ChemmineR

March 24, 2012

`AP-class` *Class "AP"*

Description

Container for storing the atom pair descriptors of a single compound as numeric vector. The atom pairs are used as structural similarity measures and for compound similarity searching.

Objects from the Class

Objects can be created by calls of the form `new("AP", ...)`.

Slots

`AP`**:** Object of class `"numeric"`

Methods

ap `signature(x = "AP")`: returns atom pairs as numeric vector

coerce `signature(from = "APset", to = "AP")`: `as(apset, "AP")`

show `signature(object = "AP")`: prints summary of `AP`

Author(s)

Thomas Girke

References

Chen X and Reynolds CH (2002). "Performance of similarity measures in 2D fragment-based similarity searching: comparison of structural descriptors and similarity coefficients", J Chem Inf Comput Sci.

See Also

Related classes: SDF, SDFset, SDFstr, APset.

Functions: `SDF2apcmp`, `apset2descdb`, `cmp.search`, `cmp.similarity`

Examples

```
showClass("AP")

## Instance of SDFset class
data(sdfsample)
sdfset <- sdfsample[1:50]
sdf <- sdfsample[[1]]

## Compute atom pair library
ap <- sdf2ap(sdf)
(apset <- sdf2ap(sdfset))
view(apset[1:4])

## Return main components of APset object
cid(apset[1:4]) # compound IDs
ap(apset[1:4]) # atom pair descriptors

## Return atom pairs in human readable format
db.explain(apset[1])

## Coerce APset to other objects
apset2descdb(apset) # returns old list-style AP database
tmp <- as(apset, "list") # returns list
as(tmp, "APset") # converst list back to APset

## Compound similarity searching with APset
cmp.search(apset, apset[1], type=3, cutoff=0.2)
plot(sdfset[names(cmp.search(apset, apset[6], type=2, cutoff=0.4))])

## Identify compounds with identical AP sets
cmp.duplicated(apset, type=2)

## Structure similarity clustering
cmp.cluster(db=apset, cutoff = c(0.65, 0.5))[1:20,]
```

`APset-class` *Class "APset"*

Description

List-like container for storing the atom pair descriptors of a many compounds as objects of class `AP`. This container is used for structure similarity searching of compounds.

Objects from the Class

Objects can be created by calls of the form `new("APset", ...)`.

Slots

`AP`: Object of class `"list"`

`ID`: Object of class `"character"`

Methods

[`signature(x = "APset")`: subsetting of class with bracket operator

[[`signature(x = "APset")`: returns single component as `AP` object

[[<- `signature(x = "APset")`: replacement method for single `AP` component

[<- `signature(x = "APset")`: replacement method for several `AP` components

ap `signature(x = "APset")`: returns atom pair list from AP slot

c `signature(x = "APset")`: concatenates two `APset` containers

cid `signature(x = "APset")`: returns all compound identifiers from ID slot

cid<- `signature(x = "APset")`: replacement method for compound identifiers in ID slot

coerce `signature(from = "APset", to = "AP")`: `as(apset, "AP")`

coerce `signature(from = "APset", to = "list")`: `as(apset, "list")`

coerce `signature(from = "list", to = "APset")`: `as(list, "APset")`

length `signature(x = "APset")`: returns number of entries stored in object

show `signature(object = "APset")`: prints summary of `APset`

view `signature(x = "APset")`: prints extended summary of `APset`

Author(s)

Thomas Girke

References

Chen X and Reynolds CH (2002). "Performance of similarity measures in 2D fragment-based similarity searching: comparison of structural descriptors and similarity coefficients", in J Chem Inf Comput Sci.

See Also

Related classes: SDF, SDFset, SDFstr, AP.

Functions: `SDF2apcmp`, `apset2descdb`, `cmp.search`, `cmp.similarity`

Examples

```
showClass("APset")

## Instance of SDFset class
data(sdfsample)
sdfset <- sdfsample[1:50]
sdf <- sdfsample[[1]]

## Compute atom pair library
ap <- sdf2ap(sdf)
(apset <- sdf2ap(sdfset))
view(apset[1:4])

## Return main components of APset object
cid(apset[1:4]) # compound IDs
ap(apset[1:4]) # atom pair descriptors

## Return atom pairs in human readable format
```

```
db.explain(apset[1])

## Coerce APset to other objects
apset2descdb(apset) # returns old list-style AP database
tmp <- as(apset, "list") # returns list
as(tmp, "APset") # converst list back to APset

## Compound similarity searching with APset
cmp.search(apset, apset[1], type=3, cutoff=0.2)
plot(sdfset[names(cmp.search(apset, apset[6], type=2, cutoff=0.4))])

## Identify compounds with identical AP sets
cmp.duplicated(apset, type=2)

## Structure similarity clustering
cmp.cluster(db=apset, cutoff = c(0.65, 0.5))[1:20,]
```

SDF-class *Class "SDF"*

Description

Container for storing every element of a single molecule defined in an SD/MOL file without information loss in a list-like container. The import occurs via the `SDFstr` container class. The header block is stored as named character vector, the atom/bond blocks as matrices and the data block as named character vector.

Objects from the Class

Objects can be created by calls of the form `new("SDF", ...)`.

Slots

`header`**:** Object of class `"character"`

`atomblock`**:** Object of class `"matrix"`

`bondblock`**:** Object of class `"matrix"`

`datablock`**:** Object of class `"character"`

Methods

[`signature(x = "SDF")`: subsetting of class with bracket operator

[[`signature(x = "SDF")`: returns one of the four object components

[[<- `signature(x = "SDF")`: replacement method for the four sub-components

[<- `signature(x = "SDF")`: replacement method for the four sub-components

atomblock `signature(x = "SDF")`: returns atom block as matrix

atomcount `signature(x = "SDF")`: returns atom frequency

bondblock `signature(x = "SDF")`: returns bond block as matrix

coerce `signature(from = "character", to = "SDF")`: `as(character, "SDF")`

coerce `signature(from = "list", to = "SDF"): as(list, "SDF")`

coerce `signature(from = "SDF", to = "character"): as(sdf, "character")`

coerce `signature(from = "SDF", to = "list"): as(sdf, "list")`

coerce `signature(from = "SDF", to = "SDFset"): as(sdf, "SDFset")`

coerce `signature(from = "SDF", to = "SDFstr"): as(SDF, "SDFstr")`

coerce `signature(from = "SDFset", to = "SDF"): as(sdfset, "SDF")`

datablock `signature(x = "SDF")`: returns data block as named character vector

datablocktag `signature(x = "SDF")`: returns data block as named character vector with subsetting support

header `signature(x = "SDF")`: returns header block as named character vector

plot `signature(x = "SDF")`: plots molecule structure for `SDF` object

sdf2list `signature(x = "SDF")`: returns `SDF` object as `list`

sdf2str `signature(sdf = "SDF")`: returns `SDF` object as `character` vector

sdfid `signature(x = "SDF")`: returns molecule ID field from header block

show `signature(object = "SDF")`: prints summary of `SDF`

Author(s)

Thomas Girke

References

SDF format definition: http://www.symyx.com/downloads/public/ctfile/ctfile.jsp

See Also

Related classes: SDFset, SDFstr, AP, APset

Examples

```
showClass("SDF")

## Instances of SDF class
data(sdfsample); sdfset <- sdfsample
(sdf <- sdfset[[1]]) # returns first molecule in sdfset as SDF object

## Accessing SDF components
header(sdf); atomblock(sdf); bondblock(sdf); datablock(sdf)
sdfid(sdf)

## Plot molecule structure of SDF
plot(sdf) # plots to R graphics device
# sdf.visualize(sdf) # viewing in browser
```

SDF2apcmp *'SDF' to 'list' for AP generation*

Description

Returns SDF class as list containing the components for generating atom pair descriptors.

Usage

```
SDF2apcmp(SDF)
```

Arguments

SDF	SDF

Details

...

Value

list	with atom and bond components

Author(s)

Thomas Girke

References

Chen X and Reynolds CH (2002). "Performance of similarity measures in 2D fragment-based similarity searching: comparison of structural descriptors and similarity coefficients", J Chem Inf Comput Sci.

See Also

Functions: sdf2ap, apset2descdb, cmp.search, cmp.similarity

Examples

```
## Instances of SDFset class
data(sdfsample)
sdf <- sdfsample[[1]]

## Return list
SDF2apcmp(sdf)
```

SDFset-class *Class "SDFset"*

Description

List-like container for storing one or many objects of class SDF each containing the structure definition information of molecules provided by an SD/MOL file. The SDFset is the most important class in the ChemmmineR package for accessing and manipulating information stored in SD files.

Objects from the Class

Objects can be created by calls of the form new("SDFset", ...).

Slots

SDF: Object of class "list" storing SDF components

ID: Object of class "character" storing compound identifiers

Methods

[signature(x = "SDFset"): subsetting of class with bracket operator

[[signature(x = "SDFset"): returns single component as SDF object

[[<- signature(x = "SDFset"): replacement method for single SDF component

[<- signature(x = "SDFset"): replacement method for several SDF components

atomblock signature(x = "SDFset"): returns all atom blocks as list

atomcount signature(x = "SDFset"): returns all atom frequencies as list

bondblock signature(x = "SDFset"): returns all bond blocks as list

c signature(x = "SDFset"): concatenates two SDFset containers

cid signature(x = "SDFset"): returns all compound identifiers from ID slot

header<- signature(x = "SDFset"): replacement method for header block

atomblock<- signature(x = "SDFset"): replacement method for atom block

bondblock<- signature(x = "SDFset"): replacement method for bond block

datablock<- signature(x = "SDFset"): replacement method for data block

coerce signature(from = "list", to = "SDFset"): as(list, "SDFset")

coerce signature(from = "SDF", to = "SDFset"): as(sdf, "SDFset")

coerce signature(from = "SDFset", to = "list"): as(sdfset, "list")

coerce signature(from = "SDFset", to = "SDF"): as(sdfset, "SDF")

coerce signature(from = "SDFset", to = "SDFstr"): as(sdfset, "SDFstr")

coerce signature(from = "SDFstr", to = "SDFset"): as(sdfstr, "SDFset")

datablock signature(x = "SDFset"): returns all data blocks as list

datablocktag signature(x = "SDFset"): returns all data blocks as named as list with subsetting support

header signature(x = "SDFset"): returns all header blocks as list

length signature(x = "SDFset"): returns number of entries stored in object

plot `signature(x = "SDFset")`: plots one or many molecule structures from `SDFset` object

sdfid `signature(x = "SDFset")`: returns molecule ID field from header block

SDFset2list `signature(x = "SDFset")`: returns `SDFset` object as `list`

SDFset2SDF `signature(x = "SDFset")`: returns `SDFset` object as `list` with `SDF` components

SDFset2SDF<- `signature(x = "SDFset")`: replacement method for `SDFset` component in `SDFset` using accessor method

show `signature(object = "SDFset")`: prints summary of `SDFset`

view `signature(x = "SDFset")`: prints extended summary of `SDFset`

SDFset `SDFset(SDF, ID)`: interface to `SDFset` constructor

Author(s)

Thomas Girke

References

SDF format definition: http://www.symyx.com/downloads/public/ctfile/ctfile.jsp

See Also

Related classes: SDF, SDFstr, AP, APset

Import function: read.SDFset("some_SDF_file")

Export function: write.SDF(sdfset, "some_file.sdf")

Examples

```
showClass("SDFset")

## Instances of SDFset class
data(sdfsample); sdfset <- sdfsample
sdfset; view(sdfset[1:4])
sdfset[[1]]

## Import and store SD File in SDFset container
# sdfset <- read.SDFset("some_SDF_file")

## Miscellaneous accessor methods
header(sdfset[1:4])
atomblock(sdfset[1:4])
atomcount(sdfset[1:4])
bondblock(sdfset[1:4])
datablock(sdfset[1:4])

## Assigning compound IDs and keeping them unique
cid(sdfset); sdfid(sdfset)
unique_ids <- makeUnique(sdfid(sdfset))
cid(sdfset) <- unique_ids

## Convert data block to matrix
blockmatrix <- datablock2ma(datablocklist=datablock(sdfset)) # Converts data block to mat
numchar <- splitNumChar(blockmatrix=blockmatrix) # Splits to numeric and character matrix
```

```
numchar[[1]][1:4,]; numchar[[2]][1:4,]

## Compute atom frequency matrix, molecular weight and formula
propma <- data.frame(MF=MF(sdfset), MW=MW(sdfset), atomcountMA(sdfset))
propma[1:4, ]

## Assign matrix data to data block
datablock(sdfset) <- propma
view(sdfset[1:4])

## String Searching in SDFset
grepSDFset("650001", sdfset, field="datablock", mode="subset") # To return index, set mod

## Export SDFset to SD file
# write.SDF(sdfset[1:4], file="sub.sdf", sig=TRUE)

## Plot molecule structure of SDF
plot(sdfset[1:4]) # plots to R graphics device
# sdf.visualize(sdfset[1:4]) # viewing in browser
```

`SDFset2SDF` *'SDFset' to list with many 'SDF'*

Description

Returns object of class `SDFset` as `list` were each component consists of an `SDF` object.

Usage

```
SDFset2SDF(x)
```

Arguments

`x`	object of class `SDFset`

Details

...

Value

`list`	containing one or many `SDF` objects

Author(s)

Thomas Girke

References

SDF format definition: http://www.symyx.com/downloads/public/ctfile/ctfile.jsp

See Also

Functions: `sdfstr2list`, `sdf2str`, `sdf2list`, `SDFset2list`

Examples

```
## Instance of SDFset class
data(sdfsample); sdfset <- sdfsample
sdfset

## Returns sdfset as list
SDFset2SDF(sdfset[1:4])
as(sdfset, "SDF")[1:4] # similar result
view(sdfset[1:4]) # same result
```

SDFset2list *'SDFset' to 'list'*

Description

Returns object of class `SDFset` as `list` where each component conists of a `list` of the four SDF sub-components: header block, atom block, bond block and data block.

Usage

```
SDFset2list(x)
```

Arguments

`x`	object of class `SDFset`

Details

...

Value

`list`	containing one or many lists each with following components:
`character`	SDF header block
`matrix`	SDF bond block
`matrix`	SDF atom block
`character`	SDF data block

Author(s)

Thomas Girke

References

SDF format definition: http://www.symyx.com/downloads/public/ctfile/ctfile.jsp

See Also

Functions: `sdfstr2list`, `sdf2str`, `sdf2list`, `SDFset2SDF`

Examples

```
## Instance of SDFset class
data(sdfsample); sdfset <- sdfsample
sdfset

## Returns sdfset as list
SDFset2list(sdfset[1:4])
as(sdfset, "list")[1:4] # similar result
```

SDFstr-class *Class "SDFstr"*

Description

List-like container for storing one or many molecules from an SD (or MOL) file. Each component of an `SDFstr` object stores the SD data line by line from a single molecule in a character vector. The `SDFstr` class is an intermediate container to import SD files into the more important `SDFset` object or to export the data back from an `SDFset` container to a valid SD file.

Objects from the Class

Objects can be created by calls of the form `new("SDFstr", ...)`.

Slots

a: Object of class `"list"` with `character` components

Methods

[`signature(x = "SDFstr")`: subsetting of class with bracket operator

[[`signature(x = "SDFstr")`: returns single component as character vector

[[<- `signature(x = "SDFstr")`: replacement method for single `SDFstr` component

[<- `signature(x = "SDFstr")`: replacement method for several `SDFstr` components

coerce `signature(from = "character", to = "SDFstr"): as(character, "SDFstr")`

coerce `signature(from = "list", to = "SDFstr"): as(list, "SDFstr")`

coerce `signature(from = "SDF", to = "SDFstr"): as(sdf, "SDFstr")`

coerce `signature(from = "SDFset", to = "SDFstr"): as(sdfset, "SDFstr")`

coerce `signature(from = "SDFstr", to = "list"): as(sdfstr, "list")`

coerce `signature(from = "SDFstr", to = "SDFset"): as(sdfstr, "SDFset")`

length `signature(x = "SDFstr")`: returns length of `SDFstr`

sdfstr2list `signature(x = "SDFstr")`: accessor method to return `SDFstr` as `list`

sdfstr2list<- `signature(x = "SDFstr")`: replacement method for several `SDFstr` components

show `signature(object = "SDFstr")`: prints summary of `SDFstr`

Author(s)

Thomas Girke

References

SDF format definition: http://www.symyx.com/downloads/public/ctfile/ctfile.jsp

See Also

Related classes: SDFset, AP, APset

Import function: read.SDFstr("some_SDF_file")

Examples

```
showClass("SDFstr")

## Instances of SDFstr class
data(sdfsample); sdfset <- sdfsample
sdfstr <- as(sdfset, "SDFstr")
sdfstr[1:4] # print summary of container content
sdfstr[[1]] # returns character vector

## Import: sdfstr <- read.SDFstr("some_SDF_file")
## Export: write.SDF(sdfstr, "some_file.sdf")
```

---

ap *Return atom pair component of 'AP/APset'*

Description

Returns atom pair component of objects of class `AP` or `APset` as list of vectors.

Usage

```
ap(x)
```

Arguments

`x`	Object of class `AP` and `APset`

Details

...

Value

`List`	with one to many of following components:
`numeric`	atom pairs

Author(s)

Thomas Girke

References

Chen X and Reynolds CH (2002). "Performance of similarity measures in 2D fragment-based similarity searching: comparison of structural descriptors and similarity coefficients", J Chem Inf Comput Sci.

See Also

Functions: `SDF2apcmp`, `apset2descdb`, `cmp.search`, `cmp.similarity`

Examples

```
## Instance of SDFset class
data(sdfsample)
sdfset <- sdfsample[1:50]
sdf <- sdfset[[1]]

## Compute atom pair library
ap <- sdf2ap(sdf)
(apset <- sdf2ap(sdfset))
view(apset[1:4])

## Return main components of APset object
cid(apset[1:4]) # compound IDs
ap(apset[1:4]) # atom pair descriptors

## Return atom pairs in human readable format
db.explain(apset[1])
```

`apset` *Atom pairs stored in 'APset' object*

Description

Atom pairs for 100 molecules stored in `sdfsample`.

Usage

```
data(apset)
```

Format

Object of class `apset`

Details

Object stores atom pairs of 100 molecules.

Source

```
apset <- sdf2ap(sdfsample)
```

References

Chen X and Reynolds CH (2002). "Performance of similarity measures in 2D fragment-based similarity searching: comparison of structural descriptors and similarity coefficients", J Chem Inf Comput Sci.

Examples

```
data(apset)
apset[1:4]
view(apset[1:4])
```

`apset2descdb` *'APset' to list-style AP database*

Description

Coerces APset to old list-style descriptor database used by search/cluster functions.

Usage

```
apset2descdb(apset)
```

Arguments

`apset`	Object of class `apset`

Details

...

Value

`list`	with following components
`descdb`	list of atom pair sets
`cids`	compound IDs
`sdfsegs`	start/end coordinates for each molecule in SD file; only populated when `cmp.parse` is used for import
`source`	path/name of SD file
`type`	import method

Author(s)

Thomas Girke

References

Chen X and Reynolds CH (2002). "Performance of similarity measures in 2D fragment-based similarity searching: comparison of structural descriptors and similarity coefficients", J Chem Inf Comput Sci.

See Also

Functions: SDF2apcmp, sdf2ap, cmp.search, cmp.similarity

Examples

```
## Instance of SDFset class
data(sdfsample)
sdfset <- sdfsample[1:50]
sdf <- sdfsample[[1]]

## Compute atom pair library
ap <- sdf2ap(sdf)
(apset <- sdf2ap(sdfset))
view(apset[1:4])

## Return main components of APset object
cid(apset[1:4]) # compound IDs
ap(apset[1:4]) # atom pair descriptors

## Return atom pairs in human readable format
db.explain(apset[1])

## Coerce APset to other objects
apset2descdb(apset) # returns old list-style AP database
tmp <- as(apset, "list") # returns list
as(tmp, "APset") # converst list back to APset

## Compound similarity searching with APset
cmp.search(apset, apset[1], type=3, cutoff=0.2)
plot(sdfset[names(cmp.search(apset, apset[6], type=2, cutoff=0.4))])

## Identify compounds with identical AP sets
cmp.duplicated(apset, type=2)

## Structure similarity clustering
cmp.cluster(db=apset, cutoff = c(0.65, 0.5))[1:20,]
```

---

atomblock *Return atom block*

Description

Returns atom block(s) from an object of class SDF or SDFset.

Usage

```
atomblock(x)
```

Arguments

x	object of class SDF or SDFset

Details

...

Value

`matrix` if `SDF` is provided or `list` of matrices if `SDFset` is provided

Author(s)

Thomas Girke

References

...

See Also

`header`, `atomcount`, `bondblock`, `datablock`, `cid`, `sdfid`

Examples

```
## SDF/SDFset instances
data(sdfsample)
sdfset <- sdfsample
sdf <- sdfset[[1]]

## Extract atome block
atomblock(sdf)
atomblock(sdfset[1:4])

## Replacement methods
sdfset[[1]][[2]][1,1] <- 999
sdfset[[1]]
atomblock(sdfset)[1:2] <- atomblock(sdfset)[3:4]
atomblock(sdfset[[1]]) == atomblock(sdfset[[3]])
view(sdfset[1:2])
```

`atomcount` *Molecular property functions*

Description

Functions to compute molecular properties: weight, formula, atom frequencies, etc.

Usage

```
atomcount(x, addH = FALSE, ...)

atomcountMA(x, ...)

MW(x, mw=atomprop, ...)

MF(x, ...)
```

Arguments

x	object of class SDFset or SDF
mw	data.frame with atomic weights; imported by default with data(atomprop); supports custom data sets
addH	'addH = TRUE' should be passed on to any of these function to add hydrogens that are often not specified in SD files
...	Arguments to be passed to/from other methods.

Details

...

Value

named vector	MW and MF
list	atomcount
matrix	atomcountMA

Author(s)

Thomas Girke

References

Standard atomic weights (2005) from: http://iupac.org/publications/pac/78/11/2051/

See Also

Functions: datablock, datablocktag

Examples

```
## Instance of SDFset class
data(sdfsample)
sdfset <- sdfsample

## Compute properties; to consider missing hydrogens, set 'addH = TRUE'
MW(sdfset[1:4], addH = FALSE)
MF(sdfset[1:4], addH = FALSE)
atomcount(sdfset[1:4], addH = FALSE)
propma <- atomcountMA(sdfset[1:4], addH = FALSE)
boxplot(propma, main="Atom Frequency")

## Example for injecting a custom matrix/data frame into the data block of an
## SDFset and then writing it to an SD file
props <- data.frame(MF=MF(sdfset), MW=MW(sdfset), atomcountMA(sdfset))
datablock(sdfset) <- props
view(sdfset[1:4])
# write.SDF(sdfset[1:4], file="sub.sdf", sig=TRUE, cid=TRUE)
```

atomprop *Standard atomic weights*

Description

Data frame with atom names, symbols, standard atomic weights, group number and period number.

Usage

```
data(atomprop)
```

Format

The format is a data frame with 117 rows and 6 columns.

Source

Columns 1 to 4 from: http://iupac.org/publications/pac/78/11/2051/ Columns 5 to 6 from: http://en.wikipedia.org/wiki/Li

References

Pure Appl. Chem., 2006, Vol. 78, No. 11, pp. 2051-2066

Examples

```
data(atomprop)
atomprop[1:4,]
```

atomsubset *Subset SDF/SDFset Objects by Atom Index to Obtain Substructure*

Description

Function to obtain a substructure from SDF/SDFset objects by providing a row index for the atom block in an SDF referencing the atoms of interest. The function subsets both the atom and bond block(s) accordingly.

Usage

```
atomsubset(x, atomrows, datablock = FALSE)
```

Arguments

x	object of class SDFset or SDF
atomrows	The argument atomrows can be assigned a numeric index referencing the atoms in the atom block of x. If x is of class SDF, the index needs to be provided as vector. If x is of class SDFset, the same number of index vectors as molecules stored in x need to be passed on in a list with component names identical to the component (molecule) names stored in x.
datablock	By default the data block(s) in SDF/SDFset objects are removed after atom subsetting. The setting datablock=TRUE will maintain the data block information in the subsetted result.

Details

...

Value

object of class `SDF` or `SDFset`

Author(s)

Thomas Girke

References

...

See Also

...

Examples

```
## Instance of SDFset class
data(sdfsample)
sdfset <- sdfsample

## Subset one or more molecules with atom index(es) to obtain substructure(s)
atomsubset(sdfset[[1]], atomrows=1:18)
indexlist <- list(1:18, 1:12)
names(indexlist) <- cid(sdfset[1:2])
atomsubset(sdfset[1:2], atomrows=indexlist)
```

`bondblock` *Return bond block*

Description

Returns bond block(s) from an object of class SDF or SDFset.

Usage

```
bondblock(x)
```

Arguments

`x`	object of class `SDF` or `SDFset`

Details

...

Value

`matrix` if `SDF` is provided or `list` of matrices if `SDFset` is provided

Author(s)

Thomas Girke

References

...

See Also

header, atomcount, atomblock, datablock, cid, sdfid

Examples

```
## SDF/SDFset instances
data(sdfsample)
sdfset <- sdfsample
sdf <- sdfset[[1]]

## Extract bond block
bondblock(sdf)
bondblock(sdfset[1:4])

## Replacement methods
sdfset[[1]][[3]][1,1] <- 999
sdfset[[1]]
bondblock(sdfset)[1:2] <- bondblock(sdfset)[3:4]
bondblock(sdfset[[1]]) == bondblock(sdfset[[3]])
view(sdfset[1:2])
```

---

bonds *Bonds, charges and missing hydrogens*

Description

Returns information about bonds, charges and missing hydrogens in SDF and SDFset objects.

Usage

```
bonds(x, type = "bonds")
```

Arguments

x	SDF or SDFset containers
type	If type="bonds" (default), a data.frame is returned with columns: atom (atom labels), Nbondcount (observed bond count), Nbondrule (bond count according to position in periodic table) and charge (charge of each atom). If type="charge", all charged atoms are returned and if type="addNH", the number of missing hydrogens are returned for each molecule.

Details

It is used by many other functions (e.g. MW, MF, atomcount, atomcuntMA and plot) to correct for missing hydrogens that are often not specified in SD files.

Value

If x is of class SDF, then a single data.frame or vector is returned. If x is of class SDFset, then a list of data.frames or vecotors is returned that has the same length and order as x.

Author(s)

Thomas Girke

References

...

See Also

Functions: conMA

Class: SDF and SDFset

Examples

```
## Instances of SDFset class
data(sdfsample)
sdfset <- sdfsample

## Returns data frames with bonds and charges
bonds(sdfset[1:2], type="bonds")

## Returns charged atoms in each molecule
bonds(sdfset[1:2], type="charge")

## Returns the number of missing hydrogens in each molecule
bonds(sdfset[1:2], type="addNH")
```

---

cid *Return compound IDs*

Description

Returns the compound identifiers from the ID slot of an SDFset object.

Usage

```
cid(x)
```

Arguments

x	object of class SDFset or APset

Details

...

Value

`character` vector

Author(s)

Thomas Girke

References

...

See Also

`atomblock`, `atomcount`, `bondblock`, `datablock`, `header`, `sdfid`

Examples

```
## SDFset/APset instances
data(sdfsample)
sdfset <- sdfsample
apset <- sdf2ap(sdfset[1:4])

## Extract compound IDs from SDFset/APset
cid(sdfset[1:4])
cid(apset[1:4])

## Extract IDs defined in SD file
sdfid(sdfset[1:4])

## Assigning compound IDs and keeping them unique
unique_ids <- makeUnique(sdfid(sdfset))
cid(sdfset) <- unique_ids
cid(sdfset[1:4])

## Replacement Method
cid(sdfset) <- as.character(1:100)
```

`cluster.sizestat` *generate statistics on sizes of clusters*

Description

'cluster.sizestat' is used to do simple statistics on sizes of clusters generated by 'cmp.cluster'. It will return a dataframe which maps a cluster size to the number of clusters with that size. It is often used along with 'cluster.visualize'.

Usage

```
cluster.sizestat(cls, cluster.result=1)
```

Arguments

`cls`	The clustering result returned by 'cmp.cluster'
`cluster.result`	If multiple cutoff values are used in clustering process, this argument tells which cutoff value is to be considered here.

Details

'cluster.sizestat' depends on the format that is returned by 'cmp.cluster' - it will treat the first column as the indecies, and the second column as the cluster sizes of effective clustering. Because of this, when multiple cutoffs are used when 'cmp.cluster' is called, 'cluster.sizestat' will only consider the clustering result of the first cutoff. If you want to work on an alternative cutoff, you have to manually reorder/remove columns.

Value

Returns a data frame of two columns.

`cluster size`	This column lists cluster sizes
`count`	This column lists number of clusters of a cluster size

Author(s)

Y. Eddie Cao

See Also

`cmp.cluster`, `cluster.visualize`

Examples

```
## Load sample SD file
# data(sdfsample); sdfset <- sdfsample

## Generate atom pair descriptor database for searching
# apset <- sdf2ap(sdfset)

## Loads same atom pair sample data set provided by library
data(apset)

## Binning clustering using variable similarity cutoffs.
cluster <- cmp.cluster(db=apset, cutoff = c(0.65, 0.5))

## Statistics on sizes of clusters
cluster.sizestat(cluster[,c(1,2,3)])
cluster.sizestat(cluster[,c(1,4,5)])
```

`cluster.visualize` *visualize clustering result using multi-dimensional scaling*

Description

'cluster.visualize' takes clustering result returned by 'cmp.cluster' and generate multi-dimensional scaling plot for visualization purpose.

Usage

```
cluster.visualize(db, cls, size.cutoff, distmat=NULL, color.vector=NULL, non.int
```

Arguments

`db`	The desciptor database, in the format returned by 'cmp.parse'.
`cls`	The clustering result returned by 'cmp.cluster'.
`size.cutoff`	The cutoff size for clusters considered in this visualization. Clusters of size smaller than the cutoff will not be considered.
`distmat`	A distance matrix that corresponds to the 'db'. If not provided, it will be computed on-the-fly in an efficient manner.
`color.vector`	Colors to be used in the plot. If the number of colors in the vector is not enough for the plot, colors will be reused. If not provided, color will be generated and randomly sampled from 'rainbow'.
`non.interactive`	If provided, will enable the non-interactive mode, and the plot will be in an eps file named after this value.
`cluster.result`	Used to select the clustering result if multiple clustering results are present in 'cls'.
`dimensions`	Dimensionality to be used in visualization. See details.
`quiet`	Whether to supress the progress bar.
`highlight.compounds`	A vector of compound IDs, corresponding to compounds to be highlighted in the plot. A highlighted compound is represented as a filled circle.
`highlight.color`	Color used for highlighted compounds. If not set, a highlighted compounds will have the same color as that used for other compounds in the same cluster.
`...`	Further arguments will be passed to 'cmp.similarity' to calculate similarity matrix.

Details

'cluster.visualize' internally calls the 'cmdscale' function to generate a set of points in 2-D for the compounds in selected clusters. Note that for compounds in clusters smaller than the cutoff size, they will not be considered in this calculation - their entries in 'distmat' will be discarded if 'distmat' is provided, and distances involving them will not be computed if 'distmat' is not provided.

To determine the value for 'size.cutoff', you can use 'cluster.sizestat' to see the size distribution of clusters.

Because 'cmp.cluster' function allows you to perform multiple clustering processes simultaneously with different cutoff values, the 'cls' parameter may point to a data frame containing multiple clustering results. The user can use 'cluster.result' to specify which result to use. By default, this is set to 1, and the first clustering result will be used in visualization. Whatever the value is, in interactive mode (described below), all clustering result will be displayed when a compound is selected in the interactive plot.

If the colors provided in 'color.vector' are not enough to distinguish clusters by colors, the function will silently reuse the colors, resulting multiple clusters colored in the same color. We suggest you use 'cluster.sizestat' to see how many clusters will be selected using your 'size.cutoff', or simply provide no 'color.vector'.

If 'non.interative' is not set, the final plot is interactive. You will be able to select points by clicking them. When you click on any point, information about the compound represented by that point will be displayed. This includes the cluster ID, cluster size, compound index in the SDF and compound name if any. You can then perform another selection. To exit this process, right click on X11 device or press ESC in non-X11 device (Quartz and Windows).

By default, 'dimensions' is set to 2, and the built-in 'plot' function will be used for plotting. If you need to do 3-Dimensional plotting, set 'dimensions' to 3, and pass the returned value to 3D plot utilities, such as 'scatterplot3d' or 'rggobi'. This package does not perform 3D plot on its own.

Value

This function returns a data frame of MDS coordinates and clustering result. This value can be passed to 3D plot utilities such as 'scatterplot3d' and 'rggobi'.

The last column of the output gives whether the compounds have been clicked in the interactive mode.

Author(s)

Y. Eddie Cao

See Also

cmp.parse, cmp.cluster, cluster.sizestat

Examples

```
## Load sample SD file
# data(sdfsample); sdfset <- sdfsample

## Generate atom pair descriptor database for searching
# apset <- sdf2ap(sdfset)

## Loads same atom pair sample data set provided by library
data(apset)
db <- apset

## cluster db with 2 cutoffs
clusters <- cmp.cluster(db, cutoff=c(0.5, 0.4))

## Return size stats
sizestat <- cluster.sizestat(clusters)

## Visualize results, using a cutoff of 3, write to file 'test.eps'
coord <- cluster.visualize(db, clusters, 2, non.interactive="test.eps")
```

```
## Not run:
## visualize it in interactive mode, using a cutoff of 3 and the 2nd clustering result
coord <- cluster.visualize(db, clusters, cluster.result=2, 3)

## 3D visualization with scatterplot3d
coord <- cluster.visualize(db, clusters, 3, dimensions=3)
library(scatterplot3d)
scatterplot3d(coord)


## End(Not run)
```

`cmp.cluster` *cluster compounds using a descriptor database*

Description

'cmp.cluster' uses compound descriptors in a database and clusters these compounds based on their pairwise distances. 'cmp.cluster' uses single linkage to measure distance between clusters when it merges clusters. 'cmp.cluster' accepts both a single cutoff and a cutoff vector. By using a cutoff vector, it can generate the same result as hierachical clustering.

Usage

```
cmp.cluster(db, cutoff, is.similarity = TRUE, save.distances = FALSE,
        use.distances = NULL, quiet = FALSE, ...)
```

Arguments

`db`	The desciptor database, in the format returned by 'cmp.parse'.
`cutoff`	The clustering cutoff. Can be a single value or a vector. The cutoff gives the maximum distance between two compounds in order to group them in the same clsuter.
`is.similarity`	Set when the cutoff supplied is a similarity cutoff. This cutoff is the mimumum similarity value between two compounds such that they will be grouped in the same cluster.
`save.distances`	whether to save distance for future clustering. See details below.
`use.distances`	Supply pre-computed distance matrix.
`quiet`	Whether to supress the progress information.
`...`	Further arguments to be passed to 'cmp.similarity' to calculate similarities if necessary.

Details

'cmp.cluster' will compute distance on the fly if 'use.distances' is not set. Furthermore, if 'save.distances' is not set, the distance will never be stored and distance between any two compounds is guaranteed not to be computed twice. Using this method, 'cmp.cluster' can deal with large database, when a distance matrix in memory is not feasible. The speed of this cluster function should be slowed because of using this transient distance value.

When 'save.distances' is set, 'cmp.cluster' will be forced to compute the distance matrix and save it in memory before doing clustering. This is useful when you need to do further clustering in the future and do not want the distance to be re-computed then. Set 'save.distances' to TRUE if you only want to force the clustering to use this 2-step approach; otherwise, set it to the filename under which you want the distance matrix to be saved. After you save it, when you need to reuse the distance matrix, you can 'load' it, and supply to 'cmp.cluster' via the 'use.distances' argument.

'cmp.cluster' supports vector of cutoffs. When you have multiple cutoffs, 'cmp.cluster' still guarantees that pairwise distances will never be recomputed, and no copy of distances is kept in memory. It is guaranteed to be as fast as calling 'cmp.cluster' with a single cutoff that results in the longest processing time, plus some small overhead linear in that processing time.

Value

Returns a data frame. Besides a variable giving compound ID, each of the other variables in the data frame will either give the cluster IDs of compounds under some clustering cutoff, or the size of clusters that the compounds belong to. When N cutoffs are given, in total 2*N+1 variables will be generated, with N of them giving the cluster ID of each compound under each of the N cutoffs, and the other N of them giving the cluster size under each of the N cutoffs. The rows are sorted by the cluster sizes.

Author(s)

Y. Eddie Cao, Li-Chang Cheng

See Also

cmp.parse1, cmp.parse, cmp.search, cmp.similarity

Examples

```
## Load sample SD file
# data(sdfsample); sdfset <- sdfsample

## Generate atom pair descriptor database for searching
# apset <- sdf2ap(sdfset)

## Loads same atom pair sample data set provided by library
data(apset)
db <- apset

## cluster using multiple cutoffs
clusters <- cmp.cluster(db, cutoff=c(0.5, 0.85))

## or save the distance before clustering:
clusters <- cmp.cluster(db, cutoff=0.65, save.distances="distmat.rda")
# later, you can load the matrix and pass it to do clustering. Load will load
# the variable 'distmat' that contains the distance matrix
load("distmat.rda")
```

```
clusters <- cmp.cluster(db, cutoff=0.60, use.distances=distmat)
```

`cmp.duplicated` *quickly detect compound duplication in a descriptor database*

Description

'cmp.duplicated' detects duplicated compounds from a descriptor database generated by 'cmp.parse'. Two compounds are said to duplicate each other when their descriptors are the same.

Usage

```
cmp.duplicated(db, sort = FALSE, type=1)
```

Arguments

`db`	The desciptor database, in the format returned by 'cmp.parse'.
`sort`	Whether to sort the descriptors for a compound. See details.
`type`	Returns results as vector (type=1) or data frame (type=2).

Details

'cmp.duplicated' will take the descriptors in the descriptor database, concatenate all descriptors for the same compound into a string, and use this string as the identification of a compound. If two compounds share the same identification string, they are said to duplicate each other.

'cmp.duplicated' assume the the database passed in as argument to follow the format generated by 'cmp.parse'. That is, 'db' is a list, 'db$descdb' is a list, and each entry of 'db$descdb' is an array of numeric values that give descriptors for one compound.

By default, 'cmp.duplicated' will assume the descriptors for a compound is already sorted. That is each entry in 'db\$descdb' is a sorted array. This is true for database generated by 'cmp.parse'. If you generate the database using some other tools, you might want to enable sorting.

Value

Returns a logic array, telling whether a compound in the database is a duplication of a compound appearing before this one. For example, if the i-th element of the array is TRUE, it means that the i-th compound in the database is a duplication of a compound listed before this compound in the database.

The returned array can be used to remove duplication. Simply use it to index the descriptor database.

If you are interested in what compound is duplicated, you can do a search in the database with cutoff set to 1.

Author(s)

Y. Eddie Cao

See Also

`cmp.parse`, `cmp.search`

Examples

```
## Load sample SD file
# data(sdfsample); sdfset <- sdfsample

## Generate atom pair descriptor database for searching
# apset <- sdf2ap(sdfset)

## Loads same atom pair sample data set provided by library
data(apset)
db <- apset

## Manually create a duplication (here compound 1 and 10)
db[10] <- db[1]

## Find duplication
dup <- cmp.duplicated(db)
dup
cid(db[dup])

## Remove all duplications
db <- db[!dup]
```

---

cmp.parse *Parse an SDF file and compute descriptors for all compounds*

Description

'cmp.parse' will take a SDF file, parse all the compounds encoded, compute their atom-pair descriptors, and return the descriptors as a list. The list contains two names, 'descdb' and 'cids'. 'descdb' is a vector of descriptors, and 'cids' is a list of names of compounds found in the SDF file. The returned list is usually used to a database, against which similarity search can be performed using the 'search' function. These two functions will parse all compounds in the SDF file. To parse a single compound, use 'cmp.parse1' instead.

Usage

```
cmp.parse(filename, quiet=FALSE, type="normal", dbname="")
```

Arguments

filename	The file name of the SDF file
quiet	Whether to silent the output of progress information
type	Database type. Use the default value, or set to 'file-backed' when the library is large. See below.
dbname	Datbase name. Only used when the type is set to 'file-backed'.

Details

The 'filename' can be a local file or an URL. It is interactive, and will display the parsing progress. Since the parsing will also compute of atom-pair descriptors, it is time consuming. You will be reminded to save the parsing result for future use at the end of parsing.

'type' is either set to the default value 'normal' or 'file-backed'. When set to 'file-backed', the parsing work will be delegated to a separate package called 'ChemmineRpp', and the database will be stored in a file instead of in the primary memory. Therefore, 'file-backed' mode can handle larger compound libraries. In 'file-backed' mode, 'dbname' will be used to name the database file. A suffix '.cdb' will be appended to the given name.

The type of the database is transparent to other part of the package. For example, calling 'cmp.search' against a database in 'file-backed' mode will cause the package to load the descriptors from the database file progressively.

Value

Return a list that can be used as the database against which similarity search can be performed. The 'search' and 'cmp.cluster' functions both expect a database returned by 'cmp.parse'.

`descdb`	A vector containing the descriptors for all the compounds.
`cids`	Compound ID information found in the SDF file. It is the first line of SDF of a compound.

Author(s)

Y. Eddie Cao, Li-Chang Cheng

References

Chen X and Reynolds CH (2002). "Performance of similarity measures in 2D fragment-based similarity searching: comparison of structural descriptors and similarity coefficients", in *J Chem Inf Comput Sci*.

See Also

`cmp.parse1`, `cmp.search`, `cmp.cluster`, `cmp.similarity`

Examples

```
## Load sample SD file
# data(sdfsample); sdfset <- sdfsample

## Generate atom pair descriptor database for searching
# apset <- sdf2ap(sdfset)

## Loads same atom pair sample data set provided by library
data(apset)
db <- apset
# (optinally) save the db for future use
save(db, file="db.rda", compress=TRUE)
# ...
# later, in a separate session, you can load it back:
load("db.rda")
```

`cmp.parse1` *Parsing an SDF file and calculate the descriptor for one compound*

Description

Read SDF information from an SDF file or connection, parse the first compound, and calculate the descriptor for that compound. The returned descriptor can be added to database returned by 'cmp.parse' or be used as the query structure when calling 'search'. This function will only parse one compound and return only the descriptor. To parse all compounds in an SDF file, use 'cmp.parse'.

Usage

```
cmp.parse1(filename)
```

Arguments

`filename`	The file name of the SDF file or a URL or a connection.

Details

'cmp.parse1' can take a file name or a URL or a connection. When a connection is used, the current line must be the first line of SDF of the compound to be parsed. 'cmp.parse1' will skip the header and parse from the 4th line. Therefore, the compound ID information will be skipped. After the parsing is done, if 'filename' is a connection, it will then point to the line after the connection table of SDF. You can use some other procedure to parse the annotation block.

Value

Return the descriptor, which is encoded as a vector.

Author(s)

Y. Eddie Cao, Li-Chang Cheng

References

Chen X and Reynolds CH (2002). "Performance of similarity measures in 2D fragment-based similarity searching: comparison of structural descriptors and similarity coefficients", in *J Chem Inf Comput Sci*.

See Also

`cmp.parse`, `cmp.search`, `cmp.cluster`, `cmp.similarity`

Examples

```
# load an SDF file from web and parse it
## Not run: structure <- cmp.parse1("http://bioweb.ucr.edu/ChemMineV2/compound/Aurora/b32
```

`cmp.search` *Search a descriptor database for compounds similar to query compound*

Description

Given descriptor of a query compound and a database of compound descriptors, search for compounds that are similar to the query compound. User can limit the output by supplying a cutoff similarity score or a cutoff that limits the number of returned compounds. The function can also return the scores together with the compounds.

Usage

```
cmp.search(db, query, type=1, cutoff = 0.5, return.score = FALSE, quiet = FA
mode = 1, visualize=FALSE, visualize.browse=TRUE, visualize.query=NULL)
```

Arguments

`db`	The compound descriptor database returned by 'cmp.parse'.
`query`	The query descriptor, which is usually returned by 'cmp.parse1'.
`type`	Returns results in form of position indices (type=1), named vector with compound IDs (type=2) or data frame (type=3).
`cutoff`	The cutoff similarity (when cutoff <= 1) or the number of maximum compounds to be returned (when cutoff > 1).
`return.score`	Whether to return similarity scores. If set to TRUE, a data frame will be returned; otherwise, only the compounds' indices in the database will be returned in the order of decreasing scores.
`quiet`	Whether to disable progress information.
`mode`	Mode used when computing similarity scores. This value is passed to 'cmp.similarity'.
`visualize`	Whether to visualize the search result in a webpage.
`visualize.browse`	Whether to open the browser automatically if you choose to visualize the search result.
`visualize.query`	Filename/URL or a character string containing the SDF of the query structure if you also want to visualize the query in the search result visualization webpage.

Details

'cmp.search' will go through all the compound descriptors in the database and calculate the similarity between the query compound and compounds in the database. When cutoff similarity score is set, compounds having a similarity score higher than the cutoff will be returned. When maximum number of compounds to return is set to N via 'cutoff', the compounds having the highest N similarity scores will be returned.

If 'visualize' is set to a TRUE value, `sdf.visualize` will be called to send the search results and the scores to ChemMine website. If 'visualize.browse' is set to a TRUE value, the browser will open to show the structures in the search result with their corresponding scores. Otherwise, a URL pointing to that webpage will be printed. By default, 'visualize.query' is not set, and the query

structure will not be uploaded. If you want that to be included in the visualization webpage as well, you must set this argument to a character string containing the SDF of the query, or a filename pointing to a file containing the SDF of the query. If the character string or the file containing multiple SDFs, only the first will be considered as the SDF of the query.

Value

When 'return.score' is set to FALSE, a vector of matching compounds' indices in the database will be returned. Otherwise, a data frame will be returned:

`ids`	The indices of matching compounds in the database.
`scores`	The similarity scores between the matching compounds and the query compound

Author(s)

Y. Eddie Cao, Li-Chang Cheng

References

Chen X and Reynolds CH (2002). "Performance of similarity measures in 2D fragment-based similarity searching: comparison of structural descriptors and similarity coefficients", in *J Chem Inf Comput Sci*.

See Also

`cmp.parse1`, `cmp.parse`, `cmp.search`, `cmp.cluster`, `cmp.similarity`, `sdf.visualize`

Examples

```
## Load sample SD file
# data(sdfsample); sdfset <- sdfsample

## Generate atom pair descriptor database for searching
# apset <- sdf2ap(sdfset)

## Loads same atom pair sample data set provided by library
data(apset)
db <- apset
query <- db[1]

## Ooptinally, save the db for future use
save(db, file="db.rda", compress=TRUE)

## Search for similar compounds using similarity cutoff
cmp.search(db, query, cutoff=0.2, type=1) # returns index
cmp.search(db, query, cutoff=0.2, type=2) # returns named vector
cmp.search(db, query, cutoff=0.2, type=3) # returns data frame

# you may visualize the search result in ChemMine
## Not run: cmp.search(db, query, cutoff=10, visualize=TRUE, visualize.browse=FALSE, visu

## in the next session, you may use load a saved db and do the search:
load("db.rda")
cmp.search(db, query, cutoff=3)
## you may also use the loaded db to do clustering:
```

```
cmp.cluster(db, cutoff=0.35)
```

`cmp.similarity` *Compute similarity between two compounds using their descriptors*

Description

Given descriptors for two compounds, 'cmp.similarity' returns the similarity measure between the two compounds.

Usage

```
cmp.similarity(a, b, mode = 1, worst = 0)
```

Arguments

`a`	Descriptor of the first compound.
`b`	Descriptor of the second compound.
`mode`	Mode used when computing the distance. See details below.
`worst`	The worst value you are expecting. If 'cmp.similarity' finds the upper bound of similarity is worse than it, it will return a 0 and potentially save some computation.

Details

'cmp.similarity' uses descriptor information generated by 'cmp.parse' and 'cmp.parse1'. Basically, a descriptor is a vector of numbers. The vector actually reprsents the set of descriptors of structural fragment. Similarity measurement uses Tanimoto coefficient.

'cmp.similarity' supports 3 different modes. In mode 1, normal Tanimoto coefficient is used. In mode 2, it uses the size of descriptor intersection over the size of the smaller descriptor, mainly to deal with compounds that vary a lot in size. In mode 3, it is similar to mode 2, except that it raises the similarity to the power 3 to penalize small values. When mode is 0, 'cmp.similarity' will select mode 1 or mode 3, based on the size differences between the two descriptors.

When 'cmp.similarity' is used in searching compounds with a threshold similarity value, or in clustering with a cutoff distance, the threshold similarity and cutoff distance can be used to decide a 'worse' value. 'cmp.similarity' can compute an upper bound of similarity easier, and by comparing this upper bound to the 'worst' value, it can potentially skip the real computation if it finds the similarity will be below the 'worst' value and will be useless to the caller.

Value

Return a numeric value between 0 and 1 which gives the similarity between the two compounds.

Author(s)

Y. Eddie Cao, Li-Chang Cheng

References

Chen X and Reynolds CH (2002). "Performance of similarity measures in 2D fragment-based similarity searching: comparison of structural descriptors and similarity coefficients", in *J Chem Inf Comput Sci*.

Peter Willett (1998). "Chemical Similarity Searching", in *J. Chem. Inf. Comput. Sci*.

See Also

cmp.parse1, cmp.parse, cmp.search, cmp.cluster

Examples

```
## Load sample SD file
# data(sdfsample); sdfset <- sdfsample

## Generate atom pair descriptor database for searching
# apset <- sdf2ap(sdfset)

## Loads same atom pair sample data set provided by library
data(apset)

## Compute similarities among two compounds
cmp.similarity(apset[1], apset[2])

## Search apset database with a query compound
cmp.search(apset, apset[1], type=3, cutoff = 0.3)
```

---

conMA *Bond Matrices*

Description

Creates a bond matrix from SDF and SDFset objects. The matrix contains the atom labels in the row and column titles and the bond types are given in the data part as follows: 0 is no connection, 1 is a single bond, 2 is a double bond and 3 is a triple bond.

Usage

```
conMA(x, exclude = "none")
```

Arguments

x	SDF or SDFset containers
exclude	if exclude="none", then all atoms will be considered in the resulting connection table; if exclude=c("H"), then the H atoms will be excluded. Any number of atom labels to be excluded can be passed on to this argument in form of a character vector.

Details

...

Value

If x is of class SDF, then a single bond matrix is returned. If x is of class SDFset, then a list of matrices is returned that has the same length as x.

Author(s)

Thomas Girke

References

...

See Also

Functions: bonds

Class: SDF and SDFset

Examples

```
## Instances of SDFset class
data(sdfsample)
sdfset <- sdfsample

## Create bond matrix for first two molecules in sdfset
conMA(sdfset[1:2], exclude=c("H"))

## Return bond matrix for first molecule and plot its structure with atom numbering
conMA(sdfset[[1]], exclude=c("H"))
plot(sdfset[1], atomnum = TRUE, noHbonds=FALSE , no_print_atoms = "", atomcex=0.8)

## Return number of non-H bonds for each atom
rowSums(conMA(sdfset[[1]], exclude=c("H")))
```

---

datablock *Return data block*

Description

Returns data block(s) from an object of class SDF or SDFset.

Usage

```
datablock(x)

datablocktag(x, tag)
```

Arguments

x	object of class SDF or SDFset
tag	numeric position (index) or character name of entry in data block vector

Details

...

Value

named `character` vector if `SDF` is provided or `list` of named `character` vectors if `SDFset` is provided

Author(s)

Thomas Girke

References

...

See Also

`atomblock`, `atomcount`, `bondblock`, `header`, `cid`, `sdfid`

Examples

```
## SDF/SDFset instances
data(sdfsample)
sdfset <- sdfsample
sdf <- sdfset[[1]]

## Extract data block
datablock(sdf)
datablock(sdfset[1:4])
datablocktag(sdfset, tag="PUBCHEM_OPENEYE_CAN_SMILES")

## Replacement methods
sdfset[[1]][[1]][1] <- "test"
sdfset[[1]]
datablock(sdfset)[1] <- datablock(sdfset[2])
view(sdfset[1:2])

## Example for injecting a custom matrix/data frame into the data block of an
## SDFset and then writing it to an SD file
props <- data.frame(MF=MF(sdfset), MW=MW(sdfset), atomcountMA(sdfset))
datablock(sdfset) <- props
view(sdfset[1:4])
# write.SDF(sdfset[1:4], file="sub.sdf", sig=TRUE, cid=TRUE)
```

`datablock2ma` *SDF data blocks to matrix*

Description

Convert data blocks in `SDFset` to character matrix with `datablock2ma`, then store its numeric columns as numeric matrix and its character columns as character matrix.

Usage

```
datablock2ma(datablocklist = datablock(sdfset), cleanup = " \\(.*", ...)

splitNumChar(blockmatrix = blockmatrix)
```

Arguments

datablocklist	list of data block vectors; can be created with datablock(sdfset)
blockmatrix	matrix returned by datablock2ma
cleanup	character pattern to be used to clean up the name fields of the data block vectors; the exact pattern matches are replaced by nothing (deleted).
...	option to pass on additional arguments

Details

...

Value

datablock2ma character matrix

splitNumChar list with two components, a numeric matrix and a character matrix

Author(s)

Thomas Girke

References

...

See Also

Classes: SDFset

Examples

```
## SDFset instance
data(sdfsample)
sdfset <- sdfsample

# Convert data block to matrix
blockmatrix <- datablock2ma(datablocklist=datablock(sdfset))
blockmatrix[1:4, 1:4]

# Split matrix to numeric matrix and character matrix
numchar <- splitNumChar(blockmatrix=blockmatrix)
names(numchar)
numchar[[1]][1:4,]
numchar[[2]][1:4,]
```

`db.explain` *Explain an atom-pair descriptor or an array of atom-pair descriptors*

Description

'db.explain' will take an atom-pair descriptor in numeric or a set of such descriptors, and interpret what they represent in a more human readable way.

Usage

```
db.explain(desc)
```

Arguments

`desc`	The descriptor or the array/vector of descriptors

Details

'desc' can be a single numeric giving a single descriptor or can be any container data type, such as vector or array, such that 'length(desc)' returns 2 or larger.

Value

Return a character vector describing the descriptors.

See Also

`cmp.parse`

Examples

```
## Load sample SD file
# data(sdfsample); sdfset <- sdfsample

## Generate atom pair descriptor database for searching
# apset <- sdf2ap(sdfset)

## Loads same atom pair sample data set provided by library
data(apset)
db <- apset

## Return atom pairs of first compound in human readable format
db.explain(db[1])
```

db.subset *Subset a descriptor database and return a sub-database for the selected compounds*

Description

'db.subset' will take a descriptor database generated by 'cmp.parse' and an array of indecies, and return a new database for compounds corresponding to these indecies. The returned value is a descriptor database as returned by the cmp.parse function.

Usage

```
db.subset(db, cmps)
```

Arguments

db	The database generated by 'cmp.parse'
cmps	An array of indecies that correspond to a set of selected compounds from the database

Details

'db.subset' creates a sub-database from 'db' by only including infomration that is relevant to compounds indexed by 'cmps'.

Value

Return a descriptor database for the selected compounds. The format of the database is compatible with the one returned by cmp.parse.

See Also

cmp.parse, sdf.subset

Examples

```
## Note: this functionality has become obsolete since the introduction of the
## 'apset' S4 class.

## Load sample SD file
# data(sdfsample); sdfset <- sdfsample

## Generate atom pair descriptor database for searching
# apset <- sdf2ap(sdfset)

## Loads same atom pair sample data set provided by library
data(apset)
db <- apset
olddb <- apset2descdb(db)

## Create a sub-database for the 1st and 2nd compound in that SDF
db_sub <- db.subset(olddb, c(1, 2))
```

fp2bit *Convert base 64 fingerprints to binary*

Description

The function converts the base 64 encoded PubChem fingerprints to a binary `matrix` or a `character` vector. If applied to a `SDFset` object, then its data block needs to contain the PubChem fingerprint information.

Usage

```
fp2bit(x, type = 2, fptag = "PUBCHEM_CACTVS_SUBSKEYS")
```

Arguments

`x`	Object of class `SDFset` or `matrix`
`type`	If set to `1`, the results are returned as binary *matrix*. If set to `2` (default), the results are returned as `character` strings in a named *vector*.
`fptag`	Name tag in SDF data block where the PubChem fingerprints are stored. Default is set to "PUBCHEM_CACTVS_SUBSKEYS".

Details

...

Value

Returns results as binary `matrix` when `type=2` or as `character` strings stored in a named `vector` when `type=1`.

Author(s)

Thomas Girke

References

See PubChem fingerprint specification at: ftp://ftp.ncbi.nih.gov/pubchem/specifications/pubchem_fingerprints.txt

See Also

Functions: `fpSim`

Examples

```
## Load PubChem SDFset sample
data(sdfsample); sdfset <- sdfsample
cid(sdfset) <- sdfid(sdfset)

## Convert base 64 encoded fingerprints to character vector or binary matrix
fpset <- fp2bit(x=sdfset, type=1)
fpset <- fp2bit(x=sdfset, type=2)
```

```
## Pairwise compound structure comparisons
fpSim(x=fpset[1,], y=fpset[2,])

## Structure similarity searching: x is query and y is fingerprint database
fpSim(x=fpset[1,], y=fpset)

## Compute fingerprint based Tanimoto similarity matrix
simMA <- sapply(rownames(fpset), function(x) fpSim(x=fpset[x,], fpset))

## Hierarchical clustering with simMA as input
hc <- hclust(as.dist(simMA), method="single")

## Plot hierarchical clustering tree
plot(as.dendrogram(hc), edgePar=list(col=4, lwd=2), horiz=TRUE)
```

`fpSim` *PubChem Fingerprint Search*

Description

Function to use PubChem fingerprints for structure similarity comparisons, searching and clustering.

Usage

```
fpSim(x, y)
```

Arguments

`x`	`vector` containing binary fingerprint data. Needs to have the same length as `y` (`vector` or `matrix` row).
`y`	`vector` or `matrix` containing binary fingerprint data.

Details

The function computes the Tanimoto coefficients for pairwise comparisons of binary fingerprints. The coefficient is defined as c/(a+b+c), which is the proportion of the "on-bits" shared among the fingerprints of two compounds divided by their union. The variable c is the number of "on-bits" common in both compounds, while a and b are the number of "on-bits" that are unique in one or the other compound, respectively.

Value

Returns `numeric vector` with Tanimoto coefficients as values and compound identifiers as names.

Note

Limitation: PubChem fingerprints need to be provided, such as in PubChem's SD files.

Author(s)

Thomas Girke

References

Tanimoto similarity coefficient: Tanimoto TT (1957) IBM Internal Report 17th Nov see also Jaccard P (1901) Bulletin del la Societe Vaudoisedes Sciences Naturelles 37, 241-272.

PubChem fingerprint specification: ftp://ftp.ncbi.nih.gov/pubchem/specifications/pubchem_fingerprints.txt

See Also

Functions: `fp2bit`

Examples

```
## Load PubChem SDFset sample
data(sdfsample); sdfset <- sdfsample
cid(sdfset) <- sdfid(sdfset)

## Convert base 64 encoded fingerprints to character vector or binary matrix
fpset <- fp2bit(x=sdfset, type=1)
fpset <- fp2bit(x=sdfset, type=2)

## Pairwise compound structure comparisons
fpSim(x=fpset[1,], y=fpset[2,])

## Structure similarity searching: x is query and y is fingerprint database
fpSim(x=fpset[1,], y=fpset)

## Compute fingerprint-based Tanimoto similarity matrix
simMA <- sapply(rownames(fpset), function(x) fpSim(x=fpset[x,], fpset))

## Hierarchical clustering with simMA as input
hc <- hclust(as.dist(simMA), method="single")

## Plot hierarchical clustering tree
plot(as.dendrogram(hc), edgePar=list(col=4, lwd=2), horiz=TRUE)
```

`getIds` *Import Compounds from PubChem*

Description

Accepts one or more PubChem compound ids and downloads the corresponding compounds from PubChem Power User Gateway (PUG) returning results in an `SDFset` container. The ChemMine Tools web service is used as an intermediate, to translate queries from plain HTTP POST to a PUG SOAP query.

Usage

```
getIds(cids)
```

Arguments

`cids`	A `numeric` object which contains one or more PubChem cids

Value

SDFset — for details see ?"SDFset-class"

Author(s)

Tyler Backman

References

PubChem PUG SOAP: http://pubchem.ncbi.nlm.nih.gov/pug_soap/pug_soap_help.html

Chemmine web service: http://chemmine.ucr.edu

PubChem help: http://pubchem.ncbi.nlm.nih.gov/search/help_search.html

Examples

```
## Not run:
## fetch 2 compounds from PubChem
compounds <- getIds(c(111,123))
## End(Not run)
```

---

grepSDFset *String search in 'SDFset'*

Description

Convenience grep function for string searching in SDFset containers.

Usage

```
grepSDFset(pattern, x, field = "datablock", mode = "subset", ignore.case = TRUE,
```

Arguments

pattern	search pattern
x	SDFset
field	delimits search to specific section in SDF; can be header, atomblock, bondblock or datablock
mode	if mode = "index", then the match positions are returned as vector; if mode = "subset", a list with SDF components is returned where every entry has at least one query match
ignore.case	TRUE turns off case sensitivity
...	option to pass on additional arguments

Details

...

Value

numeric	index vector where the name field contains the component positions in the SDFset and the values the row positions in each sub-component.
list	if mode = "subset"

Author(s)

Thomas Girke

References

...

See Also

Class: SDFset

Examples

```
## Instances of SDFset class
data(sdfsample)
sdfset <- sdfsample

## String Searching in SDFset
q <- grepSDFset("65000", sdfset, field="datablock", mode="subset")
as(q, "SDFset")
grepSDFset("65000", sdfset, field="datablock", mode="index")
```

---

groups *Enumeration of Functional Groups and Atom Neighbors*

Description

Returns frequency information of functional groups in molecules provided as SDF or SDFset objects. Alternatively, the function can return for each atom its atom/bond neighbor information.

Usage

```
groups(x, groups = "fctgroup", type)
```

Arguments

x	SDF or SDFset containers
groups	if groups="fctgroup", frequencies of functional groups are returned; if groups="neighbors", atom/bond neighbor information is returned.
type	if type="all", then the complete neighbor information is generated for each atom in a molecule; if type="count", the neighbors are enumerated in a list and if type="countMA", then the counts of atom neighbors or functional groups are returned in a frequency matrix.

Details

At this point this function is in an experimental stage.

Value

...

Author(s)

Thomas Girke

References

...

See Also

...

Examples

```
## Instances of SDFset class
data(sdfsample)
sdfset <- sdfsample

## Enumerate functional groups
groups(sdfset[1:20], groups="fctgroup", type="countMA")

## Report atom/bond neighbors
groups(sdfset[1:4], groups="neighbors", type="countMA")
groups(sdfset[1:4], groups="neighbors", type="count")
groups(sdfset[1:4], groups="neighbors", type="all")
```

---

header *Return header block*

Description

Returns header block(s) from an object of class SDF or SDFset.

Usage

```
header(x)
```

Arguments

x	object of class SDF or SDFset

Details

...

Value

named `character` vector if `SDF` is provided or `list` of named `character` vectors if `SDFset` is provided

Author(s)

Thomas Girke

References

...

See Also

`atomblock`, `atomcount`, `bondblock`, `datablock`, `cid`, `sdfid`

Examples

```
## SDF/SDFset instances
data(sdfsample)
sdfset <- sdfsample
sdf <- sdfset[[1]]

## Extract header block
header(sdf)
header(sdfset[1:4])

## Replacement methods
sdfset[[1]][[1]][1] <- "test"
sdfset[[1]]
header(sdfset)[1] <- header(sdfset[2])
view(sdfset[1:2])
```

`makeUnique` *Uniquify CMP names*

Description

Creates unique CMP names by appending a counter to each duplicatation set. The function can be used for any character vector.

Usage

```
makeUnique(x, silent = FALSE)
```

Arguments

`x`	`character` vector
`silent`	`silent = TRUE` suppresses message about duplicate count

Details

The function is important to maintain unique compound names in the ID slot of `SDFset` containers.

Value

`character` of same length as x but without duplications

Author(s)

Thomas Girke

References

...

See Also

Functions: `cid`, `sdfid`

Examples

```
## SDFset instance
data(sdfsample)
sdfset <- sdfsample

## Create unique compound IDs
unique_ids <- makeUnique(sdfid(sdfset))
cid(sdfset) <- unique_ids
cid(sdfset[1:4])
```

---

plotStruc *Plot compound structures*

Description

Plots compound structure(s) for molecules stored in SDF and SDFset containers.

Usage

```
## Convenience plot method
# plot(x, griddim, print_cid=cid(x), print=TRUE, ...)

## Less important for user
plotStruc(sdf, atomcex = 1.2, atomnum = FALSE, no_print_atoms = c("C"),
          noHbonds = TRUE, bondspacer = 0.12, ...)
```

Arguments

`sdf`	Object of class `SDF`
`atomcex`	Font size for atom labels
`atomnum`	If `TRUE`, then the atom numbers are included in the plot. They are the position numbers of each atom in the atom block of an SDF.
`no_print_atoms`	Excludes specified atoms from being plotted.

noHbonds	If TRUE, then the C-hydrogens and their bonds - explicitly defined in an SDF - are excluded from the plot.
bondspacer	Numeric value specifying the plotting distance for double/triple bonds.
...	Arguments to be passed to/from other methods.

Details

The function plotStruc depicts a single 2D compound structure based on the XY-coordinates specified in the atom block of an SDF. The generic method plot can be used as a convenient shorthand to plot one or many structures at once. Both functions depend on the availability of the XY-coordinates in the source SD file and only 2D (not 3D) representations are plotted correctly.

Additional arguments that can only be passed on to the plot function when supplied with an SDFset object:

griddim: numeric vector of length two to define the dimensions for arranging several structures in one plot.

print_cid: character vector for printing custom compound labels. Default is print_cid=cid(sdfset).

print: if print=TRUE, then a summary of the SDF content for each supplied compound is printed to the screen. This behavior is turned off with print=TRUE.

Value

Prints summary of SDF/SDFset to screen and plots their structures to graphics device.

Note

The compound depictions created by this function are not as pretty as the structure representations generated with the sdf.visualize function. This will be improved in the future.

Author(s)

Thomas Girke

References

...

See Also

sdf.visualize

Examples

```
## Import SDFset sample set
data(sdfsample)
(sdfset <- sdfsample)

## Plot single compound structure
plotStruc(sdfset[[1]])

## Plot several compounds structures
plot(sdfset[1:4])

## Customize plot
plot(sdfset[1:4], griddim=c(2,2), print_cid=letters[1:4], print=FALSE, noHbonds=FALSE)
```

pubchemFPencoding *Enncoding of PubChem Fingerprints*

Description

Data frame with bit positions and substructure specifications.

Usage

```
data(pubchemFPencoding)
```

Format

The format is a data frame with 881 rows and 2 columns.

Source

From: ftp://ftp.ncbi.nih.gov/pubchem/specifications/pubchem_fingerprints.txt

References

See: ftp://ftp.ncbi.nih.gov/pubchem/specifications/pubchem_fingerprints.txt

Examples

```
data(pubchemFPencoding)
pubchemFPencoding[1:4,]
```

read.SDFset *SD file to 'SDFset'*

Description

Imports one or many molecules from an SD/MOL file and stores it in an `SDFset` container.

Usage

```
read.SDFset(sdfstr = sdfstr, ...)
```

Arguments

`sdfstr`	path/name to an SD file; alternatively an `SDFstr` object can be provided
`...`	option to pass on additional arguments

Details

...

Value

`SDFset`	for details see ?"SDFset-class"

Author(s)

Thomas Girke

References

SDF format defintion: http://www.symyx.com/downloads/public/ctfile/ctfile.jsp

See Also

Functions: `read.SDFstr`

Examples

```
## Write instance of SDFset class to SD file
data(sdfsample); sdfset <- sdfsample
# write.SDF(sdfset[1:4], file="sub.sdf")

## Import SD file
# read.SDFset("sub.sdf")

## Pass on SDFstr object
sdfstr <- as(sdfset, "SDFstr")
read.SDFset(sdfstr)
```

`read.SDFstr` *SD file to 'SDFstr'*

Description

Imports one or many molecules from an SD/MOL file and stores it in an `SDFstr` container.

Usage

```
read.SDFstr(sdfstr)
```

Arguments

`sdfstr`	path/name to an SD file

Details

...

Value

`SDFstr`	for details see ?"SDFstr-class"

Author(s)

Thomas Girke

References

SDF format defintion: http://www.symyx.com/downloads/public/ctfile/ctfile.jsp

See Also

Functions: `read.SDFset`

Examples

```
## Write instance of SDFstr class to SD file
data(sdfsample); sdfset <- sdfsample
sdfstr <- as(sdfset, "SDFstr")
# write.SDF(sdfset[1:4], file="sub.sdf")

## Import SD file
# read.SDFstr("sub.sdf")

## Pass on SDFstr object
sdfstr <- as(sdfset, "SDFstr")
read.SDFset(sdfstr)
```

---

rings *Ring and Aromaticity Perception*

Description

Identifies all possible rings in molecules using the exhaustive ring perception algorithm from Hanser et al (1996). In addition, the function can return all smallest possible rings as well as aromaticity information for each ring.

Usage

```
rings(x, upper = Inf, type = "all", arom = FALSE, inner = FALSE)
```

Arguments

`x`	`SDF` or `SDFset` containers
`upper`	allows to specify an upper length limit for ring predictions. The default setting `upper=Inf` will return all possible rings. Smaller length limits will reduce the search space resulting in shortened compute times.
`type`	if `type="all"`, the function returns each ring of a compound as character vector of atom symbols that are numbered by their position in the atom block of an `SDF/SDFset` object. Note: the example below shows how to plot structures with the same numbering information for visual inspection. If `type="arom"`, only aromatic rings are returned, while `type="count"` returns the ring and/or aromaticity counts for each compound in a matrix.
`arom`	if `arom="TRUE"`, ring aromaticity information will be computed. If `type="all"`, the output is a logical vector where 'TRUE' values indicate aromatic rings in the associated ring list. If `type="arom"`, then the function returns only aromatic rings. A ring is considered aromatic if it meets the following requirements: (i) all atoms in the ring need to be sp2 hybridized. This means each atom has to

have a double bond or at least one lone electron pair and it needs to be attached to an sp2 hybridized atom. (ii) In addition, Hueckel's rule '4n + 2' needs to be true, where 'n' is either zero or any positive integer.

`inner` if `inner="TRUE"`, only inner (smallest possible) rings will be returned. They are identified by first computing all possible rings and then selecting only the inner rings. Note: this requires the setting `upper=Inf`. If only rings below a certain size limit (e.g. 6) are of interest, then it will be more time efficient to set this limmit under the `upper` argument than identifying all smallest rings.

Details

...

Value

The settings `type="all"` and `type="arom"` return `lists`, and `type="count"` returns a `matrix`.

Author(s)

Thomas Girke

References

Hanser, Jauffret and Kaufmann (1996) A New Algorithm for Exhaustive Ring Perception in a Molecular Graph. Journal of Chemical Information and Computer Sciences, 36: 1146-1152. URL: http://pubs.acs.org/doi/abs/10.1021/ci960322f

See Also

...

Examples

```
## Instances of SDFset class
data(sdfsample)
sdfset <- sdfsample

## Return all possible rings for a single compound
rings(sdfset[1], upper=Inf, type="all", arom=FALSE, inner=FALSE)
plot(sdfset[1], print=FALSE, atomnum=TRUE, no_print_atoms="H")

## Return all possible rings for several compounds plus their
## aromaticity information
rings(sdfset[1:4], upper=Inf, type="all", arom=TRUE, inner=FALSE)

## Return rings with no more than 6 atoms
rings(sdfset[1:4], upper=6, type="all", arom=TRUE, inner=FALSE)

## Return rings with no more than 6 atoms that are also armomatic
rings(sdfset[1:4], upper=6, type="arom", arom=TRUE, inner=FALSE)

## Return shortest possible rings (no complex rings)
rings(sdfset[1:4], upper=Inf, type="all", arom=TRUE, inner=TRUE)
```

```
## Count shortest possible rings
rings(sdfset[1:4], upper=Inf, type="count", arom=TRUE, inner=TRUE)
```

`sdf.subset` *Subset a SDF and return SDF segements for selected compounds*

Description

'sdf.subset' will take a descriptor database generated by 'cmp.parse' and an array of indices, and return an SDF string consisting of SDFs for compounds corresponding to that list of indices. The returned value is a character string.

Usage

```
sdf.subset(db, cmps)
```

Arguments

`db`	The database generated by 'cmp.parse'
`cmps`	An array of indecies that correspond to a set of selected compounds from the database

Details

'sdf.subset' depends on information embedded in the descriptor database returned by 'cmp.parse'. It also relies on the availability of the original SDF where the database has been generated from. Basically, when 'cmp.parse' parses the original SDF file, it will store the path of that SDF file as well as offset information for SDF segment in that file. Therefore, if the SDF file has been changed or deleted, 'sdf.subset' cannot function properly.

The result SDF will also have names added to compounds if they are not present in the original SDF.

Value

Return a character string whose content is the concatenation of SDFs for the selected compounds.

See Also

`cmp.parse`, `sdf.visualize`

Examples

```
## Note: this functionality has become obsolete since the introduction of the
## 'SDFset' and 'apset' S4 classes.

# load sample database from web
# db <- cmp.parse("http://bioweb.ucr.edu/ChemMineV2/static/example_db.sdf")
# select SDF for 1st and 2nd compound in that SDF
# sdf_segments <- sdf.subset(db, c(1, 2))
# now sdf_segments containt the 2 SDFs for those 2 compounds
```

`sdf.visualize` *Subset a SDFset and visualize selected compounds in a webpage*

Description

'sdf.visualize' will take a descriptor database generated by 'cmp.parse' and an array of indices, send an SDF consisting structure information of compounds indexed by this array to ChemMine (http://bioweb.ucr.edu/ChemMineV2), and open a webpage that shows the structures of these compounds. It returns the URL of that page.

Usage

```
sdf.visualize(db, cmps, extra=NULL, reference.sdf=NULL, reference.note=NULL, bro
```

Arguments

`db`	The database generated by 'cmp.parse'
`cmps`	A vector of indecies that correspond to a set of selected compounds from the database
`extra`	A vector or list of character strings or matrices or data frames, each entry of which gives extra description on the compounds being visualized.
`reference.sdf`	A character string of SDF or a filename of an SDF file for the reference compound.
`reference.note`	Note to be displayed with the reference compound.
`browse`	Whether to open the webpage automatically after the upload is finished
`quiet`	Whether to display the progress information

Details

'sdf.visualize' uses `sdf.subset` to extract the SDF for the selected compounds. Therefore, 'sdf.visualize' also depends on information embedded in the descriptor database returned by 'cmp.parse'. It also relies on the availability of the original SDF file where the database has been generated from. Basically, when 'cmp.parse' parses the original SDF file, it will store the path of that SDF file as well as offset information for SDF segment in that file. Therefore, if the SDF file has been changed or deleted, 'sdf.visualize' cannot function properly.

After extracting the SDF segments for the selected compounds, 'sdf.visualize' will send the SDF to ChemMine (http://bioweb.ucr.edu/ChemMineV2) using HTTP POST method. ChemMine will generate the 2D images for the selected compounds and a webpage containing these images as well as the SDFs. The URL is returned by 'sdf.visualize'. If 'browse' is set to TRUE, the URL will be opened by your default browser.

If the argument 'extra' is given, it must be a vector or list of character strings or data frames or matrices. The length of the vector or list must be the same as that of the indices. Each entry may be named or not. Each entry of this vector is a character string giving extra description on a compound. This vector will be sent to ChemMine, and the extra description for a compound will be listed at the right hand side of the compound. Data frames or matrices will be formatted and displayed as they would be formatted by the 'print' function.

The 'reference.sdf' argument is given when you want to upload an extra compound as a reference compound. This compound will be displayed at the top of the visualization web page. This argument can be a character string of SDF(s), or it can be a filename or URL that points to an SDF file. If the string or the file contains multiple SDFs, this function will use the first one.

If a reference compound is uploaded, note about this compound can be set via the 'reference.note' argument. This note will be displayed next to the structure of the compound on the resulting webpage.

Value

Returns the URL of the webpage containing all the SDFs and 2D images corresponding to the selected compounds.

See Also

cmp.parse, sdf.subset, plotStruc

Examples

```
## Load sample SD file
data(sdfsample)
sdfset <- sdfsample

## Not run:
## Plot structures using web service ChemMine Tools
sdf.visualize(sdfset[1:4])

## Add extra annotation as vector
sdf.visualize(sdfset[1:4], extra=month.name[1:4])

## Add extra annotation as matrix
extra <- apply(propma[1:4,], 1, function(x) data.frame(Property=colnames(propma), Value=x
sdf.visualize(sdfset[1:4], extra=extra)

## Add extra annotation as list
sdf.visualize(sdfset[1:4], extra=bondblock(sdfset[1:4]))

## End(Not run)
```

---

sdf2ap *Atom pair library*

Description

Creates from a SDFset a searchable atom pair library that is stored in a container of class APset.

Usage

```
sdf2ap(sdfset)
```

Arguments

sdfset	Objects of classes SDFset or SDF

Details

...

Value

`APset`	if input is `SDFset`
`AP`	if input is `SDF`

Author(s)

Thomas Girke

References

Chen X and Reynolds CH (2002). "Performance of similarity measures in 2D fragment-based similarity searching: comparison of structural descriptors and similarity coefficients", J Chem Inf Comput Sci.

See Also

Functions: `SDF2apcmp`, `apset2descdb`, `cmp.search`, `cmp.similarity`

Related classes: SDF, SDFset, SDFstr, APset.

Examples

```
## Instance of SDFset class
data(sdfsample)
sdfset <- sdfsample[1:50]
sdf <- sdfsample[[1]]

## Compute atom pair library
ap <- sdf2ap(sdf)
(apset <- sdf2ap(sdfset))
view(apset[1:4])

## Return main components of APset object
cid(apset[1:4]) # compound IDs
ap(apset[1:4]) # atom pair descriptors

## Return atom pairs in human readable format
db.explain(apset[1])

## Coerce APset to other objects
apset2descdb(apset) # returns old list-style AP database
tmp <- as(apset, "list") # returns list
as(tmp, "APset") # converst list back to APset

## Compound similarity searching with APset
cmp.search(apset, apset[1], type=3, cutoff=0.2)
plot(sdfset[names(cmp.search(apset, apset[6], type=2, cutoff=0.4))])

## Identify compounds with identical AP sets
cmp.duplicated(apset, type=2)
```

```
## Structure similarity clustering
cmp.cluster(db=apset, cutoff = c(0.65, 0.5))[1:20,]
```

`sdf2list` *'SDF' to 'list'*

Description

Returns objects of class `SDF` as `list`.

Usage

```
sdf2list(x)
```

Arguments

`x`	object of class `SDF`

Details

...

Value

`list`	with following components:
`character`	SDF header block
`matrix`	SDF bond block
`matrix`	SDF atom block
`character`	SDF data block

Author(s)

Thomas Girke

References

SDF format definition: http://www.symyx.com/downloads/public/ctfile/ctfile.jsp

See Also

Functions: `sdfstr2list`, `sdf2str`, `SDFset2list`, `SDFset2SDF`

Examples

```
## Instance of SDF class
data(sdfsample); sdfset <- sdfsample
sdf <- sdfset[[1]]

## Return as list
sdf2list(sdf)
as(sdf, "list") # similar result
```

`sdf2smiles` *'SDFset' to 'character' Convert 'SDFset' to SMILES ('character')*

Description

Accepts one compound in an `SDFset` container and returns the corresponding SMILES string (Simplified Molecular Input Line Entry Specification). The compound is submitted to the ChemMine Tools web service for conversion with the Open Babel Open Source Chemistry Toolbox. If the input object contains multiple items, only the first is converted.

Usage

```
sdf2smiles(sdf)
```

Arguments

`sdf`	A `SDFset` object which contains one compound

Value

`character`	for details see ?"character"

Author(s)

Tyler Backman

References

Chemmine web service: http://chemmine.ucr.edu

Open Babel: http://openbabel.org

SMILES Format: http://en.wikipedia.org/wiki/Chemical_file_format#SMILES

Examples

```
## Not run:
## get a sample compound
data(sdfsample); sdfset <- sdfsample[1]
## convert to smiles
smiles <- sdf2smiles(sdfset)
## End(Not run)
```

`sdf2str` *'SDF' to 'SDFstr'*

Description

Converts `SDF` to `SDFstr`. Its main use is to facilitate the export to SD files. It contains optional arguments to generate custom SDF output.

Usage

```
sdf2str(sdf, head, ab, bb, db, cid = NULL, sig = FALSE, ...)
```

Arguments

`sdf`	object of class `SDF`
`head`	optional `character` vector to supply custom header block
`ab`	optional `matrix` to supply custom atom block
`bb`	optional `matrix` to supply custom bond block
`db`	optional `character` vector to supply custom data block
`cid`	`character` can be provided to inject custom compound ID into header block
`sig`	`if = TRUE` then the ChemmineR signature will be injected into the header block for tracking purposes
`...`	option to pass on additional arguments

Details

If the export function `write.SDF` is supplied with an `SDFset` object, then `sdf2str` is used internally to customize the export of many molecules to a single SD file using the same optional arguments.

Value

`sdfstr`	SDF data of one molecule collapsed to character vector

Author(s)

Thomas Girke

References

SDF format definition: http://www.symyx.com/downloads/public/ctfile/ctfile.jsp

See Also

Coerce functions: `sdfstr2list`, `sdf2str`, `SDFset2list`, `SDFset2SDF`

Export function: `write.SDF`

Examples

```
## Instance of SDF class
data(sdfsample); sdfset <- sdfsample
sdf <- sdfset[[1]]

## Customize SDF blocks for export to SD file
sdf2str(sdf=sdf, sig=TRUE, cid=TRUE) # uses default SDF components
sdf2str(sdf=sdf, head=letters[1:4], db=NULL) # uses custom components for header and data

## The same arguments can be supplied to the write.SDF function for
## batch export of custom SDFs
# write.SDF(sdfset[1:4], file="sub.sdf", sig=TRUE, cid=TRUE, db=NULL)
```

---

sdfid *Return SDF compound IDs*

Description

Returns the compound identifiers from the header block of `SDF` or `SDFset` objects.

Usage

```
sdfid(x, tag = 1)
```

Arguments

`x`	object of class `SDFset` or `SDF`
`tag`	values from 1-4 to extract different header block fields; SDF ID is in first one (default)

Details

...

Value

`character` vector

Author(s)

Thomas Girke

References

...

See Also

`atomblock`, `atomcount`, `bondblock`, `datablock`, `header`, `cid`

Examples

```
## SDF/SDFset instances
data(sdfsample)
sdfset <- sdfsample
sdf <- sdfset[[1]]

## Extract IDs from header block
sdfid(sdf, tag=1)
sdfid(sdfset[1:4])

## Extract compound IDs from ID slot in SDFset container
cid(sdfset[1:4])

## Assigning compound IDs and keeping them unique
unique_ids <- makeUnique(sdfid(sdfset))
cid(sdfset) <- unique_ids
cid(sdfset[1:4])
```

---

sdfsample *SD file in 'SDFset' object*

Description

First 100 compounds from PubChem SD file: Compound_00650001_00675000.sdf.gz

Usage

```
data(sdfsample)
```

Format

Object of class `sdfset`

Details

Object stores 100 molecules from a sample SD file.

Source

ftp://ftp.ncbi.nih.gov/pubchem/Compound/CURRENT-Full/SDF/Compound_00650001_00675000.sdf.gz

References

SDF format definition: http://www.symyx.com/downloads/public/ctfile/ctfile.jsp

Examples

```
data(sdfsample)
sdfset <- sdfsample
view(sdfset[1:4])
```

sdfstr2list *'SDFstr' to 'list'*

Description

Returns objects of class SDFstr as list.

Usage

```
sdfstr2list(x)
```

Arguments

x	object of class SDFstr

Details

...

Value

list	with many of the following components:
character	SDF content of one molecule vectorized line by line

Author(s)

Thomas Girke

References

SDF format definition: http://www.symyx.com/downloads/public/ctfile/ctfile.jsp

See Also

Functions: sdf2list, sdf2str, SDFset2list, SDFset2SDF

Examples

```
## Instance of SDFstr class
data(sdfsample); sdfset <- sdfsample
sdfstr <- as(sdfset, "SDFstr")

## Return as list
sdfstr2list(sdfstr)
as(sdfstr, "list") # similar result
```

searchSim *PubChem Similarity (Fingerprint) Search*

Description

Accepts one SDFset container and performs a >0.95 similarity PubChem fingerprint search, returning the hits in an SDFset container. The ChemMine Tools web service is used as an intermediate, to translate queries from plain HTTP POST to a PubChem Power User Gateway (PUG) query. If the input object contains multiple items, only the first is used as a query.

Usage

```
searchSim(sdf)
```

Arguments

sdf	A SDFset object which contains one compound

Value

SDFset	for details see ?"SDFset-class"

Author(s)

Tyler Backman

References

PubChem PUG SOAP: http://pubchem.ncbi.nlm.nih.gov/pug_soap/pug_soap_help.html

Chemmine web service: http://chemmine.ucr.edu

PubChem help: http://pubchem.ncbi.nlm.nih.gov/search/help_search.html

SMILES Format: http://en.wikipedia.org/wiki/Chemical_file_format#SMILES

Examples

```
## Not run:
## get a sample compound
data(sdfsample); sdfset <- sdfsample[1]
## search a compound on PubChem
compounds <- searchSim(sdfset)
## End(Not run)
```

searchString *PubChem Similarity (Fingerprint) SMILES Search*

Description

Accepts one SMILES string (Simplified Molecular Input Line Entry Specification) and performs a >0.95 similarity PubChem fingerprint search, returning the hits in an `SDFset` container. The ChemMine Tools web service is used as an intermediate, to translate queries from plain HTTP POST to a PubChem Power User Gateway (PUG) query.

Usage

```
searchString(smiles)
```

Arguments

`smiles`	A `character` object which contains one SMILES string

Value

`SDFset`	for details see ?"SDFset-class"

Author(s)

Tyler Backman

References

PubChem PUG SOAP: http://pubchem.ncbi.nlm.nih.gov/pug_soap/pug_soap_help.html

Chemmine web service: http://chemmine.ucr.edu

PubChem help: http://pubchem.ncbi.nlm.nih.gov/search/help_search.html

SMILES Format: http://en.wikipedia.org/wiki/Chemical_file_format#SMILES

Examples

```
## Not run:
## search a compound on PubChem
compounds <- searchString("CC(=O)OC1=CC=CC=C1C(=O)O")
## End(Not run)
```

smiles2sdf *Convert SMILES ('character') to 'SDFset'*

Description

Accepts one compound as a SMILES string (Simplified Molecular Input Line Entry Specification) and returns it's equivalent as an `SDFset` container. The compound is submitted to the ChemMine Tools web service for conversion with the Open Babel Open Source Chemistry Toolbox.

Usage

```
smiles2sdf(smiles)
```

Arguments

`smiles`	A `character` object which contains one SMILES string

Value

`SDFset`	for details see ?"SDFset-class"

Author(s)

Tyler Backman

References

Chemmine web service: http://chemmine.ucr.edu

Open Babel: http://openbabel.org

SMILES Format: http://en.wikipedia.org/wiki/Chemical_file_format#SMILES

Examples

```
## Not run:
## convert to sdf
sdf <- smiles2sdf("CC(=O)OC1=CC=CC=C1C(=O)O\tname")
## End(Not run)
```

validSDF *Validity check of SDFset*

Description

Performs validity check of SDFs stored in `SDFset` objects. Currently, the function tests whether the atom block and the bond block in each `SDF` component of an `SDFset` have at least `Nabcol` and `Nbbcol` columns (default is 3 for both). The function returns a logical vector with `TRUE` values for valid compounds and `FALSE` values for invalid ones.

Usage

```
validSDF(x, Nabcol = 3, Nbbcol = 3, logic = "&")
```

Arguments

x	x object of class SDFset
Nabcol	minimum number of columns in atom block
Nbbcol	minimum number of columns in bond block
logic	logical connection (& or |) among Nabcol and Nbbcol cutoffs

Details

The function is important to remove invalid compounds from SDFset containers.

Value

logical vector of length x with TRUE for valid compounds and FALSE for invalid compounds.

Author(s)

Thomas Girke

References

...

See Also

Functions: read.SDFset

Examples

```
## SDFset instance
data(sdfsample)
sdfset <- sdfsample

## Detect and remove invalid SDFs in SDFset.
valid <- validSDF(sdfset)
which(!valid) # Returns index for invalid SDFs
sdfset <- sdfset[valid] # Returns only valid SDFs.
```

---

view *Viewing of complex objects*

Description

Convenience function for viewing the content of complex objects like SDFset and APset containers. The function is a shorthand wrapper for as(sdfset, "SDF") and as(apset, "AP").

Usage

```
view(x)
```

Arguments

x	object of class SDFset or APset

Details

...

Value

List populated with SDF and AP components.

Author(s)

Thomas Girke

References

...

See Also

Classes: SDF, SDFset, AP, APset

Examples

```
## Viewing content of SDFset
data(sdfsample); sdfset <- sdfsample
view(sdfset[1:4])

## Viewing content of APset
apset <- sdf2ap(sdfset[1:10])
view(apset)
```

---

write.SDF *SDF export function*

Description

Writes one or many molecules stored in a SDFset, SDFstr or SDF object to SD file.

Usage

```
write.SDF(sdf, file, cid = FALSE, ...)
```

Arguments

sdf	object of class SDFset, SDFstr or SDF
file	name of SD file to write to
cid	if cid = TRUE and an SDFset object is provide as input, then the compound IDs in the ID slot of the SDFset are used for compound naming
...	the optional arguments of the sdf2str function can be provided here, including head, ab, bb, db; details are provided in the help page for the sdf2str function

Details

If the `write.SDF` function is supplied with an `SDFset` object, then it uses internally the `sdf2str` function to allow customizing the resulting SD file. For this all optional arguments of the `sdf2str` function can be passed on to `write.SDF`.

Author(s)

Thomas Girke

References

SDF format definition: http://www.symyx.com/downloads/public/ctfile/ctfile.jsp

See Also

Import function: `read.SDFset`, `read.SDFstr`

Examples

```
## Instance of SDFset class
data(sdfsample); sdfset <- sdfsample

## Write objects of classes SDFset/SDFstr/SDF to file
# write.SDF(sdfset[1:4], file="sub.sdf")

## Example for writing customized SDFset to file containing
## ChemmineR signature, IDs from SDFset and no data block
# write.SDF(sdfset[1:4], file="sub.sdf", sig=TRUE, cid=TRUE, db=NULL)

## Example for injecting a custom matrix/data frame into the data block of an
## SDFset and then writing it to an SD file
props <- data.frame(MF=MF(sdfset), MW=MW(sdfset), atomcountMA(sdfset))
datablock(sdfset) <- props
view(sdfset[1:4])
# write.SDF(sdfset[1:4], file="sub.sdf", sig=TRUE, cid=TRUE)
```

Index