Package 'cry'

May 3, 2021

Type Package

Title Statistics for Structural Crystallography

Version 0.4.1

Date 2021-05-01

Description Reading and writing of files in the most commonly used formats of structural crystallography. It includes functions to work with a variety of statistics used in this field and functions to perform basic crystallographic computing. References: D. G. Waterman, J. Foadi, G. Evans (2011) <doi:10.1107/S0108767311084303>.

License GPL-2

LazyLoad yes

RoxygenNote 7.1.1

VignetteBuilder knitr

Encoding UTF-8

Depends R (>= 3.6.0)

Imports methods, zoo

Suggests knitr, rmarkdown, markdown, ggplot2

NeedsCompilation no

Author James Foadi [cre, aut],
David Waterman [aut],
Rita Giordano [aut],
Kutumbarao Nidamarthi [aut]

Maintainer James Foadi <james_foadi@yahoo.co.uk>

Repository CRAN

Date/Publication 2021-05-03 07:30:02 UTC

R topics documented:

`angle` *Constructor for an S3 object of class "angle"*

Description

Constructor for an S3 object of class "angle"

Usage

```
angle(ang, rad_flag = TRUE)
```

Arguments

`ang`	A real number, in degrees or radians depending on rad_flag.
`rad_flag`	A logical flag. If FALSE, the value is meant to be in radians.

Value

An object of class "angle" whose numerical value is always in degrees.

Examples

```
# Create an angle of 60 degrees
ang1 <- angle(60)
class(ang1)
```

avei_vs_res *Mean and standard deviation in resolution shells.*

Description

Calculates averages and standard deviations of the input vector quantity for all reflections, corresponding to shells of resolution.

Usage

```
avei_vs_res(nbin, resos, II = NULL, m = max(resos), M = min(resos))
```

Arguments

nbin	A positive integer. The number of resolution shells.
resos	A vector of real quantities. These are the resolutions (in angstroms) corresponding to the data vector, II. If the data vector is missing, the averages will be computed just for resos.
II	A vector of real quantities. This is the key quantity whose averages and standard deviations are calculated. If II is set to NULL, resolutions averages and standard deviations will be the calculated quantities.
m	Minimum (highest) resolution (in angstroms). Data with resolution smaller than m will be ignored when calculating the averages.
M	Maximum (lowest) resolution (in angstroms). Data with resolution larger than M will be ignored when calculating the averages.

Details

Binning is done with inverse resolutions in order to have lower resolutions correspond to small numbers and high resolutions to large numbers. The output mids, ave and sd correspond to inverse resolutions.

Value

A named list of length 4. counts is a vector of integers, the number of reflections in each resolution shell. mids is the representative inverse resolution for each resolution shell; the value is decided by the function hist. ave is the average value in each resolution shell and sd is the corresponding standard deviation.

Examples

```
datadir <- system.file("extdata",package="cry")
filename <- file.path(datadir,"1dei_phases.mtz")
lmtz <- readMTZ(filename)
hkl <- lmtz$reflections[,1:3]
II <- lmtz$reflections[,4]
cpars <- lmtz$header$CELL
```

```
resos <- hkl_to_reso(hkl[,1],hkl[,2],hkl[,3],
                     cpars[1],cpars[2],cpars[3],
                     cpars[4],cpars[5],cpars[6])
ltmp <- avei_vs_res(20,resos,II)
plot(ltmp$mids,ltmp$ave,type="b",pch=16)
```

`bravais` *Constructor for an S3 object of class "bravais"*

Description

There are 14 Bravais lattices. They are represented by a two-letter symbol: aP, mP, mS, oP, oS, oF, oI, tP, tI, hP, hR, cP, cF, cI

Usage

```
bravais(bt = NULL)
```

Arguments

`bt`	A two-letter character, denoting the Bravais type.

Value

An object of class "bravais". It is a named list of length 4. The first slot, "bt", is the universally-used two-letter symbol. The second, third and fourth slots are, respectively, "cr_fam" (the corresponding crystal family), "cr_sys" (the corresponding crystal system) and "lt_sys" (the corresponding lattice system).

Examples

```
# mS is a monoclinic, face-centred Bravais lattice
bt <- bravais("mS")
class(bt)
bt[1:4]
```

`calculate_cell_volume` *S3 generic to compute cell volume*

Description

The volume of a unit cell and a reciprocal unit cell can be calculated starting from specific objects, files, etc.

Usage

```
calculate_cell_volume(x, ...)
```

Arguments

x	An object used to select a method. Either an object of class unit_cell or an object of class rec_unit_cell.
...	Further arguments passed to or from other methods.

Value

V A real number. The volume (in angstroms^3 or angstroms^-3) of the input unit cell or reciprocal unit cell.

Examples

```
# Calculate the volume of a unit cell
uc <- unit_cell(20)
V <- calculate_cell_volume(uc)

# Calculate the volume of the corresponding reciprocal cell
ruc <- create_rec_unit_cell(uc)
Vrec <- calculate_cell_volume(ruc)
V*Vrec  # Should be 1!
```

---

calculate_cell_volume.rec_unit_cell

Volume of a reciprocal unit cell (in angstroms^(-3))

Description

Method of the S3 generic class "calculate_cell_volume", to calculate the volume, in reciprocal cubic angstroms, of the reciprocal unit cell corresponding to the input object of class "rec_unit_cell".

Usage

```
## S3 method for class 'rec_unit_cell'
calculate_cell_volume(x, ...)
```

Arguments

x	An object of class "rec_unit_cell".
...	Additional arguments passed to the calculate_cell_volume methods

Value

A positive numeric, the volume in reciprocal cubic angstroms of the reciprocal unit cell corresponding to the input.

Examples

```
# Create a monoclinic cell and the corresponding reciprocal
bt <- bravais("mP")
uc <- create_unit_cell(bt)
ruc <- create_rec_unit_cell(uc)

# Calculate reciprocall cell volume
calculate_cell_volume(ruc)
```

---

calculate_cell_volume.unit_cell

Volume of a unit cell (in angstroms^3)

Description

Method of the S3 generic class "calculate_cell_volume", to calculate the volume, in cubic angstroms, of the unit cell corresponding to the input object of class "unit_cell".

Usage

```
## S3 method for class 'unit_cell'
calculate_cell_volume(x, ...)
```

Arguments

x	An object of class "unit_cell".
...	Additional arguments passed to the calculate_cell_volume methods

Value

A positive numeric, the volume in cubic angstroms of the unit cell corresponding to the input.

Examples

```
# Create a monoclinic cell
bt <- bravais("mP")
uc <- create_unit_cell(bt)
print(uc)

# Calculate cell volume
calculate_cell_volume(uc)
```

change_COLSRC *Change COLSRC date and time stamp*

Description

Function to update the `created` column of the data frame `COLSRC` with current date and time.

Usage

```
change_COLSRC(hdr)
```

Arguments

`hdr`	A data frame. The `COLSRC` data frame included in the `header` component of the named list obtained with `readMTZ` or `readMTZHeader`.

Details

The COLSRC data frame of an MTZ header has a column called `created` which displays the date and time at which the MTZ file data columns were created. When writing out a modified list obtained from reading an MTZ file, one might want to change the `created` column with the current date and time. Other specific types of change can be operated by handling the `COLSRC` data frame in an *ad hoc* manner.

Value

The `hdr` input data frame with the `created` column of the `COLSRC` data frame changed to display the current date and time.

Examples

```
# Read a sample MTZ file
datadir <- system.file("extdata",package="cry")
filename <- file.path(datadir,"1dei_phases.mtz")
lMTZ <- readMTZ(filename)

# Original COLSRC
print(lMTZ$header$COLSRC)

# Update date and time stamp
lMTZ$header <- change_COLSRC(lMTZ$header)

# New COLSRC
print(lMTZ$header$COLSRC)
```

check_angle_validity *Validity and compatibility of a cry object of class 'angle'*

Description

An object of class 'angle' is a numeric with logical attribute "rad_flag".

Usage

```
check_angle_validity(x, message = FALSE)
```

Arguments

x	Object of class angle.
message	A logical variable. If TRUE, the function prints a message on the errors, if any (default is FALSE, i.e. no message printed).

Value

ans A logical value. TRUE means that the input is a valid object of class 'angle'.

Examples

```
# Create an object of class angle
x <- angle(80)

# Check its validity
check_angle_validity(x)

# Modify the 'rad_flag' attribute
attr(x,"rad_flag") <- 12.5

# Check its validity
check_angle_validity(x,TRUE)
```

check_bravais_validity *Validity and compatibility of a cry object of class 'bravais'*

Description

An object of class 'bravais' is a named list of length 4. The first slot, "bt", is the universally-used two-letter symbol. The second, third and fourth slots are, respectively, "cr_fam" (the corresponding crystal family), "cr_sys" (the corresponding crystal system) and "lt_sys" (the corresponding lattice system).

Usage

```
check_bravais_validity(x, message = FALSE)
```

Arguments

x	Object of class 'bravais'.
message	A logical variable. If TRUE, the function prints a message on the errors, if any (default is FALSE, i.e. no message printed).

Value

ans A logical value. TRUE means that the input is a valid object of class 'bravais'.

Examples

```
# Create an object of class 'bravais'
x <- bravais("mP")

# Check its validity
check_bravais_validity(x,TRUE)
```

---

check_cryst_symm_validity

Validity and compatibility of a cry object of class 'cryst_symm'

Description

An object of class 'cryst_symm' is a named list of length 4. The first field is a character string, the second field is a 3×3 array and the third and fourth field are 3×1 arrays.

Usage

```
check_cryst_symm_validity(x, message = FALSE)
```

Arguments

x	Object of class 'cryst_symm'.
message	A logical variable. If TRUE, the function prints a message on the errors, if any (default is FALSE, i.e. no message printed).

Value

ans A logical value. TRUE means that the input is a valid object of class'cryst_symm'.

Examples

```
# Create an object of class 'cryst_symm'
x <- cryst_symm(15)

# Check its validity
check_cryst_symm_validity(x)

# Now change a field
x$PG[[1]] <- matrix(rep(0,times=9),ncol=3)

# Check validity again
check_cryst_symm_validity(x,TRUE)
```

---

check_merged_reflections_validity

Validity and compatibility of a cry object of class 'merged_reflections'

Description

An object of class 'merged_reflections' is a named list of length 4:

ruc An object of class "rec_unit_cell".

csym An object of class "cryst_symm".

records A data frame containing the data.

dtypes A character vector containing the type of data (Miller indices, structure factors, etc).

Internal consistency must be displayed between the object 'ruc' and the object 'csym' because groups of crystallographic symmetries are compatible only with certain unit cells (and, accordingly, certain reciprocal cells). It is not possible to check consistency between dtypes and the nature of data in each column of the data frame 'records', but a check about length of 'dtypes' and number of columns is possible. Therefore, the user should pay attention to the nature of his/her data. Also, merged reflection data, having to be compatible with crystal symmetry, have to display the appropriate systematic absences. Users interested in keeping systematic absences in the object, might want to look at the object of class "raw_reflections".

Usage

```
check_merged_reflections_validity(x, message = FALSE)
```

Arguments

x	Object of class 'merged_reflections'.
message	A logical variable. If TRUE, the function prints a message on the errors, if any.

Value

ans A logical value. TRUE means that the input is a valid object of class'merged_reflections'.

Examples

```
# Create an object of class 'merged_reflections'
# (default ara data associated with a cubic cell)
x <- merged_reflections()

# Check its validity
check_merged_reflections_validity(x)

# Now change reciprocal unit cell (to triclinic)
uc <- unit_cell(10,20,30,30,50,70)
ruc <- create_rec_unit_cell(uc)
x$ruc <- ruc

# Check validity again
check_merged_reflections_validity(x)
```

---

check_rec_unit_cell_validity

Validity and compatibility of a cry object of class 'rec_unit_cell'

Description

An object of class 'rec_unit_cell' is a named list of length 6. The first three fields are numeric, the last three of class 'angle'.

Usage

```
check_rec_unit_cell_validity(x, message = FALSE)
```

Arguments

x	Object of class 'rec_unit_cell'.
message	A logical variable. If TRUE, the function prints a message on the errors, if any (default is FALSE, i.e. no message printed).

Value

ans A logical value. TRUE means that the input is a valid object of class'rec_unit_cell'.

Examples

```
# Create an object of class 'rec_unit_cell'
x <- create_rec_unit_cell()

# Check its validity
check_rec_unit_cell_validity(x)
```

```
# Now change a field
x$alphar <- 123

# Check validity again
check_rec_unit_cell_validity(x,TRUE)
```

`check_unit_cell_validity` *Validity and compatibility of a cry object of class 'unit_cell'*

Description

An object of class 'unit_cell' is a named list of length 6. The first three fields are numeric, the last three of class 'angle'.

Usage

```
check_unit_cell_validity(x, message = FALSE)
```

Arguments

`x`	Object of class 'unit_cell'.
`message`	A logical variable. If TRUE, the function prints a message on the errors, if any (default is FALSE, i.e. no message printed).

Value

ans A logical value. TRUE means that the input is a valid object of class'unit_cell'.

Examples

```
# Create an object of class 'unit_cell'
x <- create_unit_cell()

# Check its validity
check_unit_cell_validity(x)

# Now change a field
x$alpha <- 123

# Check validity again
check_unit_cell_validity(x,TRUE)
```

check_validity *Validity and compatibility of cry objects*

Description

Compatibility of cry objects The objects that can be created in cry are subject to issues of compatibility of the parameters forming them. For example, the unit cell of a cubic system cannot have the a, b, c sides different from each other. The present function returns TRUE if all parts composing the object are compatible with each other. Otherwise it returns FALSE and one or more warnings with details.

Usage

```
check_validity(x, y = NULL, message = FALSE)
```

Arguments

x	A first object of one of the new cry classes.
y	A second object of one of the new cry classes.
message	A logical variable. If TRUE, the function prints a message on the errors, if any (default is FALSE, i.e. no message printed).

Details

The available checks for individual objects are:

- **angle** (unit cell angle)
- **bravais** (Bravais system)
- **unit_cell** (unit cell)
- **rec_unit_cell** (reciprocal unit cell)
- **cryst_symm** (crystallographic symmetry)
- **merged_reflections** (scaled and merged data)

The available checks for couple of objects are:

- **bravais** and **unit_cell**
- **unit_cell** and **cryst_symm**

Value

ans A logical value. ans = TRUE means that either the parameters of the single object (if only one input is provided) are valid parameters, or that the two objects are compatible with each other.

Examples

```
# Create a cubic cell with side 50
uc <- create_unit_cell(50)

# Create an object of class "cryst_symm"
crsym <- cryst_symm("P m -3")

# Are they compatible with each other?
check_validity(uc,crsym,TRUE)
```

`create_merged_reflections` *S3 generic to create merged_reflections objects*

Description

The merged_reflections object can be created starting from specific objects, files, etc.

Usage

```
create_merged_reflections(ruc, ...)
```

Arguments

`ruc`	An object used to select a method.
`...`	Further arguments passed to or from other methods.

Value

mrefs An object of class "merged_reflections". It is a named list of length 4 whose names are:

ruc An object of class "rec_unit_cell".

csym An object of class "cryst_symm".

records A data frame containing the data.

dtypes A character vector containing the type of data (Miller indices, structure factors, etc).

Examples

```
# Create a default merged_reflections object (no arguments)
mrefs <- create_merged_reflections()
print(mrefs)

# Create merged_reflections object from symmetry
csym <- cryst_symm("P 3")
mrefs <- create_merged_reflections(csym=csym)
print(mrefs)
```

`create_merged_reflections.default` *Default method for generic "create_merged_reflections"*

Description

This method is an alternative call to 'merged_reflections'. .

Usage

```
## Default S3 method:
create_merged_reflections(
  ruc = NULL,
  csym = NULL,
  records = NULL,
  dtypes = NULL,
  ...
)
```

Arguments

`ruc`	An object of class 'rec_unit_cell'.
`csym`	An object of class 'cryst_symm'.
`records`	A data frame containing all reflections coming from the x-ray data collection on the crystal. This data frame must include at least the three Miller indices, H, K, L (of dtype "H").
`dtypes`	A character vector whose length is the same as the number of columns in 'records'. One character (a capital letter) is associated with each type of data. For example, a Miller index is of dtype "H"; a structure amplitude is of dtype "F"; an anomalous difference is of dtype "D"; etc (see details later).
`...`	Additional arguments passed to the create_merged_reflections methods.

Value

An object of class "merged_reflections". It is a named list of length 4 whose names are:

ruc An object of class "rec_unit_cell".

csym An object of class "cryst_symm".

records A data frame containing the data.

dtypes A character vector containing the type of data (Miller indices, structure factors, etc).

See Also

`merged_reflections`

Examples

```
# Create merged data for a cubic (10 angstrom) unit cell
# of space group P 2 3. Data up to 5 angstroms resolution
mrefs <- create_merged_reflections()
print(mrefs)
```

`create_rec_unit_cell` *S3 generic to create rec_unit_cell objects*

Description

The rec_unit_cell object can be created starting from specific objects, files, etc.

Usage

```
create_rec_unit_cell(ar, ...)
```

Arguments

`ar`	An object or objects used to select a method. These can be reciprocal unit cell parameters, an object of class rec_unit_cell, etc.
`...`	Further arguments passed to or from other methods.

Value

An object of class "rec_unit_cell". It is a named list of length 6 whose last three slots are of "angle" class.

Examples

```
# Create a rec_unit_cell in default (no arguments)
ruc <- create_rec_unit_cell()
print(ruc)
```

`create_rec_unit_cell.bravais`
Reciprocal unit cell starting from a Bravais symbol

Description

Method to create a "rec_unit_cell" object starting from a "bravais" object. The Bravais symbols indicate the 14 possible Bravais lattices. A few examples are "aP", "oF", etc. The cell parameters assigned are assigned randomly, but are compatible with the Bravais lattice.

Usage

```
## S3 method for class 'bravais'
create_rec_unit_cell(ar, ...)
```

Arguments

ar	An object of class "bravais".
...	Additional arguments passed to the create_rec_unit_cell methods

Value

An object of class "rec_unit_cell". It is a named list of length 6 whose last three slots are of "angle" class.

Examples

```
# Create a "rec_unit_cell" object from a monoclinic primitive Bravais lattice
# Cell parameters generated automatically.
bt <- bravais("mP")
ruc <- create_rec_unit_cell(bt)
print(ruc)
```

---

create_rec_unit_cell.cryst_symm

Reciprocal unit cell from a 'cryst_symm' object

Description

Method to create an object of class "rec_unit_cell" starting from an object of class 'cryst_symm'.

Usage

```
## S3 method for class 'cryst_symm'
create_rec_unit_cell(ar, ...)
```

Arguments

ar	An object of class 'cryst_symm'.
...	Additional arguments passed to the create_rec_unit_cell.

Details

The symmetry of a space group imposes constrains on the parameters of unit cells. For example, the cubic group P 2 3 means that all cell sides have to be equal and all angles have to be equal to 90 degrees. This function suggests the appropriate reciprocal cell compatible with the given space group.

Value

An object of class "rec_unit_cell". It is a named list of length 6 whose last three slots are of class 'angle'. The cell parameters are calculated from those of the corresponding unit cell. The default unit cell parameters are a=10, b=20, c=15, alpha=70, beta=80, gamma=100. When constrains due to symmetry are required, b and c might be equaled to a, alpha, beta and gamma might be set to 90, gamma might be set to 120 and the three angles might be set equal to each other.

Examples

```
# Symmetry "C 1 2/c 1"
csym <- cryst_symm("C 1 2/c 1")

# Reciprocal unit_cell
ruc <- create_rec_unit_cell(csym)
print(ruc)
```

---

create_rec_unit_cell.default

Default method for generic "create_rec_unit_cell"

Description

This method is an alternative call to "rec_unit_cell" .

Usage

```
## Default S3 method:
create_rec_unit_cell(
  ar = NULL,
  br = NULL,
  cr = NULL,
  aar = NULL,
  bbr = NULL,
  ccr = NULL,
  ...
)
```

Arguments

ar	A real number. One of the reciprocal unit cell's side lengths, in 1/angstroms.
br	A real number. One of the reciprocal unit cell's side lengths, in 1/angstroms.
cr	A real number. One of the reciprocal unit cell's side lengths, in 1/angstroms.
aar	A real number. One of the reciprocal unit cell's angles, in degrees.
bbr	A real number. One of the reciprocal unit cell's angles, in degrees.
ccr	A real number. One of the reciprocal unit cell's angles, in degrees.
...	Additional arguments passed to the create_rec_unit_cell methods

Value

An object of class "rec_unit_cell". It is a named list of length 6 whose last three slots are of "angle" class.

See Also

rec_unit_cell

Examples

```
# Create a reciprocal cubic cell with side 1/50
ruc <- create_rec_unit_cell(1/50)
print(ruc)
```

---

create_rec_unit_cell.unit_cell

Reciprocal unit cell starting from a unit cell

Description

Method to create an object of class "rec_unit_cell" starting from an object of class "unit_cell".

Usage

```
## S3 method for class 'unit_cell'
create_rec_unit_cell(ar, ...)
```

Arguments

ar	An object of class "unit_cell".
...	Additional arguments passed to the create_rec_unit_cell methods

Value

An object of class "rec_unit_cell". It is a named list of length 6 whose last three slots are of "angle" class.

Examples

```
# Create a "rec_unit_cell" object starting from a cubic cell object
uc <- unit_cell()
print(uc)
ruc <- create_rec_unit_cell(uc)
print(ruc)
```

create_unit_cell *S3 generic to create unit_cell objects*

Description

The unit_cell object can be created starting from specific objects, files, etc.

Usage

```
create_unit_cell(a, ...)
```

Arguments

a	An object or objects used to select a method. These can be cell parameters, an object of class rec_unit_cell, etc.
...	Further arguments passed to or from other methods.

Value

An object of class "unit_cell". It is a named list of length 6 whose last three slots are of class "angle".

Examples

```
# Create a unit_cell in default (no arguments)
uc <- create_unit_cell()
print(uc)
```

create_unit_cell.bravais *Unit cell starting from a Bravais symbol*

Description

Method to create a "unit_cell" object starting from a "bravais" object. The Bravais symbols indicate the 14 possible Bravais lattices. A few examples are "aP", "oF", etc. The cell parameters assigned are assigned randomly, but are compatible with the Bravais lattice.

Usage

```
## S3 method for class 'bravais'
create_unit_cell(a, ...)
```

Arguments

a	An object of class "bravais".
...	Additional arguments passed to the create_unit_cell methods

Value

An object of class "unit_cell". It is a named list of length 6 whose last three slots are of "angle" class.

Examples

```
# Create a "unit_cell" object from a monoclinic primitive Bravais lattice
# Cell parameters generated automatically.
bt <- bravais("mP")
uc <- create_unit_cell(bt)
print(uc)
```

`create_unit_cell.cryst_symm` *Unit cell from a 'cryst_symm' object*

Description

Method to create an object of class "unit_cell" starting from an object of class 'cryst_symm'.

Usage

```
## S3 method for class 'cryst_symm'
create_unit_cell(a, ...)
```

Arguments

a	An object of class 'cryst_symm'.
...	Additional arguments passed to the create_unit_cell.

Details

The symmetry of a space group imposes constrains on the parameters of unit cells. For example, the cubic group P 2 3 means that all cell sides have to be equal and all angles have to be equal to 90 degrees. This function suggests the appropriate unit cell compatible with the given space group.

Value

An object of class "unit_cell". It is a named list of length 6 whose last three slots are of class 'angle'. Default cell parameters are a=10, b=20, c=15, alpha=70, beta=80, gamma=100. When constrains due to symmetry are required, b and c might be equaled to a, alpha, beta and gamma might be set to 90, gamma might be set to 120 and the three angles might be set equal to each other.

Examples

```
# Symmetry "C 1 2/c 1"
csym <- cryst_symm("C 1 2/c 1")

# Unit_cell
uc <- create_unit_cell(csym)
print(uc)
```

`create_unit_cell.default` *Default method for generic "create_unit_cell"*

Description

This method is an alternative call to "unit_cell" .

Usage

```
## Default S3 method:
create_unit_cell(
  a = NULL,
  b = NULL,
  c = NULL,
  aa = NULL,
  bb = NULL,
  cc = NULL,
  ...
)
```

Arguments

a	A real number. One of the unit cell's side lengths, in angstroms.
b	A real number. One of the unit cell's side lengths, in angstroms.
c	A real number. One of the unit cell's side lengths, in angstroms.
aa	A real number. One of the unit cell's angles, in degrees.
bb	A real number. One of the unit cell's angles, in degrees.
cc	A real number. One of the unit cell's angles, in degrees.
...	Additional arguments passed to the create_unit_cell methods

Value

An object of class "unit_cell". It is a named list of length 6 whose last three slots are of "angle" class.

See Also

unit_cell

Examples

```
# Create a cubic cell with side 50
uc <- create_unit_cell(50)
print(uc)
```

`create_unit_cell.rec_unit_cell` *Unit cell starting from a reciprocal unit cell*

Description

Method to create an object of class "unit_cell" starting from an object of class "rec_unit_cell".

Usage

```
## S3 method for class 'rec_unit_cell'
create_unit_cell(a, ...)
```

Arguments

a	An object of class "rec_unit_cell".
...	Additional arguments passed to the create_unit_cell methods

Value

An object of class "unit_cell". It is a named list of length 6 whose last three slots are of "angle" class.

Examples

```
# Create a "unit_cell" object starting from a reciprocal cubic cell object
ruc <- rec_unit_cell()
print(ruc)
uc <- create_unit_cell(ruc)
print(uc)
```

crystal_family *Crystal family corresponding to given space group.*

Description

Crystal family corresponding to given space group.

Usage

```
crystal_family(gn)
```

Arguments

gn	A natural integer (1,2,3,...). the space group number.

Value

A character string, the name of the crystal family associated with the given space group. If the input integer does not correspond any existing space group, the function returns NULL and throws a warning.

Examples

```
# P1 is part of the TRICLINIC family
crystal_family(1)

# The object returned is a string
cfam <- crystal_family(1)
class(cfam)
```

crystal_system *Crystal system corresponding to given space group.*

Description

Crystal system corresponding to given space group.

Usage

```
crystal_system(gn)
```

Arguments

gn	A natural integer (1,2,3,...). the space group number.

Value

A character string, the name of the crystal system associated with the given space group. If the input integer does not correspond any existing space group, the function returns NULL and throws a warning.

Examples

```
# P1 is part of the TRICLINIC system
crystal_system(1)

# The object returned is a string
csys <- crystal_system(1)
class(csys)
```

`cryst_symm` *Constructor for an S3 object of class "cryst_symm".*

Description

This represents a crystallographic space group.

Usage

```
cryst_symm(SG = NULL, set = NULL)
```

Arguments

`SG`	A character string or an integer identifying the space group. There are 230 used space group in crystallography and each one corresponds to a unique and so-called extended Hermann-Mauguin symbol. An example is space group number 19, identified by the extended Hermann-Mauguin symbol "P 21 21 21". Several formats are possible and some of them are now rarely used. Attempts are made to transform the input into a correct Hermann-Mauguin symbol, but if all fails, a warning is raised and the space group P 1 is assigned.
`set`	An integer defining which setting of many possible for the given space group. Some crystallographic space groups can be implemented with small variants known as "settings". If the input SG is an extended Hermann-Mauguin symbol, set is ignored, as it is already specified by the xHM symbol.

Details

The constructor can be used with less than the full set of two input parameters, to create an object of class cryst_symm corresponding to space group P 1. If the input string is not recognised, a warning is raised and space group P 1 is assigned.

Value

An object of class "cryst_symm". It is a named list of length 4. The names are, "SG", "PG", "T" and "C".

- 1) SG. This is a string containing the correct extended Hermann-Mauguin symbol.
- 2) PG. This is a list whose elements are all the 3×3 matrices forming the point-group part of the symmetry transformation.
- 3) T. This is a list whose elements are all the 3×1 vectors forming the translational part of the symmetry transformation.
- 4) C. This is a list whose elements are all the 3×1 vectors forming the centering of the unit cell.

Examples

```
# The symplest symmetry: P 1
crsym <- cryst_symm("P 1")
print(crsym)

# The second space group: P -1
crsym <- cryst_symm(2)
print(crsym)

# Something more complicated
crsym <- cryst_symm("P 21 21 21")
print(crsym)
```

---

deplete_systematic_absences

Deplete systematic absences

Description

Remove systematically-absent reflections from a data frame in which Miller indices are in the first three columns. The systematically-absent reflections are determined by the specific space group.

Usage

```
deplete_systematic_absences(hkl, SG)
```

Arguments

hkl	A data frame with first three columns H, K, L corresponding to the three Miller indices. This is normally the 'record' data frame in an object of class "merged_reflections".
SG	A character. The extended Hermann-Mauguin symbol of the crystallographic space group.

Details

Crystallography symmetry forces constraints on data from x-ray diffraction. One of these constraints consists in the full cancellation of reflections with certain Miller indices. It is said that the reflection with that specific Miller index is systematically absent. For example, in data corresponding to a crystal with space group C 2, general reflections like (h,k,l) must obey h+k=2n (even number). Thus, the Miller indices (2,3,1) are a systematic absence because 2+3=5 (odd).

Value

hkl The same data frame acquired from input, depleted of all systematic absences.

Examples

```
# C 2 monoclinic space group
SG <-"C 1 2 1"

# Create an arbitrary cell compatible with C 2
uc <- unit_cell(10,15,10,90,110,90)

# Crete the related reciprocal cell
ruc <- create_rec_unit_cell(uc)

# Create a full data frame of Miller indices
hkl <- expand.grid(H=-4:4,K=-4:4,L=-4:4)

# Get rid of systematic absences
new_hkl <- deplete_systematic_absences(hkl,SG)

# Compare first 10 items of original and depleted arrays
hkl[1:10,]
new_hkl[1:10,]
```

---

extract_symmetry_info *Information on a specific space group*

Description

Returns human-readable information on a specific input space group.

Usage

```
extract_symmetry_info(SG)
```

Arguments

SG	A character string. The extended Hermann-Mauguin symbol (e.g. 'P 1 1 21')

Details

Crystallographic symmetry is fundamental in crystallography. It affects the way atoms are arranged in a unit cell, the pattern of reflections in reciprocal space and many other common occurrences in crystallography. This function returns a named list with human-readable character strings which detail key symmetry information.

Value

infostring A named list with fields corresponding to those in the CCP4 symmetry library. The fields' name are:

- **NUMBER** standard spacegroup number
- **BASISOP** change of basis operator
- **CCP4** CCP4 spacegroup number e.g. 1003 (0 if not a CCP4 group)
- **HALL** Hall symbol
- **xHM** extended Hermann Mauguin symbol
- **OLD** CCP4 spacegroup name (blank if not a CCP4 group)
- **LAUE** Laue group symbol
- **PATT** Patterson group symbol
- **PGRP** Point group symbol
- **HKLASU** reciprocal space asymmetric unit (with respect to standard setting)
- **MAPASU_CCP4** CCP4 real space asymmetric unit (with respect to standard setting. Negative ranges if not a CCP4 group)
- **MAPASU_ZERO** origin based real space asymmetric unit (with respect to current setting)
- **MAPASU_NONZ** non-origin based real space asymmetric uni (with respect to current setting)
- **CHESHIRE** Cheshire cell (with respect to standard setting)
- **SYMOP** list of primitive symmetry operators
- **CENOP** list of centering operators

Examples

```
# This is the full information for space group number 19, P 21 21 21
SG <- translate_SG(19)
ltmp <- extract_symmetry_info(SG)
ltmp
```

findHM *Correct spelling for Herman-Mauguin space groups symbols*

Description

The commonly-used spelling of a crystallographic space group does not match the correct definition given by the Herman-Mauguin symbols which define all space groups in a unique and precise way. This function attempt to translate a tentative string into a possible Herman-Mauguin symbol, if it finds one. If the input string is already in the extended Herman-Mauguin form, the same string is returned as output.

Usage

```
findHM(sym_xHM)
```

Arguments

sym_xHM	A character string. The space group symbol in its commonly-used spelling.

Value

SG A character string. The extended Hermann-Mauguin symbol (e.g. 'P 1 1 21').

Examples

```
# P21
print(find("P 21"))

# P -1
print(find("P-1"))
```

find_symm_setting *Find specific space group setting*

Description

Although a space group is uniquely defined, i.e. the symmetry operations defining it are uniquely given, the choice of vectors that defines a unit cell for that symmetry is not unique. The different choices are known as settings. Most space groups have only one setting, but it is possible to find space groups with several settings. For example, "C 1 2/c 1" has 18 settings. The xHM symbol for setting 1 is "C 1 2/c 1", the symbol for setting 2 is "A 1 2/n 1", etc.

Usage

```
find_symm_setting(SG)
```

Arguments

SG	A character string indicating the extended Hermann-Mauguin symbol for the space group.

Value

set An integer equal to the specific setting corresponding to the given xHM symbol.

Examples

```
# P2 (group number 4) has three settings
nsets <- find_symm_setting("P 1 2 1")
print(nsets)
```

---

frac_to_orth *From fractional to orthogonal coordinates*

Description

This function transforms any number of fractional coordinates (x_f, y_f, z_f), arranged as a vector or in a matrix or data frame, into the corresponding number of orthogonal coordinates (x, y, z), arranged in the same format.

1. ochoice = 1: X axis along a; Y axis normal to a, in the (a,b) plane; Z axis normal to X and Y (and therefore parallel to c*).
2. ochoice = 2: this is also called "Cambridge setting". The X axis is along a*; the Y axis lies in the (a*,b*) plane; the Z axis is, consequently, along c.

Usage

```
frac_to_orth(xyzf, a, b, c, aa, bb, cc, ochoice = 1)
```

Arguments

xyzf	A vector or $n \times 3$ matrix or data frame of fractional crystal coordinates.
a	A real number. One of the unit cell's side lengths, in angstroms.
b	A real number. One of the unit cell's side lengths, in angstroms.
c	A real number. One of the unit cell's side lengths, in angstroms.
aa	A real number. One of the unit cell's angles, in degrees.
bb	A real number. One of the unit cell's angles, in degrees.
cc	A real number. One of the unit cell's angles, in degrees.
ochoice	A natural integer indicating the choice of orthogonal transformation. 1 corresponds to the first choice and 2 to the second choice in Giacovazzo's book (see xtal_mat01 and xtal_mat02).

Value

A $n \times 3$ matrix or data frame of orthogonal coordinates corresponding to the fractional coordinates provided in the input.

Examples

```
# Matrix containing 3 fractional coordinates
xyzf <- matrix(c(0.1,0.2,0.3,0.2,0.6,0.7,0.15,0.28,0.55),ncol=3,byrow=TRUE)

# Cartesian coordinates
xyz <- frac_to_orth(xyzf,10,30,20,90,90,90,1)
```

---

full_symm_strings *Symmetry operations in human readable form*

Description

This function returns the full set of symmetry operations in human-readable form, each one as a character string starting with 'SYMM'. These are the common crystallographic symmetry operations.

Usage

```
full_symm_strings(SG)
```

Arguments

SG	A character string. The extended Hermann-Mauguin symbol (e.g. 'P 1 1 21')

Value

Symm_string A character vector whose components are strings starting by 'SYMM' and containing the symmetry operations of the given group in human-readable form.

Examples

```
# P1 has only one symmetry operation
SG <- "P 1"
symm_string <- full_symm_strings(SG)
print(symm_string)

# P 21 21 21 is has many more operations
SG <- "P 21 21 21"
symm_string <- full_symm_strings(SG)
print(symm_string)
```

generate_miller *Generate Miller indices*

Description

Function to create a data frame with complete set of Miller indices, up to a given resolution (in angstroms).

Usage

```
generate_miller(uc, SG, reso)
```

Arguments

uc	An object of class "unit_cell".
SG	A character string or a number indicating the extended Hermann-Mauguin symbol for the space group.
reso	A real number. The highest data resolution, in angstroms.

Details

Miller indices are named H, K, L in the data frame. Only values of (H,K,L) corresponding to a resolution d(h,k,l) >= reso (in angstroms), are included. The full list does not include systematic absences corresponding to the specific symmetry of the crystal.

Value

hkl A data frame with columns H, K, L corresponding to the three Miller indices, and a columns S corresponding to their inverse resolutions (in angstroms).

Examples

```
# C 2 monoclinic space group
SG <- "C 1 2 1"

# Create an arbitrary cell compatible with C 2
uc <- unit_cell(10,15,10,90,110,90)

# Generate Miller indices to 5 angstroms resolution
reso <- 5
hkl <- generate_miller(uc,SG,reso)

# Display first 10 indices
hkl[1:10,]
```

hkl_to_reso *Calculates resolution, given the Miller indices*

Description

Calculates resolution, given the Miller indices

Usage

```
hkl_to_reso(h, k, l, a, b, c, aa, bb, cc)
```

Arguments

h	An integer, A Miller index.
k	An integer, A Miller index.
l	An integer, A Miller index.
a	A real number. One of the unit cell's side lengths, in angstroms.
b	A real number. One of the unit cell's side lengths, in angstroms.
c	A real number. One of the unit cell's side lengths, in angstroms.
aa	A real number. One of the unit cell's angles, in degrees.
bb	A real number. One of the unit cell's angles, in degrees.
cc	A real number. One of the unit cell's angles, in degrees.

Value

A positive, real number. The resolution associated with (h,k,l), in angstroms.

Examples

```
datadir <- system.file("extdata",package="cry")
fname <- file.path(datadir,"1dei_phases.mtz")
hdr <- readMTZHeader(fname,message=FALSE)
ucell <- hdr$CELL
reso1 <- hkl_to_reso(1,0,0,ucell[1],ucell[2],ucell[3],ucell[4],ucell[5],ucell[6])
print(reso1)  # Low resolution
reso2 <- hkl_to_reso(20,20,20,ucell[1],ucell[2],ucell[3],ucell[4],ucell[5],ucell[6])
reso2  # High resolution
```

`lattice_stuff` *Calculation of useful lattice parameters*

Description

Calculation of useful lattice parameters

Usage

```
lattice_stuff(a, b, c, aa, bb, cc)
```

Arguments

`a`	A real number. One of the unit cell's side lengths, in angstroms.
`b`	A real number. One of the unit cell's side lengths, in angstroms.
`c`	A real number. One of the unit cell's side lengths, in angstroms.
`aa`	A real number. One of the unit cell's angles, in degrees.
`bb`	A real number. One of the unit cell's angles, in degrees.
`cc`	A real number. One of the unit cell's angles, in degrees.

Value

A named vector of real numbers and length 16. The names are:

- sa. Sine(aa)
- sb. Sine(bb)
- sc. Sine(cc)
- ca. Cosine(aa)
- cb. Cosine(bb)
- cc. Cosine(cc)
- ar. a* (reciprocal cell side length)
- br. b* (reciprocal cell side length)
- cr. c* (reciprocal cell side length)
- sar. Sine of a reciprocal cell angle
- sbr. Sine of a reciprocal cell angle
- scr. Sine of a reciprocal cell angle
- car. Cosine of a reciprocal cell angle
- cbr. Cosine of a reciprocal cell angle
- ccr. Cosine of a reciprocal cell angle
- V. Volume of the unit cell in cubic angstroms

Examples

```
datadir <- system.file("extdata",package="cry")
fname <- file.path(datadir,"1dei_phases.mtz")
hdr <- readMTZHeader(fname,message=FALSE)
ucell <- hdr$CELL
vtmp <- lattice_stuff(ucell[1],ucell[2],ucell[3],ucell[4],ucell[5],ucell[6])
vtmp[1:3]
vtmp[4:6]
vtmp[7:9]
vtmp[10:12]
vtmp[13:15]
vtmp[16]
```

---

lowest_uc_compatible_SG

Space group compatible with given cell

Description

This function returns the space group, among those compatible with the given unit cell, with the lowest symmetry group number.

Usage

```
lowest_uc_compatible_SG(uc)
```

Arguments

uc	An object of class 'unit_cell'.

Details

A given unit cell is compatible with several symmetries. For example, a cell with different sides and different angles, also different from 90 degrees, is compatible with the triclinic lattice system, which corresponds to space groups P1 and P -1. A cell with different sides, alpha = 90, gamma=90 and beta different from 90, is compatible with the monoclinic lattice system, which corresponds to space groups from number 3 ("P 1 2 1") to number 15 (C 1 2/c 1"). In the first case this function returns "P 1", while in the second case it returns "P 1 2 1".

Value

csym An object of class 'cryst_symm', corresponding to the selected, lowest symmetry.

Examples

```
# Monoclinic cell
uc <- unit_cell(10,20,15,90,110,90)

# The selected space group is "P 1 2 1"
csym <- lowest_uc_compatible_SG(uc)
print(csym)
```

`merged_reflections` *Constructor for an S3 object of class "merged_reflections".*

Description

This represents scaled and merged x-ray data from one crystal.

Usage

```
merged_reflections(ruc = NULL, csym = NULL, records = NULL, dtypes = NULL)
```

Arguments

`ruc`	An object of class "rec_unit_cell" (which represents a reciprocal unit cell).
`csym`	An object of class "cryst_symm" (which represents a crystallographic symmetry group).
`records`	A data frame containing all reflections coming from the x-ray data collection on the crystal. This data frame must include at least the three Miller indices, H, K, L (of dtype "H").
`dtypes`	A character vector whose length is the same as the number of columns in 'records'. One character (a capital letter) is associated with each type of data. For example, a Miller index is of dtype "H"; a structure amplitude is of dtype "F"; an anomalous difference is of dtype "D"; etc (see details later).

Details

If the constructor is used without arguments, the default object created will be create reflections for a cubic crystal with cell of side 10 angstroms, and symmetry P 2 3, up to 5 angstroms resolution. The only available columns will be of dtype "H", named H, K, L (the Miller indices), and of dtype "S", inverse resoluton, named S.

The possible dtypes are:

H Miller index

S Inverse resolution (1/angstroms)

J Reflection intensity

F Amplitude of a structure factor

D Anomalous difference

Q Standard deviation of J, F, D

G Amplitude associated with a Friedel pair (F(+), F(-))

L Standard deviation of G

K Intensity associated with G (I(+), I(-))

M Standard deviation of K

E Amplitude of the normalised structure factors

P Phase angle (in degrees)

W Weight of some sort

A Phase probability coefficients (Hendrickson-Lattman)

B Batch number (from raw data)

I Any other integer

R Any other real

More values can become available in a future release.

Value

An object of class "merged_reflections". It is a named list of length 4 whose names are:

ruc An object of class "rec_unit_cell".

csym An object of class "cryst_symm".

records A data frame containing the data.

dtypes A character vector containing the type of data (Miller indices, structure factors, etc).

Examples

```
# Create an orthorombic (default) cell
uc <- unit_cell(10,30,15)

# Create the related reciprocal cell
ruc <- create_rec_unit_cell(uc)

# Create symmetry (P n c 2)
csym <- cryst_symm(30)

# Create a few records (these include syst. absences)
records <- expand.grid(H=-2:2,K=-2:2,L=-2:2)
print(length(records[,1]))

# dtypes are all H
dtypes <- c("H","H","H")

# Create merged_reflections object with H, K, L
# Systematic absences have been eliminated
mrefs <- merged_reflections(ruc,csym,records,dtypes)
print(length(mrefs$records[,1]))
```

num_symm_settings *Number of space group settings*

Description

Although a space group is uniquely defined, i.e. the symmetry operations defining it is uniquely given, the choice of vectors that defines a unit cell for that symmetry is not unique. The different choices are known as settings. Most space groups have only one setting, but it is possible to find space groups with several settings. For example, "C 1 2/c 1" has 18 settings. While the xHM symbol for setting 1 is "C 1 2/c 1", the symbol for setting 2 is "A 1 2/n 1", etc.

Usage

```
num_symm_settings(SG = NULL)
```

Arguments

SG	A character string or a number indicating the space group.

Value

nsett The number of setting for the given space group.

Examples

```
# P 1 21 1 (group number 4) has three settings
num_symm_settings(4)

# Find the extended Hermann-Mauguin symbols
translate_SG(4,"number","xHM",1)$msg
translate_SG(4,"number","xHM",2)$msg
translate_SG(4,"number","xHM",3)$msg
```

op_xyz_list_to_matrix_list
List of matrices and vectors of a specific space group

Description

Returns 3×3 matrices and 3×1 vectors corresponding to point group operations, group translations and cell centring of a given space group.

Usage

```
op_xyz_list_to_matrix_list(op_xyz_list)
```

Arguments

op_xyz_list	A named list made of two vectors. The first vector, SYMOP, contains strings describing the symmetry operators. The second vector, CENOP, contains strings describing the centring of the unit cell.

Details

A crystallographic space group consists of a series of transformations on a point (x_f, y_f, z_f) in space that are mathematically implemented as the product of a 3×3 point-group matrix and the point fractional coordinates, (x_f, y_f, z_f), followed by a sum with a 3×1 translation vector. The complete set of points thus produced can be cloned into a new and shifted set translated of an amount represented by a 3×1 centring vector.

Value

mat_ops_list A named list consisting of 3 lists. The first list, PG, contains 3×3 point group matrices; the second list, T, contains the same number of 3×1 translation vectors. The first matrix is always the identity matrix, the first translation vector is always the null vector. The third list, C, consists of centering vectors; the first centering vector is always the null vector. To summarize, the output looks like the following:

[[[[I,M2,M3,...,Mn]] , [[O,V2,V3,...,Vn]] , [[O,C2,C3,...,Cm]]]] where: I = identity 3X3 matrix 0 = null 3X1 vector M2,M3,...,Mn = point group 3X3 matrices V2,V3,...,Cn = translation 3X1 vectors C2,C3,...,Cm = centering 3X1 vectors

Examples

```
# Symmetry operators for space group number 3, P 1 2 1
SG <- "P 1 2 1"
op_xyz_list <- syminfo_to_op_xyz_list(SG)
mat_ops_list <- op_xyz_list_to_matrix_list(op_xyz_list)
names(mat_ops_list)
```

---

op_xyz_to_matrix *Human-readable symmetry operator into matrix and vector*

Description

Returns a 3×3 matrix and 3×1 vector corresponding to either a symmetry operator or a centering operator in human-readable, string form.

Usage

```
op_xyz_to_matrix(op_xyz)
```

Arguments

op_xyz	A symmetry or centering operation in the form of a human-readable string, e.g. '-x+1/2,-y,z+1/2'.

Details

A string describing a symmetry or a centering operation has a format similar to, for instance, '-x+1/2,-y,z+1/2'. Such a string corresponds to the symmetry operation represented mathematically by the following matrix and vector:

$$\begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad , \quad \begin{pmatrix} 1/2 \\ 0 \\ 1/2 \end{pmatrix}$$

Where symmetry operations in human-readable form are useful for the subjective reasoning in crystallography, their mathematical counterpart is needed for all practical calculations.

Value

mat_ops A named list including a 3×3 matrix 'R' and a 3×1 vector 'T'.

Examples

```
# Reflection and translation
sop <- '-x,y+1/2,z'
mat_ops <- op_xyz_to_matrix(sop)
print(mat_ops)
```

---

orth_to_frac *From orthogonal to fractional coordinates*

Description

This function transforms any number of orthogonal coordinates (x, y, z), arranged as a vector or in a matrix or data frame, into the corresponding number of fractional coordinates (x_f, y_f, z_f), arranged in the same format.

1. ochoice = 1: X axis along a; Y axis normal to a, in the (a,b) plane; Z axis normal to X and Y (and therefore parallel to c*).
2. ochoice = 2: this is also called "Cambridge setting". The X axis is along a*; the Y axis lies in the (a*,b*) plane; the Z axis is, consequently, along c.

Usage

```
orth_to_frac(xyz, a, b, c, aa, bb, cc, ochoice = 1)
```

Arguments

xyz	A vector or $n \times 3$ matrix or data frame of orthogonal crystal coordinates.
a	A real number. One of the unit cell's side lengths, in angstroms.
b	A real number. One of the unit cell's side lengths, in angstroms.
c	A real number. One of the unit cell's side lengths, in angstroms.
aa	A real number. One of the unit cell's angles, in degrees.
bb	A real number. One of the unit cell's angles, in degrees.
cc	A real number. One of the unit cell's angles, in degrees.
ochoice	A natural integer indicating the choice of orthogonal transformation. 1 corresponds to the first choice and 2 to the second choice in Giacovazzo's book (see xtal_mat01 and xtal_mat02).

Value

A $n \times 3$ matrix or data frame of fractional coordinates corresponding to the orthogonal coordinates provided in the input.

Examples

```
# Matrix containing 3 orthogonal coordinates
xyz <- matrix(c(5, 15, 10, 2, 10, 8, 1, 1, 1),ncol=3,byrow=TRUE)

# Fractional coordinates
xyzf <- orth_to_frac(xyz,10,30,20,90,90,90,1)
```

---

print.angle *Print method for an object of class "angle".*

Description

The value is displayed in degrees

Usage

```
## S3 method for class 'angle'
print(x, ...)
```

Arguments

x	An object of class "angle".
...	Additional arguments passed to the print methods

Value

No values. A message is displayed which includes information on the angle.

Examples

```
# Create an angle of 90 degrees using radians
ang1 <- angle(pi/2,FALSE)

# Display its value
print(ang1)
```

`print.bravais` *Print method for an object of class "bravais".*

Description

The Bravais lattice and related crystal family, crystal system and lattice system are displayed.

Usage

```
## S3 method for class 'bravais'
print(x, ...)
```

Arguments

`x`	An object of class "bravais".
`...`	Additional arguments passed to the print methods

Value

No values. A message is displayed which includes information on the Bravais lattice.

Examples

```
# Create a triclinic Bravais lattice
bt <- bravais()

# Display its value
print(bt)
```

print.cryst_symm *Print method for an object of class "cryst_symm".*

Description

xxx

Usage

```
## S3 method for class 'cryst_symm'
print(x, ...)
```

Arguments

x	An object of class "cryst_symm".
...	Additional arguments passed to the print methods

Value

No values. A message is displayed which includes information on the crystallographic symmetry.

Examples

```
# Create an object of P 2 symmetry
crsym <- cryst_symm("P 2")

# Display its value
print(crsym)
```

print.rec_unit_cell *Print method for an object of class "rec_unit_cell".*

Description

The values are displayed in 1/angstroms and degrees

Usage

```
## S3 method for class 'rec_unit_cell'
print(x, ...)
```

Arguments

x	An object of class "rec_unit_cell".
...	Additional arguments passed to the print methods

Value

No values. A message is displayed which includes information on the reciprocal unit cell.

Examples

```
# Create a cubic reciprocal unit cell
ruc <- rec_unit_cell(1/10,1/10,1/10,90,90,90)

# Display its value
print(ruc)
```

---

print.unit_cell *Print method for an object of class "unit_cell".*

Description

The values are displayed in angstroms and degrees

The output includes details on the unit cell, the crystal symmetry and the first 10 records (data).

Usage

```
## S3 method for class 'unit_cell'
print(x, ...)

## S3 method for class 'merged_reflections'
print(x, ...)
```

Arguments

x	An object of class "merged_reflections".
...	Additional arguments passed to the print methods

Value

No values. A message is displayed which includes information on the unit cell.

No values. A message is displayed which includes information on the reflections contained in this object and the crystal structure they relate to.

Examples

```
# Create a cubic unit cell
uc <- unit_cell(10,10,10,90,90,90)

# Display its value
print(uc)
# Create a default 'merged_reflections' object
mrefs <- merged_reflections()
```

```
# Display its value
print(mrefs)
```

---

readCIF *Reads and output an CIF file*

Description

Reads and output an CIF file

Usage

```
readCIF(filename, message = FALSE)
```

Arguments

filename	A character string. The path to a valid CIF file.
message	A logical variable. If TRUE (default) the function prints a message highlighting what is included in the cif file.

Value

A named list. Each name correspond to a valid field in the cif.

Examples

```
datadir <- system.file("extdata",package="cry")
filename <- file.path(datadir,"AMS_DATA.cif")
lCIF <- readCIF(filename)
print(names(lCIF))
print(lCIF$INTRO$CELL)
print(lCIF$INTRO$HALL)
print(lCIF$INTRO$HM)
print(lCIF$SYMM)
```

readMTZ *Load an MTZ file*

Description

Reads mtz files and store both header information and reflection data records in named lists. A third list is used, if the mtz file is an unmerged file, for storing batch headers.

Usage

```
readMTZ(filename, message = FALSE)
```

Arguments

filename	A character string. The path to a valid mtz file.
message	A logical variable. If TRUE the function prints a message highlighting data included in the mtz file. Default value is message=FALSE.

Value

A named list of length 3. The first element is called "reflections" and is a dataframe with as many columns as are included in the mtz file. The name of each column of the dataframe coincides with the name of the corresponding column in the mtz. The second element is called "header" and is a named list in which each name correspond to a valid field in the mtz header (see details in readMTZHeader). The third element is called "batch_header" and is a list with as many elements as the number of batches (images) included in the mtz file. Each list element is, itself, a named list including all the useful variables stored in batch headers. If no batch headers are contained in the file (merged mtz), the batch_header element is NULL.

Examples

```
datadir <- system.file("extdata",package="cry")
filename <- file.path(datadir,"1dei_phases.mtz")
ltmp <- readMTZ(filename)
print(names(ltmp))
print(class(ltmp$reflections))
str(ltmp$reflections)
print(class(ltmp$header))
print(class(ltmp$batch_header))

refs <- ltmp$reflections
print(colnames(refs))
print(range(refs$H))
```

`readMTZHeader` *Reads and output an MTZ header*

Description

An MTZ file is a binary file created to carry information on x-ray diffraction experiments on crystals. It includes x-ray diffraction data and information on the experiment and the crystal.

Usage

```
readMTZHeader(filename, message = FALSE)
```

Arguments

`filename`	A character string. The path to a valid mtz file.
`message`	A logical variable. If TRUE the function prints a message highlighting what is included in the mtz header. Default value is `message=FALSE`.

Details

The function returns a named list whose components are the `reflections`, the `header` and the `batch_header`. The `header` is a named list whose components are:

TITLE A character string containing the title of the MTZ file.

NCOL Number of columns in data frame `reflections`.

CELL A numeric vector of length 6, containing the unit cell parameters.

SORT An integer vector of length 5, containing the sort order of the first 5 columns of data.

SYMINF Un-named list with 6 components: the number of symmetry operations (an integer), the number of primitive operations (an integer), the lattice type (a one-letter character), the space group number (an integer), the space group name (a 10-letter character string) and the point group name (a 6-letter character).

RESO Minimum and maximum data resolution, stored as $1/d^2$.

NDIF Number of datasets whose reflection data are present in the file.

SYMM A character vector whose length depends on the type of symmetry. It describes the symmetry operations in human-readable format, International Tables style. Each string is 80 characters long.

PROJECT A data frame whose rows provide an ID and a name (called "pname") for the projects for which the data contained have been produced.

CRYSTAL A data frame whose rows provide an ID and a name (called "pname") for the crystals for which the data contained have been produced.

DATASET A data frame whose rows provide an ID and a name (called "pname") for the datasets included in the reflections record.

DCELL A data frame whose rows contain the `CRYSTAL` IDs and cell parameters of each crystal that has contributed to the data in the reflections record.

DWAVEL A data frame whose rows contain the DATASET IDs and the wavelength with which the reflection data were collected.

COLUMN A data frame describing the type of data included in the reflections record. The data frame includes the labels for each column, the dtype (see merged_reflections) for each column, min and max values and the DATASET ID.

COLSRC A data frame with three columns. The first includes the labels of each reflections record column. The second includes a time stamp of when each data column was created. The third is the dataset ID as a string.

COLGRP A character string vector where each component is an 80-letters string describing the name and type of data.

HISTORY A character string vector of variable length. Each component is an 80-letter string summarising the steps that lead to the current reflections record. HISTORY can contain a maximum of 30 lines.

Value

A named list. Each name correspond to a valid field in the mtz header (see details).

Examples

```
datadir <- system.file("extdata",package="cry")
filename <- file.path(datadir,"1dei_phases.mtz")
ltmp <- readMTZHeader(filename)
print(names(ltmp))
print(ltmp$CELL)
print(ltmp$SYMM)
```

---

readpd_rtv *Reads and output an CIF file*

Description

Reads and output an CIF file

Usage

```
readpd_rtv(filename, messages = FALSE)
```

Arguments

filename	A character string. The path to a valid CIF file.
messages	A logical variable. If TRUE (default) the function prints a message highlighting what is included in the cif file.

Value

A named list. Each name correspond to a valid field in the powder diffraction Rietveld processed CIF.

Examples

```
datadir <- system.file("extdata",package="cry")
filename <- file.path(datadir,"e-65-00i60-Isup2.rtv")
lCIF <- readpd_rtv(filename)
print(names(lCIF))
print(lCIF$INTRO$CELL)
print(lCIF$INTRO$HALL)
print(lCIF$INTRO$HM)
print(lCIF$REFL)
```

---

readSF_CIF *Reads and output an CIF file*

Description

Reads and output an CIF file

Usage

```
readSF_CIF(filename, message = FALSE)
```

Arguments

filename	A character string. The path to a valid CIF file.
message	A logical variable. If TRUE (default) the function prints a message highlighting what is included in the cif file.

Value

A named list. Each name correspond to a valid field in the SF cif.

Examples

```
datadir <- system.file("extdata",package="cry")
filename <- file.path(datadir,"1dei-sf.cif")
lCIF <- readSF_CIF(filename)
print(names(lCIF))
print(lCIF$INTRO$CELL)
print(lCIF$INTRO$HALL)
print(lCIF$INTRO$HM)
print(lCIF$REFL)
```

readSHELXlog *Reads and SHELXD log files*

Description

Reads and SHELXD log files

Usage

```
readSHELXlog(filename)
```

Arguments

filename	A character string. The path to a valid log file.

Value

A named list. Each name correspond to a valid field in the log header.

Examples

```
datadir <- system.file("extdata",package="cry")
filename <- file.path(datadir,"shelxc.log")
ltmp <- readSHELXlog(filename)
print(names(ltmp))
```

readXDS_ASCII *Load an XDS_ASCII file.*

Description

Function to load XDS_ASCII.HKL files into a named list with three components called *processing_info*, *header* and *reflections* (see details further down).

Usage

```
readXDS_ASCII(filename, message = FALSE)
```

Arguments

filename	A character string. The path to a valid XDS ASCII file.
message	A logical variable. If TRUE (default) the function prints a message highlighting what is included in the xds header. If filename is not a valid XDS ascii file, the function returns 'NULL' and prints out a warning message.

Details

This function reads in all data from an XDS_ASCII data file and organises them into a named list. The list's name are:

processing_info This list component includes three logical variables, MERGE, FRIEDEL and PROFILE. Their TRUE/FALSE value reflect features of the XDS_ASCII file connected with the specific processing performed to obtain the file itself (for more details see https://xds.mr.mpg.de/).

header This list includes several components, like for instance SPACE_GROUP_NUMBER or UNIT_CELL_CONSTANTS, which give informations on the crystal and the experiment generating the data.

reflections This data.frame includes the actual experimental data, i.e. the observations collected during the X-ray diffraction experiment on the crystal (or crystals). The number and type of columns can vary.

Value

A named list (see details).

Examples

```
# Load one of the XDS ASCII files included with
# this distribution of cry
datadir <- system.file("extdata",package="cry")
filename <- file.path(datadir,"xds00_ascii.hkl")
ltmp <- readXDS_ASCII(filename,message=FALSE)
print(names(ltmp))
print(ltmp$reflections[1:5,])
```

---

readXDS_ASCIIHeader *Load an XDS_ASCII file's header.*

Description

This function reads information from the header of an XDS_ASCII.HKL data file and organises it into a named list with a variable number of components, according to the type of XDS_ASCII.HKL file (see details in readXDS_ASCII).

Usage

```
readXDS_ASCIIHeader(filename)
```

Arguments

filename	A character string. The path to a valid XDS ASCII file.

Value

A named list. Each name correspond to a valid field in the xds header. If `filename` is not a valid XDS ascii file, the function returns 'NULL' and prints out a warning message.

Examples

```
# Load one of the XDS ASCII files included with
# this distribution of cry
datadir <- system.file("extdata",package="cry")
filename <- file.path(datadir,"xds00_ascii.hkl")
ltmp <- readXDS_ASCIIHeader(filename)
print(names(ltmp))
```

---

rec_unit_cell *Constructor for an S3 object of class "rec_unit_cell.*

Description

This represents a crystal reciprocal unit cell.

Usage

```
rec_unit_cell(
  ar = NULL,
  br = NULL,
  cr = NULL,
  aar = NULL,
  bbr = NULL,
  ccr = NULL
)
```

Arguments

ar	A real number. One of the reciprocal unit cell's side lengths, in 1/angstroms.
br	A real number. One of the reciprocal unit cell's side lengths, in 1/angstroms.
cr	A real number. One of the reciprocal unit cell's side lengths, in 1/angstroms.
aar	A real number. One of the reciprocal unit cell's angles, in degrees.
bbr	A real number. One of the reciprocal unit cell's angles, in degrees.
ccr	A real number. One of the reciprocal unit cell's angles, in degrees.

Details

The constructor can be used with less than the full set of six input parameters, to create reciprocal unit cells for the cubic, tetragonal and orthogonal systems. Objects of "rec_unit_cell" class can also be created with no parameters (default to a reciprocal cubic cell of side length 0.1 1/angstroms).

Value

An object of class "rec_unit_cell". It is a named list of length 6 whose last three slots are of "angle" class.

Examples

```
# Create a monoclinic reciprocal unit cell
ruc <- unit_cell(0.115,0.033,0.077,90,120,90)
print(ruc)

# Create a cubic cell (default)
ruc <- rec_unit_cell()
print(ruc)

# Create a reciprocal cubic cell with side 1/20
ruc <- rec_unit_cell(1/20)
print(ruc)

# Create a reciprocal tetragonal unit cell with sides 1/20 and 1/60
ruc <- rec_unit_cell(1/20,1/60)
print(ruc)

# Create a reciprocal orthogonal unit cell
ruc <- rec_unit_cell(1/40,1/15,1/30)
print(ruc)
```

---

syminfo_to_matrix_list

Operators of a specific space group in matrix form

Description

Returns 3×3 matrices and 3×1 vectors corresponding to point group operations, group translations and cell centring of a given space group.

Usage

```
syminfo_to_matrix_list(SG)
```

Arguments

SG	A character string. The extended Hermann-Mauguin symbol (e.g. 'P 1 1 21')

Details

A crystallographic space group consists of a series of transformations on a point (x_f, y_f, z_f) in space that are mathematically implemented as the product of a 3×3 point-group matrix and the point fractional coordinates, (x_f, y_f, z_f), followed by a sum with a 3×1 translation vector. The complete set of points thus produced can be cloned into a new and shifted set translated of an amount represented by a 3×1 centring vector.

Value

mat_ops_list A named list consisting of 3 lists. The first list, PG, contains 3×3 point group matrices; the second list, T, contains the same number of 3×1 translation vectors. The first matrix is always the identity matrix, the first translation vector is always the null vector. The third list, C, consists of centering vectors; the first centering vector is always the null vector. To summarize, the output looks like the following:

[[[[I,M2,M3,...,Mn]] , [[O,V2,V3,...,Vn]] , [[O,C2,C3,...,Cm]]]] where: I = identity 3X3 matrix 0 = null 3X1 vector M2,M3,...,Mn = point group 3X3 matrices V2,V3,...,Cn = translation 3X1 vectors C2,C3,...,Cm = centering 3X1 vectors

Examples

```
# Symmetry operators for space group number 4, P 1 21 1
SG <- "P 1 21 1"
mat_ops <- syminfo_to_matrix_list(SG)
print(mat_ops)
```

---

syminfo_to_op_xyz_list

Operators of a specific space group

Description

Returns human-readable symmetry operators corresponding to a specific input space group.

Usage

```
syminfo_to_op_xyz_list(SG)
```

Arguments

SG	A character string. The extended Hermann-Mauguin symbol (e.g. 'P 1 1 21')

Details

A crystallographic space group includes a set of symmetry operators that can be expressed like operations on the (x,y,z) fractional coordinates of atoms in a unit cell. So, for example, The only operator associated with the space group P 1 is "x,y,z", while the four operators associated with P 21 21 21 are "symop x,y,z", "symop -x+1/2,-y,z+1/2", "symop x+1/2,-y+1/2,-z", "symop -x,y+1/2,-z+1/2".

Value

op_xyz_list A named list made of two vectors. The first vector, SYMOP, contains strings describing the symmetry operators. The second vector, CENOP, contains strings describing the centring of the unit cell.

Examples

```
# Symmetry operators for space group number 3, P 1 2 1
SG <- "P 1 2 1"
ltmp <- syminfo_to_op_xyz_list(SG)
ltmp
```

`symm_to_cell_const` *Cell parameter constrains from symmetry*

Description

This function returns a set of constrains, as string character expressions, imposed by the specific symmetry group on the given unit cell.

Usage

```
symm_to_cell_const(SG)
```

Arguments

`SG`	A character string indicating the extended Hermann-Mauguin symbol for the space group.

Details

Space group symmetry imposes certain constraints on the values that unit cell parameters can take. For example, the symmetry represented by the monoclinic space group of extended Hermann-Mauguin symbol "P 1 2 1" is compatible with a unit cell in which alpha=gamma=90.

There is just a handful of constrains for unit cells. Here they are indicated with the following set of specific strings:

- **'No constrains'** Like in a triclinic cell.
- **'alpha=90'** The alpha angle is fixed at 90 degrees.
- **'beta=90'** The beta angle is fixed at 90 degrees.
- **'gamma=90'** The gamma angle is fixed at 90 degrees.
- **'gamma=120'** The gamma angle is fixed at 120 degrees.
- **'alpha=beta=gamma'** The three angle have the same value, different from 90 degrees.
- **'a=b'** Cell side a is equal to cell side b.
- **'a=b=c'** The three cell sides are equal.

Value

vcons A character vector. Each component is a string, like 'alpha=90' or 'a=b', that describes the type of constrain to be applied to a unit cell of a crystal structure with given space group symmetry (see above).

Examples

```
# P 1 1 2 (group number 3) corresponds to setting 2
SG <- translate_SG(3,set=2)

# Constrains for this symmetry
stmp <- symm_to_cell_const(SG)
print(stmp)

# R 3 (rombohedral setting)
stmp <- symm_to_cell_const("R 3 :R")
print(stmp)
```

---

sysabs *Locate systematic absences*

Description

Given an mX3 matrix of Miller indices, this function returns those indices corresponding to valid reflections, i.e. to reflections which are not systematic absences.

Usage

```
sysabs(hkl, SG)
```

Arguments

hkl	An mX3 matrix or a data frame whose rows are the three integers corresponding to the Miller indices.
SG	A character. The extended Hermann-Mauguin symbol of the crystallographic space group.

Details

Crystallography symmetry forces constraints on data from x-ray diffraction. One of these constraints consists in the full cancellation of reflections with certain Miller indices. It is said that the reflection with that specific Miller index is systematically absent. For example, in data corresponding to a crystal with space group C 2, general reflections like (h,k,l) must obey h+k=2n (even number). Thus, the Miller indices (2,3,1) are a systematic absence because 2+3=5 (odd).

Value

idx A vector of integers corresponding to the position, in the array mhkl, in which the Miller indices ARE NOT systematically absent. The position of systematically-absent reflections can be found using !idx.

Examples

```
# C 2 monoclinic space group (special setting)
csym <- cryst_symm(15,set=5)
print(csym$SG)

# Create a full data frame of Miller indices
hkl <- expand.grid(H=-4:4,K=-4:4,L=-4:4)

# Index corresponding to valid reflections
# (not systematic absences)
idx <- sysabs(hkl,csym$SG)

# Indices for all reflections
fulldx <- 1:length(hkl[,1])

# Index corresponding to systematic absences
jdx <- fulldx[-idx]

# A couple of systematic absences
hkl[jdx[1:2],]
```

---

translate_SG *Translation of space group symbols, numbers, etc.*

Description

Function to find out space group symbol given number and vice-versa.

Usage

```
translate_SG(value, SG_in = "number", SG_out = "xHM", set = 1)
```

Arguments

value	A string or an integer number corresponding to the space group being investigated.
SG_in	A string representing the space group format for the input. Possible values are: • 1) "number" • 2) "ccp4" • 3) "Hall"

- 4) "xHM"
- 5) "old"

SG_out — A string representing the space group format for the output. Possible values are:

- 1) "number"
- 2) "ccp4"
- 3) "Hall"
- 4) "xHM"
- 5) "old"

set — Specific setting for the given space group. A number like 1,2,... It is used if for a same symbol there are more than one choice.

Details

This function returns either a number of a specific symbol corresponding to a crystallographic space group. The input is an integer number or a character symbol identifying a specific space group. The output is, similarly, the corresponding character symbol or number, according to what is specified in the input. Possible formats are:

- 1) Space group number
- 2) Hall symbol (e.g. ' P 2yb (z,x,y)')
- 3) Extended Hermann-Mauguin symbol (e.g. 'P 1 1 21')

If more than one setting is implied in an ambiguous way in the input value, then the first setting will be selected by default for the output value, unless argument "set" is set to another value.

Value

list_SG A named list with two fields. The first field, "msg", is a character string representing the space group format needed as output. Possible values are the same as those for SG_in. The second field, "ans", is TRUE only if a valid symbol for "msg" is found.

Examples

```
# Space Group P1 corresponds to number 1
translate_SG(value=1,SG_in="number",SG_out="xHM")
```

---

unit_cell *Constructor for an S3 object of class "unit_cell.*

Description

This represents a crystal unit cell.

Usage

```
unit_cell(a = NULL, b = NULL, c = NULL, aa = NULL, bb = NULL, cc = NULL)
```

Arguments

a	A real number. One of the unit cell's side lengths, in angstroms.
b	A real number. One of the unit cell's side lengths, in angstroms.
c	A real number. One of the unit cell's side lengths, in angstroms.
aa	A real number. One of the unit cell's angles, in degrees.
bb	A real number. One of the unit cell's angles, in degrees.
cc	A real number. One of the unit cell's angles, in degrees.

Details

The constructor can be used with less than the full set of six input parameters, to create unit cells for the cubic, tetragonal and orthogonal systems. Objects of "unit_cell" class can also be created with no parameters (default to a cubic cell of side length 10 angstroms).

Value

An object of class "unit_cell". It is a named list of length 6 whose last three slots are of "angle" class.

Examples

```
# Create a monoclinic unit cell
uc <- unit_cell(10,30,15,90,60,90)
print(uc)

# Create a cubic cell (default)
uc <- unit_cell()
print(uc)

# Create a cubic cell with side 20
uc <- unit_cell(20)
print(uc)

# Create a tetragonal unit cell with sides 20 and 60
uc <- unit_cell(20,60)
print(uc)

# Create an orthogonal unit cell
uc <- unit_cell(40,15,30)
print(uc)
```

writeMTZ *Write data to an MTZ file*

Description

Write reflections and experimental information to an MTZ file

Usage

```
writeMTZ(reflections, header, filename, title = NULL, batch_header = NULL)
```

Arguments

reflections	A data frame containing all reflection records in columns. This is usually derived from modifications of a previously existing data frame obtained using readMTZ.
header	A list whose components are other R objects. This is normally derived from the reading of another MTZ file using readMTZ. See further details at readMTZHeader.
filename	A character string. The path to a valid mtz file. If a file with the same name exists, it will be deleted.
title	A character string. The character string associated with the TITLE keyword in an MTZ file. This feature makes it easy to quickly identify the data file in CCP4 programs. Default (NULL) is for the output file to have the same title as the input file.
batch_header	A named list including information at data collection time. This component is present only for raw (unmerged) intensity data produce after the diffraction images integration. Merged MTZ reflection files have batch_header=NULL. Names and types depend on the type of experiment (more information on this can be found at CCP4.)

Value

This function does not return any R object. It outputs an MTZ reflection file to some target location.

Examples

```
# Read the 1dei_phases data included in the package
datadir <- system.file("extdata",package="cry")
filename <- file.path(datadir,"1dei_phases.mtz")
lMTZ <- readMTZ(filename)

# Change dataset name
print(lMTZ$header$DATASET)
lMTZ$header$DATASET[2,2] <- "New CRY dataset"

# Add one HISTORY line (string has to be 80-letters long)
addhist <- "From CRY 0.3.0 - run on Apr 2 20:12:00 2021"
n <- nchar(addhist)
```

```
nblanks <- 80-n
for (i in 1:nblanks) addhist <- paste0(addhist," ")
lMTZ$header$HISTORY <- c(lMTZ$header$HISTORY,addhist)

# Write to a new MTZ file
wd <- tempdir()
fname <- file.path(wd,"new.mtz")
writeMTZ(lMTZ$reflections,lMTZ$header,fname)
```

`writeXDS_ASCII` *Write data to an XDS_ASCII file.*

Description

Function to write an XDS_ASCII-tye named list to a file with XDS_ASCII format (unmerged or merged).

Usage

```
writeXDS_ASCII(proc_info, header, reflections, filename)
```

Arguments

`proc_info`	The first component of an XDS_ASCII-type object. It includes up to three components, `MERGE`, `FRIEDEL` and `PROFILE_FITTING` (this last component is missing for files with merged observations, obtained with the program XSCALE).
`header`	The second component of an XDS_ASCII-type object. This object includes several other objects (see `readXDS_ASCII`).
`reflections`	The third component of an XDS_ASCII-type object. It contains the data (the experimental observations). See `readXDS_ASCII` for more details.
`filename`	A character string. The path to a valid XDS_ASCII file. If a file with the same name exists, it will be deleted.

Details

The XDS_ASCII-type named list includes three components, `processing_info`, `header` and `reflections` (see `readXDS_ASCII`).

Value

This function does not return any R object. It outputs an XDS_ASCII reflection file to some target location.

Examples

```
# Load one of the XDS ASCII files included with
# this distribution of cry
datadir <- system.file("extdata",package="cry")
filename <- file.path(datadir,"xds00_ascii.hkl")
lXDS <- readXDS_ASCII(filename)

# Change date
print(lXDS$header$DATE)
lXDS$header$DATE <- "7-Apr-2021"

# Write to a file called "new.hkl"
wd <- tempdir()
fname <- file.path(wd,"new.hkl")
writeXDS_ASCII(lXDS$processing_info,lXDS$header,
               lXDS$reflections,fname)
```

---

xtal_mat01 *Matrix for cell orthogonalisation (first choice)*

Description

Given the cell parameters, this function returns a matrix for transforming fractional to orthogonal coordinates, corresponding to the first choice in Giacovazzo's book.

Usage

```
xtal_mat01(a, b, c, aa, bb, cc)
```

Arguments

a	A real number. One of the unit cell's side lengths, in angstroms.
b	A real number. One of the unit cell's side lengths, in angstroms.
c	A real number. One of the unit cell's side lengths, in angstroms.
aa	A real number. One of the unit cell's angles, in degrees.
bb	A real number. One of the unit cell's angles, in degrees.
cc	A real number. One of the unit cell's angles, in degrees.

Value

A $3\times$\$ matrix M that transforms a 3×1 vector of fractional coordinates into a 3×1 vector of orthogonal coordinates.

Examples

```
# Fractional coordinates
Xf = c(0.1,0.4,0.8)

# Orthorombic unit cell
M = xtal_mat01(10,40,20,90,90,90)

# Cartesian coordinates
Xc = M%*%Xf
```

`xtal_mat02` *Matrix for cell orthogonalisation (second choice)*

Description

Given the cell parameters, this function returns a matrix for transforming fractional to orthogonal coordinates, corresponding to the second choice in Giacovazzo's book.

Usage

```
xtal_mat02(a, b, c, aa, bb, cc)
```

Arguments

a	A real number. One of the unit cell's side lengths, in angstroms.
b	A real number. One of the unit cell's side lengths, in angstroms.
c	A real number. One of the unit cell's side lengths, in angstroms.
aa	A real number. One of the unit cell's angles, in degrees.
bb	A real number. One of the unit cell's angles, in degrees.
cc	A real number. One of the unit cell's angles, in degrees.

Value

A $3\times$\$ matrix M that transforms a 3×1 vector of fractional coordinates into a 3×1 vector of orthogonal coordinates.

Examples

```
# Fractional coordinates
Xf = c(0.1,0.4,0.8)

# Orthorombic unit cell
M = xtal_mat02(10,40,20,90,90,90)

# Cartesian coordinates
Xc = M%*%Xf
```

Index