virtual smatrix computeU (reflection &r1, reflection &r2)=0

  Compute the orientation matrix from two basic non-parallel reflections.

- virtual ∼diffractometer ()

  The destructor.

- virtual void printOnScreen () const

  Print the content of the fields.

- virtual void setMode (mode::diffractometer_mode currentMode)=0

  Change the current computational mode.

- smatrix get_U () const

  Return the orientation matrix.

- smatrix get_UB () const

  Return the product of the orientation matrix by the crystal matrix.

- double getReflection_h (int index) const

  Get h from the array of experimental reflections.

- double getReflection_k (int index) const

  Get k from the array of experimental reflections.

- double getReflection_l (int index) const

  Get l from the array of experimental reflections.

- void setCrystal (double alpha1, double alpha2, double alpha3, double a1, double a2, double a3)

  Change the crystal from the direct lattice parameters.

- void setCrystal (const cristal &C)

  Change the crystal where the reciprocal lattice and matrix have already been computed.

- void setWaveLength (double wl)

  Change the light source wave length as it is something usual in an experiment.

Protected Member Functions

- diffractometer (cristal currentCristal, source currentSource, reflection &reflection1, reflection &reflection2)
- diffractometer (cristal currentCristal, source currentSource)
- diffractometer ()

Protected Attributes

- smatrix m_U

  The orientation matrix.

- smatrix m_UB

  Product $U * B$.

- mode ∗ m_currentMode

  The mode describes the way we use the diffractometer.

- source m_currentSource

  The light source and its wave length.

- cristal m_currentCristal

  The crystal direct and reciprocal parameters and its matrix B.

- reflection ∗ m_reflectionList

  The array to store up to 100 experiment results.

- const int m_sizeOfArray

  Size of the reflection array.

- int m_numberOfInsertedElements

  The number of reflections inserted into m_reflectionList.

- angleConfiguration ∗ m_currentConfiguration

  The current diffractometer angle configuration.

3.4.1 Detailed Description

The abstract base class to define all different kinds of diffractometers and drive experiments.

Definition at line 13 of file diffractometer.h.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 diffractometer::diffractometer (cristal *currentCristal*, source *currentSource*, reflection & *reflection1*, reflection & *reflection2*) `[protected]`

Commun constructor

- protected to make sure this class is abstract.

Definition at line 22 of file diffractometer.cpp.

References reflection::get_h(), reflection::get_k(), reflection::get_l(), reflection::getAngleConfiguration(), reflection::getRelevance(), m_currentConfiguration, and m_reflectionList.

3.4.2.2 diffractometer::diffractometer (cristal *currentCristal*, source *currentSource*) `[protected]`

Constructor designed for testing purposes

- protected to make sure this class is abstract.

Definition at line 69 of file diffractometer.cpp.

References m_reflectionList.

3.4.2.3 diffractometer::diffractometer () `[protected]`

Default constructor

- protected to make sure this class is abstract.

Definition at line 59 of file diffractometer.cpp.

References m_reflectionList, m_UB, and reflection::set().

3.4.3 Member Function Documentation

3.4.3.1 virtual angleConfiguration∗ diffractometer::computeAngles (double *h*, double *k*, double *l*) `[pure virtual]`

The main function to get a sample of angles from (h,k,l).

Parameters:

h The scaterring vector first element.

k The scaterring vector second element.

l The scaterring vector third element.

Returns:

The computed sample of angles.

Implemented in eulerianDiffractometer4C, and eulerianDiffractometer6C.

3.4.3.2 virtual angleConfiguration∗ diffractometer::computeAngles_Rafin (double *h*, double *k*, double *l*) `[pure virtual]`

Designed for testing with Rafin algorithm.

Parameters:

h The scaterring vector first element.

k The scaterring vector second element.

l The scaterring vector third element.

Returns:

The computed sample of angles.

Implemented in eulerianDiffractometer4C.

3.4.3.3 virtual void diffractometer::computeHKL (double & *h*, double & *k*, double & *l*, angleConfiguration ∗ *ac*) `[pure virtual]`

Compute (h,k,l) from a sample of angles.

Solve a linear system Ax = b where A is the product of the rotation matrices OMEGA, CHI, PHI by the orientation matrix U and the crystal matrix B. b is the scattering vector (q,0,0) and x = (h,k,l). Raise an exception when det(A)=0.

Parameters:

h The scaterring vector first element.

k The scaterring vector second element.

l The scaterring vector third element.

ac The diffractometer current angle configuration.

Exceptions:

when det(A)=0.

Implemented in eulerianDiffractometer4C, and eulerianDiffractometer6C.

3.4.3.4 virtual smatrix diffractometer::computeR () `[pure virtual]`

Return the rotation matrix R for the current configuration.

Compute the matrix R describing a complex rotation involving all the diffractometer circles.

Implemented in eulerianDiffractometer4C, kappaDiffractometer4C, and eulerianDiffractometer6C.

3.4.3.5 virtual smatrix diffractometer::computeU (reflection & *r1*, reflection & *r2*) `[pure virtual]`

Compute the orientation matrix from two basic non-parallel reflections.

Parameters:

r1 The first reflection.

r2 The second reflection.

Returns:

The orientation matrix U.

Implemented in eulerianDiffractometer4C, kappaDiffractometer4C, and eulerianDiffractometer6C.

3.4.3.6 virtual smatrix diffractometer::computeU (angleConfiguration ∗ *ac1*, double *h1*, double *k1*, double *l1*, angleConfiguration ∗ *ac2*, double *h2*, double *k2*, double *l2*) `[pure virtual]`

Compute the orientation matrix from two basic non-parallel reflections.

Parameters:

ac1 The first angle configuration corresponding to (h1,k1,l1).

h1 The first reflection (h,k,l) first component.

k1 The first reflection (h,k,l) second component.

l1 The first reflection (h,k,l) third component.

h2 The second reflection (h,k,l) first component.

k2 The second reflection (h,k,l) second component.

l2 The second reflection (h,k,l) third component.

ac2 The second angle configuration corresponding to (h2,k2,l2).

Returns:

The orientation matrix U.

Implemented in eulerianDiffractometer4C, kappaDiffractometer4C, and eulerianDiffractometer6C.

3.4.4 Member Data Documentation

3.4.4.1 smatrix diffractometer::m_U [protected]

The orientation matrix.

This orthogonal matrix relates the crystal cartesian system to the phi-axis system. It is computed from at least two relevant reflections.

Definition at line 126 of file diffractometer.h.

Referenced by printOnScreen().

3.4.4.2 smatrix diffractometer::m_UB [protected]

Product U $*$ B.

UB = U$*$B where B defines the crystal matrix and U the orientation matrix. UB relates the reciprocal space to the PHI-axis system.

Definition at line 131 of file diffractometer.h.

Referenced by diffractometer(), and printOnScreen().

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

- D:/DS-sources/HKL/src/diffractometer.h
- D:/DS-sources/HKL/src/diffractometer.cpp

3.5 Error Class Reference

The HKL exception abstraction base class.

`#include <HKLException.h>`

Public Member Functions

- Error (void)
- Error (const char ∗reason, const char ∗desc, const char ∗origin, int severity=ERR)
- Error (const std::string &reason, const std::string &desc, const std::string &origin, int severity=ERR)
- Error (const Error &src)
- virtual ∼Error (void)
- Error & operator= (const Error &_src)

Public Attributes

- std::string reason
- std::string desc
- std::string origin
- int severity

3.5.1 Detailed Description

The HKL exception abstraction base class.

Definition at line 40 of file HKLException.h.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 Error::Error (void)

Initialization.

Definition at line 26 of file HKLException.cpp.

3.5.2.2 Error::Error (const char ∗ *reason*, const char ∗ *desc*, const char ∗ *origin*, int *severity* = ERR)

Initialization.

Definition at line 38 of file HKLException.cpp.

3.5.2.3 Error::Error (const std::string & *reason*, const std::string & *desc*, const std::string & *origin*, int *severity* = ERR)

Initialization.

Definition at line 53 of file HKLException.cpp.

3.5.2.4 Error::Error (const Error & *src*)

Copy constructor.

Definition at line 68 of file HKLException.cpp.

3.5.2.5 Error::∼Error (void) `[virtual]`

Error details: code

Definition at line 80 of file HKLException.cpp.

3.5.3 Member Function Documentation

3.5.3.1 Error & Error::operator= (const Error & *_src*)

operator=

Definition at line 88 of file HKLException.cpp.

References desc, origin, reason, and severity.

3.5.4 Member Data Documentation

3.5.4.1 std::string Error::desc

Error details: description

Definition at line 89 of file HKLException.h.

Referenced by operator=().

3.5.4.2 std::string Error::origin

Error details: origin

Definition at line 94 of file HKLException.h.

Referenced by operator=().

3.5.4.3 std::string Error::reason

Error details: reason

Definition at line 84 of file HKLException.h.

Referenced by operator=().

3.5.4.4 int Error::severity

Error details: severity

Definition at line 99 of file HKLException.h.

Referenced by operator=().

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

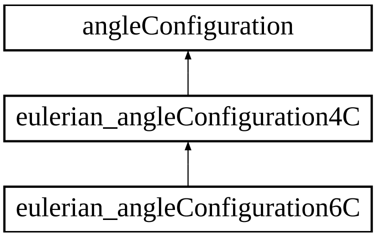
- D:/DS-sources/HKL/src/HKLException.h
- D:/DS-sources/HKL/src/HKLException.cpp

3.6 eulerian_angleConfiguration4C Class Reference

A set of four angles to define the crystal and detector positions.

`#include <angleconfig.h>`

Inheritance diagram for eulerian_angleConfiguration4C::



Public Member Functions

- eulerian_angleConfiguration4C ()

  Empty constructor setting all the fields to zero.

- eulerian_angleConfiguration4C (double, double, double, double)

  Constructor with an already made configuration.

- eulerian_angleConfiguration4C (double o, double c, double p, double t, double oi, double os, double ci, double cs, double pi, double ps, double ti, double ts)

  Constructor with an already made configuration and the angle intervals.

- virtual angleConfiguration ∗ makeCopy () const

  This redefined function builds a copy of the class.

- virtual void printOnScreen ()

  Print the angle values in radians.

- virtual void printDegreesOnScreen ()

  Print the angle values in degrees.

- virtual void printStaticOnScreen ()

  Print only static fields.

Protected Attributes

- double m_omega

  The first angle in an eulerian 4-circle diffractometer.

- double m_chi

  The second angle in an eulerian 4-circle diffractometer.

- double m_phi

The third angle in an eulerian 4-circle diffractometer.

- double m_2theta

  The detector angle in an eulerian 4-circle diffractometer.

Static Protected Attributes

- double m_omegaInf = 0.

  The first angle lower bound.

- double m_omegaSup = 0.

  The first angle upper bound.

- double m_chiInf = 0.

  The second angle lower bound.

- double m_chiSup = 0.

  The second angle upper bound.

- double m_phiInf = 0.

  The third angle lower bound.

- double m_phiSup = 0.

  The third angle upper bound.

- double m_2thetaInf = 0.

  The detector angle lower bound.

- double m_2thetaSup = 0.

  The detector angle upper bound.

3.6.1 Detailed Description

A set of four angles to define the crystal and detector positions.

The 4C eulerian diffractometer is defined by a set of 3 angles omega, chi and phi to move the crystal and also a fourth angle to move the detector by 2theta. Angles are in radians. Conventions are from William R. Busing and Henri A. Levy "Angle calculation for 3- and 4- Circle X-ray and Neutron Diffractometer" (1967) `Acta Cryst.`, **22**, 457-464.

Definition at line 27 of file angleconfig.h.

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

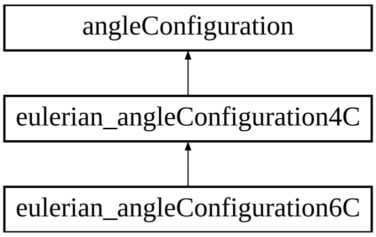
- D:/DS-sources/HKL/src/angleconfig.h
- D:/DS-sources/HKL/src/eulerian_angleconfiguration4C.cpp

3.7 eulerian_angleConfiguration6C Class Reference

A set of six angles to define the crystal and detector positions.

```
#include <angleconfig.h>
```

Inheritance diagram for eulerian_angleConfiguration6C::



Public Member Functions

- eulerian_angleConfiguration6C ()

  Empty constructor setting all the fields to zero.

- eulerian_angleConfiguration6C (double, double, double, double, double, double)

  Constructor with an already made configuration.

- eulerian_angleConfiguration6C (double m, double e, double c, double p, double n, double d, double mi, double ms, double ei, double es, double ci, double cs, double pi, double ps, double ni, double ns, double di, double ds)

  Constructor with an already made configuration and the angle intervals.

- virtual angleConfiguration $*$ makeCopy () const

  This redefined function builds a copy of the class.

- double getEta () const

  Use the protected field omega as eta.

- void setEta (double eta)

  Use the protected field omega as eta.

- double getDelta () const

  Use the protected field 2theta as delta.

- void setDelta (double delta)

  Use the protected field 2theta as delta.

- virtual void printOnScreen ()

  Print the angle values in radians.

- virtual void printDegreesOnScreen ()

  Print the angle values in degrees.

- virtual void printStaticOnScreen ()

  Print only static fields.

Static Public Member Functions

- void setEtaInf (double e)

  Use the protected field omega as eta.

- void setEtaSup (double e)

  Use the protected field omega as eta.

- void setDeltaInf (double t)

  Use the protected field 2theta as delta.

- void setDeltaSup (double t)

  Use the protected field 2theta as delta.

Protected Attributes

- double m_mu

  The new crystal angle in an eulerian 6-circle diffractometer.

- double m_nu

  The new detector angle in an eulerian 6-circle diffractometer.

Static Protected Attributes

- double m_muInf = 0.

  m_mu angle lower bound.

- double m_muSup = 0.

  m_mu angle upper bound.

- double m_nuInf = 0.

  m_nu angle lower bound.

- double m_nuSup = 0.

  m_nu angle upper bound.

3.7.1 Detailed Description

A set of six angles to define the crystal and detector positions.

The 6C eulerian diffractometer is defined by a set of 4 angles mu, eta, chi and phi to move the crystal and also nu and delta to move the detector. Angles are in radians. The angle omega previously defined in a

4-circle is now called eta and the detector angle 2theta has been renamed delta according to conventions from H. You "Angle calculations for a ‘4S+2D’ six-circle diffractometer" (1999) `J. Appl. Cryst.`, **32**, 614-623.

Definition at line 113 of file angleconfig.h.

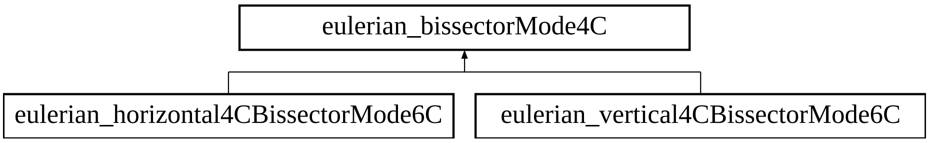
The documentation for this class was generated from the following files:

- D:/DS-sources/HKL/src/angleconfig.h
- D:/DS-sources/HKL/src/eulerian_angleConfiguration6C.cpp

3.8 eulerian_bissectorMode4C Class Reference

`#include <mode.h>`

Inheritance diagram for eulerian_bissectorMode4C::


eulerian_bissectorMode4C
eulerian_horizontal4CBissectorMode6C
eulerian_vertical4CBissectorMode6C

Public Member Functions

- eulerian_bissectorMode4C ()

  Default constructor.

- virtual angleConfiguration ∗ computeAngles (double h, double k, double l, const smatrix &UB, double lambda) const

  The main function to get a sample of angles from (h,k,l).

- angleConfiguration ∗ computeAngles_Rafin (double h, double k, double l, const smatrix &UB, double lambda) const

  Designed for testing with Rafin algorithm. Based on a geometric approach.

- virtual void computeHKL (double &h, double &k, double &l, const smatrix &UB, double lambda, angleConfiguration ∗ac) const

  Compute (h,k,l) from a sample of angles.

3.8.1 Detailed Description

The eulerian 4-circle diffractometer in bisector mode. William R. Busing and Henri A. Levy "Angle calculation for 3- and 4- Circle X-ray and Neutron Diffractometer" (1967) `Acta Cryst.`, **22**, 457-464.

Definition at line 193 of file mode.h.

3.8.2 Member Function Documentation

3.8.2.1 angleConfiguration ∗ eulerian_bissectorMode4C::computeAngles (double *h*, double *k*, double *l*, const smatrix & *UB*, double *lambda*) const `[virtual]`

The main function to get a sample of angles from (h,k,l).

Solving equation (19) from : William R. Busing and Henri A. Levy "Angle calculation for 3- and 4- Circle X-ray and Neutron Diffractometer" (1967) `Acta Cryst.`, **22**, 457-464.

- R11 ∗ hphi1 + R12 ∗ hphi2 + R13 ∗ hphi3 = q
- R21 ∗ hphi1 + R22 ∗ hphi2 + R23 ∗ hphi3 = 0

- R31 ∗ hphi1 + R32 ∗ hphi2 + R33 ∗ hphi3 = 0

- hphi1 = q(-sin(omega)∗sin(phi)+cos(omega)∗cos(chi)∗cos(phi))
- hphi2 = q(sin(omega)∗cos(phi)+cos(omega)∗cos(chi)∗sin(phi))
- hphi3 = q∗cos(omega)∗sin(chi)

If omega is constant :

- chi = arcsin(hphi3 / q∗cos(omega))
- sin(phi) = (hphi1∗sin(omega)-hphi2∗cos(omega)∗cos(chi)) / D
- cos(phi) = (hphi2∗sin(omega)+hphi1∗cos(omega)∗cos(chi)) / D

where D = q∗[cos(omega)∗cos(omega)∗cos(chi)∗cos(chi) + sin(omega)∗sin(omega)]

Parameters:

h The scaterring vector first element.

k The scaterring vector second element.

l The scaterring vector third element.

UB The product of the orientation matrix U by the crystal matrix B.

lambda The wave length.

Returns:

The computed sample of angles.

See also:

computeAngles_Rafin(), eulerianDiffractometer4C::test_eulerian4C()

Reimplemented in eulerian_horizontal4CBissectorMode6C, and eulerian_vertical4CBissectorMode6C.

Definition at line 51 of file eulerian_bissectormode4C.cpp.

References eulerian_angleConfiguration4C::set2Theta(), eulerian_angleConfiguration4C::setChi(), eulerian_angleConfiguration4C::setOmega(), and eulerian_angleConfiguration4C::setPhi().

3.8.2.2 angleConfiguration ∗ eulerian_bissectorMode4C::computeAngles_Rafin (double *h*, double *k*, double *l*, const smatrix & *UB*, double *lambda*) const

Designed for testing with Rafin algorithm. Based on a geometric approach.

Parameters:

h The scaterring vector first element.

k The scaterring vector second element.

l The scaterring vector third element.

UB The product of the orientation matrix U by the crystal matrix B.

lambda The wave length.

Returns:

The computed sample of angles.

See also:

computeAngles(), eulerianDiffractometer4C::test_eulerian4C()

Definition at line 181 of file eulerian_bissectormode4C.cpp.

References eulerian_angleConfiguration4C::set2Theta(), eulerian_angleConfiguration4C::setChi(), eulerian_angleConfiguration4C::setOmega(), and eulerian_angleConfiguration4C::setPhi().

3.8.2.3 void eulerian_bissectorMode4C::computeHKL (double & *h*, double & *k*, double & *l*, const smatrix & *UB*, double *lambda*, angleConfiguration ∗ *ac*) const [virtual]

Compute (h,k,l) from a sample of angles.

Solve a linear system Ax = b where A is the product of the rotation matrices OMEGA, CHI, PHI by the orientation matrix U and the crystal matrix B. b is the scattering vector (q,0,0) and x = (h,k,l). Raise an exception when det(A)=0.

Parameters:

h The scaterring vector first element.

k The scaterring vector second element.

l The scaterring vector third element.

UB The product of the orientation matrix U by the crystal matrix B.

lambda The wave length.

ac The diffractometer current angle configuration.

Exceptions:

when det(A)=0.

Reimplemented in eulerian_horizontal4CBissectorMode6C, and eulerian_vertical4CBissectorMode6C.

Definition at line 263 of file eulerian_bissectormode4C.cpp.

References eulerian_angleConfiguration4C::get2Theta(), eulerian_angleConfiguration4C::getChi(), eulerian_angleConfiguration4C::getOmega(), and eulerian_angleConfiguration4C::getPhi().

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

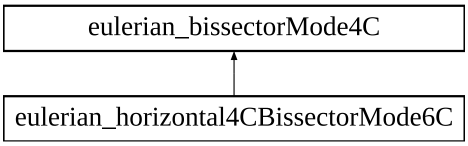
- D:/DS-sources/HKL/src/mode.h
- D:/DS-sources/HKL/src/eulerian_bissectormode4C.cpp

3.9 eulerian_horizontal4CBissectorMode6C Class Reference

Using an eulerian 6-circle diffractometer as an horizontal 4C eulerian one in bisector mode.

`#include <mode.h>`

Inheritance diagram for eulerian_horizontal4CBissectorMode6C::



Public Member Functions

- eulerian_horizontal4CBissectorMode6C ()

  Default constructor.

- virtual angleConfiguration ∗ computeAngles (double h, double k, double l, const smatrix &UB, double lambda) const

  The main function to get a sample of angles from (h,k,l).

- virtual void computeHKL (double &h, double &k, double &l, const smatrix &UB, double lambda, angleConfiguration ∗ac) const

  Compute (h,k,l) from a sample of angles.

3.9.1 Detailed Description

Using an eulerian 6-circle diffractometer as an horizontal 4C eulerian one in bisector mode.

The eulerian 6-circle diffractometer in horizontal bisector mode as described in

H. You "Angle calculations for a ‘4S+2D’ six-circle diffractometer" (1999) `J. Appl. Cryst.`, **32**, 614-623.

In this mode delta = eta = 0, so the scattering vector formula becomes :

Q = ||Q|| ∗ (0., -sin(theta), cos(theta)) where theta comes from the Bragg relation :

2tau ∗ sin(theta) = ||Q|| ∗ lambda

Definition at line 273 of file mode.h.

3.9.2 Member Function Documentation

3.9.2.1 angleConfiguration ∗ eulerian_horizontal4CBissectorMode6C::computeAngles (double *h*, double *k*, double *l*, const smatrix & *UB*, double *lambda*) const `[virtual]`

The main function to get a sample of angles from (h,k,l).

Solving equation (11) from : H. You "Angle calculations for a ‘4S+2D’ six-circle diffractometer" (1999) `J. Appl. Cryst.`, **32**, 614-623. MU.ETA.CHI.PHI.U.B.(h,k,l) = Q

Parameters:

h The scaterring vector first element.

k The scaterring vector second element.

l The scaterring vector third element.

UB The product of the orientation matrix U by the crystal matrix B.

lambda The wave length.

Returns:

The computed sample of angles.

See also:

eulerianDiffractometer4C::test_eulerian4C()

Reimplemented from eulerian_bissectorMode4C.

Definition at line 36 of file eulerian_mode6C.cpp.

References eulerian_angleConfiguration4C::setChi(), eulerian_angleConfiguration6C::setDelta(), eulerian_angleConfiguration6C::setEta(), eulerian_angleConfiguration6C::setMu(), eulerian_angleConfiguration6C::setNu(), and eulerian_angleConfiguration4C::setPhi().

3.9.2.2 void eulerian_horizontal4CBissectorMode6C::computeHKL (double & *h*, double & *k*, double & *l*, const smatrix & *UB*, double *lambda*, angleConfiguration ∗ *ac*) const `[virtual]`

Compute (h,k,l) from a sample of angles.

Solve a linear system Ax = b where A is the product of the rotation matrices MU, ETA, CHI, PHI by the orientation matrix U and the crystal matrix B. b is the scattering vector ||Q|| ∗ (0., -sin(theta), cos(theta)) and x = (h,k,l). Raise an exception when det(A)=0.

Parameters:

h The scaterring vector first element.

k The scaterring vector second element.

l The scaterring vector third element.

UB The product of the orientation matrix U by the crystal matrix B.

lambda The wave length.

ac The diffractometer current angle configuration.

Exceptions:

when det(A)=0.

Reimplemented from eulerian_bissectorMode4C.

Definition at line 152 of file eulerian_mode6C.cpp.

References eulerian_angleConfiguration4C::getChi(), eulerian_angleConfiguration6C::getDelta(), eulerian_angleConfiguration6C::getEta(), eulerian_angleConfiguration6C::getMu(), eulerian_angleConfiguration6C::getNu(), and eulerian_angleConfiguration4C::getPhi().

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

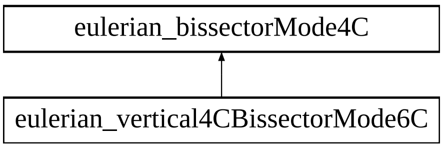
- D:/DS-sources/HKL/src/mode.h
- D:/DS-sources/HKL/src/eulerian_mode6C.cpp

3.10 eulerian_vertical4CBissectorMode6C Class Reference

Using an eulerian 6-circle diffractometer as an vertical 4C eulerian one in bisector mode.

`#include <mode.h>`

Inheritance diagram for eulerian_vertical4CBissectorMode6C::



Public Member Functions

- eulerian_vertical4CBissectorMode6C ()

  Default constructor.

- virtual angleConfiguration ∗ computeAngles (double h, double k, double l, const smatrix &UB, double lambda) const

  The main function to get a sample of angles from (h,k,l).

- virtual void computeHKL (double &h, double &k, double &l, const smatrix &UB, double lambda, angleConfiguration ∗ac) const

  Compute (h,k,l) from a sample of angles.

3.10.1 Detailed Description

Using an eulerian 6-circle diffractometer as an vertical 4C eulerian one in bisector mode.

The eulerian 6-circle diffractometer in vertical bisector mode as described in

H. You "Angle calculations for a ‘4S+2D’ six-circle diffractometer" (1999) `J. Appl. Cryst.`, **32**, 614-623.

In this mode mu = nu = 0, so the scattering vector formula becomes :

Q = ||Q|| ∗ (cos(theta), -sin(theta), 0.) where theta comes from the Bragg relation :

2tau ∗ sin(theta) = ||Q|| ∗ lambda

Definition at line 326 of file mode.h.

3.10.2 Member Function Documentation

3.10.2.1 angleConfiguration ∗ eulerian_vertical4CBissectorMode6C::computeAngles (double *h*, double *k*, double *l*, const smatrix & *UB*, double *lambda*) const `[virtual]`

The main function to get a sample of angles from (h,k,l).

Solving equation (11) from : H. You "Angle calculations for a ‘4S+2D’ six-circle diffractometer" (1999) `J. Appl. Cryst.`, **32**, 614-623. MU.ETA.CHI.PHI.U.B.(h,k,l) = Q

Parameters:

h The scaterring vector first element.

k The scaterring vector second element.

l The scaterring vector third element.

UB The product of the orientation matrix U by the crystal matrix B.

lambda The wave length.

Returns:

The computed sample of angles.

See also:

eulerianDiffractometer4C::test_eulerian4C()

Reimplemented from eulerian_bissectorMode4C.

Definition at line 304 of file eulerian_mode6C.cpp.

References eulerian_angleConfiguration4C::setChi(), eulerian_angleConfiguration6C::setDelta(), eulerian_angleConfiguration6C::setEta(), eulerian_angleConfiguration6C::setMu(), eulerian_angleConfiguration6C::setNu(), and eulerian_angleConfiguration4C::setPhi().

3.10.2.2 void eulerian_vertical4CBissectorMode6C::computeHKL (double & *h*, double & *k*, double & *l*, const smatrix & *UB*, double *lambda*, angleConfiguration ∗ *ac*) const `[virtual]`

Compute (h,k,l) from a sample of angles.

Solve a linear system Ax = b where A is the product of the rotation matrices MU, ETA, CHI, PHI by the orientation matrix U and the crystal matrix B. b is the scattering vector ||Q|| ∗ (cos(theta), -sin(theta), 0.) and x = (h,k,l). Raise an exception when det(A)=0.

Parameters:

h The scaterring vector first element.

k The scaterring vector second element.

l The scaterring vector third element.

UB The product of the orientation matrix U by the crystal matrix B.

lambda The wave length.

ac The diffractometer current angle configuration.

Exceptions:

when det(A)=0.

Reimplemented from eulerian_bissectorMode4C.

Definition at line 422 of file eulerian_mode6C.cpp.

References eulerian_angleConfiguration4C::getChi(), eulerian_angleConfiguration6C::getDelta(), eulerian_angleConfiguration6C::getEta(), eulerian_angleConfiguration6C::getMu(), eulerian_angleConfiguration6C::getNu(), and eulerian_angleConfiguration4C::getPhi().

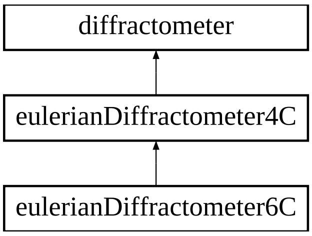
The documentation for this class was generated from the following files:

- D:/DS-sources/HKL/src/mode.h
- D:/DS-sources/HKL/src/eulerian_mode6C.cpp

3.11 eulerianDiffractometer4C Class Reference

```
#include <diffractometer.h>
```

Inheritance diagram for eulerianDiffractometer4C::



Public Member Functions

- eulerianDiffractometer4C (cristal currentCristal, source currentSource, reflection &reflection1, reflection &reflection2, mode::diffractometer_mode currentMode)

  Commun constructor.

- eulerianDiffractometer4C (cristal currentCristal, source currentSource, mode::diffractometer_mode currentMode)

  Constructor designed for testing purposes.

- eulerianDiffractometer4C (cristal currentCristal, source currentSource)

  Constructor designed for the 6C diffractometer.

- eulerianDiffractometer4C ()

  Default constructor.

- virtual smatrix computeR ()

  Compute the rotation matrix R for the current configuration.

- virtual smatrix computeR (angleConfiguration *ac1)

  Compute the rotation matrix R for a given configuration.

- virtual void setMode (mode::diffractometer_mode currentMode)

  Change the current computational mode.

- virtual void setAngleConfiguration (angleConfiguration *ac1)

  Set the angle configuration and compute the corresponding rotation matrices.

- virtual smatrix computeU (angleConfiguration *ac1, double h1, double k1, double l1, angleConfiguration *ac2, double h2, double k2, double l2)

  Compute the orientation matrix from two non-parallel reflections and set the UB matrix.

- virtual smatrix computeU (reflection &r1, reflection &r2)

  Compute the orientation matrix U from two non-parallel reflections and set the UB matrix.

- virtual angleConfiguration * computeAngles (double h, double k, double l)

The main function to compute a diffractometer configuration from a given (h, k, l).

- angleConfiguration ∗ computeAngles_Rafin (double h, double k, double l)

  Test function to compute a diffractometer configuration from a given (h, k, l).

- virtual void computeHKL (double &h, double &k, double &l, angleConfiguration ∗ac)

  Compute (h,k,l) from a sample of angles.

- virtual void printOnScreen () const

  Print the content of the fields.

Static Public Member Functions

- int test_eulerian4C ()

  Test all the main functionnalities.

Protected Attributes

- smatrix m_OMEGA

  The matrix corresponding to the first circle.

- smatrix m_CHI

  The matrix corresponding to the second circle.

- smatrix m_PHI

  The matrix corresponding to the third circle.

- smatrix m_2THETA

  The matrix corresponding to the fourth circle i.e. the detector.

- bool m_directOmega

  To reverse the first circle rotation sense.

- bool m_directChi

  To reverse the second circle rotation sense.

- bool m_directPhi

  To reverse the third circle rotation sense.

- bool m_direct2Theta

  To reverse the fourth circle rotation sense i.e. the detector.

3.11.1 Detailed Description

The eulerian 4-circle diffractometer. William R. Busing and Henri A. Levy "Angle calculation for 3- and 4- Circle X-ray and Neutron Diffractometer" (1967) `Acta Cryst.`, **22**, 457-464.

Definition at line 168 of file diffractometer.h.

3.11.2 Member Function Documentation

3.11.2.1 angleConfiguration ∗ eulerianDiffractometer4C::computeAngles (double *h*, double *k*, double *l*) `[virtual]`

The main function to compute a diffractometer configuration from a given (h, k, l).

Parameters:

- ***h*** The scaterring vector first element.
- ***k*** The scaterring vector second element.
- ***l*** The scaterring vector third element.

Returns:

The computed sample of angles.

See also:

eulerian_bissectorMode4C::computeAngles()

Implements diffractometer.

Reimplemented in eulerianDiffractometer6C.

Definition at line 241 of file diffractometer.cpp.

3.11.2.2 angleConfiguration ∗ eulerianDiffractometer4C::computeAngles_Rafin (double *h*, double *k*, double *l*) `[virtual]`

Test function to compute a diffractometer configuration from a given (h, k, l).

Parameters:

- ***h*** The scaterring vector first element.
- ***k*** The scaterring vector second element.
- ***l*** The scaterring vector third element.

Returns:

The computed sample of angles.

See also:

eulerian_bissectorMode4C::computeAngles_Rafin()

Implements diffractometer.

Definition at line 276 of file diffractometer.cpp.

3.11.2.3 void eulerianDiffractometer4C::computeHKL (double & *h*, double & *k*, double & *l*, angleConfiguration ∗ *ac*) `[virtual]`

Compute (h,k,l) from a sample of angles.

Solve a linear system Ax = b where A is the product of the rotation matrices OMEGA, CHI, PHI by the orientation matrix U and the crystal matrix B. b is the scattering vector (q,0,0) and x = (h,k,l). Raise an exception when det(A)=0.

Parameters:

h The scaterring vector first element.

k The scaterring vector second element.

l The scaterring vector third element.

ac The diffractometer current angle configuration.

Exceptions:

when det(A)=0.

Implements diffractometer.

Reimplemented in eulerianDiffractometer6C.

Definition at line 311 of file diffractometer.cpp.

3.11.2.4 smatrix eulerianDiffractometer4C::computeU (reflection & *r1*, reflection & *r2*) `[virtual]`

Compute the orientation matrix U from two non-parallel reflections and set the UB matrix.

William R. Busing and Henri A. Levy "Angle calculation for 3- and 4- Circle X-ray and Neutron Diffractometer" (1967) `Acta Cryst.`, **22**, 457-464. Compute h1c and h2c from equation (17)

h1c = B.h1

h2c = B.h2

h1phi = U.h1c

h2phi = U.h2c

u1phi = R1t.(1,0,0)

u2phi = R2t.(1,0,0)

h1phi // u1phi

h2phi // P(u1phi,u2phi)

Parameters:

r1 The first reflection.

r2 The second reflection.

Returns:

The orientation matrix U.

Implements diffractometer.

Reimplemented in eulerianDiffractometer6C.

Definition at line 344 of file diffractometer.cpp.

References computeR(), cristal::get_B(), reflection::get_h(), reflection::get_k(), reflection::get_l(), reflection::getAngleConfiguration(), and reflection::set().

3.11.2.5 smatrix eulerianDiffractometer4C::computeU (angleConfiguration ∗ *ac1*, double *h1*, double *k1*, double *l1*, angleConfiguration ∗ *ac2*, double *h2*, double *k2*, double *l2*) `[virtual]`

Compute the orientation matrix from two non-parallel reflections and set the UB matrix.

William R. Busing and Henri A. Levy "Angle calculation for 3- and 4- Circle X-ray and Neutron Diffractometer" (1967) `Acta Cryst.`, **22**, 457-464. Compute h1c and h2c from equation (17)

h1c = B.h1

h2c = B.h2

h1phi = U.h1c

h2phi = U.h2c

u1phi = R1t.(1,0,0)

u2phi = R2t.(1,0,0)

h1phi // u1phi

h2phi // P(u1phi,u2phi)

Parameters:

ac1 The first angle configuration corresponding to (h1,k1,l1).
h1 The first reflection (h,k,l) first component.
k1 The first reflection (h,k,l) second component.
l1 The first reflection (h,k,l) third component.
h2 The second reflection (h,k,l) first component.
k2 The second reflection (h,k,l) second component.
l2 The second reflection (h,k,l) third component.
ac2 The second angle configuration corresponding to (h2,k2,l2).

Returns:

The orientation matrix U.

Implements diffractometer.

Reimplemented in eulerianDiffractometer6C.

Definition at line 335 of file diffractometer.cpp.

Referenced by eulerianDiffractometer4C().

3.11.2.6 int eulerianDiffractometer4C::test_eulerian4C () `[static]`

Test all the main functionnalities.

Tests from 01 to 10 are basic tests to make sure computing B, U and angles from (h,k,l) are OK.

Tests from 11 to 20 are the same with differed settings.

Tests from 21 to 30 use Rafin algorithm to perform checks.

Tests from 31 to 40 compute angles from (h,k,l) and then (h,k,l) from these angles.

Returns:

0 if everything's fine, otherwise the number of the failing test.

Definition at line 2698 of file diffractometer.cpp.

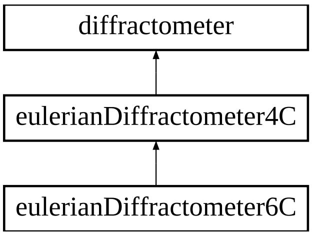
The documentation for this class was generated from the following files:

- D:/DS-sources/HKL/src/diffractometer.h
- D:/DS-sources/HKL/src/diffractometer.cpp

3.12 eulerianDiffractometer6C Class Reference

```
#include <diffractometer.h>
```

Inheritance diagram for eulerianDiffractometer6C::



Public Member Functions

- eulerianDiffractometer6C (cristal currentCristal, source currentSource, reflection &reflection1, reflection &reflection2, mode::diffractometer_mode currentMode)

  Commun constructor.

- eulerianDiffractometer6C ()

  Default constructor.

- virtual smatrix computeR ()

  Compute the rotation matrix R for the current configuration.

- virtual smatrix computeR (angleConfiguration ∗ac1)

  Compute the rotation matrix R for a given configuration.

- virtual void setMode (mode::diffractometer_mode currentMode)

  Change the current computational mode.

- virtual void setAngleConfiguration (angleConfiguration ∗ac1)

  Set the angle configuration and compute the corresponding rotation matrices.

- virtual smatrix computeU (angleConfiguration ∗ac1, double h1, double k1, double l1, angleConfiguration ∗ac2, double h2, double k2, double l2)

  Compute the orientation matrix from two non-parallel reflections and set the UB matrix.

- virtual smatrix computeU (reflection &r1, reflection &r2)

  Compute the orientation matrix U from two non-parallel reflections and set the UB matrix.

- virtual angleConfiguration ∗ computeAngles (double h, double k, double l)

  The main function to compute a diffractometer configuration from a given (h, k, l).

- virtual void computeHKL (double &h, double &k, double &l, angleConfiguration ∗ac)

  Compute (h,k,l) from a sample of angles.

- virtual void printOnScreen () const

  Print the content of the fields.

Static Public Member Functions

- int test_eulerian6C ()

  Test all the main functionnalities.

Protected Attributes

- smatrix m_MU

  The matrix corresponding to the fourth circle.

- smatrix m_NU

  The matrix corresponding to the detector second circle.

- bool m_directMu

  To reverse the fourth circle rotation sense.

- bool m_directNu

  To reverse the rotation sense of the detector second circle.

3.12.1 Detailed Description

The eulerian 6-circle diffractometer as described in H. You "Angle calculations for a ‘4S+2D’ six-circle diffractometer" (1999) `J. Appl. Cryst.`, **32**, 614-623. Two circles have been added from a 4C diffractometer, MU for the crystal and NU for the detector.According to H. You conventions the circle previously called Omega has been renamed Eta and the detector circle called 2Theta has been renamed Delta.

Definition at line 370 of file diffractometer.h.

3.12.2 Member Function Documentation

3.12.2.1 angleConfiguration ∗ eulerianDiffractometer6C::computeAngles (double *h*, double *k*, double *l*) `[virtual]`

The main function to compute a diffractometer configuration from a given (h, k, l).

Parameters:

h The scaterring vector first element.

k The scaterring vector second element.

l The scaterring vector third element.

Returns:

The computed sample of angles.

See also:

eulerian_bissectorMode4C::computeAngles()

Reimplemented from eulerianDiffractometer4C.

Definition at line 59 of file eulerian_ diffractometer6C.cpp.

Referenced by test_eulerian6C().

3.12.2.2 void eulerianDiffractometer6C::computeHKL (double & *h*, double & *k*, double & *l*, angleConfiguration ∗ *ac*) `[virtual]`

Compute (h,k,l) from a sample of angles.

Solve a linear system Ax = b where A is the product of the rotation matrices MU, ETA, CHI, PHI by the orientation matrix U and the crystal matrix B. b is the scattering vector and x = (h,k,l). Raise an exception when det(A)=0.

Parameters:

h The scaterring vector first element.

k The scaterring vector second element.

l The scaterring vector third element.

ac The diffractometer current angle configuration.

Exceptions:

when det(A)=0.

Reimplemented from eulerianDiffractometer4C.

Definition at line 81 of file eulerian_ diffractometer6C.cpp.

3.12.2.3 smatrix eulerianDiffractometer6C::computeU (reflection & *r1*, reflection & *r2*) `[virtual]`

Compute the orientation matrix U from two non-parallel reflections and set the UB matrix.

Parameters:

r1 The first reflection.

r2 The second reflection.

Returns:

The orientation matrix U.

Reimplemented from eulerianDiffractometer4C.

Definition at line 109 of file eulerian_ diffractometer6C.cpp.

References computeR(), cristal::get_B(), reflection::get_h(), reflection::get_k(), reflection::get_l(), reflection::getAngleConfiguration(), and source::getWaveLength().

3.12.2.4 smatrix eulerianDiffractometer6C::computeU (angleConfiguration ∗ *ac1*, double *h1*, double *k1*, double *l1*, angleConfiguration ∗ *ac2*, double *h2*, double *k2*, double *l2*) `[virtual]`

Compute the orientation matrix from two non-parallel reflections and set the UB matrix.

Parameters:

ac1 The first angle configuration corresponding to (h1,k1,l1).

h1 The first reflection (h,k,l) first component.

k1 The first reflection (h,k,l) second component.

l1 The first reflection (h,k,l) third component.

h2 The second reflection (h,k,l) first component.

k2 The second reflection (h,k,l) second component.

l2 The second reflection (h,k,l) third component.

ac2 The second angle configuration corresponding to (h2,k2,l2).

Returns:

The orientation matrix U.

Reimplemented from eulerianDiffractometer4C.

Definition at line 100 of file eulerian_ diffractometer6C.cpp.

Referenced by eulerianDiffractometer6C().

3.12.2.5 int eulerianDiffractometer6C::test_eulerian6C () `[static]`

Test all the main functionnalities.

Returns:

0 if everything's fine, otherwise the number of the failing test.

Definition at line 314 of file eulerian_ diffractometer6C.cpp.

References computeAngles(), eulerian_angleConfiguration4C::getChi(), eulerian_angleConfiguration6C::getDelta(), eulerian_angleConfiguration6C::getEta(), and eulerian_angleConfiguration4C::getPhi().

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

- D:/DS-sources/HKL/src/diffractometer.h
- D:/DS-sources/HKL/src/eulerian_ diffractometer6C.cpp

3.13 HKLException Class Reference

The HKL exception abstraction base class.

`#include <HKLException.h>`

Public Member Functions

- HKLException (void)
- HKLException (const char ∗reason, const char ∗desc, const char ∗origin, int severity=ERR)
- HKLException (const std::string &reason, const std::string &desc, const std::string &origin, int severity=ERR)
- HKLException (const Error &error)
- HKLException (const HKLException &src)
- HKLException & operator= (const HKLException &_src)
- virtual ∼HKLException (void)
- void push_error (const char ∗reason, const char ∗desc, const char ∗origin, int severity=ERR)
- void push_error (const std::string &reason, const std::string &desc, const std::string &origin, int severity=ERR)
- void push_error (const Error &error)

Public Attributes

- ErrorList errors

3.13.1 Detailed Description

The HKL exception abstraction base class.

Definition at line 115 of file HKLException.h.

3.13.2 Constructor & Destructor Documentation

3.13.2.1 HKLException::HKLException (void)

Initialization.

Definition at line 106 of file HKLException.cpp.

References push_error().

3.13.2.2 HKLException::HKLException (const char ∗ *reason*, const char ∗ *desc*, const char ∗ *origin*, int *severity* = ERR)

Initialization.

Definition at line 115 of file HKLException.cpp.

References push_error().

3.13.2.3 HKLException::HKLException (const std::string & *reason*, const std::string & *desc*, const std::string & *origin*, int *severity* = ERR)

Initialization.

Definition at line 127 of file HKLException.cpp.

References push_error().

3.13.2.4 HKLException::HKLException (const Error & *error*)

Initialization.

3.13.2.5 HKLException::HKLException (const HKLException & *src*)

Copy constructor.

Definition at line 139 of file HKLException.cpp.

References errors, and push_error().

3.13.2.6 HKLException::∼HKLException (void) [virtual]

Release resources.

Definition at line 169 of file HKLException.cpp.

References errors.

3.13.3 Member Function Documentation

3.13.3.1 HKLException & HKLException::operator= (const HKLException & *_src*)

operator=

Definition at line 150 of file HKLException.cpp.

References errors, and push_error().

3.13.3.2 void HKLException::push_error (const Error & *error*)

Push the specified error into the errors list.

Definition at line 200 of file HKLException.cpp.

References errors.

3.13.3.3 void HKLException::push_error (const std::string & *reason*, const std::string & *desc*, const std::string & *origin*, int *severity* = ERR)

Push the specified error into the errors list.

Definition at line 189 of file HKLException.cpp.

References errors.

3.13.3.4 void HKLException::push_error (const char ∗ *reason*, const char ∗ *desc*, const char ∗ *origin*, int *severity* = ERR)

Push the specified error into the errors list.

Definition at line 178 of file HKLException.cpp.

References errors.

Referenced by HKLException(), and operator=().

3.13.4 Member Data Documentation

3.13.4.1 ErrorList HKLException::errors

The errors list

Definition at line 185 of file HKLException.h.

Referenced by HKLException(), operator=(), push_error(), and ∼HKLException().

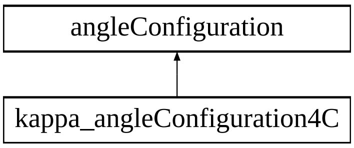
The documentation for this class was generated from the following files:

- D:/DS-sources/HKL/src/HKLException.h
- D:/DS-sources/HKL/src/HKLException.cpp

3.14 kappa_angleConfiguration4C Class Reference

```
#include <angleconfig.h>
```

Inheritance diagram for kappa_angleConfiguration4C::



Public Member Functions

- kappa_angleConfiguration4C ()

  Empty constructor which sets everything to zero.

- kappa_angleConfiguration4C (double, double, double, double)

  Constructor with an already made Configuration.

- kappa_angleConfiguration4C (double o, double c, double p, double t, double oi, double os, double ci, double cs, double pi, double ps, double ti, double ts)
- angleConfiguration ∗ makeCopy () const

  This redefined function builds a copy of the class.

- void printStaticOnScreen ()

  Print only static fields.

Protected Attributes

- double m_omega

  The four angles.

Static Protected Attributes

- double m_omegaInf = 0.

  The intervals associated to the angles.

3.14.1 Detailed Description

A space position in a 4C Kappa diffractometer is also defined by a set of of three angles omega, kappa and phi but the geometry axes are different. The fourth angle to move the detector is 2theta. Angles are in radians.

Definition at line 195 of file angleconfig.h.

3.14.2 Constructor & Destructor Documentation

3.14.2.1 kappa_angleConfiguration4C::kappa_angleConfiguration4C (double *o*, double *c*, double *p*, double *t*, double *oi*, double *os*, double *ci*, double *cs*, double *pi*, double *ps*, double *ti*, double *ts*)

Constructor with an already made configuration and the angle intervals.

Definition at line 54 of file kappa_angleconfiguration4C.cpp.

References m_omega, and m_omegaInf.

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

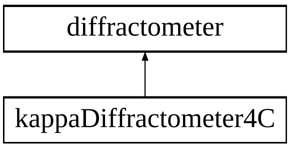
- D:/DS-sources/HKL/src/angleconfig.h
- D:/DS-sources/HKL/src/kappa_angleconfiguration4C.cpp

3.15 kappaDiffractometer4C Class Reference

This class describes a four-circle Kappa diffractometer.

```
#include <diffractometer.h>
```

Inheritance diagram for kappaDiffractometer4C::



Public Member Functions

- virtual smatrix computeR ()

  Compute the rotation for the current configuration.

- virtual smatrix computeR (angleConfiguration ∗ac1)

  Compute the rotation for a given configuration.

- virtual void setAngleConfiguration (angleConfiguration ∗ac1)

  Set the angle configuration and compute the corresponding rotation matrices.

- virtual smatrix computeU (angleConfiguration ∗ac1, double h1, double k1, double l1, angleConfiguration ∗ac2, double h2, double k2, double l2)

  Compute the orientation matrix from two basic non-parallel reflections.

- virtual smatrix computeU (reflection &r1, reflection &r2)

  Compute the orientation matrix from two basic non-parallel reflections.

- virtual void printOnScreen () const

  Print the content of the fields.

Protected Attributes

- smatrix m_OMEGA

  The matrix corresponding to the first circle.

- smatrix m_KAPPA

  The matrix corresponding to the second circle.

- smatrix m_PHI

  The matrix corresponding to the third circle.

- smatrix m_2THETA

  The matrix corresponding to the detector circle.

- smatrix m_ALPHA

  The matrix corresponding to the diffractometer inclination.

- smatrix m_OPP_ALPHA

  The opposite matrix corresponding to the diffractometer inclination m_OPP_ALPHA = -m_ALPHA.

- bool m_directOmega

  To reverse the first circle rotation sense.

- bool m_directKappa

  To reverse the second circle rotation sense.

- bool m_directPhi

  To reverse the third circle rotation sense.

- bool m_direct2Theta

  To reverse the detector circle rotation sense.

- double m_kappa

  The incident angle.

3.15.1 Detailed Description

This class describes a four-circle Kappa diffractometer.

The 4C Kappa diffractometer can be seen as a 4C eulerian one provided that we use some formula from the MHATT-CAT, Advanced Photon Source, Argonne National Laboratory (`MHATT-CAT146s Newport Kappa Diffractometer` written by Donald A. Walko). Other interesting documentation can be found at the `Brookhaven National Laboratory`

Definition at line 311 of file diffractometer.h.

3.15.2 Member Function Documentation

3.15.2.1 smatrix kappaDiffractometer4C::computeU (reflection & *r1*, reflection & *r2*) `[virtual]`

Compute the orientation matrix from two basic non-parallel reflections.

Parameters:

r1 The first reflection.

r2 The second reflection.

Returns:

The orientation matrix U.

Implements diffractometer.

Definition at line 2759 of file diffractometer.cpp.

3.15.2.2 smatrix kappaDiffractometer4C::computeU (angleConfiguration ∗ *ac1*, double *h1*, double *k1*, double *l1*, angleConfiguration ∗ *ac2*, double *h2*, double *k2*, double *l2*) `[virtual]`

Compute the orientation matrix from two basic non-parallel reflections.

Parameters:

ac1 The first angle configuration corresponding to (h1,k1,l1).

h1 The first reflection (h,k,l) first component.

k1 The first reflection (h,k,l) second component.

l1 The first reflection (h,k,l) third component.

h2 The second reflection (h,k,l) first component.

k2 The second reflection (h,k,l) second component.

l2 The second reflection (h,k,l) third component.

ac2 The second angle configuration corresponding to (h2,k2,l2).

Returns:

The orientation matrix U.

Implements diffractometer.

Definition at line 2765 of file diffractometer.cpp.

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

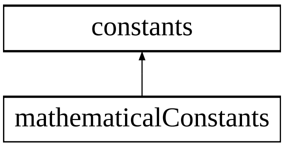
- D:/DS-sources/HKL/src/diffractometer.h
- D:/DS-sources/HKL/src/diffractometer.cpp

3.16 mathematicalConstants Class Reference

Store all the basic mathematical constants we need.

`#include <constants.h>`

Inheritance diagram for mathematicalConstants::



Static Protected Attributes

- double m_PI = 3.14159265358979323846

  The usual value of pi 3.14159265358979323846.

- double m_EPSILON_0 = 1.e-6

  The first precision factor.

- double m_EPSILON_1 = 1.e-10

  The second precision factor.

- double m_convertAnglesToDegrees = 57.2957795130823208

  To convert an angle in degrees (180 / PI).

- double m_convertAnglesToRadians = 0.01745329251994330

  To convert an angle in radians (PI / 180).

3.16.1 Detailed Description

Store all the basic mathematical constants we need.

Definition at line 40 of file constants.h.

3.16.2 Member Data Documentation

3.16.2.1 double mathematicalConstants::m_convertAnglesToDegrees = 57.2957795130823208 `[static, protected]`

To convert an angle in degrees (180 / PI).

All the computations are performed in radians, however if we want to have them in degrees (to print them out for example) we just need to multiply them by its value.

Definition at line 21 of file constants.cpp.

3.16.2.2 double mathematicalConstants::m_EPSILON_0 = 1.e-6 `[static, protected]`

The first precision factor.

This precision factor is used to test if two angles are the same.

Definition at line 11 of file constants.cpp.

3.16.2.3 double mathematicalConstants::m_EPSILON_1 = 1.e-10 `[static, protected]`

The second precision factor.

This precision factor is used to test if a double precision number is null.

Definition at line 15 of file constants.cpp.

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

- D:/DS-sources/HKL/src/constants.h
- D:/DS-sources/HKL/src/constants.cpp

3.17 mode Class Reference

`#include <mode.h>`

Public Member Functions

- virtual angleConfiguration ∗ computeAngles (double h, double k, double l, const smatrix &UB, double lambda) const =0

  The main function to get a sample of angles from (h,k,l).

- virtual angleConfiguration ∗ computeAngles_Rafin (double h, double k, double l, const smatrix &UB, double lambda) const =0

  Designed for testing with Rafin algorithm.

- virtual void computeHKL (double &h, double &k, double &l, const smatrix &UB, double lambda, angleConfiguration ∗ac) const =0

  Compute (h,k,l) from a sample of angles.

Protected Member Functions

- mode ()

3.17.1 Detailed Description

This file defines the mode telling how to use the diffractometer.

Definition at line 26 of file mode.h.

3.17.2 Constructor & Destructor Documentation

3.17.2.1 mode::mode () `[protected]`

Default constructor.

- protected to make sure this class is abstract.

Definition at line 9 of file eulerian_bissectormode4C.cpp.

3.17.3 Member Function Documentation

3.17.3.1 virtual angleConfiguration∗ mode::computeAngles (double *h*, double *k*, double *l*, const smatrix & *UB*, double *lambda*) const `[pure virtual]`

The main function to get a sample of angles from (h,k,l).

Parameters:

h The scaterring vector first element.

k The scaterring vector second element.

l The scaterring vector third element.

UB The product of the orientation matrix U by the crystal matrix B.

lambda The wave length.

Returns:

The computed sample of angles.

3.17.3.2 virtual angleConfiguration∗ mode::computeAngles_Rafin (double *h*, double *k*, double *l*, const smatrix & *UB*, double *lambda*) const [pure virtual]

Designed for testing with Rafin algorithm.

Parameters:

h The scaterring vector first element.

k The scaterring vector second element.

l The scaterring vector third element.

UB The product of the orientation matrix U by the crystal matrix B.

lambda The wave length.

Returns:

The computed sample of angles.

3.17.3.3 virtual void mode::computeHKL (double & *h*, double & *k*, double & *l*, const smatrix & *UB*, double *lambda*, angleConfiguration ∗ *ac*) const [pure virtual]

Compute (h,k,l) from a sample of angles.

Parameters:

h The scaterring vector first element.

k The scaterring vector second element.

l The scaterring vector third element.

UB The product of the orientation matrix U by the crystal matrix B.

lambda The wave length.

ac The diffractometer current angle configuration.

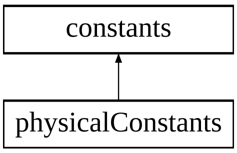
The documentation for this class was generated from the following files:

- D:/DS-sources/HKL/src/mode.h
- D:/DS-sources/HKL/src/eulerian_bissectormode4C.cpp

3.18 physicalConstants Class Reference

`#include <constants.h>`

Inheritance diagram for physicalConstants::



Static Protected Attributes

- double m_tau = 1.

3.18.1 Detailed Description

Store all the basic physical constants we need to define conventions.

Definition at line 20 of file constants.h.

3.18.2 Member Data Documentation

3.18.2.1 double physicalConstants::m_tau = 1. `[static, protected]`

We have to deal with two different conventions, if (a1,a2,a3) is the direct lattice and (b1,b2,b3) the reciprocal one, a1 $*$ b1 = 1 or a1 $*$ b1 = 2PI so we introduce tau = 1 or 2PI according to the user's choice.

Definition at line 3 of file constants.cpp.

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

- D:/DS-sources/HKL/src/constants.h
- D:/DS-sources/HKL/src/constants.cpp

3.19 reflection Class Reference

```
#include <reflection.h>
```

Public Types

- enum relevance

Public Member Functions

- reflection (angleConfiguration *this_angleConfiguration, double h, double k, double l, relevance this_relevance)
- double computeAngle (double h2, double k2, double l2) const
- void set (angleConfiguration *this_angleConfiguration, double h, double k, double l, relevance this_relevance)

Static Public Member Functions

- double test_computeAngle ()

  Designed to test computeAngle().

3.19.1 Detailed Description

The class reflection defines a configuration where a diffraction occurs. It is defined by a set of angles, the 3 integers associated to the reciprocal lattice and its relevance to make sure we only take into account significant reflections.

Definition at line 12 of file reflection.h.

3.19.2 Member Enumeration Documentation

3.19.2.1 enum reflection::relevance

The enumeration "relevance" to make sure we only take into account significant reflections.

Definition at line 17 of file reflection.h.

3.19.3 Constructor & Destructor Documentation

3.19.3.1 reflection::reflection (angleConfiguration * *this_angleConfiguration*, double *h*, double *k*, double *l*, relevance *this_relevance*)

Make a copy of the angle configuration to make sure we don't share it in memory.

Definition at line 19 of file reflection.cpp.

References angleConfiguration::makeCopy().

3.19.4 Member Function Documentation

3.19.4.1 double reflection::computeAngle (double *h2*, double *k2*, double *l2*) const

Compute the angle between two reflections to get an idea about their level of relevance (return the absolute value). As an example it can detect if (m_h, m_k, m_l) and (h2, k2, l2) are parallel.

Definition at line 47 of file reflection.cpp.

Referenced by test_computeAngle().

3.19.4.2 void reflection::set (angleConfiguration ∗ *this_angleConfiguration*, double *h*, double *k*, double *l*, relevance *this_relevance*)

Make a copy of the angle configuration to make sure we don't share it in memory.

Definition at line 32 of file reflection.cpp.

References angleConfiguration::makeCopy().

Referenced by eulerianDiffractometer4C::computeU(), diffractometer::diffractometer(), and eulerianDiffractometer4C::setAngleConfiguration().

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

- D:/DS-sources/HKL/src/reflection.h
- D:/DS-sources/HKL/src/reflection.cpp

3.20 source Class Reference

The class source defines a light ray and its main characteristics.

```
#include <source.h>
```

Public Member Functions

- source (source &S)

  Check if S units are consistent with the crystal units for the diffractometry computations.

- source (double _waveLength, double _monoAngle, double _undGap)

  _waveLength unit must be consistent with the crystal length units.

- void setWaveLength (double _wl)

  _wl unit must be consistent with the crystal length units.

3.20.1 Detailed Description

The class source defines a light ray and its main characteristics.

Definition at line 6 of file source.h.

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

- D:/DS-sources/HKL/src/source.h
- D:/DS-sources/HKL/src/source.cpp

Index