

# Package ‘rcdk’

February 27, 2012

**Version** 3.1.7

**Date** 2012-02-22

**Title** rcdk - Interface to the CDK Libraries

**Author** Rajarshi Guha <rajarshi.guha@gmail.com>

**Maintainer** Rajarshi Guha <rajarshi.guha@gmail.com>

**Depends** R (>= 2.11.0), rJava, rcdklibs, fingerprint, methods, png.iterators

**Suggests** xtable, RUnit

**License** LGPL

**LazyLoad** yes

**Description** This package allows the user to access functionality in the CDK, a Java framework for cheminformatics. 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.

**Repository** CRAN

**Date/Publication** 2012-02-27 18:58:44

## R topics documented:

Atoms . . . . .	2
bpdata . . . . .	4
cdk.version . . . . .	4
cdkFormula-class . . . . .	5
do.aromaticity . . . . .	5
eval.atomic.desc . . . . .	6
eval.desc . . . . .	7
generate.formula . . . . .	8
get.atomic.desc.names . . . . .	9

get.atoms . . . . .	9
get.bonds . . . . .	10
get.connected.atom . . . . .	11
get.desc.categories . . . . .	11
get.desc.names . . . . .	12
get.fingerprint . . . . .	13
get.formula . . . . .	14
get.isotopes.pattern . . . . .	15
get.mol2formula . . . . .	16
get.murcko.fragments . . . . .	16
get.properties . . . . .	17
get.property . . . . .	18
get.smiles . . . . .	19
get.smiles.parser . . . . .	20
get.total.charge . . . . .	20
get.total.hydrogen.count . . . . .	21
get.tpsa . . . . .	21
hasNext . . . . .	22
is.connected . . . . .	23
invalid.formula . . . . .	24
load.molecules . . . . .	25
matches . . . . .	26
Molecule . . . . .	27
parse.smiles . . . . .	29
remove.hydrogens . . . . .	30
remove.property . . . . .	30
set.charge.formula . . . . .	31
set.property . . . . .	32
view.molecule.2d . . . . .	33
view.table . . . . .	34
write.molecules . . . . .	35
<b>Index</b>	<b>37</b>

---

 Atoms

*Operations on atoms*


---

### 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` is 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

### 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)
is.aromatic(atom)
is.aliphatic(atom)
is.in.ring(atom)
```

### Arguments

`atom`                    A jobjRef representing an IAtom 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>)

### See Also

[get.atoms](#)

---

bpdata                      *Boiling Point Data*

---

**Description**

Structures and associated boiling points for 277 molecules, primarily alkanes and substituted alkanes.

**Usage**

bpdata

**Format**

A data.frame with two columns:

[,1]	SMILES	character	Structure in SMILES format
[,2]	BP	numeric	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                      *Get Current CDK Version*

---

**Description**

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

**Usage**

cdk.version()

**Value**

A string representing the CDK version

**Author(s)**

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

---

cdkFormula-class	<i>Class cdkFormula, a 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.

**Note**

No notes yet.

**Author(s)**

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

**References**

A parallel effort to expand the Chemistry Development Kit: <http://cdk.sourceforge.net>

**See Also**

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

---

do.aromaticity	<i>Perform Aromaticity Detection, atom typing or isotopic configuration</i>
----------------	---

---

**Description**

These methods can be used to perform aromaticity detection, atom typing or isotopic configuration on a molecule object. In general, when molecules are loaded via [load.molecules](#) these are performed by default. If molecules are obtained via [parse.smiles](#) these operations are not performed and so the user should call one or both of these methods to correctly configure a molecule.

**Usage**

```
do.aromaticity(molecule)
do.typing(molecule)
do.isotopes(molecule)
```

**Arguments**

molecule      The molecule on which the operation is to be performed. Should of class jobjRef with a jclass attribute of IAtomContainer

**Value**

No return value. If the operations fail an exception is thrown and an error message is printed

**Author(s)**

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

**See Also**

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

---

eval.atomic.desc      *Evaluate an Atomic Descriptor*

---

**Description**

The CDK implements a number of descriptors divided into three main groups - atomic, molecular and bond. This method evaluates the specified atomic descriptor(s) for a molecule

**Usage**

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

**Arguments**

molecule      A reference to a CDK IAtomContainer object  
which.desc      The fully qualified class name of the descriptor to evaluate or a vector such names  
verbose      If TRUE, progress will be written to the screen, otherwise the function performs silently

**Value**

A data.frame is returned.

**Author(s)**

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

**See Also**

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

---

eval.desc	<i>Evaluate a Molecular Descriptor</i>
-----------	--

---

### Description

The CDK implements a number of descriptors divided into three main groups - atomic, molecular and bond. This method evaluates the specified molecular descriptor(s) for a molecule

### Usage

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

### Arguments

molecules	A single IAtomContainer object or a list of references to CDK IAtomContainer objects
which.desc	The fully qualified class name of the descriptor to evaluate or a vector such names
verbose	If TRUE, progress will be written to the screen, otherwise the function performs silently

### Value

A data.frame is returned. For a single molecule it will have one row, for multiple molecules it will have the number of rows equal to the number of molecules

### Author(s)

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

### See Also

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

### Examples

```
smiles <- c('CCC', 'c1ccccc1', 'CC(=O)C')
mols <- sapply(smiles, parse.smiles)

dnames <- get.desc.names('topological')
descs <- eval.desc(mols, dnames, verbose=TRUE)
```



---

get.atomic.desc.names *Get the names of the available atomic descriptors*

---

**Description**

The CDK implements a number of descriptors divided into three main groups - atomic, molecular and bond. This method returns the names of the available atomic descriptors.

**Usage**

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

**Arguments**

type	A string which can be one of "all", "topological", "geometrical", "hybrid", "constitutional", "electronic", allowing you to choose atomic descriptors of specific categories. The keyword "all" will return all available descriptors
------	---

**Value**

A vector of fully qualified descriptor names.

**Author(s)**

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

**See Also**

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

---

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

---

**Description**

This function returns a list containing IAtom objects from a molecule or a bond.

**Usage**

```
get.atoms(object)  
get.atom.count(molecule)
```

**Arguments**

object	A jobRef representing an IAtomContainer, IMolecule or IBond object
molecule	A jobRef representing an IAtomContainer

**Value**

A list containing jobjRef's to a CDK IAtom object or else the number of atoms in the molecule

**Author(s)**

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

**See Also**

[get.bonds](#), [get.point3d](#), [get.symbol](#)

---

get.bonds

*Get the bonds from a molecule*

---

**Description**

This function returns a list containing IABond objects from a molecule

**Usage**

```
get.bonds(molecule)
```

**Arguments**

molecule      A jobjRef representing an IAtomContainer, IMolecule

**Value**

A list containing jobjRef's to a CDK IBond object

**Author(s)**

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

**See Also**

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

---

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.desc.categories     *Get Descriptor Class Names*

---

**Description**

This function returns the broad descriptor categories that are available. Examples include topolgal, geometrical and so on. You can use a specific category to avoid calculating all descriptors for a set of molecules and saves you having to select individual descriptors by hand.

**Usage**

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

**Value**

A character vector of descriptor category names

**Author(s)**

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

**See Also**

[eval.desc](#), [get.desc.names](#)

---

get.desc.names

*Get Descriptor Class Names*

---

**Description**

The CDK implements a number of descriptors divided into three main groups - atomic, molecular and bond. Currently the package will only evaluate molecular descriptors. This function returns the class names of the available descriptors, which can then be used to calculate descriptors for a specific molecule.

By default all available descriptor class names are returned. However it is possible to specify that a subset of the descriptors should be considered. The subset is specified by keyword and can be one of: topological, geometrical, hybrid, constitutional, protein, electronic.

**Usage**

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

**Arguments**

type                    Indicates which subset of molecular descriptors should be considered

**Value**

A character vector of descriptor class names

**Author(s)**

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

**See Also**

[eval.desc](#), [get.desc.categories](#)

---

get.fingerprint      *Evaluate Fingerprints*

---

### Description

This function evaluates fingerprints of a specified type for a set of molecules or a single molecule. Depending on the nature of the fingerprint, parameters can be specified. Currently five different fingerprints can be specified:

- standard - Considers paths of a given length. The default is but can be changed. These are hashed fingerprints, with a default length of 1024
- extended - Similar to the standard type, but takes rings and atomic properties into account
- graph - Similar to the standard type by simply considers connectivity
- 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

Depending on whether the input is a single IAtomContainer object, a list or single vector is returned. Each element of the list is an S4 object of class fingerprint, which can be manipulated with the fingerprint package.

### Usage

```
get.fingerprint(molecule, type = 'standard', depth=6, size=1024, verbose=FALSE)
```

### Arguments

molecule	An IAtomContainer object that can be obtained by loading them from disk or drawing them in the editor.
type	The type of fingerprint. Alternatives include 'extended', 'graph', 'maccs', 'estate', 'hybridization'
depth	The search depth. This argument is ignored for the 'pubchem', 'maccs' and 'estate' fingerprints
size	The length of the fingerprint bit string. This argument is ignored for the 'pubchem', 'maccs' and 'estate' fingerprints
verbose	If TRUE, exceptions, if they occur, will be printed

### Value

Objects of class fingerprint, from the fingerprint package. If there is a problem during fingerprint calculation, NULL is returned.

**Author(s)**

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

**See Also**

[load.molecules](#)

**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.formula

*Get the formula object from a formula character.*

---

**Description**

This function returns a formula object containing mass, string character and isotopes when is given a character/string formula.

**Usage**

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

**Arguments**

mf                    A string containing the formula of the molecular formula of chemical object.  
charge                The charge of the molecular formula.

**Value**

Objects of class `cdkFormula`, from the `IMolecularFormula` package

**Author(s)**

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

**References**

A parallel effort to expand the Chemistry Development Kit: <http://cdk.sourceforge.net>

**See Also**

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

**Examples**

```
formula <- get.formula('NH4', charge = 1)
formula
```

---

`get.isotopes.pattern`    *Generate the isotope pattern.*

---

**Description**

This function get the isotope pattern given a `cdkFormula` object. It modifies as the `IMolecularFormula` Java object as the its mass.

**Usage**

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

**Arguments**

<code>formula</code>	A <code>cdkFormula</code> object.
<code>minAbund</code>	Minimal abundance of the isotopes to be added in the combinatorial search.

**Value**

Objects of class `IsotopePatternGenerator`, from the `IMolecularFormula` package

**Author(s)**

Miguel Rojas-Cherto (<[miguelrojasch@yahoo.es](mailto:miguelrojasch@yahoo.es)>)

**References**

A parallel effort to expand the Chemistry Development Kit: <http://cdk.sourceforge.net>

**See Also**

[get.formula](#), [set.charge.formula](#), [isvalid.formula](#), [generate.formula](#)

get.mol2formula      *Parser a molecule to formula object.*

---

**Description**

This function convert a molecule object to a formula object.

**Usage**

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

**Arguments**

molecule      The molecule to be parsed.,  
charge          The charge characterizing the molecule.,

**Value**

Objects of class MolecularFormulaManipulator, from the IMolecularFormulaManipulator package

**Author(s)**

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

**See Also**

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

**Examples**

```
molecule <- parse.smiles("N")[[1]]  
convert.implicit.to.explicit(molecule)  
formula <- get.mol2formula(molecule,charge=0)
```

---

get.murcko.fragments      *Molecule Fragmentation Methods*

---

**Description**

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.

**Usage**

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

**Arguments**

mols	A molecule object or list of molecule objects. Each object should have a jclass of IAtomContainer
min.frag.size	The size of the smallest fragments to be considered
as.smiles	If TRUE, the fragments are returned as SMILES strings, otherwise as IAtomContainer 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

**Value**

get.murcko.fragments 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. get.exhaustive.fragments 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 IAtomContainer objects.

**Author(s)**

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

**See Also**

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

**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.properties

*Get All Property Values of a Molecule*

---

**Description**

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            A Java object of class IAtomContainer or IMolecule

**Value**

A list of the property values, with names equal to the property names. NULL property values are returned as NA

**Author(s)**

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

**See Also**

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

**Examples**

```
smiles <- 'c1ccccc1'  
mol <- parse.smiles(smiles)[[1]]  
set.property(mol, 'prop1', 23.45)  
set.property(mol, 'prop2', 'inactive')  
get.properties(mol)
```

---

get.property

*Get the Value of a Molecule Property*

---

**Description**

This function retrieves the value of a keyed property that has previously been set on the molecule.

The `get.title` function is simply a wrapper around `get.property` that directly provides access to the molecule title.

**Usage**

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

**Arguments**

molecule            A Java object of class IAtomContainer  
key                    A string naming the property

**Value**

The value of the property is the key is found else NA. For get.title, the title of the molecule if available otherwise NA

**Author(s)**

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

**See Also**

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

**Examples**

```
smiles <- 'c1ccccc1'
mol <- parse.smiles(smiles)[[1]]
set.property(mol, 'prop1', 23.45)
set.property(mol, 'prop2', 'inactive')
get.property(mol, 'prop1')
```

---

get.smiles

*Get the SMILES for 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)
```

**Arguments**

molecule      A Java object of class IAtomContainer

**Value**

An R character object containing the SMILES

**Author(s)**

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

**Examples**

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

---

get.smiles.parser      *Get a SMILES Parser*

---

### Description

This function returns a reference to a SMILES parser object. If you are parsing multiple SMILES strings, 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 to a CDK SmilesParser object

### Author(s)

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

### See Also

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

---

get.total.charge      *Get the Total Charges for the Molecule*

---

### Description

`get.total.charge` returns the summed partial charges for a molecule and `get.total.formal.charge` returns the summed formal charges. Currently, if one or more partial charges are unset, the function simply returns the sum of formal charges (via `get.total.formal.charge`). This is slightly different from how the CDK evaluates the total charge of a molecule (via `AtomContainerManipulator.getTotalCharge()`), but is in line with how OEChem determines net charge on a molecule.

In general, you will want to use the `get.total.charge` function.

### Usage

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

### Arguments

molecule      A Java object of class IAtomContainer

**Value**

A double value indicating the total partial charge or total formal charge

**Author(s)**

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

---

`get.total.hydrogen.count`

*Get the Total Hydrogen Count for a Molecule*

---

**Description**

The function will return the summed implicit hydrogens of all atoms in the specified AtomContainer

**Usage**

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

**Arguments**

molecule          A Java object of class IAtomContainer

**Value**

An integer value indicating the number of implicit hydrogens

**Author(s)**

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

---

`get.tpsa`

*Commonly Used Molecular Descriptors*

---

**Description**

These methods will return the value for the corresponding descriptors. While they can always be evaluated using [eval.desc](#), they are common enough that separate functions are provided.

**Usage**

`get.tpsa(molecule)`  
`get.alogp(molecule)`  
`get.xlogp(molecule)`  
`get.volume(molecule)`

**Arguments**

molecule            A jObjRef representing an IAtomContainer object

**Details**

It's important to note that ALogP and XLogP assumes that the molecule has explicit hydrogens. If the molecule is read from an SD file, explicit H's are usually present. On the other hand, if the molecule is obtained from a SMILES, explicit hydrogens must be added.

The molecular volume is calculated using a group contribution method rather than the an analytical method. This allows to avoid the use of 3D structures.

**Value**

Single numeric value representing TPSA, ALogP, XLogP or molecular volume.

**Author(s)**

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

**See Also**

[eval.desc](#)

---

hasNext

*Does This Iterator Have A Next Element*

---

**Description**

hasNext is a generic function that indicates if the iterator has another element.

**Usage**

```
hasNext(obj, ...)
```

```
## S3 method for class 'iload.molecules'  
hasNext(obj, ...)
```

**Arguments**

obj                    an iterator object.  
...                    additional arguments that are ignored.

**Value**

Logical value indicating whether the iterator has a next element. In the context of reading a structure file, this indicates whether there are more molecules to read

**See Also**[iload.molecules](#)

---

`is.connected`*Get the Largest Component in a Disconnected Molecule*

---

**Description**

These methods allow one to check whether a molecule is fully connected or else retrieve the largest disconnected component

**Usage**

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

**Arguments**

`mol`                    A `jObjRef` representing an `IAtomContainer` object

**Value**

For `get.largest.component`, if the input molecule has more than one disconnected component, the largest is returned. Otherwise, the molecule itself is returned.

For `is.connected`, TRUE if the molecule is fully connected, FALSE otherwise

**Author(s)**

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

**Examples**

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

---

isvalid.formula	<i>Validate a cdkFormula object.</i>
-----------------	--------------------------------------

---

### Description

This function validates a cdkFormula object. At the moment is using the nitrogen Rule and RDBE Rule.

### Usage

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

### Arguments

formula	A cdkFormula object.
rule	The rules to be applied: nitrogen and RDBE.

### Value

Objects of class MolecularFormulaChecker, from the IMolecularFormula package

### Author(s)

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

### References

A parallel effort to expand the Chemistry Development Kit: <http://cdk.sourceforge.net>

### See Also

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

### Examples

```
formula <- get.formula('NH4', charge = 0)
isvalid.formula(formula, rule = c("nitrogen", "RDBE"))
```

---

load.molecules	<i>Load Molecular Structures From Disk</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

### Usage

```
load.molecules(molfiles=NA, aromaticity = TRUE, typing = TRUE, isotopes = TRUE, verbose=FALSE)
iload.molecules(molfile, type="smi", aromaticity = TRUE, typing = TRUE, isotopes = TRUE, skip=TRUE)
```

### 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://"
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
verbose	If TRUE, output (such as file download progress) will be bountiful
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 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

load.molecules returns a list of CDK Molecule objects, which can be used in other rcdk functions.

iload.molecules is an iterating version of the loader and is applicable for large SMILES or SDF files. In contrast to load.molecules this does not load all the molecules into memory at one go, and as a result lets you process arbitrarily large structure files.

**Author(s)**

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

**See Also**

[view.molecule.2d](#), [convert.implicit.to.explicit](#)

**Examples**

```
## Not run:

## load a single file
amol <- load.molecules('foo.sdf')

## load multiple files
mols <- load.molecules(c('mol1.sdf', 'mol2.smi', 'https://github.com/rajarshi/cdkr/blob/master/data/set2/dhfr00

## iterate over a large file
moliter <- iload.molecules("big.sdf", type="sdf")
while(hasNext(moliter)) {
  mol <- nextElem(moliter)
  print(get.property(mol, "cdk:Title"))
}

## End(Not run)
```

---

matches

*Perform Substructure Searching & MCS Detection*

---

**Description**

These functions perform substructure searches of a query, specified in SMILES or SMARTS forms, over one or more target molecules and maximum common substructure searches for pairs of molecules.

**Usage**

```
matches(query, target)
is.subgraph(query, target)
get.mcs(mol1, mol2, as.molecule = TRUE)
```

**Arguments**

query	A SMILES or SMARTS string
target	A single IAtomContainer object or a list of IAtomContainer objects
mol1	An IAtomContainer
mol2	An IAtomContainer
as.molecule	If TRUE the MCS is returned as a new IAtomContainer object. Otherwise a atom index mapping between the two molecules is returned as a 2D array of integers

## Details

For the case of `is.subgraph`, the query molecule must be a single `IAtomContainer` or a valid SMILES string. Note that this method can be significantly faster than `matches`, but is limited by the fact that SMARTS patterns cannot be specified. This uses the "TurboSubStructure" SMSD method and so only searches for the first substructure match.

For MCS detection, the default SMSD algorithm is employed and the best scoring MCS is returned by default. Furthermore, one can obtain the resultant MCS either as an `IAtomContainer` in which the atoms and bonds are clones of the corresponding matching atoms and bonds in one of the molecule. Or else as a 2D array of dimensions  $N \times 2$  of atom index mappings. Here  $N$  is the size of the MCS and the first column represents the atom index from the first molecule and the second column the atom index from the second molecule.

Note that since the CDK SMARTS matcher internally will perform aromaticity perception and atom typing, the target molecules need not have these operations done on them beforehand for `matches` method. However, if `is.subgraph` or `get.mcs` is being used, the molecules should have aromaticity detected and atom typing performed explicitly.

## Value

For `matches` and `is.subgraph`, a boolean vector, where each element is `TRUE` or `FALSE` depending on whether the corresponding element in targets contains the query or not.

For `get.mcs` an `IAtomContainer` object or a 2D array of atom index mappings between the two molecules.

## Author(s)

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

## See Also

[load.molecules](#), [get.smiles](#), [do.aromaticity](#), [do.typing](#), [do.isotopes](#)

## Examples

```
smiles <- c('CCC', 'c1ccccc1', 'C(C)(C=O)C(CCNC)C1CC1C(=O)')
mols <- sapply(smiles, parse.smiles)
query <- '[#6]=O'
doesMatch <- matches(query, mols)
```

### 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.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

### Usage

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

### Arguments

`molecule`            A `jObjRef` representing an `IAtomContainer` or `IMolecule` object

### Details

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). 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.

### Value

`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](#), [do.typing](#)

## Examples

```
m <- parse.smiles('c1ccccc1')[[1]]

## Need to configure the molecule
do.aromaticity(m)
do.typing(m)
do.isotopes(m)

get.exact.mass(m)
get.natural.mass(m)

convert.implicit.to.explicit(m)
get.natural.mass(m)
do.isotopes(m) # Configure isotopes of newly added hydrogens
get.exact.mass(m)

is.neutral(m)
```

---

parse.smiles

*Parse a Vector of SMILES Strings*

---

## 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, 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)
```

## Arguments

smiles            A SMILES string

## Value

A list of jobjRefs to their corresponding CDK IAtomContainer objects. If a SMILES string could not be parsed, NA is returned instead.

## Author(s)

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

**See Also**

[load.molecules](#), [get.smiles](#), [get.smiles.parser](#), [view.molecule.2d](#), [do.aromaticity](#), [do.typing](#), [do.isotopes](#)

**Examples**

```
smiles <- c('CCC', 'c1ccccc1', 'C(C)(C=O)C(CCNC)C1CC1C(=O)')
mol <- parse.smiles(smiles[1])
mols <- parse.smiles(smiles)
```

---

remove.hydrogens      *Remove Hydrogens from a Molecule*

---

**Description**

This function generate a new IAtomContainer object in which the hydrogens have been removed. This can be useful for descriptor calculations.

**Usage**

```
remove.hydrogens(molecule)
```

**Arguments**

molecule      A Java object of class IAtomContainer

**Value**

A jobref that refers to a IAtomContainer object

**Author(s)**

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

---

remove.property      *Remove A Property From a Molecule*

---

**Description**

This function removes a keyed property from a molecule object. This deletes the key and its value from the molecule

**Usage**

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

**Arguments**

molecule	A Java object of class IAtomContainer
key	A string naming the property

**Value**

None

**Author(s)**

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

**See Also**

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

---

set.charge.formula      *Set the charge to a cdkFormula object.*

---

**Description**

This function set the charge to a cdkFormula object. It modifies as the IMolecularFormula Java object as the its mass.

**Usage**

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

**Arguments**

formula	A cdkFormula object.
charge	The value of the charge to set.

**Value**

Returns the formula object with the specified charge

**Author(s)**

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

**References**

A parallel effort to expand the Chemistry Development Kit: <http://cdk.sourceforge.net>

**See Also**

[get.formula](#), [get.isotopes.pattern](#), [isvalid.formula](#), [generate.formula](#)

---

set.property	<i>Set A Property On A Molecule</i>
--------------	-------------------------------------

---

### Description

This function allows one to add a keyed property to a molecule. The key must be a string, but the value can be string, numeric or even an arbitrary Java object (of class jobjRef)

### Usage

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

### Arguments

molecule	A Java object of class IAtomContainer
key	A string naming the property
value	The value of the property. This can be character, integer, double or of class jobjRef

### Value

None

### Author(s)

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

### See Also

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

### Examples

```
smiles <- 'c1ccccc1'  
mol <- parse.smiles(smiles)[[1]]  
set.property(mol, 'prop1', 23.45)  
set.property(mol, 'prop2', 'inactive')  
get.properties(mol)
```

## Description

The CDK is capable of generating 2D structure diagrams. These methods allow one to view 2D structure diagrams. Depending on the method called a Swing JFrame is displayed which allows resizing of the image or a raster image (derived from a PNG byte stream) is returned, which can be viewed using [rasterImage](#). It is also possible to copy a 2D depiction to the system clipboard, which can then be pasted into various external applications.

## Usage

```
view.molecule.2d(molecule, ncol = 4, cellx = 200, celly = 200)
view.image.2d(molecule, width = 200, height = 200)
copy.image.to.clipboard(molecule, width = 200, height = 200)
```

## Arguments

molecule	If a single molecule is to be viewed this should be a reference to a IAtomContainer object. If multiple molecules are to be viewed this should be a list of such objects. If a character is specified then it is taken as the name of a file and the molecules are loaded from the file
ncol	The number of columns if a grid is desired
cellx	The width of the grid cells
celly	The height of the grid cells
width	The width of the image
height	The height of the image

## Details

For the case of `view.molecule.2d`, if a `jobjRef` is passed it should be a reference to an `IAtomContainer` object. In case the first argument is of class character it is assumed to be a file and is loaded by the function.

This function can be used to view a single molecule or multiple molecules. If a list of molecule objects is supplied the molecules are displayed as a grid of 2D viewers. In case a file is specified, it will display a single molecule or multiple molecules depending on how many molecules are loaded.

For `view.image.2d`, the image can be viewed via [rasterImage](#).

`copy.image.to.clipboard` copies the 2D depiction to the system clipboard in PNG format. You can then paste into other applications.

Due to event handling issues, the depiction will show on OS X, but the window will be unresponsive. Also copying images to the clipboard will not work. As a result, on OS X we make use of a standalone helper that is run via the system command. Currently, this is supported for the `view.molecule.2d` method (for a single molecule) and the `copy.image.to.clipboard` method.

In the future, other view methods will also be accessible via this mechanism. While this allows OS X users to view molecules, it is slow due to invoking a new process.

The depictions will work fine (i.e., no need to shell out) on Linux and Windows.

### Value

view.molecule.2d and copy.image.to.clipboard do not return anything. view.image.2d returns an array of the dimensions height x width x channels, from the original PNG version of the 2D depiction.

### Author(s)

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

### See Also

[view.table](#), [rasterImage](#), [link{readPNG}](#)

### Examples

```
m <- parse.smiles('c1cccc1C(=O)NC')[[1]]
## Not run:
img <- view.image.2d(m, 100,100)
plot(1:10, 1:10, pch=19)
rasterImage(img, 0,8, 2,10)

## End(Not run)
```

---

view.table

*View 2D Structures With Data*

---

### Description

The CDK is capable of generating 2D structure diagrams. This function can be used to view a set of molecules along with some associated data. The format of the output is a table, where the first column are the 2D images of the molecules, followed by the data columns.

### Usage

```
view.table(molecules, dat, cellx = 200, celly = 200)
```

### Arguments

molecules	A list of jobRef objects that represent IAtomContainer
dat	A data.frame containing numeric or character columns. If columns are named they will be used in the data table. If not, names are autogenerated. The number of rows of the data.frame should be equal to the number of molecules
cellx	Initial width of the table cells
celly	Initial height of the table cells

## Details

Due to event handling issues, the depiction will show on OS X, but the window will be unresponsive. The depictions will work fine on Linux and Windows.

## Value

Nothing

## Author(s)

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

## See Also

[view.molecule.2d](#)

## Examples

```
smiles <- c('CCC', 'CCN', 'CCN(C)(C)',
           'c1ccccc1Cc1ccccc1',
           'C1CCC1CC(CN(C)(C))CC(=O)CC')
mols <- parse.smiles(smiles)
dframe <- data.frame(x = runif(4),
                    toxicity = factor(c('Toxic', 'Toxic', 'Nontoxic', 'Nontoxic')),
                    solubility = c('yes', 'yes', 'no', 'yes'))
## Not run: view.table(mols[1:4], dframe)
```

---

write.molecules

*Write Molecules To Disk*

---

## 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 Java objects of class IAtomContainer
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	Should keyed properties be included in the SD file output

**Details**

This function can be used to write a single SD file containing multiple molecules. In case individual SD files are desired the `together` argument can be set to `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. In case, a single molecule is to be written to disk, simply specify the filename and use the default value of `together`

**Value**

The value of the property

**Author(s)**

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

**See Also**

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

# Index

- \*Topic **classes**
  - cdkFormula-class, 5
- \*Topic **datasets**
  - bpdata, 4
- \*Topic **methods**
  - hasNext, 22
- \*Topic **programming**
  - Atoms, 2
  - cdk.version, 4
  - do.aromaticity, 5
  - eval.atomic.desc, 6
  - eval.desc, 7
  - generate.formula, 8
  - get.atomic.desc.names, 9
  - get.atoms, 9
  - get.bonds, 10
  - get.connected.atom, 11
  - get.desc.categories, 11
  - get.desc.names, 12
  - get.fingerprint, 13
  - get.formula, 14
  - get.isotopes.pattern, 15
  - get.mol2formula, 16
  - get.murcko.fragments, 16
  - get.properties, 17
  - get.property, 18
  - get.smiles, 19
  - get.smiles.parser, 20
  - get.total.charge, 20
  - get.total.hydrogen.count, 21
  - get.tpsa, 21
  - is.connected, 23
  - invalid.formula, 24
  - load.molecules, 25
  - matches, 26
  - Molecule, 27
  - parse.smiles, 29
  - remove.hydrogens, 30
  - remove.property, 30
  - set.charge.formula, 31
  - set.property, 32
  - view.molecule.2d, 33
  - view.table, 34
  - write.molecules, 35
- Atoms, 2
- bpdata, 4
- cdk.version, 4
- cdkFormula-class, 5
- charge (get.total.charge), 20
- convert.implicit.to.explicit, 25, 26, 28
- convert.implicit.to.explicit (Molecule), 27
- copy.image.to.clipboard (view.molecule.2d), 33
- depict (view.molecule.2d), 33
- do.aromaticity, 5, 27, 30
- do.isotopes, 27, 30
- do.isotopes (do.aromaticity), 5
- do.typing, 27, 28, 30
- do.typing (do.aromaticity), 5
- eval.atomic.desc, 6, 9
- eval.desc, 6, 7, 9, 12, 21, 22
- fragment (get.murcko.fragments), 16
- generate.formula, 8, 15, 24, 31
- get.alogp (get.tpsa), 21
- get.atom.count (get.atoms), 9
- get.atomic.desc.names, 6, 9
- get.atomic.number (Atoms), 2
- get.atoms, 3, 9, 10, 11, 28
- get.bonds, 10, 10
- get.charge (Atoms), 2
- get.connected.atom, 10, 11
- get.desc.categories, 7, 11, 12

- get.desc.names, 6, 7, 9, 12, 12
- get.exact.mass (Molecule), 27
- get.exhaustive.fragments
  - (get.murcko.fragments), 16
- get.fingerprint, 13
- get.formal.charge (Atoms), 2
- get.formula, 5, 8, 14, 15, 24, 31
- get.hydrogen.count (Atoms), 2
- get.isotopes.pattern, 5, 8, 15, 15, 16, 24, 31
- get.largest.component (is.connected), 23
- get.mcs (matches), 26
- get.mol2formula, 16
- get.murcko.fragments, 16
- get.natural.mass (Molecule), 27
- get.point2d (Atoms), 2
- get.point3d, 10
- get.point3d (Atoms), 2
- get.properties, 17, 19, 32
- get.property, 18, 18, 31, 32, 36
- get.smiles, 19, 20, 27, 30
- get.smiles.parser, 20, 20, 30
- get.symbol, 10
- get.symbol (Atoms), 2
- get.title (get.property), 18
- get.total.charge, 20
- get.total.formal.charge
  - (get.total.charge), 20
- get.total.hydrogen.count, 21
- get.tpsa, 21
- get.volume (get.tpsa), 21
- get.xlogp (get.tpsa), 21
  
- hasNext, 22
  
- iload.molecules, 23
- iload.molecules (load.molecules), 25
- is.aliphatic (Atoms), 2
- is.aromatic (Atoms), 2
- is.connected, 23
- is.in.ring (Atoms), 2
- is.neutral (Molecule), 27
- is.subgraph (matches), 26
- isvalid.formula, 5, 8, 15, 16, 24, 31
  
- load.molecules, 5, 6, 14, 17, 25, 27, 29, 30, 36
  
- match (matches), 26
  
- matches, 26
- mcs (matches), 26
- Molecule, 27
  
- parse.smiles, 5, 6, 17, 20, 29
  
- rasterImage, 33, 34
- remove.hydrogens, 30
- remove.property, 18, 19, 30, 32, 36
  
- set.charge.formula, 5, 8, 15, 16, 24, 31
- set.property, 18, 19, 31, 32, 36
- show, cdkFormula-method
  - (cdkFormula-class), 5
- smarts (matches), 26
- substructure (matches), 26
  
- view.image.2d (view.molecule.2d), 33
- view.molecule.2d, 20, 26, 30, 33, 35
- view.table, 34, 34
  
- write.molecules, 35