Package ‘Rpdb’

February 19, 2015

Type Package
Title Read, write, visualize and manipulate PDB files
Version 2.2
Date 2014-02-25
Author Julien Idé
Maintainer Julien Idé <julien.ide.fr@gmail.com>
Depends R (>= 3.0), rgl (>= 0.93)
Description Provides tools to read, write, visualize PDB files and perform some structural manipulations.
License GPL
Encoding UTF-8
LazyData true
NeedsCompilation no
Repository CRAN
Date/Publication 2014-04-26 14:57:10

R topics documented:

Rpdb-package ......................................................... 2
addAxes ................................................................. 4
addLabels .............................................................. 5
atoms ................................................................. 6
basis ................................................................. 8
bond-angle-dihedral ............................................. 9
cellProperties .................................................. 11
centres .......................................................... 12
conect .......................................................... 14
coords ........................................................ 16
cryst1 ......................................................... 18
distances .................................................. 19
elements ................................................... 21
### Rpdb-package

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>inertia</td>
<td>22</td>
</tr>
<tr>
<td>masses</td>
<td>23</td>
</tr>
<tr>
<td>merge.coords</td>
<td>24</td>
</tr>
<tr>
<td>mirror</td>
<td>26</td>
</tr>
<tr>
<td>mirrorHelpers</td>
<td>27</td>
</tr>
<tr>
<td>natom</td>
<td>29</td>
</tr>
<tr>
<td>pdb</td>
<td>30</td>
</tr>
<tr>
<td>range.coords</td>
<td>31</td>
</tr>
<tr>
<td>read.pdb</td>
<td>32</td>
</tr>
<tr>
<td>reindex</td>
<td>34</td>
</tr>
<tr>
<td>replicate</td>
<td>35</td>
</tr>
<tr>
<td>rotation</td>
<td>36</td>
</tr>
<tr>
<td>rotationHelpers</td>
<td>37</td>
</tr>
<tr>
<td>split.pdb</td>
<td>39</td>
</tr>
<tr>
<td>subset.pdb</td>
<td>40</td>
</tr>
<tr>
<td>toSymbols</td>
<td>41</td>
</tr>
<tr>
<td>translation</td>
<td>42</td>
</tr>
<tr>
<td>translationHelpers</td>
<td>44</td>
</tr>
<tr>
<td>universalConstants</td>
<td>46</td>
</tr>
<tr>
<td>unsplit</td>
<td>47</td>
</tr>
<tr>
<td>vectorialOperations</td>
<td>48</td>
</tr>
<tr>
<td>viewAxis</td>
<td>50</td>
</tr>
<tr>
<td>visualize</td>
<td>51</td>
</tr>
<tr>
<td>wrap</td>
<td>54</td>
</tr>
<tr>
<td>write.pdb</td>
<td>55</td>
</tr>
<tr>
<td>xyz2abc</td>
<td>56</td>
</tr>
</tbody>
</table>

**Index**

**Rpdb-package**

Read, Write, Visualize and Manipulate PDB Files

**Description**

Provides tools to read, write, visualize PDB files, and perform structural manipulations.

**Details**

This package has been developed for computational chemists wishing to manipulate molecular structures stored in PDB files. PDB files can easily be read, written, visualized and some basic structural manipulations can be achieved with the present package. Conversion of Cartesian coordinates into fractional coordinates. Spliting a molecular structure into fragments. Computation of centers-of-geometry and centers-of-mass. Wrapping molecular structure using periodical boundary conditions. Translation, rotation and reflection of atomic coordinates. Calculate atomic bond lengths, angles and dihedrals.
Rpdb-package

Author(s)

Julien Idé <julien.ide.fr@gmail.com>

References

More information on the PDB format can be found here:
http://www.wwpdb.org/documentation/format33/v3.3.html

Examples

```r
## Read a PDB file included in the package
x <- read.pdb(system.file("examples/PCBM_ODCB.pdb", package="Rpdb"))

## Visualize the PDB file
visualize(x, mode = NULL)

## From Cartesian to fractional coordinates and vice versa
x <- xyz2abc(x)
basis(x)
natom(x,x$atoms$resid)
range(x)
centres(x)
x <- abc2xyz(x)
basis(x)
natom(x,x$atoms$resid)
range(x)
centres(x)

## Split and unsplit
F <- x$atoms$resid
x <- split(x, F)
x <- unsplit(x, F)

## Subset and merge
x.PCB.only <- subset(x, resname == "PCB")
x.DCB.only <- subset(x, resname == "DCB")
x <- merge(x.PCB.only, x.DCB.only)

## Duplicate and wrap
x <- replicate(x, a.ind = -1:1, b.ind = -1:1, c.ind = -1:1)
x <- wrap(x)

## Write the 'pdb' object 'x' in a temporary file.
write.pdb(x, file = tempfile())
```
**addAxes**

*Add Axes or PBC Box to the 'rgl' Scene*

**Description**

Add lattice vectors, Cartesian axes or PBC box to the current 'rgl' scene.

**Usage**

```r
addABC(x, lwd = 2, labels = TRUE, cex = 2)
addXYZ(lwd = 2, labels = TRUE, cex = 2)
addPBCBox(x, lwd = 2)
```

**Arguments**

- `x` an object of class 'cryst1' containing unit cell parameters.
- `lwd` a numeric value indicating the line width used to draw the axes or the PBC box.
- `labels` a logical value indicating whether the labels of the axes have to be drawn.
- `cex` a numeric value indicating the magnification used to draw the labels of the axes.

**Details**

- `addABC`: Add the lattice vectors a, b and c to the current rgl device.
- `addXYZ`: Add the Cartesian axes x, y and z to the current rgl device.
- `addPBCBox`: Add a box representing the Periodic Boundary Conditions of a molecular system.

**Value**

Return (using invisible) a two-column data.frame containing the IDs and type indicators of the objects added to the scene.

**See Also**

`visualize`, `rgl.cur`, `par3d`, `addLabels`

**Examples**

```r
x <- read.pdb(system.file("examples/PCBM_ODCB.pdb",package="Rpdb"))
visualize(x, type = "l", xyz = FALSE, abc = FALSE, pbc.box = FALSE, mode = NULL)
addXYZ()
addABC(x$crys1)
addPBCBox(x$crys1)
```
addLabels

Add Labels to the ‘rgl’ Scene

Description

Add Labels to the current ‘rgl’ scene.

Usage

```r
addResLab(x, ...)  # S3 method for class 'atoms'
addResLab(x, at.centre = TRUE, col = "black", ...)
addResLab(x, at.centre = TRUE, col = "black", ...)
addResLab(x, ...)  # S3 method for class 'atoms'
addEleLab(x, eleid = FALSE, col = "black", ...)  # S3 method for class 'pdb'
addEleLab(x, eleid = FALSE, col = "black", ...)
info3d(...)  # S3 method for class 'atoms'
info3d(x, id = rgl.ids(), col = "black", verbose = TRUE, adj = 0, ...)
info3d(x, id = rgl.ids(), col = "black", verbose = TRUE, adj = 0, ...)
```

Arguments

- `x`: an R object containing atomic coordinates.
- `at.centre`: a single element logical vector indicating if residue labels have to be added only at the position of the residue’s centre-of-mass instead of at each atomic position.
- `eleid`: a single element logical vector indicating if the element ids have to be concatenated with the element names to prepare the labels.
- `col`: the colors used to display the labels.
- `adj`: one value specifying the horizontal adjustment, or two, specifying horizontal and vertical adjustment respectively. See `rgl.texts`.
- `id`: vector of ID numbers of ‘rgl’ items, as returned by `rgl.ids`. The vertexes of these items are used to display the labels.
- `verbose`: a logical value specifying if information have to be printed to the terminal.
- `...`: further arguments passed to or from other methods.
Details

addResLab add residue labels to the scene. If at.centre==TRUE only one label per residue is added at the centre of the residue. Otherwise, residue labels are added at each atomic positions.

addEleLab add element labels to the scene at each atomic positions.

info3d activate an interactive mode to add labels by selecting atoms by right-clicking on the current 'rgl' scene. To escape the interactive mode press the ESC key. The labels are as follow: "ResidRes-name:EleidElename"

Value

addResLab and addEleLab return (using invisible) a two-column data.frame containing the IDs and type indicators of the objects added to the scene.

See Also

pdb, visualize, measure

Examples

x <- read.pdb(system.file("examples/PCBM_ODCB.pdb",package="RPdb"))
visualize(x, type = "l", mode = NULL)
addResLab(x)
x <- read.pdb(system.file("examples/Pentacene.pdb",package="RPdb"))
visualize(x, type = "l", mode = NULL)
addEleLab(x)

---

Create ‘atoms’ Object

Description

Creates an object of class ‘atoms’ containing the data related to ATOM and HETATM records of a PDB file.

Usage

atoms(...)

## Default S3 method:
atoms(recname, eleid, elename, alt, resname, chainid, resid, insert, x1, x2, x3, occ, temp, segid, basis = "xyz", ...)

is.atoms(x)
atoms

Arguments

... arguments passed to methods.
recname a character vector containing the record name for each element.
eleid a integer vector containing the element ID for each element.
elename a character vector containing the element name for each element.
altnm a character vector containing the alternate location indicator for each element.
resname a character vector containing the residue name for each element.
chainid a character vector containing the chain ID for each element.
resid a integer vector containing the residue ID for each element.
insert a character vector containing the codes for insertion of residue for each element.
x1, x2, x3 a numeric vector containing the first, second and third coordinate for each element.
occ a numeric vector containing the occupancy for each element.
temp a numeric vector containing the temperature factor for each element.
segid a character vector containing the segment ID for each element.
basis a single element character vector indicating the type of basis vector used to express the atomic coordinates.
x an R object to be tested.

Details

atoms is a generic function to create objects of class ‘atoms’. The purpose of this class is to store ATOM and HETATM records from PDB files. The default method creates a atoms object from its different components, i.e.; recname, eleid, elename, alt, resname, chainid, resid, insert, x1, x2, x3, occ, temp, segid and basis. All the arguments have to be specified except basis which by default is set to “xyz” (Cartesian coordinates).
isAtoms tests if an object of class ‘atoms’, i.e. if it has a “class” attribute equal to atoms.

Value

atoms returns a data.frame of class ‘atoms’ with the following components:
recname a character vector containing the record name for each element.
eleid a integer vector containing the element ID for each element.
elename a character vector containing the element name for each element.
altnm a character vector containing the alternate location indicator for each element.
resname a character vector containing the residue name for each element.
chainid a character vector containing the chain ID for each element.
resid a integer vector containing the residue ID for each element.
insert a character vector containing the codes for insertion of residue for each element.
x1, x2, x3 a numeric vector containing the first, second and third coordinate for each element.
basis

Description

Functions to get or set the basis of an object containing atomic coordinates.

Usage

basis(x)
basis(x) <- value

## Default S3 method:
basis(x)
## Default S3 replacement method:
basis(x) <- value

## S3 method for class 'pdb'
basis(x)
## S3 replacement method for class 'pdb'
basis(x) <- value

Arguments

x an R object containing atomic coordinates.
value a single element character vector use to set the basis of x.

See Also

basis, coords, pdb

Examples

x <- atoms(recname = c("ATOM", "ATOM"), eleid = 1:2, elename = c("H", "H"), alt = "",
  resname = c("H2", "H2"), chainid = "", resid = c(1, 1), insert = "",
x1 = c(0, 0), x2 = c(0, 0), x3 = c(0, 1), occ = c(0.0, 0.0)
  temp = c(1.0, 1.0),
  segid = c("H2", "H2"))
print(x)
is.atoms(x)
Details

basis and basis<- are respectively generic accessor and replacement functions. The default methods get and set the basis attribute of an object containing atomic coordinates. This attribute indicates the type basis vector used to express atomic coordinates.

Value

For basis: NULL or a single element character vector. (NULL is given if the object has no basis attribute.)

For basis<-: the updated object. (Note that the value of basis(x) <- value is that of the assignment, value, not the return value from the left-hand side.)

See Also

cords, atoms, pdb

Examples

x <- read.pdb(system.file("examples/PCBM_ODCB.pdb", package="RPdb"))
basis(x)
x <- xyz2abc(x)
basis(x)


bond-angle-dihedral Atomic Bond Lengths, Angles and Dihedrals

Description

Compute bond lengths, angles and dihedral from atomic coordinates.

Usage

bond(...)
## S3 method for class 'coords'
bond(x, sel1, sel2, ...)
## S3 method for class 'pdb'
bond(x, sel1, sel2, ...)

angle(...)
## S3 method for class 'coords'
angle(x, sel1, sel2, sel3, ...)
## S3 method for class 'pdb'
angle(x, sel1, sel2, sel3, ...)

dihedral(...)


## S3 method for class 'coords'
dihedral(x, sel1, sel2, sel3, sel4, ...)
## S3 method for class 'pdb'
dihedral(x, sel1, sel2, sel3, sel4, ...)

measure(...)
## Default S3 method:
measure(id = rgl.ids(), verbose = TRUE, ...)
## S3 method for class 'coords'
measure(x, id = rgl.ids(), verbose = TRUE, ...)
## S3 method for class 'pdb'
measure(x, id = rgl.ids(), verbose = TRUE, ...)

### Arguments

**x**
an R object containing atomic coordinates.

**sel1, sel2, sel3, sel4**
an integer or logical vector used to select atoms defining bonds, angles or dihedrals. See details.

**id**
vector of ID numbers of ‘rgl’ items, as returned by rgl.ids. The vertexes of these items are used to compute the bond lengths, angles or dihedrals.

**verbose**
a logical value specifying if the information have to be printed to the terminal.

**...**
further arguments passed to or from other methods.

### Details

The number of selected atoms with `sel1`, `sel2`, `sel3` and `sel4` must be the same. `sel1`, `sel2`, `sel3` and `sel4` respectively select the first, second, third and fourth atoms defining bonds, angles or dihedrals.

`measure` activate an interactive mode to compute bond lengths, angles and dihedrals by selecting atoms by `right-clicing` on the current ‘rgl’ scene. To escape the active mode press the ESC key.

### Value

A numeric vector containing atomic bond lengths (in Angstrom), angles or dihedrals (in degrees)

### See Also

`coords`, `pdb`, `info3d`, `visualize`

### Examples

```r
Pen <- read.pdb(system.file("examples/Pentacene.pdb", package="Rpdb"))
visualize(Pen, mode = NULL)
text3d(coords(Pen), texts=Pen$atom$eleid)
bond(Pen,3:4,1:2)
angle(Pen,3:4,1:2,5:6)
dihedral(Pen,3:4,1:2,5:6,6:5)
```
## Properties of a Unit Cell

### Description

Compute the Cartesian coordinates of lattice vectors, the volume or the density of a unit cell.

### Usage

```r
cell.coords(...)
## Default S3 method:
cell.coords(abc, abg = c(90, 90, 90), digits = 3, ...)
## S3 method for class 'cryst1'
cell.coords(x, digits = 3, ...)
## S3 method for class 'pdb'
cell.coords(x, digits = 3, ...)

cell.volume(...)
## S3 method for class 'cryst1'
cell.volume(x, ...)
## S3 method for class 'pdb'
cell.volume(x, ...)

cell.density(...)
## Default S3 method:
cell.density(masses, volume, ...)
## S3 method for class 'pdb'
cell.density(x, ...)
```

### Arguments

- `abc`: a length 3 numeric vector containing the length of the a, b and c lattice vectors.
- `abg`: a length 3 numeric vector containing the angles (degrees) between the a, b and c lattice vectors (alpha, beta, gamma).
- `digits`: an integer used to round the lattice vectors coordinates.
- `x`: an R object containing lattice parameters.
- `masses`: a numeric vector containing atomic masses.
- `volume`: a single element numeric vector containing the volume of the unit cell in Angstrom cube.
- `...`: further arguments passed to or from other methods.

### Details

`cell.coords` is a generic function which computes a 3x3 matrix whose columns contain the Cartesian coordinates of lattice vectors. The 'a' and 'b' vectors are assumed to be respectively along the
x-axis and in the xy-plane. The default method takes directly the lattice parameters as arguments. For objects of class `cryst1` the lattice parameters are first extracted from the object and then the default method is called. For objects of class `pdb` the lattice parameters are extracted from their `cryst1` component and the default method is called.

cell.volume is a generic function to compute the volume of a unit cell. For objects of class `cryst1`, the unit cell parameters are directly used to compute the volume. For objects of class `pdb`, their `cryst1` component is used.

cell.density is a generic function to compute the density of a unit cell. For objects of class `pdb`: First the volume of the unit cell is calculated by calling the cell.volume function on the `cryst1` component of the `pdb` object. Then the element names are converted into element symbols using the toSymbols function and their masses are taken from the elements data set. Finally the density is calculated using the sum of the atomic masses and the volume of the unit cell.

Value

- `cell.coords` returns a 3x3 matrix containing the Cartesian coordinates of lattice vectors arranged by columns.
- `cell.volume` returns a single element numeric vector containing the volume of the unit cell in Angstrom cube.
- `cell.density` returns a single element numeric vector containing the density of the unit cell in g.cm-3.

See Also

cryst1, pdb, xyz2abc

Examples

```r
x <- read.pdb(system.file("examples/PCBM_ODCB.pdb",package="Rpdb"))
cell.volume(x)
cell.density(x)
cell.coords(x)
```

### centres

**Centres-of-Geometry and Centres-of-Mass**

**Description**

Computes centres-of-geometry and centres-of-mass of groups of atoms.

**Usage**

`centres(...)`

## S3 method for class 'coords'
`centres(x, factor = NULL, weights = NULL, unsplit = FALSE, na.rm = FALSE, ...)`
## Arguments

- **x**
  - an R object containing atomic coordinates.

- **factor**
  - a factor used to split the atomic coordinates by groups to compute multiple centres.

- **weights**
  - a numerical vector containing atomic weights used to compute centres-of-mass.

- **unsplit**
  - a logical value indicating whether the coordinates of the centres have to be unsplit to repeat their coordinates for each atom used for their calculation (used for wrapping by groups).

- **na.rm**
  - a logical value indicating whether NA values should be stripped before the computation proceeds.

- **...**
  - further arguments passed to or from other methods.

## Details

`centres` is a generic function to compute centres-of-geometry and centres-of-mass from an object containing atomic coordinates. For objects of class ‘coords’, ‘atoms’ and ‘pdb’, the coordinates of `x` are first splitted into groups defined by `factor` using the `split` function. For each group, the weighted mean of the `x1`, `x2` and `x3` components of `x` are calculated using `weights`. By default all atoms are assumed to have the same weight (calculation of centres-of-geometry). Finally, if `unsplit = TRUE` the coordinates of the centres are unsplit using the `unsplit` function to assign to each atom the coordinates of the centre to which they are attached (used for wrapping by groups).

For objects of class ‘atoms’ and ‘pdb’ by default `factor` is set to `x$resid` and `x$coordinates$resid`, respectively, to compute the centre-of-geometry of the different residues. Notice that coordinates can be neglected for the calculation of the centres using NA values in `factor`.

## Value

Return an object of class ‘coords’ containing the coordinates of centres.

## See Also

- `coords, atoms, pdb, elements`
- and `split, unsplit, factor` for details about splitting data sets.

## Examples

```r
# First lets read a pdb file
x <- read.pdb(system.file("examples/PCBM_ODCB.pdb",package="Rpdb"))
# Centre-of-geometry of the residues
centres(x)
```
# Centre-of-geometry of the whole structure
centres(x, factor = rep(1, natom(x)))
# or
centres(coords(x))

# Centres-of-geometry of the PCB and DCB residues
centres(x, factor = x@atoms$resname)

# Knowing the name of the elements forming
# the C60 of the PCBM molecules (PCB residues)
# we can compute the centres-of-geometry of
# the C60 by neglecting the other atoms of the
# PCB residues.
C60.elename <- paste0("C", sprintf("%0.3d",1:60))

is.PCB <- x@atoms$resname == "PCB" # Produce a mask to select only the PCB residues
is.C60 <- is.PCB & x@atoms$elename %in% C60.elename # Produce a mask to keep only the C60

F <- x@atoms$resid # We use the residue IDs to split the coordinates
F[!is.C60] <- NA # We keep only the atoms of the C60
C60.centres <- centres(x, factor = F)

# Lets check the position of the C60 centres
visualize(x, mode = NULL)
spheres3d(C60.centres)
text3d(Ty(C60.centres, 2), text=paste0("PCB_, rownames(C60.centres), cex=2)

# Centres-of-mass of the residues
symb <- toSymbols(x@atoms$elename) # Convert elename into elemental symbols
# Find the mass of the element in the periodic table
w <- elements[match(symb, elements[,"symb"])"mass"]
centres(x, weights = w)

---

**conect**

*Create ‘conect’ Object*

**Description**

Creates an object of class ‘conect’ containing the IDs of bonded atoms defining the connectivity of a molecular system.

**Usage**

conect(...)

## Default S3 method:

conect(eleid.1, eleid.2, ...)

## S3 method for class ‘coords’
conect

conect(x, radii = 0.75, safety = 1.2, by.block = FALSE, ...)  
## S3 method for class 'pdb'
conect(x, safety = 1.2, by.block = FALSE, ...)

is.conect(x)

Arguments

... arguments passed to methods.

eleid.1    a integer vector containing the IDs of bonded atoms.

eleid.2    a integer vector containing the IDs of bonded atoms.

x          an R object containing atomic coordinates.

radii      a numeric vector containing atomic radii used to find neighbours.

safety     a numeric value used to extend the atomic radii.

by.block   a logical value indicating whether the connectivity has to be determine by block (see details).

Details

conect is a generic function to create objects of class ‘conect’. The purpose of this class is to store CONECT records from PDB files, indicating the connectivity of a molecular system. The default method creates a conect object from its different components, i.e.: eleid.1 and eleid.2. Both arguments have to be specified. The S3 method for object of class ‘coords’ determine the connectivity from atomic coordinates. A distance matrix is computed, then, for each pair of atom the distance is compared to a bounding distance computed from atomic radii. If this distance is lower than the bounding distance then the atoms are assumed to be connected. The S3 method for object of class ‘pdb’ first use element names to search for atomic radii in the elements data set. Then atomic coordinates and radii are passed to conect.coords. If by.block == TRUE, a grid is defined to determined the connectivity by block. The method is slow but allow to deal with very large systems.

is.conect tests if an object of class ‘conect’, i.e. if it has a “class” attribute equal to conect.

Value

conect returns a two-column data.frame of class ‘conect’ whose rows contain the IDs of bonded atoms. The columns of this data.frame are described below:

eleid.1 a integer vector containing the elements IDs defining the connectivity of the system.

eleid.2 a integer vector containing the elements IDs defining the connectivity of the system.

is.conect returns TRUE if x is an object of class ‘coords’ and FALSE otherwise.
See Also

pdb

Examples

# If atom 1 is connected to atom 2, 3, 4 and 5
# then we can prepare the following 'conect' object:
x <- conect(rep(1:4), 2:5)
print(x)
is.conect(x)

# Compute connectivity from coordinates
x <- read.pdb(system.file("examples/PCBM_ODCB.pdb", package="Rpdb"), CONECT = FALSE)
x$conect
x$conect <- conect(x)
x$conect

cords

The Atomic Coordinates of an Object

Description

Get or set the atomic coordinates (either Cartesian or fractional coordinates) of an object.

Usage

coords(...)
cords(x) <- value
is.coords(x)

## Default S3 method:
cords(x1, x2, x3, basis = "xyz", ...)

## S3 method for class 'atoms'
cords(x, ...)
## S3 replacement method for class 'atoms'
cords(x) <- value

## S3 method for class 'pdb'
cords(x, ...)
## S3 replacement method for class 'pdb'
cords(x) <- value

## S3 method for class 'data.frame'
cords(x, basis = NULL, ...)

## S3 method for class 'matrix'
cords(x, basis = NULL, ...)
Arguments

- **x1**, **x2**, **x3**: numeric vectors containing the first, second and third coordinates.
- **basis**: a single element character vector indicating the type of basis vector used to express the atomic coordinates.
- **x**: an R object containing atomic coordinates.
- **value**: an object of class ‘coords’ used for replacement
- **...**: further arguments passed to or from other methods.

Details

The purpose of the ‘coords’ class is to store the coordinates of a molecular system and facilitate their manipulation when passing from the Cartesian to fractional coordinates and vice versa. `coords` and `coords<-` are generic accessor and replacement functions.

The default method of the `coords` function is actually a builder allowing to create a ‘coords’ object from its different components, i.e.: `x1`, `x2`, `x3`, and `basis`. All the arguments have to be specified except ‘basis’ which by default is set to "xyz" (Cartesian coordinates).

For an object of class ‘atoms’, the accessor function extracts its `x1`, `x2` and `x3` components as well as its `basis` attribute to create a ‘coords’ object. The replacement function set its `x1`, `x2` and `x3` components as well as its `basis` attribute.

For an object of class ‘pdb’, the accessor function extracts the `x1`, `x2` and `x3` components as well as the `basis` attribute of its `atoms` component to create a ‘coords’ object. The replacement function set the `x1`, `x2` and `x3` components as well as the `basis` attribute of its `atoms` component.

For ‘matrix’ and ‘data.frame’ objects, when `basis` is `NULL` this function search `x`, `y`, `z` or `a`, `b`, `c` columns in `x`. If `x`, `y`, `z` columns are found they are used to set the first, second and third coordinates of the returned ‘coords’ object. In that case the basis set of `x` is set to "xyz".

If `a`, `b`, `c` columns are found they are used to set the first, second and third coordinates of the returned ‘coords’ object. In that case the basis set of `x` is set to "abc".

If the function doesn’t found neither the `x`, `y`, `z` nor the `a`, `b`, `c` columns an error is returned.

When `basis` is `NULL` it has to be equal to "xyz" or "abc" and `x` must have exactly 3 columns.

`is.coords` tests if `x` is an object of class ‘coords’, i.e. if `x` has a “class” attribute equal to `coords`.

Value

The accessor function returns a data.frame of class ‘coords’ whose columns contain the three coordinates of the atoms of a molecular system. The coordinates can either be Cartesian (`basis` attribute equal to "xyz") or fractional coordinates (`basis` attribute equal to "abc").

The replacement function returns an object of the same class as `x` with updated coordinates.

`is.coords` returns TRUE if `x` is an object of class ‘coords’ and FALSE otherwise.

See Also

`basis`
**Examples**

```
x <- read.pdb(system.file("examples/PCBM_ODCB.pdb", package="Rpdb"))
is.coords(x)
is.coords(x$atoms)

## Replace the coordinates of x by translated coordinates
coords(x) <- coords(Tz(x, 10))
coords(x)
```

---

**cryst1**  
Create ‘cryst1’ Object

**Description**

Create an object of class ‘cryst1’ containing the unit cell parameters and the name of the space group to associate with an object of class ‘pdb’.

**Usage**

```
cryst1(...)  
## Default S3 method:  
cryst1(abc, abg = c(90, 90, 90), sgroup = "P1", ...)  
is.cryst1(x)
```

**Arguments**

- `...` further arguments passed to or from other methods.
- `abc` a numeric vector of length 3 containing the norms of the lattice vectors a, b and c.
- `abg` a numeric vector of length 3 containing the angles between the lattice vectors $\alpha$, $\beta$ and $\gamma$.
- `sgroup` a character string giving the Hermann-Mauguin symbol of the space group.
- `x` an R object to be tested.

**Details**

`cryst1` is a generic function to create objects of class ‘cryst1’. The purpose of this class is to store CRST1 records from PDB files which contain the unit cell parameters and the name of the space group of a molecular system stored in a PDB file. The default method of the `cryst1` function create an object of class ‘cryst1’ from its different components, i.e.: abc, abg and sgroup. At least abc has to be specified.

`is.cryst1` tests if an object of class ‘cryst1’, i.e. if it has a “class” attribute equal to `cryst1`. 
Value

`cryst1` returns a list of class `‘cryst1’` with the following components:

- `abc` a numeric vector of length 3 containing the norms of the lattice vectors a, b and c.
- `abg` a numeric vector of length 3 containing the angles between the lattice vectors $\alpha$, $\beta$, and $\gamma$.
- `sgroup` a character string giving the Hermann-Mauguin symbol of the space group.

`is.Cryst1` returns `TRUE` if `x` is an object of class `‘cryst1’` and `FALSE` otherwise.

See Also

`cell.coords`, `pdb`

Examples

```r
x <- cryst1(abc = c(10, 10, 10), abg = c(90, 90, 90), sgroup = "P1")
is.cryst1(x)
```

---

### distances

#### Inter-Atomic Distances

Description

Computes inter-atomic distance vectors.

Usage

```r
is.distances(x)
distances(...)
```

## Default S3 method:
`distances(dx1 = numeric(0), dx2 = numeric(0), dx3 = numeric(0), basis = "xyz", ...)`

## S3 method for class 'coords'
`distances(x, sel1, sel2, ...)`

## S3 method for class 'atoms'
`distances(x, sel1, sel2, ...)`

## S3 method for class 'pdb'
`distances(x, sel1, sel2, ...)`

`norm(...)`

## S3 method for class 'distances'
`norm(x, type = "xyz", ...)"
Arguments

- dx1, dx2, dx3: numeric arrays containing the first, second and third components of the distance vectors.
- basis: a single element character vector indicating the type of basis vector used to express the coordinates.
- x: an R object containing atomic coordinates.
- selQ, selR: integer or logical vectors defining two atomic selections between which the distance vectors are computed.
- type: a single element character vector indicating how to project the distance vectors before computing the norms. See details.
- ...: further arguments passed to or from other methods.

Details

The purpose of the `distances` class is to store the inter-atomic distance vectors and facilitate their manipulation when passing from the Cartesian to fractional references and vice versa. The default method of the `distances` function is actually a builder allowing to create a `distances` object from its different components, i.e.: dx1, dx2, dx3, and basis. All the arguments have to be specified except ‘basis’ which by default is set to “xyz” (Cartesian reference).

For objects of class ‘coords’, ‘atoms’, ‘pdb’, two sets of atomic coordinates, defined by selQ and selR, are extracted and inter-atomic distance vectors are computed between these two sets.

The method of the norm function for objects of class ‘distances’ computes the norm of the distance vectors. type specify how to project the distance vectors before computing the norms. By default no projection is perform. The three dx, dy, and dz components of the distance vectors are used to compute the norm. type can take the following values:

- x: The distance vectors are projected over x.
- y: The distance vectors are projected over y.
- z: The distance vectors are projected over z.
- xy: The distance vectors are projected in the xy-plan.
- yz: The distance vectors are projected in the yz-plan.
- zx: The distance vectors are projected in the zx-plan.
- xyz: The distance vectors are not projected (The three components of the distance vectors are used to compute the norm).

`is.distances` tests if x is an object of class ‘distances’, i.e. if x has a “class” attribute equal to distances.

Value

The distance function return an object of class ‘distances’ containing inter-atomic distance vectors. The norm function return an array, with the same dimensions as the dx1, dx2, dx3 components of the ‘distances’ object for which the norms have to be computed, containing the norm of the distance vectors.

`is.distances` returns TRUE if x is an object of class ‘distances’ and FALSE otherwise.
See Also
cords, basis, xyz2abc, abc2xyz

Examples

```
x <- read.pdb(system.file("examples/PCBM_ODCB.pdb",package="Rpdb"))
is.DCB7 <- x$atoms$resname == "DCB" & x$atoms$resid == 7
is.DCB8 <- x$atoms$resname == "DCB" & x$atoms$resid == 8
d <- distances(x, is.DCB7, is.DCB8)
norm(d, type = "xyz")
norm(d, type = "xy")
norm(d, type = "x")
```

Description

This data set gives various information on chemical elements

Usage
elements

Format

A data frame containing for each chemical element the following information.

- `num` atomic number
- `symb` elemental symbol
- `areneg` Allred and Rochow electronegativity (0.0 if unknown)
- `rcov` covalent radii (in Angstrom) (1.6 if unknown)
- `rbo` "bond order" radii
- `rvdw` van der Waals radii (in Angstrom) (2.0 if unknown)
- `maxbnd` maximum bond valence (6 if unknown)
- `mass` IUPAC recommended atomic masses (in amu)
- `elneg` Pauling electronegativity (0.0 if unknown)
- `ionization` ionization potential (in eV) (0.0 if unknown)
- `elaffinity` electron affinity (in eV) (0.0 if unknown)
- `red` red value for visualization
- `green` green value for visualization
- `blue` blue value for visualization
- `name` element name
Source

Open Babel (2.3.1) file: element.txt

Created from the Blue Obelisk Cheminformatics Data Repository
Direct Source: http://www.blueobelisk.org/
http://www.blueobelisk.org/repos/blueobelisk/elements.xml includes further bibliographic citation information

- Allred and Rochow Electronegativity from http://www.hull.ac.uk/chemistry/electroneg.php?type=Allred-Rochow
- Covalent radii from http://dx.doi.org/10.1039/b801115j
- Van der Waals radii from http://dx.doi.org/10.1021/jp8111556

Examples

```r
data(elements)
elements

# Get the mass of some elements
symb <- c("C","O","H")
elements[match(symb,elements[,"symb"]),"mass"]

# Get the van der Waals radii of some elements
symb <- c("C","O","H")
elements[match(symb,elements[,"symb"]),"rvdw"]
```

---

**inertia**  

Moment of Inertia of a Molecular System

Description

Computes the inertia tensor of a molecular system from atomic coordinates and masses.

Usage

```r
inertia(...)  
## S3 method for class 'coords'
inertia(x, m, ...)
## S3 method for class 'atoms'
inertia(x, m = NULL, ...)
## S3 method for class 'pdb'
inertia(x, m = NULL, ...)
```
masses

Arguments

x  
an R object containing atomic coordinates.

m  
a numeric vector containing atomic masses.

...  
further arguments passed to or from other methods.

Details

inertia is a generic function to compute the inertia tensor of a molecular system. For object of class `coords` both atomic coordinates and masses have to be specified. For object of class `atoms` the masses are determined from the elename component of the object (see `toSymbols` and `masses`). For object of class `pdb` the atoms component is used.

Value

Return the inertia tensor in a 3x3 matrix.

See Also

toSymbols, masses, viewInertia

Examples

C70 <- read.pdb(system.file("examples/C70.pdb", package="Rpdb"))
inertia(C70)
visualize(C70, mode = NULL)
viewXY()
viewInertia(C70)

masses  

Mass of Chemical Elements

Description

Determine the mass of chemical elements

Usage

masses(...)

### Default S3 method:
masses(x, ...)

### S3 method for class 'pdb'
masses(x, ...)
Arguments

... further arguments passed to or from other methods.

x either a character or an integer vector containing element symbols or atomic numbers, or an object of class 'pdb' from which element symbols are determined (see details).

Details

masses is a generic function to determine the mass of chemical elements.

For objects of class 'pdb':

- First the element names are converted into element symbols using the toSymbols function.
- Then their masses are taken from the elements data set.

NA values are returned for unrecognized elements.

Value

Return a numeric vector containing the mass of chemical elements.

See Also

toSymbols

Examples

x <- read.pdb(system.file("examples/PCBM_ODCB.pdb", package="Rpdb"))
masses(x)
masses(c("C","Cl","N","N"))

merge.coords  Merging Molecular Systems

Description

Merge two objects containing atomic coordinates

Usage

## S3 method for class 'coords'
merge(x, y, ...)

## S3 method for class 'atoms'
merge(x, y, reindex = TRUE, ...)

## S3 method for class 'pdb'
merge(x, y, reindex = TRUE, ...)
Arguments

x, y  objects of class 'coords' to be merged.
reindex  a single element logical vector indicating if residue and element IDs have to be reindexed after merging.
...  further arguments passed to or from other methods.

Details

To merge x and y they must have the same basis attributes (see basis).

For objects of class 'coords' and 'atoms' the atomic coordinates are directly merged by row.

For objects of class 'pdb', the atoms and conect components of the two pdb objects are merged by row and the cryst1 components of x is used to build the returned object.

For objects of class 'atoms' and 'pdb' the residue and element IDs of y are shifted to avoid any confusion with those of x. If reindex==TRUE the reindex function is called to reinitialize the indexing of the returned object.

Value

Return a n object of the same class as x and y merging x and y. If x and y have different basis attributes an error is returned.

See Also

cords, atoms, pdb, basis, merge, merge.data.frame

Examples

c1 <- coords(1:3, 4:6, 7:9, basis = "xyz")
c2 <- coords(10:12, 13:15, 16:18, basis = "xyz")
merge(c1,c2)

## Not run:
## Merging objects with different basis sets return an error.
c2 <- coords(9:11, 12:14, 15:17, basis = "abc")
merge(c1,c2)

## End(Not run)

## Prepare a Pentacene/C70 dimer
C70 <- read.pdb(system.file("examples/C70.pdb",package="Rpdb"))
Pen <- read.pdb(system.file("examples/Pentacene.pdb",package="Rpdb"))
x <- merge(Tz(C70, 3.5, thickness=0.5),Pen)
**Description**

Perform a reflexion (or mirror) operation on atomic coordinates with respect to a given reflexion plan.

**Usage**

```r
mirror(...)  
## S3 method for class 'coords'
mirror(x, p1, p2 = NULL, p3 = NULL, mask = TRUE, cryst1 = NULL, ...)
  
## S3 method for class 'pdb'
mirror(x, p1, p2 = NULL, p3 = NULL, mask = TRUE, cryst1 = x$cryst1, ...)
```

**Arguments**

- `x` an R object containing atomic coordinates.
- `p1` a numeric vector of length 3 containing the coordinates of the first point defining the reflexion plan. Can also be a 3x3 matrix or data.frame containing by row `p1`, `p2` and `p3`.
- `p2` a numeric vector of length 3 containing the coordinates of the second point defining the reflexion plan.
- `p3` a numeric vector of length 3 containing the coordinates of the third point defining the reflexion plan.
- `mask` a logical vector indicating the set of coordinates to which to apply the reflexion.
- `cryst1` an object of class 'cryst1' use to convert fractional into Cartesian coordinates when need.
- `...` further arguments passed to or from other methods.

**Details**

`mirror` is generic functions. Method for objects of class ‘coords’ first convert the coordinates into Cartesian coordinates using `cryst1` if needed. Once reflected, the coordinates are reconverted back to the orginal basis set using again `cryst1`. Method for objects of class ‘pdb’ first extract coordinates from the object using the function `coords`, perform the reflection, and update the coordinates of the ‘pdb’ object using the function `coords<-.`

**Value**

An object of the same class as `x` with reflected coordinates.
See Also

Helper functions for reflection with respect to a given Cartesian plan or a plan defined by two lattice vectors:

Mxy, Myz, Mzx, Mab, Mbc, Mca

Passing from Cartesian to fractional coordinates (or Vis Versa):

xyz2abc, abc2xyz

Examples

# First lets read a pdb file
x <- read.pdb(system.file("examples/PCBM_ODCB.pdb",package="Rpdb"))
cell <- cell.coords(x)
visualize(x, mode = NULL)
# Mirror operation with respect to the ab-plan
visualize(mirror(x, rep(0,3), p1=cell[,"a"], p2=cell[,"b"], mode = NULL))
# Mirror operation with respect to the ab-plan for residue 1
visualize(mirror(x, rep(0,3), p1=cell[,"a"], p2=cell[,"b"], mask=x$atoms$resid==1, mode = NULL))

Description

Reflection of atomic coordinates with respect to a specific Cartesian plan or a plan defined by two lattice vectors.

Usage

Mxy(...)
Myz(...)
Mzx(...)
Mab(...)
Mbc(...)
Mca(...)

## S3 method for class 'coords'
Mxy(x, mask = TRUE, cryst1 = NULL, ...)
## S3 method for class 'coords'
Myz(x, mask = TRUE, cryst1 = NULL, ...)
## S3 method for class 'coords'
Mzx(x, mask = TRUE, cryst1 = NULL, ...)
## S3 method for class 'coords'
Mab(x, cryst1, mask = TRUE, ...)
## S3 method for class 'coords'
Mbc(x, cryst1, mask = TRUE, ...)
## S3 method for class 'coords'
Mca(x, cryst1, mask = TRUE, ...)
## S3 method for class 'pdb'

\[ \text{Mxy}(x, \text{mask} = \text{TRUE}, \text{cryst1} = x\$\text{cryst1}, \ldots) \]

## S3 method for class 'pdb'

\[ \text{Myz}(x, \text{mask} = \text{TRUE}, \text{cryst1} = x\$\text{cryst1}, \ldots) \]

## S3 method for class 'pdb'

\[ \text{Mzx}(x, \text{mask} = \text{TRUE}, \text{cryst1} = x\$\text{cryst1}, \ldots) \]

## S3 method for class 'pdb'

\[ \text{Mab}(x, \text{cryst1} = x\$\text{cryst1}, \text{mask} = \text{TRUE}, \ldots) \]

## S3 method for class 'pdb'

\[ \text{Mbc}(x, \text{cryst1} = x\$\text{cryst1}, \text{mask} = \text{TRUE}, \ldots) \]

## S3 method for class 'pdb'

\[ \text{Mca}(x, \text{cryst1} = x\$\text{cryst1}, \text{mask} = \text{TRUE}, \ldots) \]

### Arguments

- **x**
  - an R object containing atomic coordinates.
- **mask**
  - a logical vector indicating the set of coordinates to which to apply the reflection.
- **cryst1**
  - an object of class ‘cryst1’ use to convert fractional into Cartesian coordinates when need.
- **...**
  - further arguments passed to or from other methods.

### Details

These functions are helper functions to perform a reflection with respect to a specific Cartesian plan or a plan defined by two lattice vectors. All of them call the \textit{mirror} function.

### Value

An object of the same class as \( x \) with reflected coordinates.

### See Also

\textit{mirror} and \textit{xyz2abc, abc2xyz} for passing from Cartesian to fractional coordinates (or Vis Versa).

### Examples

```r
# First lets read a pdb file
x <- read.pdb(system.file("examples/PCBM_ODCB.pdb",package="Rpdb"))
visualize(x,mode = NULL)
# Mirror operation with respect to the ab-plan
visualize(Mab(x), mode = NULL)
# Mirror operation with respect to the ab-plan for residue 1
visualize(Mab(x, mask=x$atoms$resid==1), mode = NULL)
```
**natom**

*Number of Atoms in an Object Containing Atomic Coordinates*

### Description

Evaluates the number of atoms in an object containing atomic coordinates.

### Usage

```r
natom(x, 

# S3 method for class 'coords'
natom(x, factor = NULL, 

# S3 method for class 'atoms'
natom(x, factor = NULL, ATOM = TRUE, HETATM = TRUE, 

# S3 method for class 'pdb'
natom(x, factor = NULL, ATOM = TRUE, HETATM = TRUE, 
```

### Arguments

- **x**: an R object containing atomic coordinates.
- **factor**: a factor use to split the object and evaluate the number of atom in each group.
- **ATOM**: a single element logical vector indicating if ATOM records have to be considered or not.
- **HETATM**: a single element logical vector indicating if HETATM records have to be considered or not.
- **...**: further arguments passed to or from other methods.

### Details

`natom` is a generic function to evaluate the number of atom in an object containing atomic coordinates. The atomic coordinates of the object are first filtered to keep ATOM and/or HETATM records as indicated by the 'ATOM' and 'HETATM' arguments. Then, if `factor` is specify, the object is splitted to evaluate the number of atoms in each group defined by `factor`. If `factor` is not specify then the total number of atoms in the object is return.

### Value

Return an integer or a vector of integer of lenght equal to `nlevels(factor)` (if `factor` is specify) indication the number of atoms in the object or in the groups defined by `factor`.

### See Also

`coords, atoms, pdb, factor, split`
Examples

```r
x <- read.pdb(system.file("examples/PCBM_ODCB.pdb", package="Rpdb"))
natom(x)
natom(x, x$atoms$resid)
natom(x, x$atoms$resname)
natom(x, HETATM=FALSE)
```

---

**pdb**

*Create an Object of Class ‘pdb’*

---

**Description**

Creates an object of class ‘pdb’.

**Usage**

```r
pdb(...)  ## Default S3 method: 
pdb(atoms, cryst1 = NULL, conect = NULL, remark = NULL, title = NULL, ...)
is.pdb(x)
```

**Arguments**

- **atoms**
  - a data.frame of class `atoms` containing ATOM and HETATM records use to create the `pdb` object.

- **cryst1**
  - a list of class `cryst1` containing the periodical bondary conditions and space group use to create the `pdb` object.

- **conect**
  - a data.frame of class `conect` containing CONECT records use to create the `pdb` object.

- **remark**
  - a character vector containing some REMARK records to be added to the `pdb` object.

- **title**
  - a character vector containing some TITLE records to be added to the `pdb` object.

- **x**
  - an R object to be tested

- ... further arguments passed to or from other methods.

**Details**

This function is the generic function to create objects of class ‘pdb’. The purpose of this class is to store the data of molecular systems contained in PDB files. The default method of the `pdb` function creates an object of class ‘pdb’ from its different components, i.e.: `title`, `remark`, `cryst1`, `atoms` and `conect`. At least an object of class ‘atoms’ has to be specified.

`is.pdb` tests if `x` is an object of class ‘pdb’, i.e. if `x` has a “class” attribute equal to `pdb`. 
Value

pdb returns a list of class ‘pdb’ with the following components:

- `title`: a character vector containing the TITLE records found in a PDB file.
- `remark`: a character vector containing the REMARK records found in a PDB file.
- `cryst1`: a list of class ‘cryst1’ containing the first CRYST1 record found in a PDB file. All others are ignored.
- `atoms`: a data.frame of class ‘atoms’ containing the ATOM and HETATM records found in a PDB file.
- `conect`: a data.frame of class ‘conect’ containing the CONECT records found in a PDB file.

`is.pdb` returns TRUE if x is an object of class ‘pdb’ and FALSE otherwise.

See Also

`atoms`, `coords`, `cryst1`, `conect` and `read.pdb`

Examples

```r
title <- "This is just an example"
remark <- NULL
cryst1 <- cryst1(c(10,10,10))
atoms <- atoms(recname = c("ATOM","ATOM"), eleid = 1:2, elenamer = c("H","H"), alt = "", resname = c("H2","H2"), chainid = ", resid = c(1,1), insert = ", x1 = c(0,0), x2 = c(0,0), x3 = c(0,1), occ = c(0.8,0.0), temp = c(1.0,1.0), segid = c("H2","H2"))
conect <- conect(eleid.1 = c(1), eleid.2 = c(2))
x <- pdb(atoms = atoms, cryst1 = cryst1, conect = conect, remark = remark, title = title)
is.pdb(x)
```

Description

Determines the range of atomic coordinates.

Usage

```r
## S3 method for class 'coords'
range(x, na.rm = FALSE, finite = FALSE, ...)

## S3 method for class 'atoms'
range(x, na.rm = FALSE, finite = FALSE, ...)

## S3 method for class 'pdb'
range(x, na.rm = FALSE, finite = FALSE, ...)
```
Arguments

- `x`: an R object containing atomic coordinates.
- `na.rm`: logical, indicating if `NA`'s should be omitted.
- `finite`: logical, indicating if all non-finite elements should be omitted.
- `...`: further arguments passed to or from other methods.

Value

Return a `data.frame` whose columns contain the range of the first, second and third coordinates of `x`.

See Also

`range`, `coords`, `atoms.pdb`

Examples

```r
x <- read.pdb(system.file("examples/PCBM_ODCB.pdb", package="Rpdb"));
range(x)
range(range(x))
```

---

**read.pdb**  
**PDB File Reader**

Description

Reads a Protein Data Bank (PDB) coordinate file.

Usage

```r
read.pdb(file, ATOM = TRUE, HETATM = TRUE, CRYST1 = TRUE, CONECT = TRUE, TITLE = TRUE, REMARK = TRUE, MODEL = 1)
```

Arguments

- `file`: a single element character vector containing the name of the PDB file to be read.
- `ATOM`: a single element logical vector indicating whether ATOM records have to be read.
- `HETATM`: a single element logical vector indicating whether HETATM records have to be read.
- `CRYST1`: a single element logical vector indicating whether CRYS T1 records have to be read.
- `CONECT`: a single element logical vector indicating whether CONECT records have to be read.
**read.pdb**

TITLE a single element logical vector indicating whether TITLE records have to be read.

REMARK a single element logical vector indicating whether REMARK records have to be read.

MODEL an integer vector containing the serial number of the MODEL sections to be read. Can also be equal to NULL to read all the MODEL sections or to NA to ignore MODEL records (see details).

Details

The *read.pdb* function reads TITLE, REMARK, ATOM, HETATM, CRST1 and CONECT records from a PDB file. Three different reading modes can be used depending on the value of MODEL:

- When `model` is a vector of integers, MODEL sections whose serial numbers match these integers are read.
- When `model` = NULL, all MODEL sections are read.
- When `model` = NA, MODEL records are ignored to read all ATOM and/or HETATM records together to return a single object.

Value

When a single MODEL section is read, this function returns an object of class ‘pdb’ (a list with a class attribute equal to ‘pdb’) with the following components:

- `title` a character vector containing the TITLE records found in the PDB file.
- `remark` a character vector containing the REMARK records found in the PDB file.
- `cryst1` a list of class ‘cryst1’ containing the first CRST1 record found in the PDB file. All others are ignored.
- `atoms` a data.frame of class ‘atoms’ containing the ATOM and HETATM records found in the PDB file.
- `conect` a data.frame of class ‘conect’ containing the CONECT records found in the PDB file.

When multiple MODEL sections are read, a list of object of class ‘pdb’ is returned.

References

PDB format has been taken from: http://www.wwpdb.org/documentation/format33/v3.3.html

See Also

`write.pdb`, `pdb`, `cryst1`, `atoms`, `conect`
Examples

## Read a PDB file included with the package
```r
x <- read.pdb(system.file("examples/PCBM_ODCB.pdb",package="Rpdb"))
```

## Visualize the PDB file
```r
visualize(x, mode = NULL)
```

## Write the `pdb` object `x` in file "Rpdb.pdb" into the current directory
```r
write.pdb(x, file = "Rpdb.pdb")
```

---

**reindex**

Reinitialize Object Indexing

### Description

Reinitialize the indexing of an object.

#### Usage

```r
reindex(...)
```

#### Arguments

- `...`: further arguments passed to or from other methods
- `x`: an R object
- `eleid`: a single element logical vector indicating if elements IDs have to reindexed
- `resid`: a single element logical vector indicating if residues IDs have to reindexed

#### Details

`reindex` is a generic function to reinitialize the indexing of an object or its components. The methods for objects of class `\texttt{atoms}` reinitialize the residue and element IDs starting from 1 and avoiding gaps in the indexes. For objects of class `\texttt{pdb}` their \texttt{atoms} and \texttt{conect} components are reindexed consistently.

### Value

Return an object of the same class as `x` with updated indexes.
replicate

See Also
pdb, atoms, subset.atoms, subset.pdb

Examples
x <- read.pdb(system.file("examples/PCBM_ODCB.pdb",package="Rpdb"))
x <- subset(x, x$atoms$eleid %in% sample(x$atoms$eleid, 10))
print(x)
x <- reindex(x)
print(x)

replicate  Replicate Atomic Coordinates

Description
Replicate atomic coordinates using periodic boundary conditions.

Usage
replicate(x, ...)

## S3 method for class 'coords'
replicate(x, cryst1, a.ind = 0, b.ind = 0, c.ind = 0, ...)

## S3 method for class 'atoms'
replicate(x, cryst1, a.ind = 0, b.ind = 0, c.ind = 0, ...)

## S3 method for class 'pdb'
replicate(x, a.ind = 0, b.ind = 0, c.ind = 0, cryst1 = NULL, ...)

Arguments
x an R object containing atomic coordinates to be replicated.
cryst1 an object of class ‘cryst1’ containing periodical boundary conditions used for replicating.
a.ind a vector of integers indicating the positions of the replicated cells along the a-axis.
b.ind a vector of integers indicating the positions of the replicated cells along the b-axis.
c.ind a vector of integers indicating the positions of the replicated cells along the c-axis.
... further arguments passed to or from other methods.
rotation

Details

The replicate function replicate a unit cell along the lattice vectors a, b and c as many times as indicated by the a.ind, b.ind and c.ind arguments. Discontinuous integer vectors can be used for a.ind, b.ind and c.ind to create layered supercells (See examples).

Value

Return an object of class 'pdb' with replicated atomic coordinates.

See Also

cords, atoms, pdb, cryst1

Examples

x <- read.pdb(system.file("examples/PCBM_ODCB.pdb",package="Rpdb"))

# Create a 3x3 supercell
y <- replicate(x, a.ind= 0:2, b.ind = 0:2, c.ind = 0:2)

# Create a 3x3 supercell which might need to be wrapped (some molecules are outside the cell)
y <- replicate(x, a.ind= -1:1, b.ind = -1:1, c.ind = -1:1)

# Create a layered supercell with a vacuum layer in the bc-plan
y <- replicate(x, a.ind= c(0,2), b.ind = 0:2, c.ind = 0:2)

---

rotation

Rotation of Atomic Coordinates

Description

Rotation of atomic coordinates around a given vector.

Usage

R(...)
## S3 method for class 'coords'
R(obj, angle = 0, x = 0, y = 0, z = 1, mask = TRUE, cryst1 = NULL, ...)
## S3 method for class 'pdb'
R(obj, angle = 0, x = 0, y = 0, z = 1, mask = TRUE, cryst1 = obj$cryst1, ...)

Arguments

obj an R object containing atomic coordinates.
angle the angle of the rotation in degrees.
x the x-component of the rotation vector.
y  the y-component of the rotation vector.

z  the z-component of the rotation vector.

mask a logical vector indicating the set of coordinates to which the rotation has to be applied.

cryst1 an object of class ‘cryst1’ use to convert fractional into Cartesian coordinates when need.

... further arguments passed to or from other methods.

Details

R is generic functions. Method for objects of class ‘coords’ first convert the coordinates into Cartesian coordinates using cryst1 if needed. Once rotated, the coordinates are reconverted back to the original basis set using again cryst1. Method for objects of class ‘pdb’ first extract coordinates from the object using the function coords, perform the rotation, and update the coordinates of the ‘pdb’ object using the function coords<-.

Value

An object of the same class as x with rotated coordinates.

See Also

Helper functions for rotation around a given Cartesian vector:

Rx, Ry, Rz

Passing from Cartesian to fractional coordinates (or Vis Versa):

xyz2abc, abc2xyz

Examples

# First lets read a pdb file
x <- read.pdb(system.file("examples/PCBM_ODCB.pdb",package="Rpdb"))
cell <- cell.coords(x)
visualize(x, mode = NULL)
# Rotation of the structure around the c-axis
visualize(R(x, 90, x=cell["x","c"], y=cell["y","c"], z=cell["z","c"], mode = NULL))
# Rotation of the residue 1 around the c-axis
visualize(R(x, 90, x=cell["x","c"], y=cell["y","c"], z=cell["z","c"], mask=x$atoms$resid==1), mode = NULL)
Usage

RX(...)
RY(...)
RZ(...)

## S3 method for class 'coords'
RX(x, angle = 0, mask = TRUE, cryst1 = NULL, ...)
## S3 method for class 'coords'
RY(x, angle = 0, mask = TRUE, cryst1 = NULL, ...)
## S3 method for class 'coords'
RZ(x, angle = 0, mask = TRUE, cryst1 = NULL, ...)

## S3 method for class 'pdb'
RX(x, angle = 0, mask = TRUE, cryst1 = x$cryst1, ...)
## S3 method for class 'pdb'
RY(x, angle = 0, mask = TRUE, cryst1 = x$cryst1, ...)
## S3 method for class 'pdb'
RZ(x, angle = 0, mask = TRUE, cryst1 = x$cryst1, ...)

Arguments

x an R object containing atomic coordinates.
angle the angle of the rotation in degrees.
mask a logical vector indicating the set of coordinates to which the rotation has to be
applyed.
cryst1 an object of class ‘cryst1’ use to convert fractional into Cartesian coordinates
when need.
... further arguments passed to or from other methods.

Details

These functions are helper functions to perform a rotation around a specific Cartesian vector. All of
them call the R function.

Value

An object of the same class as x with rotated coordinates.

See Also

R and xyz2abc, abc2xyz for passing from Cartesian to fractional coordinates (or Vis Versa).

Examples

# First lets read a pdb file
x <- read.pdb(system.file("examples/PCBM_ODCB.pdb",package="Rpdb"))
cell <- cell.coords(x)
visualize(x, mode = NULL)
split.pdb

C rotation of the structure around the z-axis
visualize(Rz(x, 90), mode = NULL)
C rotation of the residue 1 around the c-axis
visualize(Rz(x, 90, mask=x$atoms$resid==1), mode = NULL)

---

### Description

split divides a ‘pdb’ object by groups of atoms defined by f. unsplit reverses the effect of split.

### Usage

```r
## S3 method for class 'pdb'
split(x, f, drop = FALSE, ...)
```

```r
## S3 method for class 'pdb'
unsplit(value, f, drop = FALSE, ...)
```

### Arguments

- **x**: an object of class ‘pdb’ to be divided into groups.
- **f**: a ‘factor’ in the sense that `as.factor(f)` defines the grouping, or a list of such factors in which case their interaction is used for the grouping.
- **drop**: logical indicating if levels that do not occur should be dropped (if `f` is a factor or a list).
- **value**: a list of ‘pdb’ objects compatible with a splitting of `x`. Recycling applies if the lengths do not match.
- **...**: further potential arguments passed to methods.

### Details

`split` produce a list of ‘pdb’ objects with the same `cryst1`, `title` and `remark` components as `x`. Only its `atoms` component is splitted while its `conect` component is cleaned to keep only the meaningful connectivity for each ‘pdb’ object of the list returned by the function. `unlist` produce a ‘pdb’ object with the same `cryst1`, `title` and `remark` components as the first element of `value`. The `atoms` and `conect` components of all the elements of `value` are combined by row.

### Value

The value returned from `split` is a list of ‘pdb’ objects containing the data for the groups of atoms. The components of the list are named by the levels of `f` (after converting to a factor, or if already a factor and `drop=TRUE`, dropping unused levels).

`unsplit` returns a ‘pdb’ object for which `split(x, f)` equals `value`. 
See Also

`split.unsplit.pdb`

Examples

```r
## Not run:

## Split a pdb file by residue IDs and write them into separated files
x <- read.pdb(system.file("examples/PCBM_ODCB.pdb", package="Rpdb"))
file.names <- paste0(x$atoms$resname,"_",x$atoms$resid,".pdb")
file.names <- unique(file.names)
pdb.resid <- split(x, x$atoms$resid)
useless <- mapply(write.pdb, pdb.resid, file.names)
```

```r
## End(Not run)
```
Value

Return a subsetted object of the same class as \( x \).

See Also

subset, pdb, atoms, reindex

Examples

```r
x <- read.pdb(system.file("examples/PCBM_ODCB.pdb", package="Rpdb"))
y <- subset(x, x$atoms$eleid %in% sample(x$atoms$eleid, 10))
  is(y)
y <- subset(x$atoms, x$atoms$eleid %in% sample(x$atoms$eleid, 10))
  is(y)
x <- coords(x)
y <- subset(x, x < 0)
  is(y)
```

toSymbols

**Atomic Symbols Converter**

Description

Converts character strings or atomic numbers into atomic symbols.

Usage

toSymbols(x, ...)

## S3 method for class 'integer'
toSymbols(x, ...)

## S3 method for class 'numeric'
toSymbols(x, ...)

## S3 method for class 'character'
toSymbols(x, nletters = 3, ...)

Arguments

- **x**
  - a vector to be converted into atomic symbols.

- **nletters**
  - an integer used to truncate the character strings before conversion.

- **...**
  - further arguments passed to or from other methods.
**Details**

Each element of x are converted into atomic symbols. When x is an integer (or numeric) vector, atomic number are search into the `elements` data set to find associated atomic symbols. When x is a character vector, `toSymbols` first removes all leading and trailing white spaces and numbers. Then translates the first character of the character strings to uppercase and all the others to lowercase. Finally, the character strings are tested for matching with element symbols provided by the `elements` data set. NA are produced for no matching.

**Value**

A character vector containing atomic symbols.

**See Also**

`elements`

**Examples**

```r
x <- c(1:10)
toSymbols(x)

x <- c("C", "O", "H", "H", "Si", "Si")
toSymbols(x)

# 'nletters' can be used to truncate the character strings before conversion, if need

toSymbols("Si", nletters=3) # return NA

toSymbols("Si", nletters=2) # return "Si"

toSymbols("Si", nletters=1) # return "S"
```

---

**translation**

*Translation of Atomic Coordinates*

**Description**

Translation of Cartesian or fractional coordinates.

**Usage**

```r
### Translation of Cartesian coordinates:
Txyz(...)

# S3 method for class 'coords'
Txyz(obj, x = 0, y = 0, z = 0, mask = TRUE, thickness = NULL, cryst1 = NULL, ...)

# S3 method for class 'pdb'
Txyz(obj, x = 0, y = 0, z = 0, mask = TRUE, thickness = NULL, cryst1 = obj$cryst1, ...)

### Translation of fractional coordinates:
```
Arguments

- **obj**: an R object containing atomic coordinates.
- **x**: the x-component of the translation vector.
- **y**: the y-component of the translation vector.
- **z**: the z-component of the translation vector.
- **a**: the a-component of the translation vector.
- **b**: the b-component of the translation vector.
- **c**: the c-component of the translation vector.
- **mask**: a logical vector indicating the set of coordinates to which to apply the translation.
- **thickness**: a numeric value indicating the fraction of the thickness of the selected atom to be added to the translation vector (Usually 0, 0.5 or 1. See details).
- **cryst1**: an object of class ‘cryst1’ use to convert Cartesian into fraction coordinates (or Vis Versa) when need.
- **...**: further arguments passed to or from other methods.

Details

Txyz and Tabc are generic functions. Method for objects of class ‘coords’ first convert the coordinates into Cartesian or fractional coordinates using cryst1 if needed to performed the translation. Once translated, the coordinates are reconvered back to the orginal basis set using again cryst1. Method for objects of class ‘pdb’ first extract coordinates from the object using the function coords, perform the translation, and update the coordinates of the ‘pdb’ object using the function coords<-.

The thickness argument can be use to translate selected atoms by a fraction of its thickness along the translation direction. This can be use when merging two fragments centered at the origin to build a dimer to avoid atomic overlap and set the inter-fragment distance (see examples).

Value

An object of the same class as x with translated coordinates.

See Also

Helper functions for translation along given Cartesian or lattice vector:

- **Tx, Ty, Tz, Ta, Tb, Tc**
- Passing from Cartesian to fractional coordinates (or Vis Versa):
  - **xyz2abc, abc2xyz**
Examples

# First lets read a pdb file
x <- read.pdb(system.file("examples/PCBM_ODCB.pdb", package="Rpdb"))
visualize(x, mode = NULL)
visualize(Txyz(x, y=10), mode = NULL)
visualize(Txyz(x, y=10, mask=x$atoms$resid==1), mode = NULL)
visualize(Tabc(x, b=1), mode = NULL)
visualize(Tabc(x, b=1, mask=x$atoms$resid==1), mode = NULL)

# Lets build a C70/Pentacene dimer with an inter-molecular distance equal to 3.5
C70 <- read.pdb(system.file("examples/C70.pdb", package="Rpdb"))
Pen <- read.pdb(system.file("examples/Pentacene.pdb",package="Rpdb"))
x <- merge(C70, Pen)
visualize(x, mode = NULL)
viewXY()
visualize(Txyz(x, x=0, y=0, z=3.5, mask=x$atoms$resname="C70", thickness=0.5), mode = NULL)
viewXY()

translationHelpers  

Helper Functions for Translation of Atomic Coordinates

Description

Translation of atomic coordinates along a specific Cartesian or lattice vector.

Usage

### Translation of Cartesian coordinates:
Tx(...)
Ty(...)
Tz(...)

## S3 method for class 'coords'
Tx(obj, x = 0, mask = TRUE, thickness = NULL, cryst1 = NULL, ...)

## S3 method for class 'coords'
Ty(obj, y = 0, mask = TRUE, thickness = NULL, cryst1 = NULL, ...)

## S3 method for class 'coords'
Tz(obj, z = 0, mask = TRUE, thickness = NULL, cryst1 = NULL, ...)

## S3 method for class 'pdb'
Tx(obj, x = 0, mask = TRUE, thickness = NULL, cryst1 = obj$cryst1, ...)

## S3 method for class 'pdb'
Ty(obj, y = 0, mask = TRUE, thickness = NULL, cryst1 = obj$cryst1, ...)

## S3 method for class 'pdb'
Tz(obj, z = 0, mask = TRUE, thickness = NULL, cryst1 = obj$cryst1, ...)

### Translation of fractional coordinates:
Ta(...)

# Translation of fractional coordinates along a specific Cartesian or lattice vector.

### S3 method for class 'coords'
Ta(x, y, z = 0, mask = FALSE, thickness = NULL, cryst1 = NULL, ...)

## S3 method for class 'pdb'
Ta(obj, x = 0, y = 0, z = 0, mask = FALSE, thickness = NULL, cryst1 = obj$cryst1, ...)

## S3 method for class 'pdb'
Ta(obj, x = 0, y = 0, z = 0, mask = FALSE, thickness = NULL, cryst1 = obj$cryst1, ...)

### Translation of fractional coordinates along a specific Cartesian or lattice vector.

### S3 method for class 'coords'
Ta(x, y, z = 0, mask = FALSE, thickness = NULL, cryst1 = NULL, ...)

## S3 method for class 'pdb'
Ta(obj, x = 0, y = 0, z = 0, mask = FALSE, thickness = NULL, cryst1 = obj$cryst1, ...)

## S3 method for class 'pdb'
Ta(obj, x = 0, y = 0, z = 0, mask = FALSE, thickness = NULL, cryst1 = obj$cryst1, ...)

### Translation of fractional coordinates along a specific Cartesian or lattice vector.

### S3 method for class 'coords'
Ta(x, y, z = 0, mask = FALSE, thickness = NULL, cryst1 = NULL, ...)

## S3 method for class 'pdb'
Ta(obj, x = 0, y = 0, z = 0, mask = FALSE, thickness = NULL, cryst1 = obj$cryst1, ...)

## S3 method for class 'pdb'
Ta(obj, x = 0, y = 0, z = 0, mask = FALSE, thickness = NULL, cryst1 = obj$cryst1, ...)

### Translation of fractional coordinates along a specific Cartesian or lattice vector.

### S3 method for class 'coords'
Ta(x, y, z = 0, mask = FALSE, thickness = NULL, cryst1 = NULL, ...)

## S3 method for class 'pdb'
Ta(obj, x = 0, y = 0, z = 0, mask = FALSE, thickness = NULL, cryst1 = obj$cryst1, ...)

## S3 method for class 'pdb'
Ta(obj, x = 0, y = 0, z = 0, mask = FALSE, thickness = NULL, cryst1 = obj$cryst1, ...)

### Translation of fractional coordinates along a specific Cartesian or lattice vector.

### S3 method for class 'coords'
Ta(x, y, z = 0, mask = FALSE, thickness = NULL, cryst1 = NULL, ...)

## S3 method for class 'pdb'
Ta(obj, x = 0, y = 0, z = 0, mask = FALSE, thickness = NULL, cryst1 = obj$cryst1, ...)

## S3 method for class 'pdb'
Ta(obj, x = 0, y = 0, z = 0, mask = FALSE, thickness = NULL, cryst1 = obj$cryst1, ...)
Tb(...)
Tc(...)

## S3 method for class 'coords'
Ta(obj, a = 0, mask = TRUE, cryst1 = NULL, ...)
## S3 method for class 'coords'
Tb(obj, b = 0, mask = TRUE, cryst1 = NULL, ...)
## S3 method for class 'coords'
Tc(obj, c = 0, mask = TRUE, cryst1 = NULL, ...)

## S3 method for class 'pdb'
Ta(obj, a = 0, mask = TRUE, cryst1 = obj$cryst1, ...)
## S3 method for class 'pdb'
Tb(obj, b = 0, mask = TRUE, cryst1 = obj$cryst1, ...)
## S3 method for class 'pdb'
Tc(obj, c = 0, mask = TRUE, cryst1 = obj$cryst1, ...)

Arguments

- **obj**: an R object containing atomic coordinates.
- **x**: the x-component of the translation vector.
- **y**: the y-component of the translation vector.
- **z**: the z-component of the translation vector.
- **a**: the a-component of the translation vector.
- **b**: the b-component of the translation vector.
- **c**: the c-component of the translation vector.
- **mask**: a logical vector indicating the set of coordinates to which to apply the translation.
- **thickness**: a numeric value indicating the fraction of the thickness of the selected atom to be added to the translation vector (Usually 0, 0.5 or 1. See details).
- **cryst1**: an object of class ‘cryst1’ use to convert Cartesian into fraction coordinates (or Vis Versa) when need.
- ... further arguments passed to or from other methods.

Details

These functions are helper functions to perform a translation along a specific Cartesian or lattice vector. All of them call either the Txyz or Tabc function.

Value

An object of the same class as x with translated coordinates.

See Also

- Txyz, Tabc
- Passing from Cartesian to fractional coordinates (or Vis Versa): xyz2abc, abc2xyz
Examples

```r
x <- read.pdb(system.file("examples/PCBM_ODCB.pdb", package="Rpdb"))
visualize(x, mode = NULL)
visualize(Ty(x, 10), mode = NULL)
visualize(Ty(x, 10, mask=x$atoms$resid==1), mode = NULL)
visualize(Tb(x, 1), mode = NULL)
visualize(Tb(x, 1, mask=x$atoms$resid==1), mode = NULL)

# Let's build a C70/Pentacene dimer with an inter-molecular distance equal to 3.5
C70 <- read.pdb(system.file("examples/C70.pdb", package="Rpdb"))
Pen <- read.pdb(system.file("examples/Pentacene.pdb", package="Rpdb"))
x <- merge(C70, Pen)
visualize(x, mode = NULL)
viewXY()
visualize(Tz(x, z=3.5, mask=x$atoms$resname=="C70", thickness=0.5), mode = NULL)
viewXY()
```

universalConstants  universal Constants

Description

This data set provides various universal constants.

Usage

```r
universalConstants
```

Format

A data frame containing for each universal constant the following information.

- **Quantity**: a character vector containing a short description of the constants.
- **Value**: a numeric vector containing the value of the constants.
- **Unit**: a character vector indicating the unit of the constants.

Source

http://www.ebyte.it/library/educards/constants/ConstantsOfPhysicsAndMath.html

Examples

```r
# Data for the speed of light
universalConstants["c",]

# Return the speed of light in m.s⁻¹
universalConstants["c","Value"]

# Return the Planck constant in J.s
```
unsplit

universalConstants["h","Value"]

unsplit

Reassemble Groups

Description

unsplit reverses the effect of split.

Usage

unsplit(value, f, drop = FALSE, ...)

Arguments

value a list of vectors or data frames compatible with a splitting of x. Recycling applies if the lengths do not match.

f a ‘factor’ in the sense that as.factor(f) defines the grouping, or a list of such factors in which case their interaction is used for the grouping.

drop logical indicating if levels that do not occur should be dropped (if f is a factor or a list).

... further potential arguments passed to methods.

Details

unsplit is a generic functions with a default method (Method dispatch takes place based on the class of the first element of value) working with lists of vectors or data frames (assumed to have compatible structure, as if created by split). It puts elements or rows back in the positions given by f. In the data frame case, row names are obtained by unsplitting the row name vectors from the elements of value.

f is recycled as necessary and if the length of x is not a multiple of the length of f a warning is printed.

Any missing values in f are dropped together with the corresponding values of x.

Value

Returns a vector or data frame for which split(x, f) equals value

References

vectorialOperations

See Also

cut to categorize numeric values.
strsplit to split strings.

Examples

```r
require(stats); require(graphics)
n <- 10; nn <- 100
g <- factor(round(n * runif(n * nn)))
x <- rnorm(n * nn) + sqrt(as.numeric(g))
xg <- split(x, g)
boxplot(xg, col = "lavender", notch = TRUE, varwidth = TRUE)
sapply(xg, length)
sapply(xg, mean)

### Calculate 'z-scores' by group (standardize to mean zero, variance one)
z <- unsplit(lapply(split(x, g), scale), g)

# or
zz <- x
split(zz, g) <- lapply(split(x, g), scale)

# and check that the within-group std dev is indeed one
tapply(z, g, sd)
tapply(zz, g, sd)

### data frame variation

## Notice that assignment form is not used since a variable is being added
g <- airquality$Month
l <- split(airquality, g)
l <- lapply(l, transform, Oz.Z = scale(Ozone))
aq2 <- unsplit(l, g)
head(aq2)
with(aq2, tapply(Oz.Z, Month, sd, na.rm=TRUE))

### Split a matrix into a list by columns
ma <- cbind(x = 1:10, y = (-4:5)^2)
split(ma, col(ma))
split(1:10, 1:2)
```
Description

Basic vectorial operations such as scalar product and vectorial product

Usage

```r
## Scalar product
dotProd(U, V)

## Norm of a vector
vectNorm(U)

## Rotation of the components of a vector
rotVect(U, n = 1)

## Vectorial product
vectProd(U, V)
```

Arguments

- `U`: a numeric vector of length 3.
- `V`: a numeric vector of length 3.
- `n`: an integer.

Value

- `dotProct`: return a single element numeric vector.
- `vectNorm`: return a single element numeric vector.
- `rotVect`: return a numeric vector of length 3.
- `vectProd`: return a numeric vector of length 3.

See Also

- `matmult`

Examples

```r
Vx <- c(3,0,0)
vectNorm(Vx)
Vx <- Vx/vectNorm(Vx)
Vy <- c(0,1,0)
Vz <- vectProd(Vx, Vy)
print(Vz)
```
viewAxis Set the View of the ‘rgl’ Scene

Description

Set the view of the current ‘rgl’ scene aligning one vector perpendicularly to the screen and placing another in the horizontal plan.

Usage

```r
viewAxis(V1, V2)
viewXY()
viewYZ()
viewZX()
viewAB(cryst1)
viewBC(cryst1)
viewCA(cryst1)
viewInertia(x, m = NULL)
```

Arguments

- **V1**: a length 3 numeric vector.
- **V2**: a length 3 numeric vector.
- **cryst1**: an object of class ‘cryst1’.
- **x**: an R object containing atomic coordinates.
- **m**: a numeric vector containing atomic masses.

Details

`viewAxis` set the view of the current rgl scene (by setting `userMatrix`). See `par3d` for more details) so that V1 is perpendicular to the screen and V2 is in the horizontal plan. The other functions documented here are helper functions calling `viewAxis` to set the view using particular Cartesian or lattice vectors. For functions `viewAB`, `viewBC` and `viewCA` a ‘cryst1’ object has to be specified to defined the lattice vectors used to set the view. The function `viewInertia` computes the inertia tensor from atomic coordinates and masses (see `inertia`) and set the view to its eigen vectors basis set.

See Also

- `visualize`
- `cell.coords`
- `par3d`
- `rgl.cur`
Examples

```r
x <- read.pdb(system.file("examples/PCBM_ODCB.pdb", package="Rpdb"))
visualize(x, mode = NULL)
viewAB(x$cryst1)

c70 <- read.pdb(system.file("examples/C70.pdb", package="Rpdb"))
visualize(c70, mode = NULL)
viewXY()
viewInertia(c70)
```

**visualize**

*Visualize a Molecular Structure*

**Description**

Use the rgl library to visualize in 3D a molecular structure.

**Usage**

```r
visualize(...)
```

```
## S3 method for class 'coords'
visualize(x, elename = NULL, cryst1 = NULL, conect = NULL, mode = NULL,
type = "1", xyz = NULL, abc = NULL, pbc.box = NULL, lwd = 2,
lwd.xyz = lwd, lwd.abc = lwd, lwd.pbc.box = lwd,
cex.xyz = 2, cex.abc = 2, col = NULL, bg = "#FAFAD2", radii = "rvdw",
add = FALSE, windowRect = c(0,0,800,600), FOV = 0, userMatrix=diag(4), ...)
```

```
## S3 method for class 'data.frame'
visualize(x, elename = NULL, cryst1 = NULL, conect = NULL, mode = NULL,
type = "1", xyz = NULL, abc = NULL, pbc.box = NULL, lwd = 2,
lwd.xyz = lwd, lwd.abc = lwd, lwd.pbc.box = lwd,
cex.xyz = 2, cex.abc = 2, col = NULL, bg = "#FAFAD2", radii = "rvdw",
add = FALSE, windowRect = c(0,0,800,600), FOV = 0, userMatrix=diag(4), ...)
```

```
## S3 method for class 'matrix'
visualize(x, elename = NULL, cryst1 = NULL, conect = NULL, mode = NULL,
type = "1", xyz = NULL, abc = NULL, pbc.box = NULL, lwd = 2,
lwd.xyz = lwd, lwd.abc = lwd, lwd.pbc.box = lwd,
cex.xyz = 2, cex.abc = 2, col = NULL, bg = "#FAFAD2", radii = "rvdw",
add = FALSE, windowRect = c(0,0,800,600), FOV = 0, userMatrix=diag(4), ...)
```

```
## S3 method for class 'atoms'
visualize(x, cryst1 = NULL, conect = NULL, mode = NULL,
type = "1", xyz = NULL, abc = NULL, pbc.box = NULL, lwd = 2,
lwd.xyz = lwd, lwd.abc = lwd, lwd.pbc.box = lwd,
cex.xyz = 2, cex.abc = 2, col = NULL, bg = "#FAFAD2", radii = "rvdw",
```
Arguments

x

an object or the name of a PDB file containing the molecular structure to visualize.
nename
~
a character vector containing the atomic names used to chose atom colors and radii.
cryst1
~
an object of class ‘cryst1’. See cryst1
conect
~
an object of class ‘conect’. See conect
mode
~
a single element character vector indicating the visualization mode (See details).
type
~
a character string indicating the visualization style (See details).
xyz
~
a logical value indicating whether the x, y and z axes have to be added to the scene. See details
abc
~
a logical value indicating whether the a, b and c axes have to be added to the scene. See details
pbc.box
~
a logical value indicating whether the pbc box has to be added to the scene. See details
lwd
~
a numeric value indication the line width used to plot the axes, the pbc box and atomic bonds when type = "l" (see details).
lwd.xyz
~
a numeric value indicating the line width used to plot the x, y and z axes.
lwd.abc
~
a numeric value indicating the line width used to plot the a, b and c axes.
lwd.pbc.box
~
a numeric value indicating the line width used to plot the pbc box.
cex.xyz
~
a numeric value indicating the magnification used to plot the labels of the x, y and z axes.
cex.abc
~
a numeric value indicating the magnification used to plot the labels of the a, b and c axes.
col
~
a vector indicating the colors to use to plot each atom.
bg
~
the color of the background
radii either a character string indicating the type of radii or a numeric vector specifying the radii of each atom to use to plot atoms as spheres (see details).

add a logical value indicating whether the plot has be to added to a existing scene (see rgl.cur and open3d).

windowRect a vector of four integers indicating the left, top, right and bottom of the displayed window in pixels (see par3d).

FOV the field of view. This controls the degree of parallax in the perspective view (see par3d).

userMatrix a 4 by 4 matrix describing user actions to display the scene (see par3d).

... further arguments passed to or from other methods.

Details

Three different visualization styles are allowed.

- When type="p": Points are drawn at each atomic positions (very light visualization mode).
- When type="l": Lines are drawn between bonded atoms. The connectivity of the system has to be specified.
- When type="s": Spheres are drawn at each atomic positions (heavy visualization mode).
  The radii of the spheres are given by radii.
  - When radii="rcov": Covalent radii, taken from the elements data set, are used.
  - When radii="rvdw": Van der Waals radii, taken from the elements data set, are used.
  - When radii is a numeric vector: The numeric values are used to assign to each atom a radius. If length(radii) != natom(pdb) radii is recycled.

When xyz, abc or pbcNbox are NULL, the axis or pbc box are are added depending if a 'cryst1' object can be found.

Two different interactive visualization modes are available:

- When mode="measure": bond lengths, angles and dihedrals can be measured by right-clicking on the atoms.
- When mode="info": atomic labels can be added to the scene by right-clicking on the atoms. The labels are as follow: "ResidResname:EleidElename"

When mode=NULL the interactive mode is disabled. To escape the interactive mode press the ESC key.

Value

Return (using invisible) a two-column data.frame containing the IDs and type indicators of the objects added to the scene.

See Also

addXYZ, addABC, addPBCBox, par3d, select3d, measure, info3d
Examples

x <- read.pdb(system.file("examples/PCBM_ODCB.pdb", package="Rpdb"))
visualize(x, type = "l", mode = NULL)
visualize(x, type = "s", radii = "rcov", mode = NULL)
visualize(x, type = "s", radii = "rvdw", mode = NULL)
visualize(x, type = "p", mode = NULL)
visualize(subset(x, resid != 1), type = "l", mode = NULL)
visualize(subset(x, resid == 1), type = "s", add = TRUE, mode = NULL)

wrap Atomic Coordinates

Description

Wraps atomic coordinates using periodic boundary conditions.

Usage

wrap(x, ...)

## S3 method for class 'coords'
wrap(x, cryst1, factor = NULL, ...)

## S3 method for class 'atoms'
wrap(x, cryst1, factor = NULL, ...)

## S3 method for class 'pdb'
wrap(x, cryst1 = x$cryst1, factor = NULL, ...)

Arguments

x       an R object containing atomic coordinates to be wrapped.
cryst1  an object of class 'cryst1' containing periodic boundary conditions used for wrapping.
factor  a factor used to wrap the atoms by groups
...     further arguments passed to or from other methods.

Details

The wrap function translates all atoms out of the unit cell back into the unit cell using periodic boundary conditions. To do so, the wrap function first converts Cartesian into fractional coordinates. Then atoms with fractional coordinates greater than 1 or lower than 0 are respectively translated by -1 or +1. Finally, if the original atomic coordinates were Cartesian coordinates their are reconverted into Cartesian coordinates.
write.pdb

Value

Return a object of class ‘pdb’ with wrapped atomic coordinates.

See Also

coords, atoms.pdb, cryst1, centres.pdb, xyz2abc.

Examples

x <- read.pdb(system.file("examples/PCBM_ODCB.pdb",package="Rpdb"))

# Translation of the atoms along x-axis
x$atoms$x1 <- x$atoms$x1 + 10

# Wrapping the structure
y <- wrap(x)

write.pdb

PDB File Writer

Description

Writes a Protein Data Bank (PDB) coordinate file from an object of class ‘pdb’.

Usage

write.pdb(x, file = "Rpdb.pdb")

Arguments

x an object, or a list of objects, of class ‘pdb’.

file a single element character vector containing the name of the PDB file to be created.

Details

All data stored in the ‘pdb’ object are written on a PDB file. A list of object of class ‘pdb’ can be provided to write multiple MODEL into a single file. In this case, each ‘pdb’ object of the list have to have the same cryst1 and conect components.

To write only a subset of a ‘pdb’ object see function subset.pdb.

References

PDB format has been taken from: http://www.wwpdb.org/documentation/format33/v3.3.html
xyz2abc

From Cartesian to Fractional Coordinates and Vis Versa

Description

Converts Cartesian coordinates into fractional coordinates and vice versa.

Usage

xyz2abc(...)
abc2xyz(...)

## S3 method for class 'coords'
xyz2abc(x, cryst1, ...)
## S3 method for class 'coords'
abc2xyz(x, cryst1, ...)

## S3 method for class 'atoms'
xyz2abc(x, cryst1, ...)
## S3 method for class 'atoms'
abc2xyz(x, cryst1, ...)

## S3 method for class 'pdb'
xyz2abc(x, cryst1 = x$cryst1, ...)
## S3 method for class 'pdb'
abc2xyz(x, cryst1 = x$cryst1, ...)

## S3 method for class 'distances'
xyz2abc(x, cryst1, ...)
## S3 method for class 'distances'
abc2xyz(x, cryst1, ...)

See Also

read.pdb, pdb, cryst1, atoms, conect, subset.pdb

Examples

## Read a PDB file included with the package
pdb <- read.pdb(system.file("examples/PCBM_ODCB.pdb", package="Rpdb"))

## Write the pdb object in file "Rpdb.pdb" into the current directory
write.pdb(pdb, file = "Rpdb.pdb")
Arguments

... arguments passed to methods.
x an R object containing atomic coordinates.
crst1 an object of class cryst1.

Details

For atoms and pdb objects, the atomic coordinates are first extracted from x using the coords function. Then, using the periodic boundary conditions stored into crst1, the coordinates are converted from Cartesian to fractional (for the xyz2abc functions) or from fractional to Cartesian (for the abc2xyz functions) coordinates. Finally, for atoms and pdb objects, the new atomic coordinates are reassigned to the original x object using the coords<- function and x is returned.

Value

Return an object of the same class as x, with atomic coordinates expressed in a different basis set.

See Also

basis, coords, atoms, pdb, cryst1

Examples

x <- read.pdb(system.file("examples/PCBM_ODCB.pdb",package="Rpdb"))
basis(x)
x <- xyz2abc(x)
basis(x)
x <- abc2xyz(x)
basis(x)

## Not run:

# This example return an error because the coordinates stored
# into the PDB file are already Cartesian coordinates.
x <- read.pdb(system.file("examples/PCBM_ODCB.pdb",package="Rpdb"))
x <- abc2xyz(x)

## End(Not run)
Index

*Topic IO
read.pdb, 32
write.pdb, 55

*Topic attribute
basis, 8

*Topic category
split.pdb, 39
unsplit, 47

*Topic classes
atoms, 6
conect, 14
cords, 16
crstl, 18
distances, 19
pdb, 30

*Topic datasets
elements, 21
universalConstants, 46

*Topic dynamic
addAxes, 4
addLabels, 5
bond-angle-dihedral, 9
viewAxis, 50
visualize, 51

*Topic manip
bond-angle-dihedral, 9
cellProperties, 11
centres, 12
cords, 16
distances, 19
inertia, 22
masses, 23
merge.coords, 24
mirror, 26
mirrorHelpers, 27
naton, 29
range.coords, 31
reindex, 34
replicate, 35
rotation, 36
rotationHelpers, 37
subset.pdb, 40
toSymbols, 41
translation, 42
translationHelpers, 44
vectorialOperations, 48
wrap, 54
xyz2abc, 56

*Topic package
Rpdb-package, 2

abc2xyz, 21, 27, 28, 37, 38, 43, 45
abc2xyz (xyz2abc), 56
addABC, 53
addABC (addAxes), 4
addAxes, 4
addEleLab (addLabels), 5
addLabels, 4, 5
addPBCBox, 53
addPBCBox (addAxes), 4
addResLab (addLabels), 5
addXYZ, 53
addXYZ (addAxes), 4
angle (bond-angle-dihedral), 9
as.factor, 39, 47
atoms, 6, 9, 13, 25, 29, 31–33, 35, 36, 41, 55–57

basis, 8, 8, 17, 21, 25, 57
basis<-b (basis), 8
bond (bond-angle-dihedral), 9
bond-angle-dihedral, 9
cell.coords, 19, 50
cell.coords (cellProperties), 11
cell-density (cellProperties), 11
cell-volume (cellProperties), 11
cellVolume, 11
centres, 12
INDEX

centres.pdb, 55
conect, 14, 31, 33, 32, 56
coords, 8–10, 13, 16, 21, 25, 29, 31, 32, 36, 55, 57
cords<- (coords), 16
cryst1, 12, 18, 31, 33, 36, 55–57
cut, 48
data.frame, 32
dihedral (bond-angle-dihedral), 9
distances, 19
dotProd (vectorialOperations), 48
elements, 13, 21, 42
factor, 13, 29
inertia, 22, 50
info3d, 10
info3d (addLabels), 5
is.atoms (atoms), 6
is.conect (conect), 14
is.coords (coords), 16
is.cryst1 (cryst1), 18
is.distances (distances), 19
is.pdb (pdb), 30
Mab, 27
Mab (mirrorHelpers), 27
masses, 23, 23
matmult, 49
Mbc, 27
Mbc (mirrorHelpers), 27
Mca, 27
Mca (mirrorHelpers), 27
measure, 6
measure (bond-angle-dihedral), 9
merge.atoms (merge.coords), 24
merge.coords, 24
merge.pdb (merge.coords), 24
mirror, 26, 28
mirrorHelpers, 27
Mxy, 27
Mxy (mirrorHelpers), 27
Myz, 27
Myz (mirrorHelpers), 27
Mzx, 27
Mzx (mirrorHelpers), 27
NA, 32

natom, 29
norm (distances), 19
par3d, 4, 50
pdb, 6, 8–10, 12, 13, 16, 19, 25, 29, 30, 32, 33, 35, 36, 40, 41, 55–57
R, 38
R (rotation), 36
range.atoms (range.coords), 31
range.coords, 31
range.pdb (range.coords), 31
read.pdb, 31, 32, 56
reindex, 25, 34, 41
replicate, 35
rgl.cur, 4, 50
rgl.texts, 5
rotation, 36
rotationHelpers, 37
rotVect (vectorialOperations), 48
Rpdb (Rpdb-package), 2
Rpdb-package, 2
Rx, 37
Ry (rotationHelpers), 37
Ry, 37
Rz (rotationHelpers), 37
Rz, 37
Rz (rotationHelpers), 37
split, 13, 29, 40
split.pdb, 39
strsplit, 48
subset, 40, 41
subset.atoms, 35
subset.atoms (subset.pdb), 40
subset.pdb, 35, 37, 40, 41, 55, 56

Ta, 43
Ta (translationHelpers), 44
Tabc, 45
Tabc (translation), 42
Tb, 43
Tb (translationHelpers), 44
Tc, 43
Tc (translationHelpers), 44
toSymbols, 23, 24, 41
translation, 42
translationHelpers, 44
Tx, 43
Tx (translationHelpers), 44
Txyz, 45
Txyz (translation), 42
Ty, 43
Ty (translationHelpers), 44
Tz, 43
Tz (translationHelpers), 44

universalConstants, 46
unsplit, 13, 40, 47
unsplit.pdb (split.pdb), 39

vectNorm (vectorialOperations), 48
vectorialOperations, 48
vectProd (vectorialOperations), 48
viewAB (viewAxis), 50
viewAxis, 50
viewBC (viewAxis), 50
viewCA (viewAxis), 50
viewInertia, 23
viewInertia (viewAxis), 50
viewXY (viewAxis), 50
viewYZ (viewAxis), 50
viewZX (viewAxis), 50
visualize, 4, 6, 10, 50, 51

wrap, 54
write.pdb, 33, 55

xyz2abc, 12, 21, 27, 28, 37, 38, 43, 45, 55, 56