

# Package: rcdk (via r-universe)

October 24, 2024

**Version** 3.8.1

**Date** 2023-06-25

**Title** Interface to the 'CDK' Libraries

**Depends** rcdklibs (>= 2.8)

**Imports** fingerprint, rJava, methods, png, iterators, itertools

**Suggests** xtable, RUnit, knitr, rmarkdown, devtools

**License** LGPL

**LazyLoad** yes

**LazyData** true

**SystemRequirements** Java (>= 8)

**Description** Allows the user to access functionality in the 'CDK', a Java framework for chemoinformatics. This allows the user to load molecules, evaluate fingerprints, calculate molecular descriptors and so on. In addition, the 'CDK' API allows the user to view structures in 2D.

**RoxygenNote** 7.2.3

**VignetteBuilder** knitr

**Encoding** UTF-8

**NeedsCompilation** no

**Author** Rajarshi Guha [aut, cph], Zachary Charlop-Powers [cre], Emma Schymanski [ctb]

**Maintainer** Zachary Charlop-Powers <zach.charlop.powers@gmail.com>

**Repository** CRAN

**Date/Publication** 2023-07-01 22:20:02 UTC

## Contents

Atoms . . . . .	3
bpdata . . . . .	4
cdk.version . . . . .	5

cdkFormula-class . . . . .	5
compare.isotope.pattern . . . . .	6
convert.implicit.to.explicit . . . . .	7
copy.image.to.clipboard . . . . .	7
do.aromaticity . . . . .	8
do.isotopes . . . . .	8
eval.atomic.desc . . . . .	8
eval.desc . . . . .	9
generate.2d.coordinates . . . . .	10
generate.formula . . . . .	10
generate.formula.iter . . . . .	11
get.adjacency.matrix . . . . .	12
get.alogp . . . . .	13
get.atom.count . . . . .	13
get.atom.index . . . . .	14
get.atomic.desc.names . . . . .	14
get.atomic.number . . . . .	15
get.atoms . . . . .	16
get.bond.order . . . . .	16
get.bonds . . . . .	17
get.charge . . . . .	18
get.chem.object.builder . . . . .	18
get.connected.atom . . . . .	19
get.connected.atoms . . . . .	20
get.connection.matrix . . . . .	20
get.depictor . . . . .	21
get.desc.categories . . . . .	22
get.desc.names . . . . .	22
get.element.types . . . . .	23
get.exact.mass . . . . .	24
get.exhaustive.fragments . . . . .	24
get.fingerprint . . . . .	25
get.formal.charge . . . . .	27
get.formula . . . . .	28
get.hydrogen.count . . . . .	28
get.isotope.pattern.generator . . . . .	29
get.isotope.pattern.similarity . . . . .	29
get.isotopes.pattern . . . . .	30
get.largest.component . . . . .	30
get.mcs . . . . .	31
get.mol2formula . . . . .	32
get.murcko.fragments . . . . .	32
get.natural.mass . . . . .	33
get.point2d . . . . .	34
get.point3d . . . . .	35
get.properties . . . . .	36
get.property . . . . .	36
get.smiles . . . . .	37

get.smiles.parser	38
get.stereo.types	39
get.stereocenters	39
get.symbol	40
get.title	41
get.total.charge	41
get.total.formal.charge	42
get.total.hydrogen.count	42
get.tpsa	43
get.volume	43
get.xlogp	44
load.molecules	44
is.aliphatic	45
is.aromatic	46
is.connected	47
is.in.ring	47
is.neutral	48
isvalid.formula	49
load.molecules	49
matches	50
Molecule	51
parse.smiles	52
rdk-deprecated	53
remove.hydrogens	53
remove.property	54
set.atom.types	55
set.charge.formula	55
set.property	56
set.title	56
smiles.flavors	57
view.image.2d	59
view.molecule.2d	59
view.table	60
write.molecules	60
<b>Index</b>	<b>62</b>

**Description**

`get.symbol` returns the chemical symbol for an atom `get.point3d` returns the 3D coordinates of the atom `get.point2d` returns the 2D coordinates of the atom `get.atomic.number` returns the atomic number of the atom `get.hydrogen.count` returns the number of implicit H's on the atom. Depending on where the molecule was read from this may be NULL or an integer greater than or equal to 0 `get.charge` returns the partial charge on the atom. If charges have not been set the return value is NULL, otherwise the appropriate charge. `get.formal.charge` returns the

formal charge on the atom. By default the formal charge will be 0 (i.e., NULL is never returned) `is.aromatic` returns TRUE if the atom is aromatic, FALSE otherwise `is.aliphatic` returns TRUE if the atom is part of an aliphatic chain, FALSE otherwise `is.in.ring` returns TRUE if the atom is in a ring, FALSE otherwise `get.atom.index` returns the index of the atom in the molecule (starting from 0) `get.connected.atoms` returns a list of atoms that are connected to the specified atom

### Usage

```
get.symbol(atom) get.point3d(atom) get.point2d(atom) get.atomic.number(atom) get.hydrogen.count(atom)
get.charge(atom) get.formal.charge(atom) get.connected.atoms(atom, mol) get.atom.index(atom, mol)
is.aromatic(atom) is.aliphatic(atom) is.in.ring(atom) set.atom.types(mol)
```

### Arguments

atom A jobjRef representing an IAtom object mol A jobjRef representing an IAtomContainer object

### Value

In the case of `get.point3d` the return value is a 3-element vector containing the X, Y and Z coordinates of the atom. If the atom does not have 3D coordinates, it returns a vector of the form `c(NA, NA, NA)`. Similarly for `get.point2d`, in which case the return vector is of length 2.

### Author(s)

Rajarshi Guha (<rajarshi.guha@gmail.com>)

---

bpdata

*Boiling Point Data*

---

### Description

A dataset containing the structures and associated boiling points for 277 molecules, primarily alkanes and substituted alkanes.

### Usage

```
bpdata
```

### Format

A data frame with 277 rows and 2 columns.:

**SMILES** Structure in SMILES format

**BP** Boiling point in Kelvin

The names of the molecules are used as the row names.

**References**

Goll, E.S. and Jurs, P.C.; "Prediction of the Normal Boiling Points of Organic Compounds From Molecular Structures with a Computational Neural Network Model", *J. Chem. Inf. Comput. Sci.*, 1999, 39, 974-983.

---

cdk.version	<i>Get the current CDK version used in the package.</i>
-------------	---

---

**Description**

Get the current CDK version used in the package.

**Usage**

```
cdk.version()
```

**Value**

Returns a character containing the version of the CDK used in this package

**Author(s)**

Rajarshi Guha (<rajarshi.guha@gmail.com>)

---

cdkFormula-class	<i>Class cdkFormula, ac class for handling molecular formula</i>
------------------	--

---

**Description**

This class handles molecular formulae. It provides extra information such as the IMolecularFormula Java object, elements contained and number of them.

**Objects from the Class**

Objects can be created using new constructor and filled with a specific mass and window accuracy

**Author(s)**

Miguel Rojas-Cherto (<miguelrojasch@yahoo.es>)

**References**

A parallel effort to expand the Chemistry Development Kit: <https://cdk.github.io/>

**See Also**

[get.formula](#) [set.charge.formula](#) [get.isotopes](#) [pattern](#) [isvalid.formula](#)

---

compare.isotope.pattern

*Compare isotope patterns.*

---

### Description

Computes a similarity score between two different isotope abundance patterns.

### Usage

```
compare.isotope.pattern(iso1, iso2, ips = NULL)
```

### Arguments

iso1	The first isotope pattern, which should be a jobjRef corresponding to the IsotopePattern class
iso2	The second isotope pattern, which should be a jobjRef corresponding to the IsotopePattern class
ips	An instance of the IsotopePatternSimilarity class. if NULL one will be constructed automatically

### Value

A numeric value between 0 and 1 indicating the similarity between the two patterns

### Author(s)

Miguel Rojas Cherto

### References

<http://cdk.github.io/cdk/2.3/docs/api/org/openscience/cdk/formula/IsotopePatternSimilarity.html>

### See Also

[get.isotope.pattern.similarity](#)

---

`convert.implicit.to.explicit`*Convert implicit hydrogens to explicit.*

---

**Description**

In some cases, a molecule may not have any hydrogens (such as when read in from an MDL MOL file that did not have hydrogens or SMILES with no explicit hydrogens). In such cases, this method will add implicit hydrogens and then convert them to explicit ones. The newly added H's will not have any 2D or 3D coordinates associated with them. Ensure that the molecule has been typed beforehand.

**Usage**`convert.implicit.to.explicit(mol)`**Arguments**

<code>mol</code>	The molecule to query. Should be a 'jobRef' representing an 'IAtomContainer'
------------------	--

**Author(s)**

Rajarshi Guha (<rajarshi.guha@gmail.com>)

**See Also**

[get.hydrogen.count](#), [remove.hydrogens](#), [set.atom.types](#)

---

`copy.image.to.clipboard`*copy.image.to.clipboard*

---

**Description**

generate an image and make it available to the system clipboard.

**Usage**`copy.image.to.clipboard(molecule, depictor = NULL)`**Arguments**

<code>molecule</code>	The molecule to query. Should be a 'jobRef' representing an 'IAtomContainer'
<code>depictor</code>	Optional. Default NULL. Depictor from <code>get.depictor</code>

do.aromaticity      *do.aromaticity*

---

**Description**

detect aromaticity of an input compound

**Usage**

do.aromaticity(mol)

**Arguments**

mol                      The molecule to query. Should be a 'jobRef' representing an 'IAtomContainer'

---

do.isotopes              *do.isotopes*

---

**Description**

configure isotopes

**Usage**

do.isotopes(mol)

**Arguments**

mol                      The molecule to query. Should be a 'jobRef' representing an 'IAtomContainer'

---

eval.atomic.desc          *Compute descriptors for each atom in a molecule*

---

**Description**

Compute descriptors for each atom in a molecule

**Usage**

eval.atomic.desc(molecule, which.desc, verbose = FALSE)



**Arguments**

molecule	A molecule object
which.desc	A character vector of atomic descriptor class names
verbose	Optional. Default FALSE. Toggle verbosity.

**Value**

A 'data.frame' with atoms in the rows and descriptors in the columns

**Author(s)**

Rajarshi Guha (<rajarshi.guha@gmail.com>)

**See Also**

[get.atomic.desc.names](#)

---

eval.desc

*Compute descriptor values for a set of molecules*

---

**Description**

Compute descriptor values for a set of molecules

**Usage**

```
eval.desc(molecules, which.desc, verbose = FALSE)
```

**Arguments**

molecules	A 'list' of molecule objects
which.desc	A character vector listing descriptor class names
verbose	If 'TRUE', verbose output

**Value**

A 'data.frame' with molecules in the rows and descriptors in the columns. If a descriptor value cannot be computed for a molecule, 'NA' is returned.

**Author(s)**

Rajarshi Guha (<rajarshi.guha@gmail.com>)

---

generate.2d.coordinates

*Generate 2D coordinates for a molecule.*

---

### Description

Some file formats such as SMILES do not support 2D (or 3D) coordinates for the atoms. Other formats such as SD or MOL have support for coordinates but may not include them. This method will generate reasonable 2D coordinates based purely on connectivity information, overwriting any existing coordinates if present.

### Usage

```
generate.2d.coordinates(mol)
```

### Arguments

mol                    The molecule to query. Should be a 'jobRef' representing an 'IAtomContainer'

### Details

Note that when depicting a molecule ([view.molecule.2d](#)), 2D coordinates are generated, but since it does not modify the input molecule, we do not have access to the generated coordinates.

### Value

The input molecule, with 2D coordinates added

### Author(s)

Rajarshi Guha (<rajarshi.guha@gmail.com>)

### See Also

[get.point2d](#), [view.molecule.2d](#)

---

generate.formula

*generate.formula*

---

### Description

generate.formula

**Usage**

```

generate.formula(
  mass,
  window = 0.01,
  elements = list(c("C", 0, 50), c("H", 0, 50), c("N", 0, 50), c("O", 0, 50), c("S", 0,
    50)),
  validation = FALSE,
  charge = 0
)

```

**Arguments**

mass	Required. Mass.
window	Optional. Default 0.01
elements	Optional. Default list(c('C', 0,50), c('H', 0,50),c('N', 0,50), c('O', 0,50), c('S', 0,50))
validation	Optional. Default FALSE
charge	Optional. Default FALSE

---

generate.formula.iter *generate.formula.iter*

---

**Description**

Generate a list of possible formula objects given a mass and a mass tolerance.

**Usage**

```

generate.formula.iter(
  mass,
  window = 0.01,
  elements = list(c("C", 0, 50), c("H", 0, 50), c("N", 0, 50), c("O", 0, 50), c("S", 0,
    50)),
  validation = FALSE,
  charge = 0,
  as.string = TRUE
)

```

**Arguments**

mass	Required. Mass.
window	Optional. Default 0.01
elements	Optional. Default list(c('C', 0,50), c('H', 0,50),c('N', 0,50), c('O', 0,50), c('S', 0,50))
validation	Optional. Default FALSE

charge	Optional. Default FALSE
as.string	Optional. Default FALSE

---

get.adjacency.matrix *Get adjacency matrix for a molecule.*

---

### Description

The adjacency matrix for a molecule with  $N$  non-hydrogen atoms is an  $N \times N$  matrix where the element  $[i,j]$  is set to 1 if atoms  $i$  and  $j$  are connected by a bond, otherwise set to 0.

### Usage

```
get.adjacency.matrix(mol)
```

### Arguments

mol            A jobjRef object with Java class IAtomContainer

### Value

A  $N \times N$  numeric matrix

### Author(s)

Rajarshi Guha <rajarshi.guha@gmail.com>

### See Also

[get.connection.matrix](#)

### Examples

```
m <- parse.smiles("CC=C")[[1]]
get.adjacency.matrix(m)
```

---

get.alogp	<i>Compute ALogP for a molecule</i>
-----------	-------------------------------------

---

**Description**

Compute ALogP for a molecule

**Usage**

```
get.alogp(molecule)
```

**Arguments**

molecule      A molecule object

**Value**

A double value representing the ALogP value

**Author(s)**

Rajarshi Guha (<rajarshi.guha@gmail.com>)

---

get.atom.count	<i>Get the number of atoms in the molecule.</i>
----------------	---

---

**Description**

Get the number of atoms in the molecule.

**Usage**

```
get.atom.count(mol)
```

**Arguments**

mol              The molecule to query. Should be a 'jobRef' representing an 'IAtomContainer'

**Value**

An integer representing the number of atoms in the molecule

**Author(s)**

Rajarshi Guha (<rajarshi.guha@gmail.com>)

---

<code>get.atom.index</code>	<i>get.atom.index</i>
-----------------------------	-----------------------

---

**Description**

Get the index of an atom in a molecule.

**Usage**

```
get.atom.index(atom, mol)
```

**Arguments**

<code>atom</code>	The atom object
<code>mol</code>	The 'IAtomContainer' object containing the atom

**Details**

Access the index of an atom in the context of an IAtomContainer. Indexing starts from 0. If the index is not known, -1 is returned.

**Value**

An integer representing the atom index.

**Author(s)**

Rajarshi Guha (<rajarshi.guha@gmail.com>)

**See Also**

[get.connected.atom](#)

---

<code>get.atomic.desc.names</code>	<i>Get class names for atomic descriptors</i>
------------------------------------	---

---

**Description**

Get class names for atomic descriptors

**Usage**

```
get.atomic.desc.names(type = "all")
```

**Arguments**

type            A string indicating which class of descriptors to return. Specifying "all" will return class names for all molecular descriptors. Options include \* topological \* geometrical \* hybrid \* constitutional \* protein \* electronic

**Value**

A character vector containing class names for atomic descriptors

**Author(s)**

Rajarshi Guha (<rajarshi.guha@gmail.com>)

---

`get.atomic.number`      *get.atomic.number*

---

**Description**

Get the atomic number of the atom.

**Usage**

`get.atomic.number(atom)`

**Arguments**

atom            The atom to query

**Value**

An integer representing the atomic number

**Author(s)**

Rajarshi Guha (<rajarshi.guha@gmail.com>)

---

get.atoms                      *Get the atoms from a molecule or bond.*

---

**Description**

Get the atoms from a molecule or bond.

**Usage**

```
get.atoms(object)
```

**Arguments**

object                      A 'jobRef' representing either a molecule ('IAtomContainer') or bond ('IBond') object.

**Value**

A list of 'jobRef' representing the 'IAtom' objects in the molecule or bond

**Author(s)**

Rajarshi Guha (<rajarshi.guha@gmail.com>)

**See Also**

[get.bonds](#), [get.connected.atoms](#)

---

get.bond.order                      *Get an object representing bond order*

---

**Description**

This function returns a Java enum representing a bond order. This can be used to modify the order of pre-existing bonds

**Usage**

```
get.bond.order(order = "single")
```

**Arguments**

order                      A character vector that can be one of single, double, triple, quadruple, quintuple, sextuple or unset. Case is ignored



**Value**

A `jobRef` representing an 'Order' enum object

**Author(s)**

Rajarshi Guha (<rajarshi.guha@gmail.com>)

**Examples**

```
## Not run:
m <- parse.smiles('CCN')[[1]]
b <- get.bonds(m)[[1]]
b$setOrder(get.bond.order("double"))

## End(Not run)
```

---

get.bonds

*Get the bonds in a molecule.*

---

**Description**

Get the bonds in a molecule.

**Usage**

```
get.bonds(mol)
```

**Arguments**

mol            A 'jobRef' representing the molecule ('IAtomContainer') object.

**Value**

A list of 'jobRef' representing the bonds ('IBond') objects in the molecule

**Author(s)**

Rajarshi Guha (<rajarshi.guha@gmail.com>)

**See Also**

[get.atoms](#), [get.connected.atoms](#)

get.charge

*get.charge*

---

**Description**

Get the charge on the atom.

**Usage**

```
get.charge(atom)
```

**Arguments**

atom                    The atom to query

**Details**

This method returns the partial charge on the atom. If charges have not been set the return value is NULL, otherwise the appropriate charge.

**Value**

An numeric representing the partial charge. If charges have not been set, 'NULL' is returned

**Author(s)**

Rajarshi Guha (<rajarshi.guha@gmail.com>)

**See Also**

[get.formal.charge](#)

---

get.chem.object.builder

*Get the default chemical object builder.*

---

**Description**

The CDK employs a builder design pattern to construct instances of new chemical objects (e.g., atoms, bonds, parsers and so on). Many methods require an instance of a builder object to function. While most functions in this package handle this internally, it is useful to be able to get an instance of a builder object when directly working with the CDK API via 'rJava'.

**Usage**

```
get.chem.object.builder()
```

**Details**

This method returns an instance of the [SilentChemObjectBuilder](#). Note that this is a static object that is created at package load time, and the same instance is returned whenever this function is called.

**Value**

An instance of [SilentChemObjectBuilder](#)

**Author(s)**

Rajarshi Guha (<rajarshi.guha@gmail.com>)

---

get.connected.atom      *Get the atom connected to an atom in a bond.*

---

**Description**

This function returns the atom that is connected to a specified in a specified bond. Note that this function assumes 2-atom bonds, mainly because the CDK does not currently support other types of bonds

**Usage**

```
get.connected.atom(bond, atom)
```

**Arguments**

bond	A jObjRef representing an 'IBond' object
atom	A jObjRef representing an 'IAtom' object

**Value**

A jObjRef representing an 'IAtom' object

**Author(s)**

Rajarshi Guha (<rajarshi.guha@gmail.com>)

**See Also**

[get.atoms](#)

get.connected.atoms    *get.connected.atoms*

---

### Description

Get atoms connected to the specified atom

### Usage

```
get.connected.atoms(atom, mol)
```

### Arguments

atom	The atom object
mol	The 'IAtomContainer' object containing the atom

### Details

Returns a 'list' of atoms that are connected to the specified atom.

### Value

A 'list' containing 'IAtom' objects, representing the atoms directly connected to the specified atom

### Author(s)

Rajarshi Guha (<rajarshi.guha@gmail.com>)

---

get.connection.matrix    *Get connection matrix for a molecule.*

---

### Description

The connection matrix for a molecule with  $N$  non-hydrogen atoms is an  $N \times N$  matrix where the element  $[i,j]$  is set to the bond order if atoms  $i$  and  $j$  are connected by a bond, otherwise set to 0.

### Usage

```
get.connection.matrix(mol)
```

### Arguments

mol	A jobRef object with Java class IAtomContainer
-----	--

### Value

A  $N \times N$  numeric matrix

**Author(s)**

Rajarshi Guha <rajarshi.guha@gmail.com>

**See Also**

[get.adjacency.matrix](#)

**Examples**

```
m <- parse.smiles("CC=C")[[1]]
get.connection.matrix(m)
```

---

get.depictor                      *get.depictor*

---

**Description**

return an RcdkDepictor.

**Usage**

```
get.depictor(
  width = 200,
  height = 200,
  zoom = 1.3,
  style = "cow",
  annotate = "off",
  abbr = "on",
  suppressh = TRUE,
  showTitle = FALSE,
  smaLimit = 100,
  sma = NULL,
  fillToFit = FALSE
)
```

**Arguments**

width	Default. 200
height	Default. 200
zoom	Default. 1.3
style	Default. cow
annotate	Default. off
abbr	Default. on
suppressh	Default. TRUE
showTitle	Default. FALSE

smaLimit	Default. 100
sma	Default. NULL
fillToFit	Default. FALSE

---

get.desc.categories     *List available descriptor categories*

---

### Description

List available descriptor categories

### Usage

```
get.desc.categories()
```

### Value

A character vector listing available descriptor categories. This can be used in [get.desc.names](#)

### Author(s)

Rajarshi Guha (<rajarshi.guha@gmail.com>)

### See Also

[get.desc.names](#)

---

get.desc.names     *Get descriptor class names*

---

### Description

Get descriptor class names

### Usage

```
get.desc.names(type = "all")
```

### Arguments

type	A string indicating which class of descriptors to return. Specifying "all" will return class names for all molecular descriptors. Options include * topological * geometrical * hybrid * constitutional * protein * electronic
------	--

**Author(s)**

Rajarshi Guha (<rajarshi.guha@gmail.com>)

**See Also**

[get.atomic.desc.names](#)

---

get.element.types      *Obtain the type of stereo element support for atom.*

---

**Description**

Supported elements types are

**Bicoordinate** an central atom involved in a cumulated system (not yet supported)

**Tricoordinate** an atom at one end of a geometric (double-bond) stereo bond or cumulated system

**Tetracoordinate** a tetrahedral atom (could also be square planar in future)

**None** the atom is not a (supported) stereo element type

**Usage**

```
get.element.types(mol)
```

**Arguments**

mol                    A jObjRef representing an IAtomContainer

**Value**

A factor of length equal in length to the number of atoms, indicating the element type

**Author(s)**

Rajarshi Guha <rajarshi.guha@gmail.com>

**See Also**

[get.stereocenters](#), [get.stereo.types](#)

---

<code>get.exact.mass</code>	<i>get.exact.mass</i>
-----------------------------	-----------------------

---

**Description**

`get.exact.mass`

**Usage**

`get.exact.mass(mol)`

**Arguments**

`mol` The molecule to query. Should be a 'jobRef' representing an 'IAtomContainer'

---

<code>get.exhaustive.fragments</code>	<i>Generate Bemis-Murcko Fragments</i>
---------------------------------------	--

---

**Description**

Fragment the input molecule using the Bemis-Murcko scheme

**Usage**

`get.exhaustive.fragments(mols, min.frag.size = 6, as.smiles = TRUE)`

**Arguments**

`mols` A list of 'jobRef' objects of Java class 'IAtomContainer'  
`min.frag.size` The smallest fragment to consider (in terms of heavy atoms)  
`as.smiles` If 'TRUE' return the fragments as SMILES strings. If not, then fragments are returned as 'jobRef' objects

**Details**

A variety of methods for fragmenting molecules are available ranging from exhaustive, rings to more specific methods such as Murcko frameworks. Fragmenting a collection of molecules can be a useful for a variety of analyses. In addition fragment based analysis can be a useful and faster alternative to traditional clustering of the whole collection, especially when it is large.

Note that exhaustive fragmentation of large molecules (with many single bonds) can become time consuming.



**Value**

returns a list of length equal to the number of input molecules. Each element is a character vector of SMILES strings or a list of 'jobRef' objects.

**Author(s)**

Rajarshi Guha (<rajarshi.guha@gmail.com>)

**See Also**

[get.murcko.fragments()]

**Examples**

```
mol <- parse.smiles('c1ccc(cc1)CN(c2cc(ccc2[N+](=O)[O-])c3c(nc(nc3CC)N)N)C')[[1]]
mf1 <- get.murcko.fragments(mol, as.smiles=TRUE, single.framework=TRUE)
mf1 <- get.murcko.fragments(mol, as.smiles=TRUE, single.framework=FALSE)
```

---

get.fingerprint	<i>Generate molecular fingerprints</i>
-----------------	--

---

**Description**

'get.fingerprint' returns a 'fingerprint' object representing molecular fingerprint of the input molecule.

**Usage**

```
get.fingerprint(  
  molecule,  
  type = "standard",  
  fp.mode = "bit",  
  depth = 6,  
  size = 1024,  
  substructure.pattern = character(),  
  circular.type = "ECFP6",  
  verbose = FALSE  
)
```

**Arguments**

molecule	A jobRef object to an IAtomContainer
type	The type of fingerprint. Possible values are: <ul style="list-style-type: none"><li>• standard - Considers paths of a given length. The default is but can be changed. These are hashed fingerprints, with a default length of 1024</li><li>• extended - Similar to the standard type, but takes rings and atomic properties into account</li><li>• graph - Similar to the standard type by simply considers connectivity</li></ul>

- hybridization - Similar to the standard type, but only consider hybridization state
- maccs - The popular 166 bit MACCS keys described by MDL
- estate - 79 bit fingerprints corresponding to the E-State atom types described by Hall and Kier
- pubchem - 881 bit fingerprints defined by PubChem
- kr - 4860 bit fingerprint defined by Klekota and Roth
- shortestpath - A fingerprint based on the shortest paths between pairs of atoms and takes into account ring systems, charges etc.
- signature - A feature,count type of fingerprint, similar in nature to circular fingerprints, but based on the signature descriptor
- circular - An implementation of the ECFP6 (default) fingerprint. Other circular types can be chosen by modifying the circular.type parameter.
- substructure - Fingerprint based on list of SMARTS pattern. By default a set of functional groups is tested.

fp.mode	The style of fingerprint. Specifying "bit" will return a binary fingerprint, "raw" returns the the original representation (usually sequence of integers) and "count" returns the fingerprint as a sequence of counts.
depth	The search depth. This argument is ignored for the 'pubchem', 'maccs', 'kr' and 'estate' fingerprints
size	The final length of the fingerprint. This argument is ignored for the 'pubchem', 'maccs', 'kr', 'signature', 'circular' and 'estate' fingerprints
substructure.pattern	List of characters containing the SMARTS pattern to match. If the an empty list is provided (default) than the functional groups substructures (default in CDK) are used.
circular.type	Name of the circular fingerprint type that should be computed given as string. Possible values are: 'ECFP0', 'ECFP2', 'ECFP4', 'ECFP6' (default), 'FCFP0', 'FCFP2', 'FCFP4' and 'FCFP6'.
verbose	Verbose output if TRUE

**Value**

an S4 object of class [fingerprint-class](#) or [featvec-class](#), which can be manipulated with the fingerprint package.

**Author(s)**

Rajarshi Guha (<rajarshi.guha@gmail.com>)

**Examples**

```
## get some molecules
sp <- get.smiles.parser()
smiles <- c('CCC', 'CCN', 'CCN(C)(C)', 'c1ccccc1Cc1ccccc1', 'C1CCC1CC(CN(C)(C))CC(=O)CC')
mols <- parse.smiles(smiles)
```

```
## get a single fingerprint using the standard
## (hashed, path based) fingerprinter
fp <- get.fingerprint(mols[[1]])

## get MACCS keys for all the molecules
fps <- lapply(mols, get.fingerprint, type='maccs')

## get Signature fingerprint
## feature, count fingerprinter
fps <- lapply(mols, get.fingerprint, type='signature', fp.mode='raw')
## get Substructure fingerprint for functional group fragments
fps <- lapply(mols, get.fingerprint, type='substructure')

## get Substructure count fingerprint for user defined fragments
mol1 <- parse.smiles("c1ccccc1CCC")[[1]]
smarts <- c("c1ccccc1", "[CX4H3][#6]", "[CX2]#[CX2]")
fps <- get.fingerprint(mol1, type='substructure', fp.mode='count',
  substructure.pattern=smarts)

## get ECFP0 count fingerprints
mol2 <- parse.smiles("C1=CC=CC(=C1)CCCC2=CC=CC=C2")[[1]]
fps <- get.fingerprint(mol2, type='circular', fp.mode='count', circular.type='ECFP0')
```

---

get.formal.charge      *get.formal.charge*

---

## Description

Get the formal charge on the atom.

## Usage

```
get.formal.charge(atom)
```

## Arguments

atom	The atom to query
------	-------------------

## Details

By default the formal charge will be 0 (i.e., NULL is never returned).

## Value

An integer representing the formal charge

## Author(s)

Rajarshi Guha (<rajarshi.guha@gmail.com>)

**See Also**[get.charge](#)

---

<code>get.formula</code>	<i>get.formula</i>
--------------------------	--------------------

---

**Description**

obtain molecular formula from formula string

**Usage**

```
get.formula(mf, charge = 0)
```

**Arguments**

<code>mf</code>	Required. Molecular formula
<code>charge</code>	Optional. Default 0

---

<code>get.hydrogen.count</code>	<i>get.hydrogen.count</i>
---------------------------------	---------------------------

---

**Description**

Get the implicit hydrogen count for the atom.

**Usage**

```
get.hydrogen.count(atom)
```

**Arguments**

<code>atom</code>	The atom to query
-------------------	-------------------

**Details**

This method returns the number of implicit H's on the atom. Depending on where the molecule was read from this may be NULL or an integer greater than or equal to 0

**Value**

An integer representing the hydrogen count

**Author(s)**

Rajarshi Guha (<rajarshi.guha@gmail.com>)

---

```
get.isotope.pattern.generator
```

*Construct an isotope pattern generator.*

---

**Description**

Constructs an instance of the CDK `IsotopePatternGenerator`, with an optional minimum abundance specified. This object can be used to generate all combinatorial chemical isotopes given a structure.

**Usage**

```
get.isotope.pattern.generator(minAbundance = NULL)
```

**Arguments**

`minAbundance`     The minimum abundance

**Value**

A `jobjRef` corresponding to an instance of `IsotopePatternGenerator`

**Author(s)**

Miguel Rojas Cherto

**References**

<http://cdk.github.io/cdk/1.5/docs/api/org/openscience/cdk/formula/IsotopePatternGenerator.html>

---

```
get.isotope.pattern.similarity
```

*Construct an isotope pattern similarity calculator.*

---

**Description**

A method that returns an instance of the CDK `IsotopePatternSimilarity` class which can be used to compute similarity scores between pairs of isotope abundance patterns.

**Usage**

```
get.isotope.pattern.similarity(tol = NULL)
```

**Arguments**

`tol`                 The tolerance

**Value**

A jobRef corresponding to an instance of IsotopePatternSimilarity

**Author(s)**

Miguel Rojas Cherto

**References**

<http://cdk.github.io/cdk/1.5/docs/api/org/openscience/cdk/formula/IsotopePatternSimilarity.html>

**See Also**

[compare.isotope.pattern](#)

---

`get.isotopes.pattern`    *get.isotopes.pattern*

---

**Description**

Generate the isotope pattern given a formula class

**Usage**

```
get.isotopes.pattern(formula, minAbund = 0.1)
```

**Arguments**

formula	Required. A CDK molecule formula
minAbund	Optional. Default 0.1

---

`get.largest.component`    *Gets the largest component in a disconnected molecular graph.*

---

**Description**

A molecule may be represented as a **disconnected graph**, such as when read in as a salt form. This method will return the largest connected component or if there is only a single component (i.e., the molecular graph is **complete** or fully connected), that component is returned.

**Usage**

```
get.largest.component(mol)
```

**Arguments**

mol                    The molecule to query. Should be a 'jobRef' representing an 'IAtomContainer'

**Value**

The largest component as an 'IAtomContainer' object or else the input molecule itself

**Author(s)**

Rajarshi Guha (<rajarshi.guha@gmail.com>)

**See Also**

[is.connected](#)

**Examples**

```
m <- parse.smiles("CC.CCCCC.CCCC")[[1]]
largest <- get.largest.component(m)
length(get.atoms(largest)) == 6
```

---

get.mcs

*get.mcs*

---

**Description**

get.mcs

**Usage**

```
get.mcs(mol1, mol2, as.molecule = TRUE)
```

**Arguments**

mol1                    Required. First molecule to compare. Should be a 'jobRef' representing an 'IAtomContainer'

mol2                    Required. Second molecule to compare. Should be a 'jobRef' representing an 'IAtomContainer'

as.molecule            Optional. Default TRUE.

---

```
get.mol2formula      get.mol2formula
```

---

**Description**

```
get.mol2formula
```

**Usage**

```
get.mol2formula(molecule, charge = 0)
```

**Arguments**

molecule	The molecule to query. Should be a 'jobRef' representing an 'IAtomContainer'
charge	Optional. Default 0

---

```
get.murcko.fragments  Generate Bemis-Murcko Fragments
```

---

**Description**

Fragment the input molecule using the Bemis-Murcko scheme

**Usage**

```
get.murcko.fragments(  
  mols,  
  min.frag.size = 6,  
  as.smiles = TRUE,  
  single.framework = FALSE  
)
```

**Arguments**

mols	A list of 'jobRef' objects of Java class 'IAtomContainer'
min.frag.size	The smallest fragment to consider (in terms of heavy atoms)
as.smiles	If 'TRUE' return the fragments as SMILES strings. If not, then fragments are returned as 'jobRef' objects
single.framework	If 'TRUE', then a single framework (i.e., the framework consisting of the union of all ring systems and linkers) is returned for each molecule. Otherwise, all combinations of ring systems and linkers are returned



## Details

A variety of methods for fragmenting molecules are available ranging from exhaustive, rings to more specific methods such as Murcko frameworks. Fragmenting a collection of molecules can be a useful for a variety of analyses. In addition fragment based analysis can be a useful and faster alternative to traditional clustering of the whole collection, especially when it is large.

Note that exhaustive fragmentation of large molecules (with many single bonds) can become time consuming.

## Value

Returns a list with each element being a list with two elements: 'rings' and 'frameworks'. Each of these elements is either a character vector of SMILES strings or a list of 'IAtomContainer' objects.

## Author(s)

Rajarshi Guha (<rajarshi.guha@gmail.com>)

## See Also

[get.exhaustive.fragments()]

## Examples

```
mol <- parse.smiles('c1ccc(cc1)CN(c2cc(ccc2[N+](=O)[O-])c3c(nc(nc3CC)N)N)C')[[1]]
mf1 <- get.murcko.fragments(mol, as.smiles=TRUE, single.framework=TRUE)
mf1 <- get.murcko.fragments(mol, as.smiles=TRUE, single.framework=FALSE)
```

---

get.natural.mass      *get.natural.mass*

---

## Description

get.natural.mass

## Usage

```
get.natural.mass(mol)
```

## Arguments

mol                    The molecule to query. Should be a 'jobRef' representing an 'IAtomContainer'

---

`get.point2d`*get.point2d*

---

### Description

Get the 2D coordinates of the atom.

### Usage

```
get.point2d(atom)
```

### Arguments

atom            The atom to query

### Details

In case, coordinates are unavailable (e.g., molecule was read in from a SMILES file) or have not been generated yet, 'NA's are returned for the X & Y coordinates.

### Value

A 2-element numeric vector representing the X & Y coordinates.

### Author(s)

Rajarshi Guha (<rajarshi.guha@gmail.com>)

### See Also

[get.point3d](#)

### Examples

```
## Not run:
atoms <- get.atoms(mol)
coords <- do.call('rbind', lapply(apply, get.point2d))

## End(Not run)
```

---

`get.point3d`*get.point3d*

---

**Description**

Get the 3D coordinates of the atom.

**Usage**

```
get.point3d(atom)
```

**Arguments**

atom            The atom to query

**Details**

In case, coordinates are unavailable (e.g., molecule was read in from a SMILES file) or have not been generated yet, 'NA's are returned for the X, Y and Z coordinates.

**Value**

A 3-element numeric vector representing the X, Y and Z coordinates.

**Author(s)**

Rajarshi Guha (<rajarshi.guha@gmail.com>)

**See Also**

[get.point2d](#)

**Examples**

```
## Not run:
atoms <- get.atoms(mol)
coords <- do.call('rbind', lapply(apply, get.point3d))

## End(Not run)
```

---

get.properties	<i>Get all properties associated with a molecule.</i>
----------------	---

---

### Description

In this context a property is a value associated with a key and stored with the molecule. This method returns a list of all the properties of a molecule. The names of the list are set to the property names.

### Usage

```
get.properties(molecule)
```

### Arguments

molecule	The molecule to query. Should be a 'jobRef' representing an 'IAtomContainer'
----------	--

### Value

A named 'list' with the property values. Element names are the keys for each property. If no properties have been defined, an empty list.

### Author(s)

Rajarshi Guha (<rajarshi.guha@gmail.com>)

### See Also

[set.property](#), [get.property](#), [remove.property](#)

### Examples

```
mol <- parse.smiles("CC1CC(C=O)CCC1")[[1]]
set.property(mol, 'prop1', 23.45)
set.property(mol, 'prop2', 'inactive')
get.properties(mol)
```

---

get.property	<i>Get a property value of the molecule.</i>
--------------	--

---

### Description

This function retrieves the value of a keyed property that has previously been set on the molecule. Properties enable us to associate arbitrary pieces of data with a molecule. Such data can be text, numeric or a Java object (represented as a 'jobRef').

**Usage**

```
get.property(molecule, key)
```

**Arguments**

molecule	The molecule to query. Should be a 'jobRef' representing an 'IAtomContainer'
key	The property key as a character string

**Value**

The value of the property. If there is no property with the specified key, 'NA' is returned

**Author(s)**

Rajarshi Guha (<rajarshi.guha@gmail.com>)

**See Also**

[set.property](#), [get.properties](#)

**Examples**

```
mol <- parse.smiles("CC1CC(C=O)CCC1")[[1]]
set.property(mol, 'prop1', 23.45)
set.property(mol, 'prop2', 'inactive')
get.property(mol, 'prop1')
```

---

get.smiles

*Generate a SMILES representation of a molecule.*

---

**Description**

The function will generate a SMILES representation of an 'IAtomContainer' object. The default parameters of the CDK SMILES generator are used. This can mean that for large ring systems the method may fail. See CDK [Javadocs](#) for more information

**Usage**

```
get.smiles(molecule, flavor = smiles.flavors(c("Generic")), smigen = NULL)
```

**Arguments**

molecule	The molecule to query. Should be a 'jobRef' representing an 'IAtomContainer'
flavor	The type of SMILES to generate. See <a href="#">smiles.flavors</a> . Default is 'Generic' SMILES
smigen	A pre-existing SMILES generator object. By default, a new one is created from the specified flavor

**Value**

A character string containing the generated SMILES

**Author(s)**

Rajarshi Guha (<rajarshi.guha@gmail.com>)

**References**

[SmilesGenerator](#)

**See Also**

[parse.smiles](#), [smiles.flavors](#)

**Examples**

```
m <- parse.smiles('C1C=CCC1N(C)c1cccc1')[[1]]
get.smiles(m)
get.smiles(m, smiles.flavors(c('Generic', 'UseAromaticSymbols')))
```

---

get.smiles.parser      *Get a SMILES parser object.*

---

**Description**

This function returns a reference to a SMILES parser object. If you are parsing multiple SMILES strings using multiple calls to [parse.smiles](#), it is preferable to create your own parser and supply it to [parse.smiles](#) rather than forcing that function to instantiate a new parser for each call

**Usage**

```
get.smiles.parser()
```

**Value**

A 'jobRef' object corresponding to the CDK [SmilesParser](#) class

**Author(s)**

Rajarshi Guha (<rajarshi.guha@gmail.com>)

**See Also**

[get.smiles](#), [parse.smiles](#)

---

get.stereo.types      *Obtain the stereocenter type for atom.*

---

### Description

Supported stereo center types are

**True** the atom has constitutionally different neighbors

**Para** the atom resembles a stereo centre but has constitutionally equivalent neighbors (e.g. inositol, decalin). The stereocenter depends on the configuration of one or more stereocenters.

**Potential** the atom can supported stereo chemistry but has not be shown ot be a true or para center

**Non** the atom is not a stereocenter (e.g. methane)

### Usage

```
get.stereo.types(mol)
```

### Arguments

mol                    A jObjRef representing an IAtomContainer

### Value

A factor of length equal in length to the number of atoms indicating the stereocenter type.

### Author(s)

Rajarshi Guha <rajarshi.guha@gmail.com>

### See Also

[get.stereocenters](#), [get.element.types](#)

---

get.stereocenters      *Identify which atoms are stereocenters.*

---

### Description

This method identifies stereocenters based on connectivity.

### Usage

```
get.stereocenters(mol)
```

**Arguments**

mol                    A jObjRef representing an IAtomContainer

**Value**

A logical vector of length equal in length to the number of atoms. The i'th element is TRUE if the i'th element is identified as a stereocenter

**Author(s)**

Rajarshi Guha <rajarshi.guha@gmail.com>

**See Also**

[get.element.types](#), [get.stereo.types](#)

---

`get.symbol`

*get.symbol*

---

**Description**

Get the atomic symbol of the atom.

**Usage**

```
get.symbol(atom)
```

**Arguments**

atom                    The atom to query

**Value**

A character representing the atomic symbol

**Author(s)**

Rajarshi Guha (<rajarshi.guha@gmail.com>)



---

get.title	<i>Get the title of the molecule.</i>
-----------	---------------------------------------

---

**Description**

Some molecules may not have a title (such as when parsing in a SMILES with not title).

**Usage**

```
get.title(mol)
```

**Arguments**

mol	The molecule to query. Should be a 'jobRef' representing an 'IAtomContainer'
-----	--

**Value**

A character string with the title, 'NA' is no title is specified

**Author(s)**

Rajarshi Guha (<rajarshi.guha@gmail.com>)

**See Also**

[set.title](#)

---

get.total.charge	<i>get.total.charge</i>
------------------	-------------------------

---

**Description**

```
get.total.charge
```

**Usage**

```
get.total.charge(mol)
```

**Arguments**

mol	The molecule to query. Should be a 'jobRef' representing an 'IAtomContainer'
-----	--

---

`get.total.formal.charge`

*get.total.formal.charge*

---

### **Description**

`get.total.formal.charge`

### **Usage**

`get.total.formal.charge(mol)`

### **Arguments**

`mol`                    The molecule to query. Should be a 'jobRef' representing an 'IAtomContainer'

---

`get.total.hydrogen.count`

*Get total number of implicit hydrogens in the molecule.*

---

### **Description**

Counts the number of hydrogens on the provided molecule. As this method will sum all implicit hydrogens on each atom it is important to ensure the molecule has already been configured (and thus each atom has an implicit hydrogen count).

### **Usage**

`get.total.hydrogen.count(mol)`

### **Arguments**

`mol`                    The molecule to query. Should be a 'jobRef' representing an 'IAtomContainer'

### **Value**

An integer representing the total number of implicit hydrogens

### **Author(s)**

Rajarshi Guha (<rajarshi.guha@gmail.com>)

### **See Also**

[get.hydrogen.count](#), [remove.hydrogens](#)

---

get.tpsa	<i>Compute TPSA for a molecule</i>
----------	------------------------------------

---

**Description**

Compute TPSA for a molecule

**Usage**

```
get.tpsa(molecule)
```

**Arguments**

molecule      A molecule object

**Value**

A double value representing the TPSA value

**Author(s)**

Rajarshi Guha (<rajarshi.guha@gmail.com>)

---

get.volume	<i>Compute volume of a molecule</i>
------------	-------------------------------------

---

**Description**

This method does not require 3D coordinates. As a result its an approximation

**Usage**

```
get.volume(molecule)
```

**Arguments**

molecule      A molecule object

**Value**

A double value representing the volume

**Author(s)**

Rajarshi Guha (<rajarshi.guha@gmail.com>)

---

get.xlogp	<i>Compute XLogP for a molecule</i>
-----------	-------------------------------------

---

**Description**

Compute XLogP for a molecule

**Usage**

```
get.xlogp(molecule)
```

**Arguments**

molecule      A molecule object

**Value**

A double value representing the XLogP value

**Author(s)**

Rajarshi Guha (<rajarshi.guha@gmail.com>)

---

iload.molecules	<i>Load molecules using an iterator.</i>
-----------------	--

---

**Description**

The CDK can read a variety of molecular structure formats. Some file formats support multiple molecules in a single file. If read using [load.molecules](#), all are read into memory. For very large structure files, this can lead to out of memory errors. Instead it is recommended to use the iterating version of the loader so that only a single molecule is read at a time.

**Usage**

```
iload.molecules(  
  molfile,  
  type = "smi",  
  aromaticity = TRUE,  
  typing = TRUE,  
  isotopes = TRUE,  
  skip = TRUE  
)
```

**Arguments**

molfile	A string containing the filename to load. Must be a local file
type	Indicates whether the input file is SMILES or SDF. Valid values are "smi" or "sdf"
aromaticity	If 'TRUE' then aromaticity detection is performed on all loaded molecules. If this fails for a given molecule, then the molecule is set to 'NA' in the return list
typing	If 'TRUE' then atom typing is performed on all loaded molecules. The assigned types will be CDK internal types. If this fails for a given molecule, then the molecule is set to 'NA' in the return list
isotopes	If 'TRUE' then atoms are configured with isotopic masses
skip	If 'TRUE', then the reader will continue reading even when faced with an invalid molecule. If 'FALSE', the reader will stop at the first invalid molecule

**Details**

Note that the iterating loader only supports SDF and SMILES file formats.

**Author(s)**

Rajarshi Guha (<rajarshi.guha@gmail.com>)

**See Also**

[write.molecules](#), [load.molecules](#), [parse.smiles](#)

**Examples**

```
## Not run:
moliter <- iload.molecules("big.sdf", type="sdf")
while(hasNext(moliter)) {
  mol <- nextElem(moliter)
  print(get.property(mol, "cdk:Title"))
}

## End(Not run)
```

---

is.aliphatic

*is.aliphatic*

---

**Description**

Tests whether an atom is aliphatic.

**Usage**

```
is.aliphatic(atom)
```

**Arguments**

atom                    The atom to test

**Details**

This assumes that the molecule containing the atom has been appropriately configured.

**Value**

'TRUE' is the atom is aliphatic, 'FALSE' otherwise

**Author(s)**

Rajarshi Guha (<rajarshi.guha@gmail.com>)

**See Also**

[is.in.ring](#), [is.aromatic](#)

---

is.aromatic

*is.aromatic*

---

**Description**

Tests whether an atom is aromatic.

**Usage**

```
is.aromatic(atom)
```

**Arguments**

atom                    The atom to test

**Details**

This assumes that the molecule containing the atom has been appropriately configured.

**Value**

'TRUE' is the atom is aromatic, 'FALSE' otherwise

**Author(s)**

Rajarshi Guha (<rajarshi.guha@gmail.com>)

**See Also**

[is.aliphatic](#), [is.in.ring](#), [do.aromaticity](#)

---

is.connected	<i>Tests whether the molecule is fully connected.</i>
--------------	---

---

### Description

A single molecule will be represented as a **complete** graph. In some cases, such as for molecules in salt form, or after certain operations such as bond splits, the molecular graph may contain **disconnected components**. This method can be used to test whether the molecule is complete (i.e. fully connected).

### Usage

```
is.connected(mol)
```

### Arguments

mol                    The molecule to query. Should be a 'jobRef' representing an 'IAtomContainer'

### Value

'TRUE' if molecule is complete, 'FALSE' otherwise

### Author(s)

Rajarshi Guha (<rajarshi.guha@gmail.com>)

### See Also

[get.largest.component](#)

### Examples

```
m <- parse.smiles("CC.CCCCC.CCCC")[[1]]
is.connected(m)
```

---

is.in.ring	<i>is.in.ring</i>
------------	-------------------

---

### Description

Tests whether an atom is in a ring.

### Usage

```
is.in.ring(atom)
```

**Arguments**

atom            The atom to test

**Details**

This assumes that the molecule containing the atom has been appropriately configured.

**Value**

'TRUE' is the atom is in a ring, 'FALSE' otherwise

**Author(s)**

Rajarshi Guha (<rajarshi.guha@gmail.com>)

**See Also**

[is.aliphatic](#), [is.aromatic](#)

---

*is.neutral*

*Tests whether the molecule is neutral.*

---

**Description**

The test checks whether all atoms in the molecule have a formal charge of 0.

**Usage**

```
is.neutral(mol)
```

**Arguments**

mol            The molecule to query. Should be a 'jobRef' representing an 'IAtomContainer'

**Value**

'TRUE' if molecule is neutral, 'FALSE' otherwise

**Author(s)**

Rajarshi Guha (<rajarshi.guha@gmail.com>)



---

isvalid.formula	<i>isvalid.formula</i>
-----------------	------------------------

---

**Description**

Validate a cdkFormula.

**Usage**

```
isvalid.formula(formula, rule = c("nitrogen", "RDBE"))
```

**Arguments**

formula	Required. A CDK Formula
rule	Optional. Default rule=c("nitrogen", "RDBE")

---

load.molecules	<i>Load molecular structures from disk or URL</i>
----------------	---

---

**Description**

The CDK can read a variety of molecular structure formats. This function encapsulates the calls to the CDK API to load a structure given its filename or a URL to a structure file.

**Usage**

```
load.molecules(
  molfiles = NA,
  aromaticity = TRUE,
  typing = TRUE,
  isotopes = TRUE,
  verbose = FALSE
)
```

**Arguments**

molfiles	A 'character' vector of filenames. Note that the full path to the files should be provided. URL's can also be used as paths. In such a case, the URL should start with "http://"
aromaticity	If 'TRUE' then aromaticity detection is performed on all loaded molecules. If this fails for a given molecule, then the molecule is set to 'NA' in the return list
typing	If 'TRUE' then atom typing is performed on all loaded molecules. The assigned types will be CDK internal types. If this fails for a given molecule, then the molecule is set to 'NA' in the return list
isotopes	If 'TRUE' then atoms are configured with isotopic masses
verbose	If 'TRUE', output (such as file download progress) will be bountiful

### Details

Note that this method will load all molecules into memory. For files containing tens of thousands of molecules this may lead to out of memory errors. In such situations consider using the iterating file readers.

Note that if molecules are read in from formats that do not have rules for handling implicit hydrogens (such as MDL MOL), the molecule will not have implicit or explicit hydrogens. To add explicit hydrogens, make sure that the molecule has been typed (this is 'TRUE' by default for this function) and then call `convert.implicit.to.explicit`. On the other hand for a format such as SMILES, implicit or explicit hydrogens will be present.

### Value

A 'list' of CDK 'IAtomContainer' objects, represented as 'jobRef' objects in R, which can be used in other 'rcdk' functions

### Author(s)

Rajarshi Guha (<rajarshi.guha@gmail.com>)

### See Also

[write.molecules](#), [parse.smiles](#), [iload.molecules](#)

### Examples

```
## Not run:
sdffile <- system.file("molfiles/dhfr00008.sdf", package="rcdk")
mols <- load.molecules(c('mol1.sdf', 'mol2.smi', sdffile))

## End(Not run)
```

---

matches

*matches*

---

### Description

matches

### Usage

```
matches(query, target, return.matches = FALSE)
```

### Arguments

query	Required. A SMARTSQuery
target	Required. The molecule to query. Should be a 'jobRef' representing an 'IAtom-Container'
return.matches	Optional. Default FALSE

---

Molecule

*Operations on molecules*

---

## Description

Various functions to perform operations on molecules.

`get.exact.mass` returns the exact mass of a molecule `get.natural.mass` returns the natural exact mass of a molecule `convert.implicit.to.explicit` converts implicit hydrogens to explicit hydrogens. This function does not return any value but rather modifies the molecule object passed to it `is.neutral` returns TRUE if all atoms in the molecule have a formal charge of 0, otherwise FALSE

## Details

In some cases, a molecule may not have any hydrogens (such as when read in from an MDL MOLfile that did not have hydrogens). In such cases, `convert.implicit.to.explicit` will add implicit hydrogens and then convert them to explicit ones. In addition, for such cases, make sure that the molecule has been typed beforehand.

## Usage

```
get.exact.mass(mol) get.natural.mass(mol) convert.implicit.to.explicit(mol) is.neutral(mol)
```

## Arguments

mol A jobjRef representing an IAtomContainer or IMolecule object

## Value

`get.exact.mass` returns a numeric `get.natural.mass` returns a numeric `convert.implicit.to.explicit` has no return value `is.neutral` returns a boolean.

## Author(s)

Rajarshi Guha (<rajarshi.guha@gmail.com>)

## See Also

`get.atoms`, `set.atom.types`

---

`parse.smiles`*Parse SMILES strings into molecule objects.*

---

### Description

This function parses a vector of SMILES strings to generate a list of 'IAtomContainer' objects. Note that the resultant molecule will not have any 2D or 3D coordinates. Note that the molecules obtained from this method will not have any aromaticity perception (unless aromatic symbols are encountered, in which case the relevant atoms are automatically set to aromatic), atom typing or isotopic configuration done on them. This is in contrast to the [load.molecules](#) method. Thus, you should perform these steps manually on the molecules.

### Usage

```
parse.smiles(smiles, kekulise = TRUE, omit.nulls = FALSE, smiles.parser = NULL)
```

### Arguments

<code>smiles</code>	A single SMILES string or a vector of SMILES strings
<code>kekulise</code>	If set to 'FALSE' disables electron checking and allows for parsing of incorrect SMILES. If a SMILES does not parse by default, try setting this to 'FALSE' - though the resultant molecule may not have consistent bonding. As an example, 'c4ccc2c(cc1=Nc3ncccc3(Cn12))c4' will not be parsed by default because it is missing a nitrogen. With this argument set to 'FALSE' it will parse successfully, but this is a hack to handle an incorrect SMILES
<code>omit.nulls</code>	If set to 'TRUE', omits SMILES which were parsed as 'NULL'
<code>smiles.parser</code>	A SMILES parser object obtained from <a href="#">get.smiles.parser</a>

### Value

A 'list' of 'jobRef's to their corresponding CDK 'IAtomContainer' objects. If a SMILES string could not be parsed and 'omit.nulls=TRUE' it is omitted from the output list.

### Author(s)

Rajarshi Guha (<rajarshi.guha@gmail.com>)

### See Also

[get.smiles](#), [parse.smiles](#)

---

rcdk-deprecated      *Deprecated functions in the rcdk package.*

---

### Description

These functions are provided for compatibility with older version of the phyloseq package. They may eventually be completely removed.

### Usage

```
deprecated_rcdk_function(x, value, ...)
```

### Arguments

x	For assignment operators, the object that will undergo a replacement (object inside parenthesis).
value	For assignment operators, the value to replace with (the right side of the assignment).
...	For functions other than assignment operators, parameters to be passed to the modern version of the function (see table).

### Details

`do.typing` now a synonym for [set.atom.types](#)

---

`remove.hydrogens`      *Remove explicit hydrogens.*

---

### Description

Create an copy of the original structure with explicit hydrogens removed. Stereochemistry is updated but up and down bonds in a depiction may need to be recalculated. This can also be useful for descriptor calculations.

### Usage

```
remove.hydrogens(mol)
```

### Arguments

mol	The molecule to query. Should be a 'jobRef' representing an 'IAtomContainer'
-----	--

**Value**

A copy of the original molecule, with explicit hydrogens removed

**Author(s)**

Rajarshi Guha (<rajarshi.guha@gmail.com>)

**See Also**

[get.hydrogen.count](#), [get.total.hydrogen.count](#)

---

remove.property

*Remove a property associated with a molecule.*

---

**Description**

In this context a property is a value associated with a key and stored with the molecule. This method will remove the property defined by the key. If there is such key, a warning is raised.

**Usage**

```
remove.property(molecule, key)
```

**Arguments**

molecule	The molecule to query. Should be a 'jobRef' representing an 'IAtomContainer'
key	The property key as a character string

**Author(s)**

Rajarshi Guha (<rajarshi.guha@gmail.com>)

**See Also**

[set.property](#), [get.property](#), [get.properties](#)

**Examples**

```
mol <- parse.smiles("CC1CC(C=O)CCC1")[[1]]
set.property(mol, 'prop1', 23.45)
set.property(mol, 'prop2', 'inactive')
get.properties(mol)
remove.property(mol, 'prop2')
get.properties(mol)
```

---

set.atom.types      *set.atom.types*

---

**Description**

Set the CDK atom types for all atoms in the molecule.

**Usage**

```
set.atom.types(mol)
```

**Arguments**

mol                      The molecule whose atoms should be typed

**Details**

Calling this method will overwrite any pre-existing type information. Currently there is no way to choose other atom typing schemes

**Value**

Nothing is returned, the molecule is modified in place

**Author(s)**

Rajarshi Guha (<rajarshi.guha@gmail.com>)

---

set.charge.formula      *set.charge.formula*

---

**Description**

Set the charge to a cdkFormula function.

**Usage**

```
set.charge.formula(formula, charge = -1)
```

**Arguments**

formula                  Required. Molecular formula  
charge                    Optional. Default -1

---

set.property	<i>Set a property value of the molecule.</i>
--------------	--

---

### Description

This function sets the value of a keyed property on the molecule. Properties enable us to associate arbitrary pieces of data with a molecule. Such data can be text, numeric or a Java object (represented as a 'jobRef').

### Usage

```
set.property(molecule, key, value)
```

### Arguments

molecule	The molecule to query. Should be a 'jobRef' representing an 'IAtomContainer'
key	The property key as a character string
value	The value of the property. This can be a character, numeric or 'jobRef' R object

### Author(s)

Rajarshi Guha (<rajarshi.guha@gmail.com>)

### See Also

[get.property](#), [get.properties](#), [remove.property](#)

### Examples

```
mol <- parse.smiles("CC1CC(C=O)CCC1")[[1]]
set.property(mol, 'prop1', 23.45)
set.property(mol, 'prop2', 'inactive')
get.property(mol, 'prop1')
```

---

set.title	<i>Set the title of the molecule.</i>
-----------	---------------------------------------

---

### Description

Set the title of the molecule.

### Usage

```
set.title(mol, title = "")
```



**Arguments**

mol	The molecule to query. Should be a 'jobRef' representing an 'IAtomContainer'
title	The title of the molecule as a character string. This will overwrite any pre-existing title. The default value is an empty string.

**Author(s)**

Rajarshi Guha (<rajarshi.guha@gmail.com>)

**See Also**

[get.title](#)

---

smiles.flavors	<i>Generate flag for customizing SMILES generation.</i>
----------------	---

---

**Description**

The CDK supports a variety of customizations for SMILES generation including the use of lower case symbols for aromatic compounds to the use of the ChemAxon **CxSmiles** format. Each 'flavor' is represented by an integer and multiple customizations are bitwise OR'ed. This method accepts the names of one or more customizations and returns the bitwise OR of them. See [CDK documentation](#) for the list of flavors and what they mean.

**Usage**

```
smiles.flavors(flavors = c("Generic"))
```

**Arguments**

flavors	A character vector of flavors. The default is Generic (output non-canonical SMILES without stereochemistry, atomic masses). Possible values are <ul style="list-style-type: none"><li>• Absolute</li><li>• AtomAtomMap</li><li>• AtomicMass</li><li>• AtomicMassStrict</li><li>• Canonical</li><li>• Cx2dCoordinates</li><li>• Cx3dCoordinates</li><li>• CxAtomLabel</li><li>• CxAtomValue</li><li>• CxCoordinates</li><li>• CxFragmentGroup</li><li>• CxMulticenter</li><li>• CxPolymer</li></ul>
---------	--

- CxRadical
- CxSmiles
- CxSmilesWithCoords
- Default
- Generic
- InChI Labelling
- Isomeric
- Stereo
- StereoCisTrans
- StereoExTetrahedral
- StereoTetrahedral
- Unique
- UniversalSmiles
- UseAromaticSymbols

**Value**

A numeric representing the bitwise ‘OR‘ of the specified flavors

**Author(s)**

Rajarshi Guha <rajarshi.guha@gmail.com>

**References**

[CDK documentation](#)

**See Also**

[get.smiles](#)

**Examples**

```
m <- parse.smiles('C1C=CCC1N(C)c1cccc1')[[1]]
get.smiles(m)
get.smiles(m, smiles.flavors(c('Generic', 'UseAromaticSymbols')))
```

  

```
m <- parse.smiles("OS(=O)(=O)c1ccc(cc1)C(CC)CC |Sg:n:13:m:ht,Sg:n:11:n:ht|")[[1]]
get.smiles(m, flavor = smiles.flavors(c("CxSmiles")))
get.smiles(m, flavor = smiles.flavors(c("CxSmiles", "UseAromaticSymbols")))
```

---

view.image.2d	<i>view.image.2d</i>
---------------	----------------------

---

**Description**

view.image.2d

**Usage**

```
view.image.2d(molecule, depictor = NULL)
```

**Arguments**

molecule	The molecule to display Should be a 'jobRef' representing an 'IAtomContainer'
depictor	Default NULL

---

view.molecule.2d	<i>view.molecule.2d</i>
------------------	-------------------------

---

**Description**

Create a 2D depiction of a molecule. If there are more than one molecules supplied, return a grid with ncol columns,.

**Usage**

```
view.molecule.2d(  
  molecule,  
  ncol = 4,  
  width = 200,  
  height = 200,  
  depictor = NULL  
)
```

**Arguments**

molecule	The molecule to query. Should be a 'jobRef' representing an 'IAtomContainer'
ncol	Default 4
width	Default 200
height	Default 200
depictor	Default NULL

---

view.table	<i>view.table</i>
------------	-------------------

---

**Description**

Create a tabular view of a set of molecules (in 2D) and associated data columns

**Usage**

```
view.table(molecules, dat, depictor = NULL)
```

**Arguments**

molecules	A list of molecule objects ('jobRef' representing an 'AtomContainer')
dat	The data.frame associated with the molecules, one per row
depictor	Default NULL

---

write.molecules	<i>Write molecules to disk.</i>
-----------------	---------------------------------

---

**Description**

This function writes one or more molecules to an SD file on disk, which can be of the single- or multi-molecule variety. In addition, if the molecule has keyed properties, they can also be written out as SD tags.

**Usage**

```
write.molecules(mols, filename, together = TRUE, write.props = FALSE)
```

**Arguments**

mols	A 'list' of 'jobRef' objects representing 'AtomContainer' objects
filename	The name of the SD file to write. Note that if 'together' is 'FALSE' then this argument is taken as a prefix for the name of the individual files
together	If 'TRUE' then all the molecules are written to a single SD file. If 'FALSE' each molecule is written to an individual file
write.props	If 'TRUE', keyed properties are included in the SD file output

**Details**

In case individual SD files are desired the together argument can be set of FALSE. In this case, the value of filename is used as a prefix, to which a numeric identifier and the suffix of ".sdf" is appended.

*write.molecules*

61

**Author(s)**

Rajarshi Guha (<rajarshi.guha@gmail.com>)

**See Also**

[load.molecules](#), [parse.smiles](#), [iload.molecules](#)

# Index

- \* **datasets**
  - bpdata, 4
- Atoms, 3
- bpdata, 4
- cdk.version, 5
- cdkFormula-class, 5
- charge (is.neutral), 48
- compare.isotope.pattern, 6, 30
- convert.implicit.to.explicit, 7, 50, 51
- copy.image.to.clipboard, 7
- deprecated\_rcdk\_function
  - (rcdk-deprecated), 53
- do.aromaticity, 8, 46
- do.isotopes, 8
- do.typing (rcdk-deprecated), 53
- eval.atomic.desc, 8
- eval.desc, 9
- generate.2d.coordinates, 10
- generate.formula, 10
- generate.formula.iter, 11
- get.adjacency.matrix, 12, 21
- get.alogp, 13
- get.atom.count, 13
- get.atom.index, 4, 14
- get.atomic.desc.names, 9, 14, 23
- get.atomic.number, 3, 15
- get.atoms, 16, 17, 19, 51
- get.bond.order, 16
- get.bonds, 16, 17
- get.charge, 3, 18, 28
- get.chem.object.builder, 18
- get.connected.atom, 14, 19
- get.connected.atoms, 4, 16, 17, 20
- get.connection.matrix, 12, 20
- get.depictor, 21
- get.desc.categories, 22
- get.desc.names, 22, 22
- get.element.types, 23, 39, 40
- get.exact.mass, 24, 51
- get.exhaustive.fragments, 24
- get.fingerprint, 25
- get.formal.charge, 3, 18, 27
- get.formula, 5, 28
- get.hydrogen.count, 3, 7, 28, 42, 54
- get.isotope.pattern.generator, 29
- get.isotope.pattern.similarity, 6, 29
- get.isotopes.pattern, 5, 30
- get.largest.component, 30, 47
- get.mcs, 31
- get.mol2formula, 32
- get.murcko.fragments, 32
- get.natural.mass, 33, 51
- get.point2d, 3, 4, 10, 34, 35
- get.point3d, 3, 4, 34, 35
- get.properties, 36, 37, 54, 56
- get.property, 36, 36, 54, 56
- get.smiles, 37, 38, 52, 58
- get.smiles.parser, 38, 52
- get.stereo.types, 23, 39, 40
- get.stereocenters, 23, 39, 39
- get.symbol, 3, 40
- get.title, 41, 57
- get.total.charge, 41
- get.total.formal.charge, 42
- get.total.hydrogen.count, 42, 54
- get.tpsa, 43
- get.volume, 43
- get.xlogp, 44
- hydrogen (get.hydrogen.count), 28
- iload.molecules, 44, 50, 61
- is.aliphatic, 4, 45, 46, 48
- is.aromatic, 4, 46, 46, 48
- is.connected, 31, 47

`is.in.ring`, [4](#), [46](#), [47](#)  
`is.neutral`, [48](#), [51](#)  
`isvalid.formula`, [5](#), [49](#)

`load.molecules`, [44](#), [45](#), [49](#), [52](#), [61](#)

`match-SMARTS (matches)`, [50](#)  
`matches`, [50](#)  
`Molecule`, [51](#)

`parse.smiles`, [38](#), [45](#), [50](#), [52](#), [52](#), [61](#)

`rcdk-deprecated`, [53](#)  
`remove.hydrogens`, [7](#), [42](#), [53](#)  
`remove.property`, [36](#), [54](#), [56](#)

`set.atom.types`, [7](#), [51](#), [53](#), [55](#)  
`set.charge.formula`, [5](#), [55](#)  
`set.property`, [36](#), [37](#), [54](#), [56](#)  
`set.title`, [41](#), [56](#)  
`show, cdKFormula-method`  
    (`cdKFormula-class`), [5](#)  
`smiles.flavors`, [37](#), [38](#), [57](#)

`view.image.2d`, [59](#)  
`view.molecule.2d`, [10](#), [59](#)  
`view.table`, [60](#)

`write.molecules`, [45](#), [50](#), [60](#)