JOELib Tutorial

A Java based cheminformatics/computational chemistry package

Dipl. Chem. Jörg K. Wegner

JOELib Tutorial: A Java based cheminformatics/computational chemistry package

by Dipl. Chem. Jörg K. Wegner

Published $Date: 2004/03/16 09:16:14 $

Updated

$Date: 2004/03/16 09:16:14 $

License

Documents

PS (JOELibTutorial.ps), PDF (JOELibTutorial.pdf), RTF (JOELibTutorial.rtf) versions of this tutorial are available.
Plucker E-Book (http://www.plkr.org) versions: HiRes-color (JOELib-HiRes-color.pdb), HiRes-grayscale (JOELib-HiRes-grayscale.pdb) (recommended), HiRes-black/white (JOELib-HiRes-bw.pdb), color (JOELib-color.pdb), grayscale (JOELib-grayscale.pdb), black/white (JOELib-bw.pdb)

Revision History

Revision $Revision: 1.5 $ $Date: 2004/03/16 09:16:14 $
$Id: JOELibTutorial.sgml,v 1.5 2004/03/16 09:16:14 wegner Exp $

Table of Contents

Preface ..... **i**

1. Installing JOELib **1**

Java Development Kit (JDK) 1
Ant - XML based makefile mechanism 1
Installing and starting JOELib 1
Linux/Unix 1
Windows 2000/NT 2
Windows 95/98/XP 2
Matlab toolbox 3
Using another JDK 3

2. JOELib basics **4**

Atoms 4
Accessing atoms 4
Chemical properties 4
Bonds 5
Accessing bonds 5
Chemical properties 6
Molecule 7
Basics 7
Chemical properties 7

3. Molecule operation methods and classes **8**

Molecule 8
Molecule data entries 8
Molecule descriptors 8
Set and get descriptor data entries 8
Using external calculated descriptors 9
Available descriptors 10
Writing your own descriptor and result classes 10
SMiles ARbitrary Target Specification (SMARTS)-substructure search 11
SMARTS basics 11
SMARTS definition 11
Programmable Atom Typer (PATTY) 12
SMARTS based structure modification 12
Processes and filters 13
Processes (internal, Java) 13
Process filters 13
External processes 13
Input and output 14
Supported input and output formats 14
XML respective Chemical Markup Language (CML) 15
CACTVS's clear text format (CTX) 16
Image writers (BMP, GIF, JPEG, PPM) 18

Joint Commitee on Atomic and Molecular Physical Data (JCAMP) 17
Protein Data Base (PDB) 17
Portable Adobe Document Format (PDF) 18
Persistence Of Vision (POV) Ray Tracer 18
Structured Data File (SDF) 18
Simplified Molecular Input Line Entry System (SMILES) 18
Sybyl (MOL2) 19
Tinker (TINKER) 19
Writing your own import/export filter 19
Assigning atom types, aromatic flags, hybridization and hydrogens 19
Assigning aromaticity flags 20
Assigning atom hybridizations 20
Assigning atom types 20
Assigning implicite hydrogens 20
Calculate descriptors and/or assign special atomtypes 21
4. Used utilities 22
Development 22
Maintenance 22
5. Descriptors 24
Native values 24
Number of acidic groups 24
Number of aliphatic hydroxy groups 24
Number of aromatic bonds 24
Number of basic groups 24
Fraction of rotatable bonds 24
Geometrical diameter 24
Geometrical radius 24
Geometrical shape coefficient 24
Graph shape coefficient 24
Number of Hydrogen Bond Acceptors (HBA) 1 25
Number of Hydrogen Bond Acceptors (HBA) 2 25
Number of Hydrogen Bond Donors (HBD) 1 25
Number of Hydrogen Bond Donors (HBD) 2 25
Number of heavy bonds 25
Number of heterocycles 25
Number of hydrophobic groups 25
Kier Shape 1 25
Kier shape 2 26
Kier shape 3 26
Octanol/Water partition coefficient (logP) 26
Molar refractivity (MR) 26
Molecular weight (MW) 26
Number of atoms 26
Number of boron atoms 26
Number of bromine atoms 26
Number of bonds 26
Number of chlorine atoms 27

Number of halogen atoms....27
Number of iodine atoms....27
Number of fluorine atoms....27
Number of nitrogen atoms....27
Number of oxygen atoms....27
Number of phosphorus atoms....27
Number of sulfur atoms....27
Number of -NO_2 groups....27
Number of -OSO atoms....27
Polar surface area (PSA)....27
Number of rotatable bonds....27
Number of -SO groups....28
Number of -SO_2 atoms....28
Topological diameter....28
Topological radius....28
Zagreb index 1....28
Zagreb index 2....28
Atom properties....28
Atom in acceptor....28
Atom in conjugated environment....28
Atom in donor or acceptor....29
Atom in donor....29
Atom in ring....29
Atom is terminal carbon....29
Atom is negative....29
Atom is positive....29
Atom masss....29
Valence....29
Van der Waals volume....29
Conjugated electrotopological state....30
Electrogeometrical state....30
Electron affinity....30
Electronegativity after Pauling....30
Electrotopological state....30
Gasteiger-Marsili....31
Graph potentials....31
Intrinsic topological state....31
Fingerprints....31
Pharmacophore fingerprint....31
Transformations....34
Moreau-Broto topological autocorrelation....34
Burden modified eigenvalues....34
Global topological charge....34
Radial distribution function (RDF)....34
6. Algorithms....36
Breadth First Search (BFS)....36
Depth First Search (DFS)....36

Topological distance matrix 37
Geometrical distance matrix 37
Morgan: Unique atom numbering 37
7. Interfaces to other libraries 39
Interfaces to Java libraries 39
Chemical Development Kit (CDK) 39
Weka data mining library 39
Interfaces to C/C++ libraries using the Java native interface (JNI) 39
LibGhemical 39
Matlab interface 39
8. Documentation 40
DocBook 40
DocBook and mathematical equations 40
DocBook and molecular structures 41
API documentation 43
9. JOELib examples and code snippets 45
Molecule 45
Molecules 45
Load and store molecules 45
Modify molecules 47
Access atoms and bonds of a molecule 48
Special atom methods 49
Access the neighbour atoms of an atom 49
Access the bonds of an atom 49
Descriptors 50
Get and calculate descriptors 50
Create own descriptor classes 52
Processes and filters 56
Applying processes and filters 56
Import/Export 57
Import 57
Get molecule import classes 57
Export 58
Get molecule export classes 58
Simple Import/Export pipeline 59
Create molecule processing pipe 59
Interfaces 61
Java interfaces 61
Chemical Development Kit (CDK) 61
Weka data mining interface 61
C/C++ interfaces using JNI 62
Some Java implementions to test Ghemical 62
Matlab interface 63
Matlab to Java connection 63
Java to Matlab connection 64
Database 64

Database access using the Java Database Connection (JDBC) 64
Miscellaneous 65
Point group symmetry calculation 65
Hash code calculation 66
Reading JCAMP-DX spectras 66
Create fragmented molecules 66
Combinatorial synthesis for generating virtual combinatorial libraries 66
10. Example applications 67
Convert 67
Options 69
Examples 69
ConvertSkip 71
Options 71
Examples 71
Comparison 71
Options 71
Descriptor calculation 72
Options 72
EuklidianComparison 74
Options 74
Statistic 74
Options 74
Point Group Symmetry 75
Options 75
11. Support 77
A. Summary of molecular structures 78
Bibliography 80
Glossary 82
Index 84

List of Tables

3-1. Predefined data types/names 8
3-2. Substructure search expressions 11
3-3. Supported file formats 14
3-4. Process of assigning atom types 20
3-5. Possible special atom type assignments (not implemented) 21
5-1. SMARTS definitions for assigning the conjugated atom property flag 29
5-2. Pharmacophore fingerprint definition 32
8-1. Molecular structures with atom numbers and special atom labels 41
8-2. Molecular structures with a delocalised ring 42
8-3. Molecular structures with retrosynthetic bond splittings 42
8-4. Molecular structures with electron transfer arrows 43
9-1. Simple database definition 64

List of Figures

6-1. Pseudocode for the BFS algorithm....36
6-2. Pseudocode for the DFS algorithm....36
6-3. Pseudocode for the Morgan labeling algorithm....37
6-4. Pseudocode for the Morgan renumbering algorithm....38

List of Examples

2-1. Using an iterator for accessing atoms 4
2-2. Using the atom index for accessing atoms 4
2-3. Bond iterator 6
3-1. Getting descriptor data entries 9
3-2. Setting descriptor data entries 9
3-3. SMARTS substructure search 11
3-4. Definition of IO types 18
9-1. Load molecules from large files sequentially 46
9-2. Load molecules from smaller files into memory 46
9-3. Add atoms to a molecule 47
9-4. Access atoms using a `for` statement 49
9-5. Access atoms using an `AtomIterator` 49
9-6. Access bonds using a `for` statement 49
9-7. Access bonds using an `BondIterator` 49
9-8. Access neighbour atoms using a `NbrAtomIterator` 49
9-9. Access bonds of an atom using a `BondIterator` 49
9-10. Get a list of all available descriptors 54
9-11. Get a list of all native value descriptors 54
9-12. Get a list of all atom property descriptors 54
9-13. Get and calculate descriptors using always a new instance (slow) 54
9-14. Get and calculate descriptors creating only one descriptor calculation instance (fast) 54
9-15. Create own native descriptor calculation classes 54
9-16. Create own atom property descriptor calculation classes 54
9-17. Get a molecule process by creating an instance 56
9-18. Get a molecule process by using the process factory 56
9-19. Use initialization method for a process (depends on the process!) 56
9-20. Get a molecule process pipe with filter functionalities 56
9-21. Apply a process to a molecule 57
9-22. Get molecule import classes 59
9-23. Get molecule export classes 59
9-24. Create molecule processing pipe 59
9-25. Chemical Development Kit (CDK) interface 61
9-26. Weka data mining interface 61
9-27. Create 3D coordinates using the Ghemical force field (molecular mechanics) 63
9-28. Load native descriptors under Matlab 63
9-29. Database access 64
9-30. Calculate point group symmetries 66
9-31. Calculate a molecule hashcode 66
9-32. Load JCAMP-DX spectras 66
9-33. Fragment molecules if they are disconnected (non-contiguous) 66
9-34. Generate molecules using R-Groups 66
10-1. Show help screen for Convert 69
10-2. Print molecules to standard output 69

10-3. Show help screen for ConvertSkip 71
10-4. Print molecules to standard output 71
10-5. Write flat file 71
10-6. Using conversion rule 71

List of Equations

5-1. Kier shape 1 25
5-2. Kier shape 1 (alternative formulation) 25
5-3. Kier shape 2 26
5-4. Kier shape 3 26
5-5. Zagreb index 1 28
5-6. Zagreb index 2 28
5-7. Conjugated electrotopological state 30
5-8. Conjugated topological distance 30
5-9. Electrogeometrical state 30
5-10. Electrotopological state 30
5-11. Orbital electronegativity based on atom charge 31
5-12. Graph potentials 31
5-13. Intrinsic topological state 31
5-14. Moreau-Broto autocorrelation 34
5-15. Radial distribution function 35

Preface

JOELib is the Java successor of the OELib library from OpenEye (http://www.eyesopen.com) and was registered 2001-11-09 at the SF sites.

There exists also OpenBabel (http://openbabel.sourceforge.net) which is the C++ successor of the OELib library from OpenEye (http://www.eyesopen.com) and was registered 2001-11-25 at the SF sites.

JOELib has additional descriptor calculation classes. Furthermore there are some classes to process molecules and use external processing programs to modify molecular data. The processing functionality can be combined with filtering classes. A really usefull feature of this Java version is the possibility to define and use dynamicly defined IO, process and filter classes. On the other side, until now, there are some important features missing, like conformer generation and PDB-file import/export.

By the way, the JOELib logo uses the Tengwar scribe of J. R. R. Tolkien. So if you now think, that the main author of this software is a fantasy fan, you are right ! He like's fantasy novels, especially book's also suitable for childrens, because they are often much more funny and uses a more straight forward writting style.

"*Some who have read the book, or at any rate have reviewed it, have found it boring, absurd, or contemptible; and I have no cause to complain, since I have similar opinions of their works, or of the kinds of writing that they evidently prefer.*" (**J. R. R. Tolkien**, foreword in lord of the rings)

"*Yet the most interesting science is to be found in the unknown world. How do you go from the known to the unknown ?*" (**Sir D. H. R. Barton**, foreword in 'Serendipity-Accidental discoveries in science, R. M. Roberts, 1989'). I think this holds also for the pharmaceutical industry, which searches for drugs (with no known patent). In contrast to this statement we hope to provide a software library presenting as much knowledge based and reproducible algorithms as possible, because we are interested to find *serendipitious* phenomens in chemistry not in the algorithms we apply.

By the way, *experimental design* algorithms (task for engineers or computer scientists) looks also in areas we have never discovered, but which similarity measure should we apply on dynamic chemical graphs ? So where are we now looking for new drugs ? "*looking in known world*" or "*looking in the unknown world*" ?

Chapter 1. Installing JOELib

Java Development Kit (JDK)

To start and compile the included examples, you must have installed JDK1.4 (http://java.sun.com/j2se/1.4/).

Ant - XML based makefile mechanism

Apache Ant (http://ant.apache.org/) is a very good tool to develop, compile, distribute and run Java applications under Linux AND Windows.

It supports a common Makefile mechanism for Java on an XML based format and is supported from IDE's like Eclipse (http://www.eclipse.org/), jEdit (http://www.jedit.org/), NetBeans (http://www.netbeans.org/) and JBuilder (http://www.borland.com/jbuilder/) (AntRunner (http://www.dieter-bogdoll.de/java/AntRunner/)).

Installing and starting JOELib

Get an up-to-date version of the JOELib library from the homepage at http://joelib.sourceforge.net.

Linux/Unix

1. Get an up-to-date version of the JOELib library (http://joelib.sourceforge.net).
2. Get an up-to-date version of the Ant library (see the Section called *Ant - XML based makefile mechanism*).
3. Set the environent variables `JAVA_HOME` and `ANT_HOME` and extend the `PATH` variable with `$ANT_HOME/bin`.

   ```
   setenv JAVA_HOME /usr/local/j2sdk1.4.0
   setenv ANT_HOME ~/tmp/ant141
   setenv PATH ${PATH}:$ANT_HOME/bin
   ```

4. Change your current directory to `joelib/ant`. Now you can compile and start an example with:

   ```
   ant compile
   ant MoleculeLoadTest
   ```

   You can use **ant -projecthelp** to get a list of other examples.

Windows 2000/NT

1. Get an up-to-date version of the JOELib library (http://joelib.sourceforge.net).
2. Get an up-to-date version of the Ant library (see the Section called *Ant - XML based makefile mechanism*).

3. Open a MS-DOS box.
4. Set the environent variables `JAVA_HOME` and `ANT_HOME` and extend the `PATH` variable with `$ANT_HOME/bin`.

   ```
   set JAVA_HOME=C:\Programme\j2sdk1.4.0
   set ANT_HOME=C:\lib\java\ant1.4.1
   set PATH=%PATH%:%ANT_HOME%\bin
   ```

5. Change your current directory to `joelib/ant`. Now you can compile and start an example with:

   ```
   ant compile
   ant MoleculeLoadTest
   ```

   You can use **ant -projecthelp** to get a list of other examples.

Windows 95/98/XP

Unfortunately Windows98 don't support the makefile-mechanism entirely so it's necessary to change the standard calling mechanism.

1. Get an up-to-date version of the JOELib library (http://joelib.sourceforge.net).
2. Get an up-to-date version of the Ant library (see the Section called *Ant - XML based makefile mechanism*).
3. Open a MS-DOS box.
4. Open the `ant/build.xml` (joelib/ant/build.xml)-file and comment the following line:

   ```
   <property environment="env"/>
   ```

   with

   ```
   <!-- <property environment="env"/> -->
   ```

   Add the paths to your Java Development Kit and Ant with the following lines:

   ```
   <property name="env.JAVA_HOME" value="C:/Programme/j2sdk1.4.0" />
   <property name="env.ANT_HOME" value="C:/lib/java/ant1.4.1" />
   ```

5. Change your current directory to `joelib`. Now you can compile and start an example with:

   ```
   build.bat compile
   build.bat MoleculeLoadTest
   ```

   You can use **build.bat -projecthelp** to get a list of other examples.

If you were using a batch file it's important to use the '\'-slash under Windows98 ! Windows XP works also with the '/'-slash.

Matlab toolbox

Mathworks-Matlab (http://www.mathworks.com/) is mathematical, scientific environment with a very good Java connection and lot's of toolboxes.

Using another JDK

1. Get an up-to-date version of the JOELib library (http://joelib.sourceforge.net) and the JOELib-Matlab toolbox (http://joelib.sourceforge.net).
2. You must add all required Java libraries to the Matlab `CLASSPATH` which are defined in the file `yourMatlabPath/toolbox/local/classpath.txt`:

   ```
   $matlabroot/java/jarext/joelib.jar
   $matlabroot/java/jarext/log4j.jar
   ```

3. Copy all the required libraries to the directory `yourMatlabPath/java/jarext`.
4. for using Java 1.4 or higher you must add/change your system variable `MATLAB_JAVA` to

   ```
   MATLAB_JAVA 1.4.1_01
   ```

   if you are using JDK1.4.1.
5. then copy your JRE directory to the matlab directory `yourMatlabPath/sys/java/jre/win32/jre1.4.1_01`.

JOELib can directly be used under Matlab (the GUI is Java based, too) or vice versa using the JNI interface JMatLink (http://www.held-mueller.de/JMatLink/) which requires some experience with the JNI.

Chapter 2. JOELib basics

Atoms

Accessing atoms

Atoms **V** in `joelib.molecule.JOEAtom` (joelib/src/joelib/molecule/JOEAtom.java) are represented as nodes of a molecular graph **G**.

For getting all atoms of a molecule you can use an iterator or a simple index access.

Example 2-1. Using an iterator for accessing atoms

```
JOEAtom atom;
AtomIterator ait = molecule.atomIterator();
while(ait.hasNext())
{
  atom = ait.nextAtom();
}
```

Example 2-2. Using the atom index for accessing atoms

```
JOEAtom atom;
for(int index=1;index<=molecule.numAtoms();index++)
{
  atom = molecule.getAtom(index);
}
```

Chemical properties

Atoms **V** are in fact unlabeled nodes of a molecular graph **G**. The chemical atom properties are assigned with a set $\mathbf{A}=\{\boldsymbol{\alpha}_1,\boldsymbol{\alpha}_1,...,\boldsymbol{\alpha}_{|A|}\}$ of labelling functions $\boldsymbol{\alpha}_i:\mathbf{G},\mathbf{V},\mathbf{A}_{j\neq i}\rightarrow\mathbf{W}_{V,i}$. We can distinguish between a set of critical labelling functions $\mathbf{A}_{chem}$ (chemical kernel) and all other optional chemical properties.

- The chemical kernel or chemical base knowledge uses the following functions for which the dependencies are also shown:
  - The chemical element $\boldsymbol{\alpha}_{elem}(\mathbf{V})$. This atom property can be accessed by `JOEElementTable.instance().getSymbol(atom.getAtomicNum())`.
  - Valence or bond order $\boldsymbol{\alpha}_{val}(\mathbf{G},\mathbf{V})$. This atom property can be accessed by `atom.getValence()`.
  - Ring size $\boldsymbol{\alpha}_{ring}(\mathbf{G},\mathbf{V})$ which is based on a ring search algorithm, e.g. the *Smallest Set of Smallest Ring* (SSSR) [fig96]. This atom property can be accessed by `atom.isInRing()` or `atom.isInRingSize(int)`.
  - Binary aromaticity flag $\boldsymbol{\alpha}_{arom}(\mathbf{G},\mathbf{V},\boldsymbol{\alpha}_{elem},\boldsymbol{\alpha}_{val})$. This atom property can be accessed by `atom.isAromatic()`.
  - Hybridisation $\boldsymbol{\alpha}_{hyb}(\mathbf{G},\mathbf{V},\boldsymbol{\alpha}_{val},\boldsymbol{\alpha}_{arom})$. This atom property can be accessed by `atom.getHyb()`.

- Chirality informations $\alpha_{stereo}(G,V,\alpha_{val},\alpha_{hyb})$. This atom property can be accessed by `atom.isClockwise()` or `atom.isAntiClockwise()`.
- The implicite valence (hydrogen saturation) $\alpha_{iVal}(G,V,\alpha_{val},\alpha_{hyb})$. This atom property can be accessed by `atom.getImplicitValence()`.

So we obtain as chemical base knowledge (chemical kernel) $A_{chem}(lib=JOELib,kernelID)=\{\alpha_{elem},\alpha_{val},\alpha_{ring},\alpha_{arom},\alpha_{hyb},\alpha_{stereo},\alpha_{iVal}\}$

- Other atom properties are available through *look-up* tables or other algorithms based on $A_{chem}(lib=JOELib,kernelID)$. The most important one is the atom type $\alpha_{type}(G,V,A_{chem})$ which uses the *Programmable ATom TYper* (PATTY) [bs93] based on the *Smiles ARbitrary Target Specification* (SMARTS) substructure search [smarts]. By using an internal *look-up* table (`joelib.data.JOETypeTable` (joelib/src/joelib/data/JOETypeTable.java)) we can now create easily chemical file formats, like Sybyl MOL2 or AMBER.

  More atom property assignment functions and algorithms are can be found in the Section called *Atom properties* in Chapter 5.

All atom properties are automatically assigned by the expert systems (the Section called *Assigning atom types, aromatic flags, hybridization and hydrogens* in Chapter 3). We can also see that there is much room for interpreting the single assignment functions and we strongly recommend an indentifier and version number for each single function. We have introduced also a chemical kernel identifier, especially to grant identically results for all kind of algorithms, like SMARTS substructure search, similarity searches, descriptor calculations and much more.

If you are going to work on graph based algorithms: please remember that typical chemical atom types are ONLY a special case of much more labels for the atoms of molecular structures. The number is possibly infinite or at least a very large finite number, and we have also properties like `atom.isHeteroatom()`, `atom.isAxial()` and much more. Especially if you take also calculated atom properties, like e.g. CESTATE, into account. So sooner or later we will face the combinatorial optimization problem to find a good set of atom property assignment functions to fit to our problem, e.g. finding descriptors using atom properties for model building, similarity searches, and so on. Finally, we can see that there is a strong relationship between atom label assignment functions and the *NP complete feature selection* problem [wz03, wfz04a,wfz04b,fwz04].

Internally atoms have special atom types, which were defined as SMARTS [smarts] patterns in the `joelib/data/plain/atomtype.txt` (joelib/src/joelib/data/plain/atomtype.txt)-file. These types can be used to develop descriptors or to export molecules easily to other file formats (e.g. force field or ab inito programs). For the last task there is the `joelib.data.JOETypeTable` (joelib/src/joelib/data/JOETypeTable.java) helper class available, which uses the default converting types defined in `joelib/data/plain/types.txt` (joelib/src/joelib/data/plain/types.txt). The atom properties can be changed with standard set and get methods.

Bonds

Accessing bonds

Bonds in `joelib.molecule.JOEBond` (joelib/src/joelib/molecule/JOEBond.java) are represented as edges of a graph. The bond properties can be changed with standard set and get methods.

If you use `molecule.getBond(int bondIdx)` you must take into account that bonds begin with the index number 0. That's one of the biggest differences between atoms and bonds, because atoms begin with the index number 1.

Example 2-3. Bond iterator

```
JBondIterator bit = molecule.bondIterator();
JOEBond bond;
while(bit.hasNext())
{
  bond = bit.nextBond();
}
```

Chemical properties

Bonds $\mathbf{E}=\subseteq\mathbf{V}\times\mathbf{V}$ are in fact unlabeled edges of a molecular graph **G**. The chemical bond properties are assigned with a set $\mathbf{B}=\{\beta_1,\beta_1,...,\beta_{|B|}\}$ of labelling functions $\beta_i:\mathbf{G},\mathbf{V},\mathbf{A},\mathbf{B}_{j\neq i}\rightarrow\mathbf{W}_{E,i}$. We can distinguish between a set of critical labelling functions $\mathbf{E}_{chem}$ (chemical kernel) and all other optional chemical properties.

- The chemical kernel or chemical base knowledge uses the following functions for which the dependencies are also shown:
  - Bond type $\beta_{type}(\mathbf{G},\mathbf{V},\mathbf{A}_{chem},\mathbf{B}_{chem})$. These bond properties can be accessed by `bond.isSingle()`,`bond.isDouble()`,`bond.isTriple()`.
  - Binary ring bond flag $\beta_{ring}(\mathbf{G},\mathbf{V},\mathbf{A}_{chem},\mathbf{B}_{chem})$ which is based on a ring search algorithm, e.g. the *Smallest Set of Smallest Ring* (SSSR) [fig96]. This bond property can be accessed by `bond.isInRing()`.
  - Kekulé bond type $\beta_{kekule}(\mathbf{G},\mathbf{V},\mathbf{A}_{chem},\mathbf{B}_{chem})$. These bond properties can be accessed by `bond.isKSingle()`,`bond.isKDouble()`,`bond.isKTriple()`. You can, e.g. for visualization purpose, force the 'normal' bond type with the 'kekulized' bond type by using `molecule.kekulize()`.
  - Binary aromaticity flag $\beta_{arom}(\mathbf{G},\mathbf{V},\mathbf{A}_{chem},\mathbf{B}_{chem})$. This bond property can be accessed by `bond.isAromatic()`.
  - E/Z isomerism $\beta_{E/Z}(\mathbf{G},\mathbf{V},\mathbf{A}_{chem},\mathbf{B}_{chem})$. This bond property can be accessed by `joelib.util.IsomerismDetection.isCisTransBond(bond)`.
  - E/Z isomerism for SMARTS substructure search and visulalization $\beta_{up/down}(\mathbf{G},\mathbf{V},\mathbf{A}_{chem},\mathbf{B}_{chem})$. This bond property can be accessed by `joelib.util.IsomerismDetection.isCisTransBond(bond, true)`.

  So we obtain as chemical base knowledge (chemical kernel)
  $\mathbf{B}_{chem}(\mathbf{lib=JOELib,kernelID})=\{\beta_{type},\beta_{ring},\beta_{kekule},\beta_{arom},\beta_{E/Z},\beta_{up/down}\}$
- Other bond properties are for example asymmetric bond dissociation energies (directed bond properties) which can be calculated by the *Parameter Estimation for the Treatment of Reactivity Application* (PETRA) library [gas88] and then imported into JOELib using the CTX file format (the Section called *Supported input and output formats* in Chapter 3).

All bond properties are automatically assigned by the expert systems (the Section called *Assigning atom types, aromatic flags, hybridization and hydrogens* in Chapter 3).

Molecule

Basics

A molecule is represented as a graph and has typical graph properties and accessing methods. For adding user defined data it's recommended to use the descriptor atom or bond properties (see the Section called *Molecule* in Chapter 3).

Molecule load example `joelib/test/TestMolecule` (joelib/src/joelib/test/TestMolecule.java).

Chemical properties

After the molecule is loaded the aromaticity and Kekule-flags are assigned automatically (the Section called *Assigning atom types, aromatic flags, hybridization and hydrogens* in Chapter 3). if you want to force to write kekulized molecular structures use `molecule.kekulize()`. Otherwise you will work on structures with aromaticity flags, which is the recommended way. Only for visualization and some search purpose you can switch to Kekule structures.

Chapter 3. Molecule operation methods and classes

Molecule

Molecule data entries

Data entries.

Table 3-1. Predefined data types/names

Data type	Data name	Allowed occurence
`JOEDataType.JOE_UNDEFINED_DATA`	Undefined	multiple
`JOEDataType.JOE_VIRTUAL_BOND_DATA`	VirtualBondData	multiple
`JOEDataType.JOE_ROTAMER_LIST`	RotamerList	multiple
`JOEDataType.JOE_EXTERNAL_BOND_DATA`	ExternalBondData	multiple
`JOEDataType.JOE_COMPRESS_DATA`	CompressData	multiple
`JOEDataType.JOE_COMMENT_DATA`	Comment	multiple
`JOEDataType.JOE_ENERGY_DATA`	EnergyData	multiple
`JOEDataType.JOE_PAIR_DATA`	PairData	single attribute name

Molecule descriptors

Set and get descriptor data entries

The typical data type to store descriptors is the `JOEDataType.JOE_PAIR_DATA`.

Every descriptor can be accessed by his name. To access the descriptor data entries efficiently the descriptor data entries are stored in a dictionary. Therefore descriptors can only occure once in a molecule.

Example 3-1. Getting descriptor data entries

```
// getting an iterator over all data elements
// including SSSR informations and other stuff
GenericDataIterator  gdit  = mol.genericDataIterator();
while ( gdit.hasNext() )
{
  // get the next data element
  genericData = gdit.nextGenericData();
  // use only the data elements which contains descriptor
  // or user defined data
  if ( genericData.getDataType() == JOEDataType.JOE_PAIR_DATA )
  {
    // write this descriptor data as typical data block
    // to an SD file
    ps.printf( ">  <%s>", genericData.getAttribute() );
    pairData = ( JOEPairData ) genericData;
    // write data in SD format, lines not longer than 80 characters
    // per line and remove empty lines in data entries with
    // ? or a character of your choice
    ps.println( pairData.toString( IOTypeHolder.instance().getIOType( "SDF" )
) );
  }
}
```

Example 3-2. Setting descriptor data entries

```
// add a user defined data entry to the molecule
JOEPairData   dp = new JOEPairData();
// the data entry has the name 'attribute'
dp.setAttribute( attribute );
// and a typical String value
// own types must have the fromString and toString method !!!
dp.setValue( dataEntry.toString() );
mol.addData( dp );
```

Using external calculated descriptors

A big advantage is that you can use descriptors from other programs. If no calculation routine in JOELib exists all unknown descriptors (e.g. additional data elements in SDF-files) are handled as String's. If you know the data type you can simply define your own data parser/writer. All known decsriptors can be defined in `joelib/data/plain/knownResults.txt` (joelib/src/joelib/data/plain/knownResults.txt). If you access data elements with `mol.getData("DataName")` the data element will be automatically parsed if the data type is known (e.g. atom or bond properties or matrices or ...).

You can supress data parsing by using `mol.getData("DataName", false)` which can be usefull if you not want to modify all data elements (should be faster !).

If you have special atom or bond properties you should always implement the `joelib.molecule.types.AtomProperties` (joelib/src/joelib/molecule/types/AtomProperties.java) or the `joelib.molecule.types.BondProperties` (joelib/src/joelib/molecule/types/BondProperties.java) classes which guarantees you to access the data elements by the atom index or bond index which were used in JOELib. All implemented result classes are available at `joelib/desc/result` (joelib/src/joelib/desc/result) and contains simple types like int

or double but also complex types like double array or int matrix. If you want use this data types in different file formats you should add your needs to the `fromString(IOType ioType, String sValue)` and `toString(IOType ioType)`.

Available descriptors

- Kier Shape 1 descriptor [tc00] (see also the Section called *Kier Shape 1* in Chapter 5)
- Number of hydrogen bond donors (see also the Section called *Number of Hydrogen Bond Donors (HBD) 1* in Chapter 5)
- Number of nitrogen atoms(see also the Section called *Number of nitrogen atoms* in Chapter 5)
- External rotational symmetry or graph potentials [wy96] (see also the Section called *Graph potentials* in Chapter 5)
- Partial charges after Gasteiger-Marsili [gm78] (see also the Section called *Gasteiger-Marsili* in Chapter 5)

Descriptor calculation example: `joelib.test.TestDescriptor` (joelib/src/joelib/test/TestDescriptor.java)

Writing your own descriptor and result classes

All new descriptors should implement the `joelib.desc.Descriptor` (joelib/src/joelib/desc/Descriptor.java)-interface and be defined in the `joelib.properties` (joelib/src/joelib.properties)-file.

A simple example is the Kier descriptor `joelib.desc.types.KierShape1` (joelib/src/joelib/desc/types/KierShape1.java). If you have a group of similar descriptors which uses the same initilization and result class you can write a wrapper class like `joelib.desc.SMARTSCounter` (joelib/src/joelib/desc/SMARTSCounter.java) which can very easily be used to create a lot of SMARTS pattern count descriptors, e.g. `joelib.desc.types.HBD1` (joelib/src/joelib/desc/types/HBD1.java) to count the number of hydrogen donors in a molecule.

To remain user and developer friendly you should always produce a simple set of documentation files (XML, HTML, RTF) in the `docs` (joelib/src/docs/)-directory:

The easiest way would be to create a XML DocBook (http://www.docbook.org/) documentation file in the `docbook/descriptors` (joelib/docs/docbook/descriptors)-directory. These files can be easily transformed to HTML, RTF and PDF files. If you want using a formation in these descriptor documentation files you must use <sect1>...</sect1> or <sect2>...</sect2>, the <chapter> entries were already used by the tutorial book. Futhermore you can use listitems, tables or analoge elements. All these single descriptor documentation files will be generated by the Ant makefile mechanism (calling `ant tutorial`) and be available as HTML- and RTF-files in the `docs/tutorial/descriptors/documentation`-directory

SMiles ARbitrary Target Specification (SMARTS)-substructure search

SMARTS basics

The SMiles ARbitrary Target Specification (SMARTS) [smarts] is based on the SMILES notation [wei88,wei89].

Example 3-3. SMARTS substructure search

```
// benzene
String smartsPattern = "c1ccccc1";
JOESmartsPattern smarts = new JOESmartsPattern();

// parse, initialize and generate SMARTS pattern
// to allow fast pattern matching
if(!smarts.init(smartsPattern())
{
  System.err.println("Invalid SMARTS pattern.");
}

// find substructures
smarts.match(mol);
Vector         matchList  = smarts.getUMapList();

System.out.println("Pattern found "+matchList.size()+" times.");
```

SMARTS definition

The standard SMARTS definition [smarts] can be obtained directly from the Daylight homepage. The extended SMARTS definitions which were not available in the Daylight tutorial are explained here. Some of this definitions are analogous the the definitions in MOE to provide as much standard as possible.

Table 3-2. Substructure search expressions

SMARTS entry	description
`G<n>`	Atom of periodic group `n`
`Q<n>`	`n` explicite bonds to heavy atoms
`D<n>`	`n` explicit bonds (including H atoms). That's the standard definition, only the old OELib counts only heavy weight bonds.
`X<n>`	`n` number of bonds (including implicit+explicit H atoms). That's the standard definition, only the old OELib counts only implicit hydrogen bonds.
`^<n>`	Hybridisation: sp^n. See `joelib/data/plain/atomtype.txt`

SMARTS entry	description
	(joelib/src/joelib/data/plain/atomtype.txt) for more informations.

To understand the substructure search and the chirality search functionality, it could be usefull to have a look at the atom type assigning process in the Section called *Assigning atom types, aromatic flags, hybridization and hydrogens*.

For assigning atom types using a geometry-based algorithm have a look at the paper of Meng and Lewis [ml91].

Programmable Atom Typer (PATTY)

Programmable Atom Typer (PATTY) [bs93].

SMARTS based structure modification

Transformation of chemical structures (`joelib.data.JOEChemTransformation` (joelib/src/joelib/data/JOEChemTransformation.java)) can be used for a PH value correction (`joelib.data.JOEPhModel` (joelib/src/joelib/data/JOEPhModel.java)). The TRANSFORMation patterns were defined in `joelib/data/plain/phmodel.txt` (joelib/src/joelib/data/plain/phmodel.txt). An important property of *SMARTS atoms* is the vector binding number. A vector binding number is the SMARTS atom delimited by ':' and *a number* which were enclosed by [] ! Don't confound with the expression for aromatic bonds ':' which must be enclosed by *two SMARTS atom expressions*. Here are some abstract examples to understand the transformation definitions:

- Delete atoms:

  ```
  TRANSFORM O=CO[#1:1] >> O=CO
  ```

  Delete all atoms which have an defined vector binding on the left side and no equivalent vector binding number on the right side. If the right side would be of type `O=CO[#1:1]` the H atom would not be deleted.

- Change atom type:

  TRANSFORM O=C[O:1] >> O=C[N:1]

  Replace all oxygen atoms from an carboxyl group with a nitrogen atom to an amide. Change only the atoms which have the same vector binding number and different atom types.

- Change atom charge:

  TRANSFORM O=C[O:1] >> O=C[O-:1]

  Change all oxygen atoms from an carboxyl group to oxygen atoms with a negative charge. Change only the atoms which have the same vector binding number and atom charges.

- Change bond type:

  TRANSFORM [O:1]=[C:2]O >> [O:1]-[C:2]O

  Change all double bonds of carboxyl group to single bonds. The bonds which must be changed should be enclosed between two SMARTS atom expressions with the equal vector binding numbers.

If you want use another PH-value model you can simple change the `joelib.data.JOEPhModel.resourceFile=joelib/data/plain/phmodel.txt` entry in the `joelib.properties` (joelib/src/joelib.properties)-file.

Processes and filters

Processes (internal, Java)

A process can be a simple process or combined processes with filtering rules, e.g. `joelib.process.ProcessPipe` (joelib/src/joelib/process/ProcessPipe.java).

Process filters

A filter allows you to define processes which are only processed if the filter rule fits. E.g. you can define a filter which checks if an descriptor value exists and a filter which accept all molecules which fits the 'Rule of 5' (By the way, i'm interested in a freely available logP calculation program, let me know if you can offer one).

For a simle example see the external process example: `joelib.test.Title2DataTest` (joelib/src/joelib/test/Title2DataTest.java).

External processes

External processes uses external programs or shared libraries AND are system specific. External processes are derived from processes and have an additional method `isThisOSsupported()` There are three different possibilities to call external code (if you also consider client/server techniques like CORBA and RMI even more):

1. Using the Java Native Interface (JNI) to use C/C++ code from shared libraries. This possiblitity is only available for programs with a very good defined programming interface or where the source code is available.
2. Using the pipe mechanism or standard input/output/error of programs to call external programs where the source code is not available. But the precondition for this mechanism is that the programs support the pipe mechanism (mostly only for linux programs)
3. The last and not very comfortable method is to use files to exchange data between JOELib and other programs. Fortran programs need this mechanism, even if the source code is available ! If anyone has a good solution for calling Fortran programs from Java i would be very interested !!!

For a simple example see the external process example:

`joelib.test.Title2DataTest` (joelib/src/joelib/test/Title2DataTest.java). This program needs the compiled linux version of `lib/linux/title2data.cpp` (joelib/lib/linux/title2data.cpp), which can be compiled with **g++ title2data.cpp -o title2data**

The binary file should be located in `joelib/lib/linux/`.

The windows version can be generated with the Visual C++ compiler.

Unfortunately there seems to be problems with JDK's pipe mechanism (JDK bug !) under Windows, therefore the example don't work properly under Windows. If you want to use the example anyway you

should not use the pipe mechanism. It would be better to write a little JNI wrapper to call external programs.

Input and output

Supported input and output formats

Supported file types can be defined in the `joelib.properties` (joelib/src/joelib.properties)-file in the `joelib/src` directory if you have a source distribution or in the `joelib/lib/joelib.jar:joelib.properties`-file if you have a binary distribution. If no represenation for an input/output-type is defined JOELib will try to load a default interpreting class which is defined in `joelib.io.IOTypeHolder` (joelib/src/joelib/io/IOTypeHolder.java). Here is an example definition:

Example 3-4. Definition of IO types

```
joelib.filetypes.1.name           = SDF
joelib.filetypes.1.representation = joelib.io.types.MDLSD
joelib.filetypes.2.name           = SMILES
joelib.filetypes.2.representation = joelib.io.types.Smiles
joelib.filetypes.3.name           = CTX
joelib.filetypes.4.name           = CML
joelib.filetypes.5.name           = POV
```

The defined input/output definitions can be easily accessed with two helper classes joelib.io.SimplerReader and joelib.io.SimplerWriter which should help you to understand the mechanism to get an suitable reader and writer object for your needs. Here is a list of supported input/output types:

Table 3-3. Supported file formats

Name	is readable	is writeable	Description
UNDEFINED	false	false	Undefined
BMP	false	true	Windows Bitmap (BMP) image
CML	true	true	Chemical Markup Language (CML)
CTX	true	true	CACTVS clear text format (CTX)
POV	false	true	Persistence Of Vision (POV) Ray Tracer
FLAT	true	true	Flat file format
Gaussian	true	true	Gaussian
GIF	false	true	CompuServe Graphics Interchange (GIF)

Name	is readable	is writeable	Description
			image
JPEG	false	true	JPEG image
JCAMP	false (but in progress)	false (but in progress)	Joint Commitee on Atomic and Molecular Physical Data
MOL2	true	true	Sybyl Mol2
MOLCONNZ	true	false	MolConnZ
MOPACOUT	true	false	MOPAC Output
PDB	false (but in progress)	true	Protein Data Bank
PDF	false	true	Portable Adobe Document Format (PDF)
PNG	false	true	Portable Network Graphics (PNG) image
PPM	false	true	Portable Pixelmap (PPM) image
PREP	true	false	Amber PREP
SDF	true	true	MDL SD file
SMILES	true	true	Simplified Molecular Input Line Entry System (SMILES)
TINKER	false	true	Tinker XYZ
XYZ	true	true	XYZ
ZIP	true	true	Compressed ZIP file format

XML respective Chemical Markup Language (CML)

XML respective Chemical Markup Language (CML) [mr99, mr01a, mr01b, wil01] (in progress)

For more informations have a look at http://www.xml-cml.org/. The CML output type can be defined in the `joelib.properties` (joelib/src/joelib.properties)-file:

```
###########
# CML
# version:            1.0 and 2.0
# ouput:              attributearray, array, large, huge
# delimiter:          if you comment this line, standard white space will be
used
```

```
# force.formalCharge: formal charges will be always written, even when they
are zero
# partialCharge:      write partial atom charge
# hydrogenCount:      write number of implicite+explicite hydrogens
############
## use slower memory saving preparser for avoiding to load the complete data
set into memory
## This flag will be automatically switched 'ON' for CML files in compressed
ZIP files !
## The basic convert does not need it, because it uses already another
sequential
## SAX reader (forced by a callback)
joelib.io.types.ChemicalMarkupLanguage.useSlowerMemorySavingPreparser=false
############
joelib.io.types.ChemicalMarkupLanguage.output.defaultVersion=2.0
joelib.io.types.ChemicalMarkupLanguage.defaultDelimiter=\u0020
#joelib.io.types.ChemicalMarkupLanguage.defaultDelimiter=|
joelib.io.types.ChemicalMarkupLanguage.output=huge
joelib.io.types.ChemicalMarkupLanguage.output.force.formalCharge=false
joelib.io.types.ChemicalMarkupLanguage.output.partialCharge=true
joelib.io.types.ChemicalMarkupLanguage.output.hydrogenCount=true
joelib.io.types.ChemicalMarkupLanguage.output.useNamespace=true
joelib.io.types.ChemicalMarkupLanguage.output.namespace=cml
joelib.io.types.ChemicalMarkupLanguage.output.xmlDeclaration=http://www.xml-
cml.org/schema/cml2/core
joelib.io.types.ChemicalMarkupLanguage.DTD.resourceDir=joelib/io/types/cml/da
ta/
############
## a first step to 'reproducable' descriptor calculation algorithms
joelib.io.types.ChemicalMarkupLanguage.output.storeChemistryKernelInfo=true
## these informations are not really a CML standard
############
joelib.io.types.ChemicalMarkupLanguage.output.symmetryInformations=false
############
```

CACTVS's clear text format (CTX)

CACTVS's clear text format (CTX) [gas95] http://www2.chemie.uni-erlangen.de/software/cactvs/index.html

Image writers (BMP, GIF, JPEG, PPM)

The image output properties can be defined in the `joelib.properties` (joelib/src/joelib.properties)-file:

```
# General image writer
joelib.gui.render.Mol2Image.defaultWidth=300
joelib.gui.render.Mol2Image.defaultHeight=200

# General 2D rendering options
joelib.gui.render.Renderer2DModel.bond.length=30.0
joelib.gui.render.Renderer2DModel.bond.distance=6.0
joelib.gui.render.Renderer2DModel.bond.width=2.0
```

```
joelib.gui.render.Renderer2DModel.drawNumbers=false
joelib.gui.render.Renderer2DModel.useKekuleStructure=false
joelib.gui.render.Renderer2DModel.showEndCarbons=true
joelib.gui.render.Renderer2DModel.atomColoring=false
joelib.gui.render.Renderer2DModel.orthoLineOffset=20
joelib.gui.render.Renderer2DModel.arrowOffset=10
joelib.gui.render.Renderer2DModel.arrowSize=5

joelib.gui.render.Renderer2DModel.background.color.r=255
joelib.gui.render.Renderer2DModel.background.color.g=255
joelib.gui.render.Renderer2DModel.background.color.b=255

joelib.gui.render.Renderer2DModel.foreground.color.r=0
joelib.gui.render.Renderer2DModel.foreground.color.g=0
joelib.gui.render.Renderer2DModel.foreground.color.b=0

joelib.gui.render.Renderer2DModel.highlight.color.r=255
joelib.gui.render.Renderer2DModel.highlight.color.g=0
joelib.gui.render.Renderer2DModel.highlight.color.b=0

joelib.gui.render.Renderer2DModel.number.color.r=0
joelib.gui.render.Renderer2DModel.number.color.g=0
joelib.gui.render.Renderer2DModel.number.color.b=255

joelib.gui.render.Renderer2DModel.conjugatedRing.color.r=0
joelib.gui.render.Renderer2DModel.conjugatedRing.color.g=0
joelib.gui.render.Renderer2DModel.conjugatedRing.color.b=0

joelib.gui.render.Renderer2DModel.arrow.color.r=0
joelib.gui.render.Renderer2DModel.arrow.color.g=255
joelib.gui.render.Renderer2DModel.arrow.color.b=0

joelib.gui.render.Renderer2DModel.orthogonalLine.color.r=0
joelib.gui.render.Renderer2DModel.orthogonalLine.color.g=0
joelib.gui.render.Renderer2DModel.orthogonalLine.color.b=255
```

Joint Commitee on Atomic and Molecular Physical Data (JCAMP)

Joint Commitee on Atomic and Molecular Physical Data (JCAMP) format [dl93, dw88, ghhjs91, lhdl94] (in progress)

Protein Data Base (PDB)

Protein Data Base (http://www.rcsb.org/pdb/docs/format/pdbguide2.2/guide2.2_frame.html)

The Protein Data Bank (PDB) is an archive of experimentally determined three-dimensional structures of biological macromolecules, serving a global community of researchers, educators, and students. The archives contain atomic coordinates, bibliographic citations, primary and secondary structure information, as well as crystallographic structure factors and NMR experimental data.

Portable Adobe Document Format (PDF)

The PDF output properties can be defined in the `joelib.properties` (joelib/src/joelib.properties)-file:

```
# PDF writer
joelib.io.types.PDF.fontSize=10
joelib.io.types.PDF.fontOffset=2
joelib.io.types.PDF.pageBorder=20
```

Additional the normal image writer properties must be set described in detail in the Section called *Image writers (BMP, GIF, JPEG, PPM)*.

Persistence Of Vision (POV) Ray Tracer

Persistence Of Vision (POV) Ray Tracer: http://www.povray.org/

Here some features:

- Three supported visualisation types: Spheres, Balls & Sticks, Sticks.
- Aromatic rings are visualized as torus elements. p orbitals can be also be visualized as simple lines. Please let me know if you have a good p orbital element under POVRay.
- You can use atom properties for atom coloring.

The PovRay output type can be defined in the `joelib.properties` (joelib/src/joelib.properties)-file:

```
# PovRay
# ouput type can be: stick, sphere, ball_and_stick
joelib.io.types.POVRay.output=ball_and_stick
joelib.io.types.POVRay.atomPropertyColoring=false
joelib.io.types.POVRay.atomProperty=Gasteiger_Marsili
```

Structured Data File (SDF)

Structured Data File (SDF) format or MDL molfile format [sdf] This is the mostly used molecule format in JOELib. Please let me know if you have further improving proposals. Golden rule: Empty lines are not allowed *in the data block* of MDL SD files, because an empty line is the signal for the end of an actual data entry. Because the internal data representation has this as precondition all empty lines in file format loaders which are contained in data entries must be converted to ?, 0.0 or whatever you want, except an empty line.

You can force JOELib to write kekulized molecular structures instead of molecular structures with assigned aromaticity, by using:

```
#SD Files
joelib.io.types.MDLSD.writeAromaticityAsConjugatedSystem=false
```

Simplified Molecular Input Line Entry System (SMILES)

Simplified Molecular Input Line Entry System (SMILES) [smiles, wei88, wei89].

The SMILES input/output-line-definition can be defined in the `joelib.properties` (joelib/src/joelib.properties)-file:

```
#SMILES
joelib.io.types.Smiles.canonical=false
joelib.io.types.Smiles.lineStructure=SMILES|TITLE
joelib.io.types.Smiles.lineStructure.delimiter=|
joelib.io.types.Smiles.lineStructure.input.delimiter=\u0020\t\n\r
joelib.io.types.Smiles.lineStructure.output.delimiter=\u0020
```

The canonical/unique SMILES representation of a molecule can be calculated, if the `canonical`-property entry is set to `true` (see the Section called *Morgan: Unique atom numbering* in Chapter 6).

Sybyl (MOL2)

Tripos Mol2 File Format (http://www.tripos.com/services/mol2/)

A mol2 file (.mol2) is a complete, portable representation of a SYBYL molecule. It is an ASCII file which contains all the information needed to reconstruct a SYBYL molecule.

Tinker (TINKER)

Tinker (http://dasher.wustl.edu/tinker/)

Some fortran programs like Tinker are very sensitive to Unix or Windows files, because the new line characters. Remember this if you have problems under Unix systems with files generated under Windows.

Writing your own import/export filter

You are missing file formats ? Write your own import and export filter! Use the MoleculeFileType class as abstract parent and fill the methods with functionality. The formatting of the input or output is pretty easy with the ScanfFormat and PrintfFormat classes from John E. Lloyd at:

http://www.cs.ubc.ca/~lloyd/java/doc/cformat.html

Internally atoms have special atom types, which were defined as SMARTS pattern in the joelib/data/plain/atomtype.txt (joelib/src/joelib/data/plain/atomtype.txt)-file. These types can be used for exporting easily to other file formats, especially force filed or ab inito programs. For the last task there is the joelib.data.JOETypeTable (joelib/src/joelib/data/JOETypeTable.java) helper class available which uses the default converting types in joelib/data/plain/types.txt (joelib/src/joelib/data/plain/types.txt).

Golden rule: Empty lines are not allowed in the data block of MDL SD files, because an empty line is the signal for the end of an actual data entry. Because the internal data representation has this as precondition all empty lines in file format loaders which are contained in data entries must be converted to ?, 0.0 or whatever you want, except an empty line.

Assigning atom types, aromatic flags, hybridization and hydrogens

Atom types can be assigned to atoms of a molecule using only topological informations and SMARTS substructure search. For more specialized atom types, like special chirality- and Z/E-isomerism-descriptors it would be a good choice to use atom property descriptors (see the Section called *Atom properties* in Chapter 5).

As already discussed in our three feature selection model building papers [wfz04a,wfz04b,fwz04] the descriptor calculation is the last step after calling four different expert systems, so you should be carefully check your descriptor results when predicting values with models not calculated on your own.

In our opinion for every expert system a 'standard' (e.g., JOELib/OpenBabel, atomTyperVersion=1.0), formulated as classification problem should exist, to be able to say simply: calculate descriptors for the already mentioned standard. The formulation as classification in a PUBLIC database is required to test your/our implemented atom typer against this standard. Let's see if we can ever find time and men-/women-power to formulate and test such a standard ...

Table 3-4. Process of assigning atom types

Molecule	→	→	→	→	→
	aromaticity	hybridization	implicite valence	atom types	descriptor
	SMARTS without				calculation
	`D<n>`				algorithm
	`^<n>`				

Assigning aromaticity flags

Aromatic flags can be assigned to atoms using SMARTS (see the Section called *SMARTS definition*) substructure patterns defined in the `joelib/data/plain/aromatic.txt` (joelib/src/joelib/data/plain/aromatic.txt)-file. All SMARTS patterns except `D<n>` (explicite bonds) and `^<n>` (hybridization) are allowed. Chiral atoms are allowed, which use the `XYZVector.calcTorsionAngle(...)` (joelib/src/joelib/math/XYZVector.java)-method.

Assigning atom hybridizations

To assign atom hybridizations it is necessary to have already assigned aromaticity flags. All `INTHYB`-definitions in the `joelib/data/plain/atomtype.txt` (joelib/src/joelib/data/plain/atomtype.txt)-file are used get the atom hybridizations.

Assigning atom types

To assign atom types it is necessary to have already assigned aromaticity flags and atom hybridizations. All `EXTTYP`-definitions in the `joelib/data/plain/atomtype.txt` (joelib/src/joelib/data/plain/atomtype.txt)-file are used get the atom types. These are mainly used for the file conversion process and for descriptor calculation algorithms.

Assigning implicite hydrogens

To assign the implicite valence to atoms it is necessary to have already assigned aromaticity flags, atom hybridizations and atom types. All `IMPVAL`-definitions in the `joelib/data/plain/atomtype.txt` (joelib/src/joelib/data/plain/atomtype.txt)-file are used to calculate the number of implicite hydrogens for each atom.

Calculate descriptors and/or assign special atomtypes

Descriptors can be simple topology descriptors without requiring any chemical informations or descriptors with requiring atom types and implicite hydrogens (see e.g. the Section called *Fingerprints* in Chapter 5). PATTY rules (see the Section called *Programmable Atom Typer (PATTY)*) can be used for simple atom type descriptors. And all kinds of other models or expert rules can be used for chirality or Z/E-isomerism descriptors.

Table 3-5. Possible special atom type assignments (not implemented)

Assignment	Reference
chirality descriptor	[gt03]
E/Z descriptor	[gbt02]
planar three-coordinate nitrogen	calculate vector product of the three neighbors
nitrogen with aromatic ligand	PATTY SMARTS rule: `[#7a]`

For all atom property descriptors there must always exist a descriptor documentation-file (see the Section called *Writing your own descriptor and result classes*). Otherwise a HTML-documentation (generated by using DocBook (http://www.docbook.org/)) error will occur every time JOELib starts. The XML- and RTF-description files are optional.

Chapter 4. Used utilities

Development

Utilities used for faciliating development and avoiding to invent the wheel twice.

- A memory efficient data structure to store flags for atoms and bonds; `joelib.util.JOEBitVec` (joelib/src/joelib/util/JOEBitVec.java) mainly derived from `java.util.BitSet`. It's also used for SMARTS substructure search and ring search.
- Comfortable logging framework with Log4J from http://jakarta.apache.org/log4j/docs/index.html
- Well known scanff and printf formating classes from John E. Lloyd at http://www.cs.ubc.ca/\~lloyd/java/doc/cformat.html

  Here a slightly modified version of the cformat.ScanfReader (joelib/src/cformat/ScanfReader.java) class is used, to take white spaces at the beginning of lines into account. To activate this modification call the `useCstandard(false)`-method.
- Good matrix operation classes with Jama from http://math.nist.gov/javanumerics/jama/
- Chemical Markup Language (CML) support based on the Chemical Data Object Programming Interface (CDOPI) (used as kernel for JOELib) http://www.openscience.org/~egonw/cdopi/.We extended the basic version for accepting descriptors and CML2 support.
- Scientific Graphics Toolkit (SGT) http://www.epic.noaa.gov/java/sgt/
- ACME Labs Java utilities for image creation (GIF, PPM) http://www.acme.com/java/
- JpegEncoder of James R. Weeks and BioElectroMech. for image creation (JPEG)
- BMPEncoder of Jean-Pierre Dube, Christian Ribeaud, and Egon Willighagen for image creation (BMP)

Maintenance

These libraries are mainly used for documentation purposes and software design (faciliating the maintenance of JOELib). Unfortunately some required software design recommended complexity reductions can not be done, because there is no time (for me!) to implement testing classes and redesign the library structure. Furthermore some speed-up's can not be done, too, e.g. H-depleted molecule caching for descriptor calculations, or improved SMARTS matching with faster search trees. So we will need more time and more developers to be able to gurantee a state-of-the-art softwaredesign. Otherwise you will be forced, until now, to read the available API and the (complex) source code on your own !

- The `Jalopy` (http://jalopy.sourceforge.net/)-library to format the source code and reduce CVS traffic when using different IDE's.
- The `Pretty-Mess-Detection (PMD)` (http://pmd.sourceforge.net/)-library to check code complexity and coding conventions.
- The `JavaNCSS` (http://www.kclee.com/clemens/java/javancss/)-library to check for duplicated code.

- The `VizANT` (http://vizant.sourceforge.net/)-library to visualize the dependencies in the ANT build file.
- The `FindBugs` (http://www.cs.umd.edu/~pugh/java/bugs/)-library to check for potential bugs in the JOELib library.

Chapter 5. Descriptors

Native values

The native values like simple atom/group counts and complexity measures are suitable to build primitive QSAR prediction models for structure property relations, which has a good correlation to the molecular complexity. for more biological relevant models we recommend to use transformation based descriptors (the Section called *Transformations*) based on atom properties (the Section called *Atom properties*).

Number of acidic groups

Number of acidic groups.

Number of aliphatic hydroxy groups

Number of aliphatic hydroxy groups.

Number of aromatic bonds

Number of aromatic bonds.

Number of basic groups

Number of basic groups.

Fraction of rotatable bonds

Fraction of rotatable bonds.

Geometrical diameter

Geometrical diameter.

Geometrical radius

Geometrical radius.

Geometrical shape coefficient

Geometrical shape coefficient.

Graph shape coefficient

Graph shape coefficient.

Number of Hydrogen Bond Acceptors (HBA) 1

Number of Hydrogen Bond Acceptors (HBA) 1. The default SMARTS pattern is: [$([!#6;+0]);!$([F,Cl,Br,I]);!$([o,s,nX3]);!$([Nv5,Pv5,Sv4,Sv6])].

Number of Hydrogen Bond Acceptors (HBA) 2

Number of Hydrogen Bond Acceptors (HBA) 2. The default SMARTS pattern is: [$([$([#8,#16]);!$(*=N~O);!$(*~N=O);X1,X2]),$([#7;v3;!$([nH]);!$(*(-a)-a)])].

Number of Hydrogen Bond Donors (HBD) 1

Number of Hydrogen Bond Donors (HBD) 1. The default SMARTS pattern is: [!#6;!H0].

Number of Hydrogen Bond Donors (HBD) 2

Number of Hydrogen Bond Donors (HBD) 2. The default SMARTS pattern is: [$([O;H1,-&!$(*-N=O)]),$([S;H1&X2,-&X1]),$([#7;H;!$(*(S(=O)=O)C(F)(F)F);!$(n1nnnc1);!$(n1nncn1)]),$([#7;-])].

Number of heavy bonds

Number of heavy bonds.

Number of heterocycles

Number of heterocycles.

Number of hydrophobic groups

Number of hydrophobic groups.

Kier Shape 1

The Kier shape $^{1}\kappa$ descriptor [tc00] can be defined as :

Equation 5-1. Kier shape 1

$$^{1}\kappa = \frac{2(\mathbf{P}_{max}\mathbf{P}_{min})}{\mathbf{P1}^{2}}$$

$\mathbf{P}_{max}$ is the maximum number of paths of length 1. $\mathbf{P}_{min}$ is the minimum number of paths of length 1 in a molecule graph of same atom number. **P1** is the number of paths in the given molecule of length 1. or:

Equation 5-2. Kier shape 1 (alternative formulation)

$$^{1}\kappa = \frac{\mathbf{A}(\mathbf{A}-1)^{2}}{\mathbf{P1}^{2}}$$

A means the number of nodes in the molecules graph. **P1** is the number of paths of length 1 in the given molecule.

Kier shape 2

The Kier shape $^{2}\kappa$ descriptor [tc00] can be defined as :

Equation 5-3. Kier shape 2

$$^{2}\kappa = \frac{(A-1)(A-2)^{2}}{P2^{2}}$$

A means the number of nodes in the molecules graph. **P2** is the number of paths of length 2 in the given molecule.

Kier shape 3

The Kier shape $^{3}\kappa$ descriptor [tc00] can be defined as :

Equation 5-4. Kier shape 3

$$^{3}\kappa = \begin{cases} \frac{(A-3)(A-2)^{2}}{P3^{2}} & \text{for even } A\ (A > 3) \\ \frac{(A-1)(A-3)^{2}}{P3^{2}} & \text{for odd } A\ (A > 3) \end{cases}$$

A means the number of nodes in the molecules graph. **P3** is the number of paths of length 3 in the given molecule.

Octanol/Water partition coefficient (logP)

Octanol/Water partition coefficient (logP) or hydrophobicity [wc99].

Molar refractivity (MR)

Molar refractivity (MR) [wc99].

Molecular weight (MW)

Molecular weight (MW).

Number of atoms

Number of atoms.

Number of boron atoms

Number of boron atoms.

Number of bromine atoms

Number of bromine atoms.

Number of bonds

Number of bonds.

Number of chlorine atoms

Number of chlorine atoms.

Number of halogen atoms

Number of halogen atoms.

Number of iodine atoms

Number of iodine atoms.

Number of fluorine atoms

Number of fluorine atoms.

Number of nitrogen atoms

Number of nitrogen atoms.

Number of oxygen atoms

Number of oxygen atoms.

Number of phosphorus atoms

Number of phosphorus atoms.

Number of sulfur atoms

Number of sulfur atoms.

Number of -NO_2 groups

Number of -NO_2 groups.

Number of -OSO atoms

Number of -OSO atoms.

Polar surface area (PSA)

Polar surface area (PSA) [ers00].

Number of rotatable bonds

Number of rotatable bonds, where the atoms are heavy atoms with bond order one and a hybridization which is not one (no sp). Additionally the bond is a non-ring-bond.

Number of -SO groups

Number of -SO groups.

Number of -SO_2 atoms

Number of -SO_2 atoms.

Topological diameter

Topological diameter.

Topological radius

Topological radius.

Zagreb index 1

Zagreb index 1 [tc00].

Equation 5-5. Zagreb index 1

$$M_1 = \sum_a \delta_a^2$$

where **a** are all atoms of the hydrogen depleted graph and δ is the vertex degree.

Zagreb index 2

Zagreb index 2 [tc00].

Equation 5-6. Zagreb index 2

$$M_2 = \sum_b (\delta_i \delta_j)_b$$

where **b** are all bonds of the hydrogen depleted graph and δ is the vertex degree. **i** and **j** are the atom indices of the atoms connected to the bond **b**.

Atom properties

The formal definition of atom properties is given in chapter Chapter 2.

Atom in acceptor

Is this atom a hydrogen acceptor atom $\boldsymbol{\alpha_{acceptor}(G,V,A_{chem}|JOELib,kernelID)}$. We used here the interpretation of Böhm/Klebe [bk02].

Atom in conjugated environment

Atom in conjugated environment $\boldsymbol{\alpha_{conj}(G,V,A_{chem}|JOELib,kernelID)}$ [wfz04a,wfz04b,fwz04].

Table 5-1. SMARTS definitions for assigning the conjugated atom property flag

SMARTS	Description
a	Aromatic atoms
=,#-,=*=,#*	All butadien analogues
[N,P,O,S]=,#*-[*;!H0]	alpha, beta unsaturated (pi effect)
=,#-[F,Cl,Br,I]	alpha, beta unsaturated (sigma effect)
=,#-[N,P,O,S;!H0]	alpha, beta unsaturated (pi effect, tautomer)

Atom in donor or acceptor

Is this atom a hydrogen donor or acceptor atom $\boldsymbol{\alpha_{don,acc}(G,V,A_{chem}|JOELib,kernelID)}$. We used here the interpretation of Böhm/Klebe [bk02].

Atom in donor

Is this atom a hydrogen donor atom $\boldsymbol{\alpha_{donor}(G,V,A_{chem}|JOELib,kernelID)}$. We used here the interpretation of Böhm/Klebe [bk02].

Atom in ring

Is this atom a ring atom $\boldsymbol{\alpha_{inRing}(G,V,A_{chem}|JOELib,kernelID)}$.

Atom is terminal carbon

Is this atom a terminal carbon atom $\boldsymbol{\alpha_{methylen}(G,V,A_{chem}|JOELib,kernelID)}$.

Atom is negative

Is this atom a negative charged atom $\boldsymbol{\alpha_{negative}(G,V,A_{chem}|JOELib,kernelID)}$.

Atom is positive

Is this atom a positive charged atom $\boldsymbol{\alpha_{positive}(G,V,A_{chem}|JOELib,kernelID)}$.

Atom masss

Atom mass.

Valence

Valence.

Van der Waals volume

Van der Waals volume.

Conjugated electrotopological state

Conjugated electrotopological state [wz03,wfz04b].

Equation 5-7. Conjugated electrotopological state

$$\mathbf{CETS_i} = \mathbf{I_i} + \Delta\mathbf{I_{i,top,conj}} = \mathbf{I_i} + \sum_{j=1}^{A} \frac{\mathbf{I_i} - \mathbf{I_j}}{(d_{ij,top}+1)^{k/CTD_i}}$$

$\mathbf{I_i}$ is the intrinsic state of atom **i** (the Section called *Intrinsic topological state*) and **k** the distance influence. The distance influence is reduced by the conjugated topological distance

Equation 5-8. Conjugated topological distance

$$\mathbf{CTD_i} = \max(d_{ij,top}|\mathbf{C_i})\forall j$$

where $\mathbf{C_i}$ is the conjugated atom **i** (the Section called *Atom in conjugated environment*) of the molecule.

Electrogeometrical state

Electrogeometrical state $\boldsymbol{\alpha_{EGstate}}$**(G,V,**$\mathbf{A_{chem}}$**|JOELib,kernelID)** [wz03,wfz04b].

Equation 5-9. Electrogeometrical state

$$\mathbf{EGS_i} = \mathbf{I_i} + \Delta\mathbf{I_{i,geom}} = \mathbf{I_i} + \sum_{j=1}^{A} \frac{\mathbf{I_i} - \mathbf{I_j}}{(d_{ij,geom}+1)^k}$$

$\mathbf{I_i}$ is the intrinsic state of atom **i** (the Section called *Intrinsic topological state*) and **k** the distance influence.

Electron affinity

Electron affinity.

Electronegativity after Pauling

Electronegativity after Pauling.

Electrotopological state

Electrotopological state $\boldsymbol{\alpha_{Estate}}$**(G,V,**$\mathbf{A_{chem}}$**|JOELib,kernelID)** [tc00,wfz04b].

Equation 5-10. Electrotopological state

$$\mathbf{ETS_i} = \mathbf{I_i} + \Delta\mathbf{I_{i,top}} = \mathbf{I_i} + \sum_{j=1}^{A} \frac{\mathbf{I_i} - \mathbf{I_j}}{(d_{ij,top}+1)^k}$$

$\mathbf{I_i}$ is the intrinsic state of atom **i** (the Section called *Intrinsic topological state*) and **k** the distance influence.

Gasteiger-Marsili

Calculation of atom partial charges (Gasteiger-Marsili) α_{GM}**(G,V,A_{chem}|JOELib,kernelID)** [gm78]. The Partial Equalization of Orbital Electronegativities (PEOE) sigma charges of atoms can be calculated using an iterative algorithm. There exists a dependency between the electronegativity $\chi_{i,A}$ and the charge q_A of an atom:

Equation 5-11. Orbital electronegativity based on atom charge

$$\chi_{i,A} = a_i + b_i q_A + c_i q_A^2$$

where **i** is the orbital and **A** the atom.

The algorithm requires the polynomial coefficients a_i, b_i, c_i the damping factor α and the maximal number of iterations N_{max}. Because the amount of charge transferred at each iteration is damped with an exponentially decreasing factor the convergence is guaranteed.

Graph potentials

Graph potentials α_{GP}**(G,V,A_{chem}|JOELib,kernelID)** [wy96] or rotational symmetry descriptor. Only the connection table is needed to calculate the external rotational symmetry, or topological equivalent atoms.

Equation 5-12. Graph potentials

$$\begin{aligned} G_{ij} &= \begin{cases} 1+v_i & \text{if i=j} \\ -1 & \text{atom}_i \text{ is connected to atom}_j \\ 0 & \text{otherwise} \end{cases} \\ \vec{P} &= G^{-1}\vec{v} \end{aligned}$$

where v_i is the valence (vertex degree) of the atom **i**.

Intrinsic topological state

Intrinsic topological state α_{Istate}**(G,V,A_{chem}|JOELib,kernelID)** [tc00,wfz04b].

Equation 5-13. Intrinsic topological state

$$I_i = \frac{(2/L_i)^2 \delta_i^v + 1}{\delta_i}$$

where L_i is the principal quantum number, δ_i^v is the number of valence electrons, and δ_i is the number of sigma electrons of the **i**th atom a_i.

Fingerprints

Pharmacophore fingerprint

Pharmacophore fingerprint [gxsb00].

Table 5-2. Pharmacophore fingerprint definition

Bit position	Set when
1	Fraction of rotatable bonds > 0.0
2	Fraction of rotatable bonds > 0.1
3	Fraction of rotatable bonds > 0.2
4	Fraction of rotatable bonds > 0.3
5	Fraction of rotatable bonds > 0.4
6	Aromatic bonds in molecule > 2
7	Aromatic bonds in molecule > 8
8	Aromatic bonds in molecule > 16
9	Aromatic bonds in molecule > 20
10	Aromatic bonds in molecule > 26
11	Aromatic bonds in molecule > 32
12	Aromatic bonds in molecule > 38
13	Molecule contains a heterocycle
14	Molecule contains an aromatic hydroxy group
15	Molecule contains an aliphatic hydroxy group
16	Molecule contains an aliphatic secondary amine
17	Molecule contains an aliphatic tertiary amine
18	Molecule contains a phenyl ring
19	Molecule contains a ring containing nitrogen
20	Molecule contains a $-SO_2$ group
21	Molecule contains a -SO
22	Molecule contains an ester
23	Molecule contains an amide
24	Molecule contains a 5-membered non-aromatic ring
25	Molecule contains a 5-membered aromatic ring
26	Molecule contains a 9-membered or larger (fused) ring
27	Molecule contains a fused ring system

Bit position	Set when
28	Molecule contains a fused aromatic ring system
29	Molecule contains a -OSO group
30	Molecule contains a halogen atom
31	Molecule contains a nitrogen atom attached to alpha-carbon of aromatic system
32	Molecule contains a $-NO_2$ group
33	Molecule contains rings separated by 2-3 non-ring atoms
34	Molecule contains rings separated by 4-5 non-ring atoms
35	Molecule contains a NN group
36	Molecule contains a carbon atom attached to 3 carbons and a hetero atom
37	Molecule contains an oxygen atom separated by 2 atoms
38	Molecule contains a methyl group attached to hetero atom
39	Molecule contains a double bond
40	Molecule contains a non-H atom linked to 3 heteroatoms
41	Molecule contains a quaternary atom
42	Molecule contains 2 methylenes separated by 2 atoms
43	Molecule contains a non-ring oxygen atom attached to aromatic system
44	Molecule contains 2 non-C,H atoms separated by 2 atoms
45	Number of hydrogen bond acceptors >= 1
46	Number of hydrogen bond acceptors >= 2
47	Number of hydrogen bond acceptors >= 3
48	Number of hydrogen bond acceptors >= 4
49	Number of hydrogen bond acceptors >= 5
50	Number of hydrogen bond acceptors >= 6

Bit position	Set when
51	Number of hydrogen bond acceptors >= 7
52	Number of hydrogen bond acceptors >= 8
53	Number of hydrogen bond acceptors >= 9
54	Number of hydrogen bond acceptors >= 10

Transformations

The descriptors obtained for the autocorrelation, BCUT and GTCI should be used carefully for QSAR models, because they are limited to the smallest molecule or more exactly the smallest topological distance in the molecule. Although in these specail cases missing values can be filled with zero values, because there is no probability for large distances in 'small' lead molecules. Nonetheless this is not a good strategy, because you will not get really good models, if you have a lot of zero values, which is obviously if you have a closer look at the data sets. Or in other words, you will get a potential 'degeneracy problem' for such molecules.

Moreau-Broto topological autocorrelation

Moreau-Broto topological autocorrelation [bmv84].

Equation 5-14. Moreau-Broto autocorrelation

$$\delta_{ij} = \begin{cases} 1 & \text{if } d_{ij} = d \\ 0 & \text{otherwise} \end{cases}$$

$$AC_d = \sum_{i=1}^{A} \sum_{j=1}^{A} \delta_{ij} w_i w_j$$

where $\mathbf{d}_{ij}$ is the topological distance between the atoms **i** and atom **j**, $\mathbf{w}_i$ and $\mathbf{w}_j$ are the atom properties of the atoms **i** and **j**.

It must be mentioned that the autocorrelation is only a special case of the radial distribution function (the Section called *Radial distribution function (RDF)*).

Burden modified eigenvalues

Burden modified eigenvalues [tc00].

Global topological charge

Global topological charge [tc00].

Radial distribution function (RDF)

The radial distribution function (RDF) [msg99,wfz04b] can be interpreted as the probability distribution of finding an atom in a spherical volume of radius **r**.

Equation 5-15. Radial distribution function

$$g(r) = \frac{1}{2}\sum_{i,j,i\neq j}^{A} p_i p_j e^{-Bd^2}$$

$$AC(d) = g(r)_{\lim B\to\infty} = \sum_{i,j,i\neq j} A(p_i p_j)_d \delta_{ij} \text{ with } \delta_{ij} = \begin{cases} 1 & \text{if } d(a_i,a_j) = d \\ 0 & \text{otherwise} \end{cases}$$

where $\mathbf{r}_{ij}$ is the geometrical distance between the atoms **i** and atom **j**, $\mathbf{w}_i$ and $\mathbf{w}_j$ are the atom properties of the atoms **i** and **j**. **B** is the smoothing parameter (fuzziness of the distance $\mathbf{r}_{ij}$) for the interatomic distance and **f** the scaling factor.

If **B** aims to infinity the RDF code approximates to the autocorrelation function (the Section called *Moreau-Broto topological autocorrelation*) and the fuzziness of the distance $\mathbf{r}_{ij}$ vanishes. So the RDF code can be treated as a generalized autocorrelation function.

The RDF user parameters can be defined in the `joelib.properties`-file, otherwise the default parameters will be used:

```
jcompchem.joelib.desc.types.RadialDistributionFunction.minSphericalVolume =
0.2
joelib.desc.types.RadialDistributionFunction.maxSphericalVolume = 10.0
joelib.desc.types.RadialDistributionFunction.sphericalVolumeResolution = 0.2
joelib.desc.types.RadialDistributionFunction.smoothingFactor = 25
```

Optional the RDF can be calculated with protonated molecules, but you must be sure that all available atom properties are calculated with hydrogens also. Because this is not the standard, this option should be only used by developers.

Chapter 6. Algorithms

Breadth First Search (BFS)

The BFS method performs a breadth-first search [clr98] of a graph. A breadth-first search visits vertices that are closer to the source before visiting vertices that are further away. In this context `distance' is defined as the number of edges in the shortest path from the source vertex.

Figure 6-1. Pseudocode for the BFS algorithm

```
paint all vertices white;
paint the source grey, set its distance to 0 and enqueue it;
repeat
  dequeue vertex v;
  if v is the target, we're done - exit this algorithm;
  paint v black;
  for each white neighbor w of v
        paint w grey;
        set distance w to (distance v + 1);
        set parent w to v;
        enqueue w
until the queue is empty
if we haven't yet exited, we didn't find the target
```

The time complexity is $\mathbf{O(E+V)}$ [clr98].

Depth First Search (DFS)

The DFS method performs a depth--first search [clr98] of a graph. A depth--first search visits vertices that are further to the source before visiting vertices that are closer away. In this context `distance' is defined as the number of edges in the shortest path from the source vertex.

Figure 6-2. Pseudocode for the DFS algorithm

```
DFS(G)
  {
    For each v in V,
    {
      color[v]=white;
      pred[u]=NULL
    }

    time=0;
    For each u in V
      If (color[u]=white) DFSVISIT(u)
  }


  DFSVISIT(u)
```

```
{
  color[u]=gray;
  d[u] = ++time;

  For each v in Adj(u)  do
    If (color[v] = white)
    {
      pred[v] = u;
      DFSVISIT(v);
    }

  color[u] = black; f[u]=++time;
}
```

The time complexity is $O(E+V)$ [clr98].

Topological distance matrix

Calculates the topological distances between all atom pairs. Here a simple Breadth First Search (BFS the Section called *Breadth First Search (BFS)*) is used to calculate these distances, which causes a running time of $O(A^3)$, where **A** is the number of atoms.

Geometrical distance matrix

The geometrical distance matrix calculates the euklidian distance between the 3D coordinates of all atom pairs.

Morgan: Unique atom numbering

Algorithm to get a unique numbering for molecules (graphs) [mor65].

Figure 6-3. Pseudocode for the Morgan labeling algorithm

```
label each atom with its degree;
labels=count the number of different labels;
hasNTchanged=5;
for all time
  label each atom with sum of label+all neighbor labels;
  actLabels=count the number of different labels;
  if actLabels equal labels then
    decrement hasNTchanged;
    if hasNTchanged is zero break loop;
  fi
rof
```

The sloppy breaking criteria is necessary, because it's possible that the number of different labels can be constant for only two iterations. But that's not so interesting, let's continue with the renumbering part of the Morgan algorithm. As you can see, it's possible, that 'symmetric' atoms in the molecule will have same labels. Is there now a possibility to solve these 'labeling/renumbering' ties ? Yes, additional

informations, like bond order and element number can be used for resolving renumbering ties or the suggested Jochum-Gasteiger canonical renumbering [tc00] informations can be used.

Figure 6-4. Pseudocode for the Morgan renumbering algorithm

```
calculate the morgan atom labels;
start breadth first search from this atom;
choose node with the highest label and set new atom index to 1;
repeat
  build deque i of atoms with same BFS traversing number i;
  if deque i contains no equal labels
    renumber atoms in order of decreasing atom labels.
  fi
  else
    try to resolve renumbering tie for the equal labels:
      1. prefer atom with higher bond order for renumbering
      2. prefer atom with higher element number for renumbering
      3. ...
    if tie solved
      renumber atoms in order of decreasing atom labels.
    fi
    else
      show renumbering tie warning;
    esle
  esle
  increment i;
until all atoms are numbered
```

The uniquely renumbered molecule can be used to calculate molecule hashcodes and canonical/unique SMILES representations (see the Section called *Simplified Molecular Input Line Entry System (SMILES)* in Chapter 3).

Chapter 7. Interfaces to other libraries

abc.

Interfaces to Java libraries

Chemical Development Kit (CDK)

The most relevant interface methods for the Chemical Development Kit (CDK) (http://sourceforge.net/projects/cdk) library are available in the `joelib/util/cdk/CDKTools.java` (joelib/src/joelib/util/cdk/CDKTools.java) helper class.

Weka data mining library

The most relevant interface methods for the Weka data mining library (http://www.cs.waikato.ac.nz/ml/weka/) library are available in the `joelib/algo/weka/InstancesHelper.java` (joelib/src/joelib/algo/weka/InstancesHelper.java) helper class. For using Weka you should modify the constructors in weka.core.Attributes from 'protected' to 'public' and use the recompiled Weka sources.

Interfaces to C/C++ libraries using the Java native interface (JNI)

LibGhemical

The most relevant interface methods for the LibGhemical (http://bioinformatics.org) interface library are available in the `joelib/util/ghemical/GhemicalInterface.java` (joelib/src/joelib/util/ghemical/GhemicalInterface.java) helper class.

Matlab interface

JOELib can directly be used under Matlab (the GUI is Java based, too) or vice versa using the JNI interface JMatLink (http://www.held-mueller.de/JMatLink/).

Chapter 8. Documentation

DocBook

This tutorial was generated by using the XML based DocBook DTD (http://www.oasis-open.org/docbook/).

You will need Jade (http://www.jclark.com/jade/), the DocBook 4.2 (http://www.oasis-open.org/docbook/sgml/4.2/) definition files and finally the ISO Entities (http://www.oasis-open.org/cover/ISOEnts.zip) for generating a HTML and RTF output from the SGML(XML) files.

`.cshrc`-settings for Unix systems

```
# DocBook
setenv DB_HTML /usr/share/sgml/docbook/docbook-dsssl-stylesheets-
1.72/html/docbook.dsl
setenv DB_PRINT /usr/share/sgml/docbook/docbook-dsssl-stylesheets-
1.72/print/docbook.dsl
setenv SGML_CATALOG_FILES
/usr/share/sgml/docbook_4/docbook.cat:/usr/share/sgml/jade_dsl/catalog
setenv JADE "jade"
setenv CONVERT "convert"
```

`.tcshrc`-settings for Windows systems with cygwin

```
# DocBook
setenv DB_HTML "C:/cygwin/usr/share/sgml/docbook/db177/html/docbook.dsl"
setenv DB_PRINT "C:/cygwin/usr/share/sgml/docbook/db177/print/docbook.dsl"
setenv SGML_CATALOG_FILES
"C:/cygwin/usr/share/sgml/docbook/docbook41/docbook.cat;\
                  C:/cygwin/usr/share/sgml/docbook/db177/common/catalog;\
                  C:/cygwin/usr/share/sgml/docbook/jade121/catalog"
setenv JADE "C:/cygwin/usr/share/sgml/docbook/jade121/jade"
setenv CONVERT "convert"
```

The environment settings can be checked by calling

```
${JADE} -t sgml -d $DB_HTML JOELibTutorial.xml
${JADE} -t rtf -d $DB_PRINT JOELibTutorial.xml
```

from the `docs/docbook` directory.

DocBook and mathematical equations

A well known 'problem' or a definitively missing feature of DocBook is the generation of mathematical equations. We used a workaraound for Windows (Cygwin) and Linux systems. Mathematical equations can be generated by using LaTeX expressions, which will be converted from Enculapsed PostScript (EPS) files to GIF files by using ImageMagick (http://www.imagemagick.org/)'s convert method. This seems at the moment one of the best solutions, because the LaTeX expressions are well known and the

conversion method is well established. To faciliate the generation of the tutorial the shell scripts will be called from the ANT Makefile.

All LaTeX equations are defined directly in the SGML files by special LATEXEQUATION entries (quoted by in SGML comments). These entries will be parsed and the .tex files and image files will be created automatically.

DEPRECATED SOLUTION:

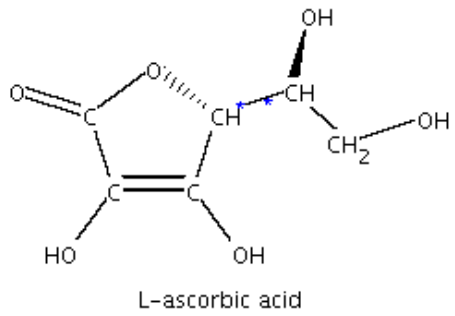
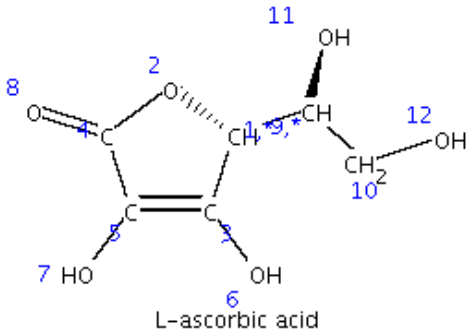
All equation definitions must be defined in the `docs/docbook/formulas` directory. All found `*.tex` files will be converted automatically to GIF images by calling the `createImages.sh` (joelib/docs/docbook/formulas/descriptors/createImages.sh) script.

Any equation in DocBook should use these images. If you will link these images from a subdirectory you must carfully check the correct links for the generated SGML or RTF output files. The image files will be copied to the correct directories using the ANT Makefile mechanism.

DocBook and molecular structures

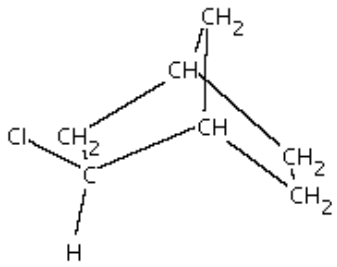
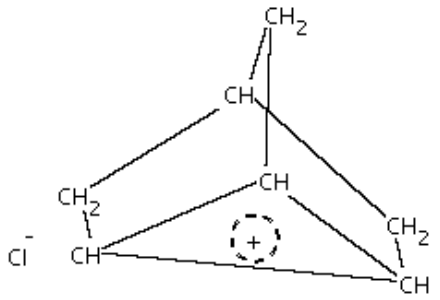
Show stereo bonds and chiral atoms by assigning atom labels.

Table 8-1. Molecular structures with atom numbers and special atom labels

Molecular structure	Options
 L-ascorbic acid	<!-- MOLECULARSTRUCTURE structures/l_ascorbic_acid.mol structures/l_ascorbic_acid: width=300 height=200 rotate=270 labels=9=*;1=*-->
 L-ascorbic acid	<!-- MOLECULARSTRUCTURE structures/l_ascorbic_acid.mol structures/l_ascorbic_acid_numbers: width=300 height=200 shownumbers rotate=270 labels=9=*;1=*-->

A norcamphan derivate can be used for substitution reactionS by obtaining a norbornyl cation with delocalised three ring with a cationic charge.

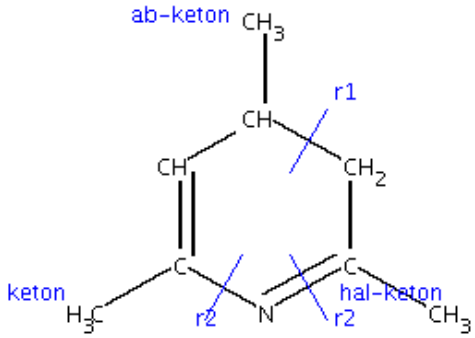
Table 8-2. Molecular structures with a delocalised ring

Molecular structure	Options
 exo-norcamphan	<!-- MOLECULARSTRUCTURE structures/exo_norcamphan.mol structures/exo_norcamphan: width=300 height=200 rotate=270-->
 norbornyl cation (SN,intermediate product)	<!-- MOLECULARSTRUCTURE structures/exo_norcamphan_norbonyl_cation.mol structures/exo_norcamphan_norbonyl_cation: width=300 height=250 rotate=270 conjRing=2,3,4,c+ -->

Retrosynthetic cuts for the michael addition (r1) and imin bond closure (r2) for the kroehnke pyridin synthesis.

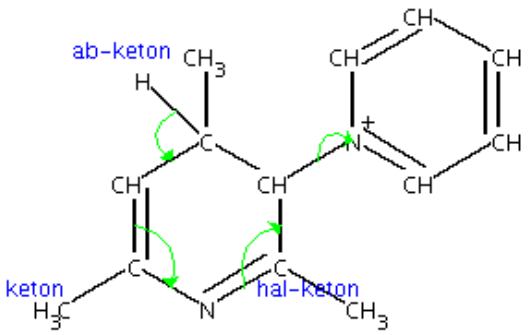
Table 8-3. Molecular structures with retrosynthetic bond splittings

Molecular structure	Options
	<!-- MOLECULARSTRUCTURE structures/pyridin_kroehnke.mol structures/pyridin_kroehnke: width=300 height=200 rotate=270 hideEndCarbon orthoLines=1,l,sr2,6;5,r,sr2,6;2,r,sr1,3 labels=8=hal-keton;7=ab-keton;9=keton-->

Molecular structure	Options
	

Last step of the kroehnke pyridin synthesis by creating the aromatic ring and splitting the positive pyridinium ring we used to activate the nucleophilic character of the alpha-keton position.

Table 8-4. Molecular structures with electron transfer arrows

Molecular structure	Options
	<!-- MOLECULARSTRUCTURE structures/pyridin_kroehnke_arom.mol structures/pyridin_kroehnke_arom: width=300 height=270 rotate=270 hideEndCarbon arrows=3,16,l,3,4;4,5,r,5,6;1,6,r,1,2;7,2,r,7 labels=13=ab-keton;14=hal-keton;15=keton-->

API documentation

The JavaDOC generated API documentation is available in the docs/api (../../api/index.html)-directory.

To be more developer friendly there were some special taglets available in this project.

- `@author`: Predefined authors (e.g., `wegnerj`) will be expanded to full names including E-Mail address.
- `@license`: Predefined license models (e.g. `GPL`) will be expanded to full license model name including a link to the web page of this license model.

- `@cvsversion`: The CVS version line `$Revision: 1.3 $, $Date: 2003/10/16 08:49:27 $` will be formatted and a link to the source code (CVS head) in the repository will be set. In my opinion this is one of the nicest features i've ever seen.
- `@cite`: Literature references (e.g. `clr98complexity`) can be used to reference articles, web pages or other BibTeX entries to describe algorithms and standards more tranparently and seriously. The HTML file was generated from a BibTeX file by using the tth (http://hutchinson.belmont.ma.us/tth/) tool.

Chapter 9. JOELib examples and code snippets

Molecule

Here we will present some code snippets for handling basic molecular operations.

Molecules

Load and store molecules

For loading molecules we can load them sequentially from molecules files or all can be loaded at once into memory, which is only recommended for smaller data sets or when you want process a (sub)set of molecules very often.

Example 9-1. Load molecules from large files sequentially

```
// open file input stream
String inputFile="yourInputFile.sdf";
FileInputStream input = null;
try
{
  input = new FileInputStream(inputFile);
}
catch (Exception ex)
{
  ex.printStackTrace();
  // TODO: handle exception
}
// create simple reader and
// estimate file format by file extension
SimpleReader reader = null;
try
{
  // estimate input file type
  IOType inType = IOTypeHolder.instance().filenameToType(inputFile);
  if (inType == null)
  {
    // TODO: handle unkown input file type
  }
  // open simple reader
  reader = new SimpleReader(input, inType);
}
catch (IOException e)
{
  e.printStackTrace();
  // TODO: handle exception
}

// load molecules sequentially
```

```
// set output type to input type
JOEMol mol = new JOEMol(inType, inType);
for (;;)
{
  try
  {
    if (!reader.readNext(mol))
    {
      // process all molecules until
      // they are all processed
      break;
    }
  }
  catch (IOException ex)
  {
    // TODO: handle input/output exception
  }
  catch (MoleculeIOException ex)
  {
    // TODO: handle molecule parsing exception
  }

  // now the molecule is loaded !;-)
  // TODO: implement your molecule operation methods

}
```

A (sub)set of molecules can be loaded a molecule vector which has vector analogue properties.

Example 9-2. Load molecules from smaller files into memory

```
// open file input stream
String inputFile="yourInputFile.sdf";
FileInputStream input = null;
try
{
  input = new FileInputStream(inputFile);
}
catch (Exception ex)
{
  ex.printStackTrace();
  // TODO: handle exception
}
// estimate file format by file extension
try
{
  // estimate input file type
  IOType inType = IOTypeHolder.instance().filenameToType(inputFile);
  if (inType == null)
  {
    // TODO: handle unkown input file type
  }
}
```

```
catch (IOException e)
{
  e.printStackTrace();
  // TODO: handle exception
}

// load molecules into memory
JOEMolVector molecules=null;
try
{
  // set output type to input type
  // skip molecules with corrupted molecule entries !
  molecules = new JOEMolVector(input, inType, inType);
}
catch (IOException e)
{
  e.printStackTrace();
  // TODO: handle exception
}
```

Modify molecules

Adding atoms is a little bit critical and the required steps will be explained in example Example 9-3. When atoms are removed `beginModify()` and `endModify()` must be called also or strange results will be obtained. Adding and removing bonds is trivial and is not explained in detail.

Example 9-3. Add atoms to a molecule

```
// mol is a new molecule or an already loaded molecule
//begin molecule modification
//modifications will only be processed if the modification counter is 0!!!
mol.beginModify();
// let's assume we will add 1 atoms
// This is only essentially if many atoms will be added
// to avoid multiple internal array swappings
int natoms=mol.numAtoms()+1;
mol.reserveAtoms(natoms);

// get last atom to which we will add the new atoms
JOEAtom atom2add=mol.getAtom(mol.numAtoms());

JOEAtom atom = new JOEAtom();
XYZVector v = new XYZVector();
boolean createCoords=true;
if(createCoords)
{
  // get non-corrected bond radii for the atoms, because for the new one
  // the hybridizations are not really available
  double bondlen
=JOEElementTable.instance().getCovalentRad(nbr.getAtomicNum())+

JOEElementTable.instance().getCovalentRad(frag1.getAtomicNum());
  atom2add.getNewBondVector(v, bondlen);
```

```
}
else{
  v.setX(0.0);
  v.setY(0.0);
  v.setZ(0.0);
}
// set atom positions
atom.setVector(v);
String elementType="C";
int atomicNumber = JOEElementTable.instance().getAtomicNum(elementType);
if (atomicNumber == 0)
{
  // TODO: handle an unknown element type
}
// set atomic nnumber
atom.setAtomicNum(atomicNumber);
// set element type
atom.setType(elementType);
// add atom to the molecule
if (!mol.addAtom(atom))
{
  // TODO: atom can not be added
}
// clear atom object if you want to use it to
// add more atoms to the molecule
atom.clear();

//modifications will only be processed if the modification counter is 0!!!
//If you have called beginModify/endModify twice you can not expect
//that these changes are already available correctly.
//This fits especially for deleted and added atoms, because endModify
//updates the atomId's, writes the atom coordinates to the rotamer
//arrays and checks the aromaticity.
mol.endModify();
```

Access atoms and bonds of a molecule

Accessing atoms

Example 9-4. Access atoms using a `for` statement

```
int atoms = mol.numAtoms();
JOEAtom atom;
for (int i=1; i<=atoms; i++)
{
  atom=mol.getAtom(i);
  // TODO: do something with atoms
}
```

Example 9-5. Access atoms using an `AtomIterator`

```
AtomIterator ait = mol.atomIterator();
JOEAtom atom;
while (ait.hasNext())
```

```
{
  atom = ait.nextAtom();
  // TODO: do something with atoms
}
```

Accessing bonds

Example 9-6. Access bonds using a `for` statement

```
int bonds = mol.numBonds();
JOEBond bond;
for (int i=0; i<bonds; i++)
{
  bond=mol.getBond(i);
  // TODO: do something with atoms
}
```

Example 9-7. Access bonds using an `BondIterator`

```
BondIterator bit = bondIterator();
JOEBond bond;
while (bit.hasNext())
{
  bond = bit.nextBond();
  // TODO: do something with atoms
}
```

Special atom methods

Access the neighbour atoms of an atom

Example 9-8. Access neighbour atoms using a `NbrAtomIterator`

```
NbrAtomIterator nait = atom.nbrAtomIterator();
JOEBond bond;
JOEAtom nbrAtom;
while (nait.hasNext())
{
  nbrAtom=nait.nextNbrAtom();
  bond = nait.actualBond();
  // TODO: do something with neighbour atoms
}
```

Access the bonds of an atom

Example 9-9. Access bonds of an atom using a `BondIterator`

```
BondIterator bit = atom.bondIterator();
JOEBond bond;
while (bit.hasNext())
{
  bond = bit.nextBond();
  // TODO: do something with atom bonds
}
```

Descriptors

The descriptor calculation and storing facility of JOELib will be explained in detail in this section.

Get and calculate descriptors

There exists different descriptor types and the two main types are: native value descriptors, atom property descriptors. There exists still some methods for getting lists of descriptor names, which,for example, are required to calculate the descriptors.

Example 9-10. Get a list of all available descriptors

```
Enumeration enum = DescriptorHelper.instance().descriptors();
System.out.println("Descriptors:");
String descName;
for (; enum.hasMoreElements();)
{
  descName = (String) enum.nextElement();
  System.out.println(descName);
}
```

Example 9-11. Get a list of all native value descriptors

```
Vector nativeDescs = DescriptorHelper.instance().getNativeDescs();
System.out.println("Native value descriptors:");
int size = nativeDescs.size();
String descName;
for (int i = 0; i < size; i++)
{
  descName = nativeDescs.get(i);
  System.out.println(descName);
}
```

Example 9-12. Get a list of all atom property descriptors

```
Vector atomPropDescs = DescriptorHelper.instance().getAtomPropDescs();
System.out.println("Atom property descriptors:");
int size = atomPropDescs.size();
String descName;
for (int i = 0; i < size; i++)
{
  descName = atomPropDescs.get(i);
  System.out.println(descName);
}
```

Get descriptors creating a new descriptor instance, which is slow when used multiple times, because it's not cached internally.

Example 9-13. Get and calculate descriptors using always a new instance (slow)

```
DescResult result = null;
// the molecule must ba already available in the mol object
// descName contains the descriptor name which should be calculated
try
```

```
{
  // calculate descriptor for the given molecule
  result = DescriptorHelper.instance().descFromMol(mol, descName);
}
catch (DescriptorException ex)
{
  // TODO: descriptor calculation preconditions are not valid
  // TODO: handle exception
}
if (result == null)
{
  // TODO: descriptor can not be calculated
  // TODO: handle this case
}
else
{
  // add calculated descriptor result to this molecule
  JOEPairData dp = new JOEPairData();
  // use default descriptor name to store result
  dp.setAttribute(descName);
  dp.setValue(result);
  // add descriptor result to molecule without
  // overwriting old result with the same descriptor name
  mol.addData(dp);
}
```

Get descriptors creating in the first step an descriptor instance, which can then be used for descriptor calculation multiple times (fast).

Example 9-14. Get and calculate descriptors creating only one descriptor calculation instance (fast)

```
Descriptor descriptor=null;
DescDescription descDescription=null;
try
{
  descriptor = DescriptorFactory.getDescriptor(descNames);
  if (descriptor == null)
  {
    // TODO: descriptor calculation method can not be loaded
  }
  else
  {
    descDescription = descriptor.getDescription();
  }
}
catch (DescriptorException ex)
{
  // TODO: descriptor calculation preconditions are not valid
  // TODO: handle exception
}

// calculate descriptors for a set of molecules
```

```
...
  // TODO: iterate over a set of molecules
  // or load a set of molecules
  // precondition for the following lines: mol contains a molecule
  DescResult results=null;
  try
  {
    // initialize descriptor calculation properties
    // we will here use no properties, this can be e.g.
    // an atom property when calculating the autocorrelation function
    Hashtable calculationProperties = new Hashtable();
    descriptor.clear();
    results = descriptor.calculate(mol, calculationProperties);
    if (result == null)
    {
      // TODO: descriptor can not be calculated
      // TODO: handle this case
    }
    else
    {
      // add calculated descriptor result to this molecule
      JOEPairData dp = new JOEPairData();
      // use default descriptor name to store result
      dp.setAttribute(descName);
      dp.setValue(result);
      // add descriptor result to molecule without
      // overwriting old result with the same descriptor name
      mol.addData(dp);
    }
  }
  catch (DescriptorException ex)
  {
    // TODO: descriptor calculation preconditions are not valid
    // TODO: handle exception
  }
...
```

Create own descriptor classes

There exists different abstraction levels to create own descriptor calculation methods. For native value descriptors there exists already some simple abstract class implementations, so we will begin with this really simple example.

The simple absract native descriptor classes exist for boolean, double and int value descriptors. The abstract simple classes are `joelib/desc/SimpleBooleanDesc.java` (joelib/src/joelib/desc/SimpleBooleanDesc.java), `joelib/desc/SimpleDoubleDesc.java` (joelib/src/joelib/desc/SimpleDoubleDesc.java), `joelib/desc/SimpleIntDesc.java` (joelib/src/joelib/desc/SimpleIntDesc.java), `joelib/desc/AtomsCounter.java` (joelib/src/joelib/desc/AtomsCounter.java) and `joelib/desc/SMARTSCounter.java` (joelib/src/joelib/desc/SMARTSCounter.java). The abstract methods which must be implemented are `getBooleanValue(JOEMol)`,`getDoubleValue(JOEMol)` and `getIntValue(JOEMol)`. All other

needed methods are already implemented and you can ignore these implementations for these simple descriptors.

Example 9-15. Create own native descriptor calculation classes

```
// use default descriptor calculation package
package joelib.desc.types;

// import base classes and the molecule class
import joelib.desc.DescriptorHelper;
import joelib.desc.DescriptorInfo;
import joelib.desc.SimpleDoubleDesc;
import joelib.molecule.JOEMol;

// import logging tool
import org.apache.log4j.Category;

//
public class MyMolecularWeight extends SimpleDoubleDesc
{
  // initialize logging tool for this class
  private static Category
logger=Category.getInstance("joelib.desc.types.MyMolecularWeight");

  // initialize public DESC_KEY (descriptor name) by which this descriptor
can be
  // calculated
  // IMPORTANT: This should be always be a 'public static final' variable
  // IMPORTANT: to avoid misinterpretations during runtime
  public static final String DESC_KEY = "My_molecular_weight";

  public My_molecular_weight()
  {
    // show basic logging message if debugging is enabled
    if (logger.isDebugEnabled())logger.debug("Initialize " +
this.getClass().getName());

    // IMPORTANT: initialize descriptor informations
    // IMPORTANT: use DescriptorHelper to facilitate this task
    // IMPORTANT: relevant parameters are the descriptor name, the
    // IMPORTANT: calculation representation and the descriptor result

    descInfo=DescriptorHelper.generateDescInfo(DESC_KEY,this.getClass(),
                              DescriptorInfo.TYPE_NO_COORDINATES,null,
                              "joelib.desc.result.DoubleResult");
  }

  // get double value for molecular weight
  public double getDoubleValue(JOEMol mol)
  {
    double mw;
    mw = mol.getMolWt();
    return mw;
```

```
  }
}
```

JOELib must now be told that there is a new descriptor calculation method. You must add the following line `joelib.descriptor.60.representation=joelib.desc.types.MyMolecularWeight` to the `joelib.properties` (joelib/src/joelib.properties)-file. If you've already implemented some other descriptors you must use another number, e.g. 61, or something else. It's important that these numbers increase monotonically by 1, because the descriptor factory class interrupts the loading process if no higher number (increased by 1) is available.

The abstract simple classes for atom properties are `joelib/desc/SimpleDoubleAtomProperty.java` (joelib/src/joelib/desc/SimpleDoubleAtomProperty.java), `joelib/desc/SimpleDynamicAtomProperty.java` (joelib/src/joelib/desc/SimpleDynamicAtomProperty.java).

Example 9-16. Create own atom property descriptor calculation classes

```
// use default descriptor calculation package
package joelib.desc.types;

// import base classes and the molecule class
import joelib.desc.DescriptorHelper;
import joelib.desc.DescriptorInfo;
import joelib.desc.SimpleDynamicAtomProperty;
import joelib.desc.result.DynamicArrayResult;
import joelib.molecule.JOEAtom;
import joelib.molecule.JOEMol;
import joelib.util.iterator.AtomIterator;

// import logging tool
import org.apache.log4j.Category;

public class ElectronegativityAllredRochow extends SimpleDynamicAtomProperty
{
        // initialize logging tool for this class
        private static Category logger =

Category.getInstance("joelib.desc.types.ElectronegativityAllredRochow");

        // initialize public DESC_KEY (descriptor name) by which this
descriptor can be
        // calculated
        // IMPORTANT: This should be always be a 'public static final'
variable
        // IMPORTANT: to avoid misinterpretations during runtime
        public static final String DESC_KEY =
"Electronegativity_allred_rochow";

        public ElectronegativityAllredRochow()
        {
                // show basic logging message if debugging is enabled
                if (logger.isDebugEnabled())
```

```
                        logger.debug("Initialize " +
this.getClass().getName());

                // IMPORTANT: initialize descriptor informations
                // IMPORTANT: use DescriptorHelper to facilitate this task
                // IMPORTANT: relevant parameters are the descriptor name,
the
                // IMPORTANT: calculation representation and the descriptor
result
                descInfo =
                        DescriptorHelper.generateDescInfo(
                                DESC_KEY,
                                this.getClass(),
                                DescriptorInfo.TYPE_NO_COORDINATES,
                                null,
                                "joelib.desc.result.AtomDynamicResult");

        }

        // get array with atom properties
        // typically we use already deprotonated
        // molecules without hydrogens
        public Object getAtomPropertiesArray(JOEMol mol)
        {
                // get partial charges for all atoms
                JOEAtom atom;
                AtomIterator ait = mol.atomIterator();
                double enAllredRochow[] =
                        (double[]) DynamicArrayResult.getNewArray(
                                DynamicArrayResult.DOUBLE,
                                mol.numAtoms());
                int i = 0;
                while (ait.hasNext())
                {
                        atom = ait.nextAtom();
                        enAllredRochow[i++] = atom.getENAllredRochow();
                }
                return enAllredRochow;
        }
}
```

For more complex descriptors, e.g. Moreau-Broto-Autocorrelation, the required interface methods for `joelib/desc/Descriptor.java` (joelib/src/joelib/desc/Descriptor.java) must be all implemented and there is no abstract helper class available. Because this is a complex task it is recommended to use an already implemented descriptor class, to copy and rename this file and modify these implementations for your requirements.

Processes and filters

Applying processes and filters

Processes can be get by using an instance or by using the factory helper class. In general getting plain instances of these process classes would be more easier, because most of them have special initialization methods, which must be called after getting an instance of these classes.

Example 9-17. Get a molecule process by creating an instance

```
JOEProcess process=new DescVarianceNorm();
```

Example 9-18. Get a molecule process by using the process factory

```
JOEProcess process=null;
try
{
  process=  ProcessFactory.instance().getProcess("VarianceNormalization");
}
catch (JOEProcessException ex)
{
  // TODO: handle exception, when process can not be found
}
```

Example 9-19. Use initialization method for a process (depends on the process!)

```
try
{
  ((DescVarianceNorm)process).init(inType,inputFile);
}
catch (Exception ex)
{
  // TODO: handle exception
}
```

For building more complex process pies with applying special filter functionalities there exists a simple process pipe. More extended things, like process trees or somethings else sj´hould not be too difficult to implement.

Example 9-20. Get a molecule process pipe with filter functionalities

```
//initialize filter
DescriptorFilter descFilter = new DescriptorFilter();
// filter accepts only molecules where all descriptors given by the
// file at the descNamesURL exists. Commented descriptors names will
// be ignored
descFilter.init(descNamesURL, false);

// initialize process and pipe
DescSelectionWriter dsw = null;
ProcessPipe processPipe = null;
dsw = new DescSelectionWriter();
processPipe = new ProcessPipe();
// initialize output options for the
```

```
// descriptor selection writer
try
{
  dsw.init(outputFile, outType, desc2write, descOutType);
  dsw.setDelimiter(delimiter);
}
catch (Exception ex)
{
}

// use descriptor selection filter only when
// all descriptor names given by the file at the
// descNamesURL are available.
// For more complex filter mechanism there
// exists still AND and OR combinations for filters
processPipe.addProcess(dsw, descFilter);
```

When a molecule has been loaded it is really simple the process:

Example 9-21. Apply a process to a molecule

```
try
{
  // call process without additional paramameters
  processPipe.process(mol, null);
}
catch (JOEProcessException ex)
{
  // TODO: catch process exception
}
```

Import/Export

The import and export types can be defined dynamically and this is really usefull for slightly modified import and export funtionalities, e.g. assigning automatically identifiers, checking for duplicates, ...

Import

Get molecule import classes

Example 9-22. Get molecule import classes

```
// open file input stream
String inputFile="yourInputFile.sdf";
FileInputStream input = null;
try
{
  input = new FileInputStream(inputFile);
}
catch (Exception ex)
{
  ex.printStackTrace();
```

```
  // TODO: handle exception
}
// create simple reader and
// estimate file format by file extension
SimpleReader reader = null;
try
{
  // estimate input file type
  IOType inType = IOTypeHolder.instance().filenameToType(inputFile);
  if (inType == null)
  {
    // TODO: handle unkown input file type
  }
  // open simple reader
  reader = new SimpleReader(input, inType);
}
catch (IOException e)
{
  e.printStackTrace();
  // TODO: handle exception
}
```

Export

Get molecule export classes

Example 9-23. Get molecule export classes

```
// open file output stream
String outputFile="yourOutputFile.sdf";
FileOutputStream output = null;
try
{
  output = new FileOutputStream(outputFile);
}
catch (Exception ex)
{
  ex.printStackTrace();
  // TODO: handle exception
}
// create simple writer and
// estimate file format by file extension
SimpleWriter writer = null;
try
{
  // estimate output file type
  IOType outType = IOTypeHolder.instance().filenameToType(outputFile);
  if (outType == null)
  {
    // TODO: handle unkown output file type
  }
  // open simple writer
  writer = new SimpleWriter(output, outType);
```

```
}
catch (IOException e)
{
  e.printStackTrace();
  // TODO: handle exception
}
```

Simple Import/Export pipeline

Create molecule processing pipe

Example 9-24. Create molecule processing pipe

```
package joelib.test;

// import base classes and the molecule class
import java.io.IOException;
import joelib.data.JOEPairData;
import joelib.io.IOTypeHolder;
import joelib.io.MoleculeIOException;
import joelib.io.SimpleReaderWriterPipe;
import joelib.molecule.JOEMol;
import org.apache.log4j.Category;

public class AddMolID extends SimpleReaderWriterPipe
{
  // initialize logging tool for this class
  private static Category logger =
Category.getInstance("joelib.test.AddMolID");

  public AddMolID(String args[]) throws IOException
  {
    // call super class for created pipe by
    // using the command line parameters
    super(args);
  }

  // you will be fored to implement these method by the
  // SimpleReaderWriterPipe parent class
  public void showUsage()
  {
    StringBuffer sb = new StringBuffer();
    String programName = this.getClass().getName();
    sb.append("Usage is :\n");
    sb.append("java -cp . ");
    sb.append(programName);
    sb.append(" [options]");
    sb.append(" <input file>");
    sb.append(" <output file>");
    sb.append("\n where options can be:");
    sb.append(" -i<inputFormat>");
    sb.append(" -o<outputFormat>");
    sb.append("\nSupported molecule types:");
```

```
    sb.append(IOTypeHolder.instance().toString());
    System.out.println(sb.toString());
    System.exit(0);
  }

  public void loadWriteAllMolecules()
  {
    logger.info("Start adding molecule identifiers ...");
    for (;;)
    {
      try
      {
        // break, if all molecules were treated
        if (!this.readWriteNext())break;
      }
      catch (IOException e)
      {
        logger.error("IO problems: "+e.getMessage()+" at molecule
"+loadedMolecule().getTitle());
      }
      catch (MoleculeIOException e)
      {
        logger.error("Molecule parsing problems: "+e.getMessage()+
                     " at molecule "+loadedMolecule().getTitle());
      }
    }
  }

  // you will be fored to implement these method by the
  // SimpleReaderWriterPipe parent class
  public void molecule2handle(JOEMol mol)
  {
    JOEPairData dp = new JOEPairData();
    dp.setAttribute("ID");
    dp.setValue(Integer.toString(moleculesLoaded()));
    // overwrite previous identifiers
    mol.addData(dp,true);
  }

  public static void main(String[] args)
  {
    AddMolID addMolID = null;
    try
    {
      addMolID = new AddMolID(args);
    }
    catch (IOException e)
    {
      e.printStackTrace();
      System.exit(1);
    }
    addMolID.loadWriteAllMolecules();
  }
```

```
}
```

Interfaces

abc.

Java interfaces

abc.

Chemical Development Kit (CDK)

The Chemical Development Kit (CDK) (http://sourceforge.net/projects/cdk) interface supports only the generation of 2D coordinates.

Example 9-25. Chemical Development Kit (CDK) interface

```
TestSMILES joeMolTest = new TestSMILES();
JOEMol mol = new JOEMol(IOTypeHolder.instance().getIOType("SMILES"),
                        IOTypeHolder.instance().getIOType("SDF"));

System.out.println("Generate molecule from SMILES: \"" + smiles + "\"");
// create molecule from SMILES string
if (!JOESmilesParser.smiToMol(mol, smiles, "Name:" + smiles))
{
  System.err.println("Molecule could not be generated from \"" + smiles +
"\".");
}

// Layout test
System.out.println("Generate 2D coordinates:");
CDKTools.generate2D(mol);
// show molecule with 2D coordinates
System.out.println(mol);
```

Weka data mining interface

The Weka data mining library (http://www.cs.waikato.ac.nz/ml/weka/) data mining interface can be used mainly designed for clustering and classification tasks. For regression it's recommended to use e.g. the JavaNNS (http://www-ra.informatik.uni-tuebingen.de/software) interface or other interfaces this group will provide.

Example 9-26. Weka data mining interface

```
// open file input stream
FileInputStream in = null;
JOEMolVector mols = null;
// create molecule vector
try
{
  in = new FileInputStream(inputFile);
  mols = new JOEMolVector(in);
}
```

```
catch (Exception ex)
{
  ex.printStackTrace();
  System.exit(1);
}

// load descriptor binning
DescBinning binning = DescBinning.getDescBinning(mols);

// create Weka attribute types
// we assume the following format:
// all attributes are numeric, except the class index, which is
// a nominal attribute.
// Please refer for further help the Weka tutorial and mailing list
Enumeration enum = binning.getDescriptors();
String attributes[] = new String[binning.numberOfDescriptors()];
int attributeTypes[] = new int[binning.numberOfDescriptors()];
int i = 0;
int classAttributeIndex = -1;
for (; enum.hasMoreElements(); i++)
{
  attributes[i] = (String) enum.nextElement();
  if (attributes[i].equals(classAttributeName))
  {
    classAttributeIndex = i;
    attributeTypes[i] = Attribute.NOMINAL;
  }
  else
  {
    attributeTypes[i] = Attribute.NUMERIC;
  }
}
// check class index
if (classAttributeIndex == -1)
{
  logger.error("Class attribute " + classAttributeName + " not found.");
  System.exit(1);
}

// create Weka instances
MolInstances instances = MolInstances.createMolInstances(mols, attributes,
attributeTypes);

// set class index
instances.setClassIndex(classAttributeIndex);
```

C/C++ interfaces using JNI

Some Java implementions to test Ghemical

To use LibGhemical (http://bioinformatics.org) you must install the system specifife JNI interfaces under `lib/linux` and `lib/windows`. The LibGhemical parameter sets for initializing the force field

must be also already installed. The default path is `/usr/local/share/libghemical/1.51/`, but these things can vary with a new release or if you have compiled LibGhemical on your own (recommended).

Example 9-27. Create 3D coordinates using the Ghemical force field (molecular mechanics)

```
TestSMILES joeMolTest = new TestSMILES();
JOEMol mol = new JOEMol(IOTypeHolder.instance().getIOType("SMILES"),
                        IOTypeHolder.instance().getIOType("SDF"));

System.out.println("Generate molecule from SMILES: \"" + smiles + "\"");
// create molecule from SMILES string
if (!JOESmilesParser.smiToMol(mol, smiles, "Name:" + smiles))
{
  System.err.println("Molecule could not be generated from \"" + smiles +
"\".");
}

// Layout test
System.out.println("Generate 3D coordinates:");
double tresholdDeltaE = 1.0e-14;
double tresholdStep = 6.0e-11;
int numSteps = 2000;
// when all coordinates are zero the molecule coordinates will be initialized
// with random coordinates
if (!GhemicalInterface.instance().doGeometryOptimization(
  mol,
  numSteps,
  tresholdDeltaE,
  tresholdStep,
  true))
{
  logger.error("Could not apply energy minimization.");
  System.exit(1);
}

// show molecule with 2D coordinates
System.out.println(mol);
```

Matlab interface

abc.

Matlab to Java connection

Download the JOELib-Matlab toolbox (http://joelib.sourceforge.net) to use this feature.

Example 9-28. Load native descriptors under Matlab

```
cd yourWorkingDirectory/
descriptors=loaddescriptors('test_moe_desc.mol', 'SDF');
```

Java to Matlab connection

This can be esablished when using the JMatLink JNI interface (see also the Section called *Matlab toolbox* in Chapter 1).

Database

abc.

Database access using the Java Database Connection (JDBC)

For the database access presented here the properties `location`, `password`, `username`, `driver` must be set in the `joelib.properties` (joelib/src/joelib.properties)-file. Additionally this example will create `MOLECULES` table with the table properties

Table 9-1. Simple database definition

Name	Data type	Description
`NAME`	`VARCHAR(100)`	Molecule name
`ID`	`BIGINT`	Molecule identifieer, should be unique
`HASH`	`BIGINT`	Hashcode for the molecule (checking for duplicates, until now without cis/trans, E/Z)
`SHASH`	`BIGINT`	Hashcode for the molecule using SMILES (checking for duplicates, including cis/trans, E/Z)
`SDF`	`LONGTEXT`	Structured Data File (SDF)
`CML`	`LONGTEXT`	Chemical Markup Language (CML)
`SMILES`	`LONGTEXT`	Simplified Molecular Input Line Entry System (SMILES)

Example 9-29. Database access

```
public static void main(String[] args)
{
  SimpleJOELibDatabase dbTest = new SimpleJOELibDatabase();

  logger.info("Open file:" + args[0]);
  logger.info("with input type:" + args[1]);


dbTest.storeMolsInDatabase(args[0],IOTypeHolder.instance().getIOType(args[1])
);
```

```
}
```

Miscellaneous

Here all examples are presented, which does not fit in another category.

Point group symmetry calculation

Here is an example for the brute force symmetry analyzer identifying primitive rotations, planes and inversion centres.

Example 9-30. Calculate point group symmetries

```
if (symmetry.readCoordinates(mol) < 0)
{
  System.err.println("Error reading in atomic coordinates");
  System.exit(1);
}

// start symmetry estimation
try
{
  symmetry.findSymmetryElements(true);
}
catch (SymmetryException e)
{
  e.printStackTrace();
  System.exit(1);
}

if (symmetry.getBadOptimization())
  System.out.println("Refinement of some symmetry elements was"+
                     " terminated before convergence was reached.\n" +
                     "Some symmetry elements may remain unidentified.\n");

symmetry.reportSymmetryElementsVerbose();
System.out.println("Molecule has the following symmetry elements:\n"+
symmetry.getSymmetryCode());
PointGroup pointGroup = symmetry.identifyPointGroup();
if (pointGroup != null)
{
  System.out.println("It seems to be the "+ pointGroup.getGroupName()+ "
point group");
}
else
{
  System.out.println("These symmetry elements match more than one group I
know of.");
}
```

Hash code calculation

Sloppy Hashcode which uses only topological informations without E/Z isomerism and S/R chirality. This hascode is really helpfull to identify duplicate topological molecule entries.

Example 9-31. Calculate a molecule hashcode

```
int hashcode =AbstractDatabase.getHashcode(mol)
```

Reading JCAMP-DX spectras

Parse JCAMP-DX data.

Example 9-32. Load JCAMP-DX spectras

```
JCAMPParser jcamp=null;
try
{
  jcamp=JCAMPParser(jcampString);
}
catch(IOException ioex)
{
  ioex.printStackTrace();
}
catch(JCAMPException jex)
{
  jex.printStackTrace();
}
```

Create fragmented molecules

Fragemted molecules can be generated by splitting contiguous fragments or using SMARTS splice definitions.

Example 9-33. Fragment molecules if they are disconnected (non-contiguous)

```
ContiguousFragments fragmenter=new ContiguousFragments();
JOEMolVector fragments=fragmenter.getFragmentation(mol);
```

Combinatorial synthesis for generating virtual combinatorial libraries

The standard procedure to generate molecules is using s scaffold and R-groups. New coordinates will be generated if the scaffold contains already 2D or 3D coordinates.

Example 9-34. Generate molecules using R-Groups

```
MolGenerationHelper generator=new MolGenerationHelper();
JOEMol newMolecule=createNewMolecule(baseMolecule,connections,rGroups);
```

Chapter 10. Example applications

Convert

Example application for file conversion, descriptor calculation and process filters (here only for converting molecules).

Options

sh convert.sh [-? | --help] [-h] [+h] [+p] [-e] [-d] [+d] [+v] [+snd] [+sad] [+sall] [-salt] [-i*inputFormat*] [-o*outputFormat*] [+x*descriptorName*] [-r*skipDescRule*] [+r*descRule*] [+s*lineStructure*] *inputFile* [*outputFile*]

[-? | --help]

Shows the help screen including all available input/output formats and all native value descriptors.

[-h]

Removes all hydrogens from the molecule.

[-h]

Adds hydrogens to molecule.

[+p]

Adds only polar hydrogens to molecule.

[-e]

Converts only non-empty molecules.

[-d]

Removes all descriptors from molecule.

[+d]

Adds all available descriptors to molecule, which can be calculated by using JOELib.

[+v]

Switches verbosity mode ON.

[+sall]

Shows all available descriptors, which can be calculated by using JOELib.

[+snd]

Shows all available native value descriptors, which can be calculated by using JOELib.

[+sad]

Shows all available atom property descriptors, which can be calculated by using JOELib.

[-salt]

Strip salts and gets only largest contigous fragment.

[-i*inputFormat*]

Format of the input file.

[-o*outputFormat*]

Format of the output file.

[+x*descriptorName*]

Converts only molecules, where this descriptor exist. This command line parameter can occur multiple times.

[-r*skipDescRule*]

Skips all molecules, where this descriptor rule fits. This command line parameter can occur multiple times.

[+r*descRule*]

Converts only molecules, where this descriptor rule fits. This command line parameter can occur multiple times.

[+s*lineStructure*]

Can be used for an alternate SMILES entry line structure.

inputFile

The input file. The input format will be estimated by using the file extension.

[*outputFile*]

The output file. The output format will be estimated by using the file extension.

Examples

Example 10-1. Show help screen for Convert

```
java -cp joelib.jar joelib.test.Convert --help
```

shows the help screen

Example 10-2. Print molecules to standard output

```
java -cp joelib.jar joelib.test.Convert joelib/test/test.mol
```

Prints the molecule file to the screen.

ConvertSkip

Example application for file conversion, descriptor calculation and process filters. Additionally this example can be used to check a molecule file for correctness and store invalid or empty molecules (if [-e] option is activated) in a skip file. Because this example uses another loading mechanism, not all input formats are supported until now. Supported input file formats are SDF, SMILES, Flat and XYZ.

Options

sh convertSkip.sh [-? | --help] [-h] [+h] [+p] [-e] [-d] [+d] [+v] [+snd] [+sad] [+sall] [-salt] [-i*inputFormat*] [-o*outputFormat*] [+x*descriptorName*] [-r*skipDescRule*] [+r*descRule*] [+f*lineStructure*] [+s*lineStructure*] *inputFile* [*outputFile*] [*skipFile*]

[-? | --help]

Shows the help screen including all available input/output formats and all native value descriptors.

[-h]

Removes all hydrogens from the molecule.

[+h]

Adds hydrogens to molecule.

[+p]

Adds only polar hydrogens to molecule.

[-e]

Converts only non-empty molecules.

[-d]

Removes all descriptors from molecule.

[+d]

Adds all available descriptors to molecule, which can be calculated by using JOELib.

[+v]

Switches verbosity mode ON.

[+sall]

Shows all available descriptors, which can be calculated by using JOELib.

[-salt]

Strip salts and gets only largest contigous fragment.

[+snd]

Shows all available native value descriptors, which can be calculated by using JOELib.

[+sad]

Shows all available atom property descriptors, which can be calculated by using JOELib.

[-i*inputFormat*]

Format of the input file.

[-o*outputFormat*]

Format of the output file.

[+x*descriptorName*]

Converts only molecules, where this descriptor exist. This command line parameter can occur multiple times.

[-r*skipDescRule*]

Skips all molecules, where this descriptor rule fits. This command line parameter can occur multiple times.

[+r*descRule*]

Converts only molecules, where this descriptor rule fits. This command line parameter can occur multiple times.

[+f*lineStructure*]

Required if you use FLAT output format which other input format.

[+s*lineStructure*]

Can be used for an alternate SMILES entry line structure.

inputFile

The input file. The input format will be estimated by using the file extension.

[*outputFile*]

The output file. The output format will be estimated by using the file extension.

[*skipFile*]

The skip file. The output format will be the same of the input format. All skipped molecules will be stored in this file.

If no output file is defined, all molecules will be written to stdout. The skip file must have the same format as the input file. All skipped entries and empty entries (if -e was selected) were stored in the skip file.

Filter rules have the form: <native value descriptor><relation><value> where <relation> is <, <=, ==, >, >= or != Example: "+rNumber_of_halogen_atoms==2"

Examples

Example 10-3. Show help screen for ConvertSkip

```
java -cp joelib.jar joelib.test.ConvertSkip --help
```
shows the help screen

Example 10-4. Print molecules to standard output

```
java -cp joelib.jar joelib.test.ConvertSkip joelib/test/test.mol
```
Prints the molecule file to the screen.

Example 10-5. Write flat file

```
java -cp joelib.jar joelib.test.ConvertSkip resources/multiple.mol test.flat
"-fID|TITLE|d1|d2|d3"
```
Writes a flat file with line structure ID TITLE d1 d2 d3 to file test.flat.

Example 10-6. Using conversion rule

```
java -cp joelib.jar joelib.test.ConvertSkip resources/multiple.mol +h -e
+xID\
         test.sdf "+rNumber_of_halogen_atoms<=2"
```
Converts all non-empty molecules where the number of halogen atoms is smaller or equal to two and the identifier ID exists. The accepted and protonated molecules will be written to test.sdf in MDL SDF file format.

Comparison

Example application for comparing molecules. Molecules can be compared using fingerprints (tanimoto) or descriptor values (euklidian distance).

Options

sh comparison.sh *typeTarget targetFile typeToCompare toCompareFile descriptorFile outputFile identifier*

typeTarget

Input/Output type of the target file.

targetFile

Target file (size T). This file will be loaded into the memory.

typeToCompare

Input/Output type of the file to which the target file will be compared.

toCompareFile

Comparison file (size C). This file will be loaded from file and can be huge.

descriptorFile

These descriptors will be used for comparing molecules. If this is a fingerprint descriptor the tanimoto coefficient will be calculated. If a list of native value descriptors is available, the euklidian distance value will be calculated for these descriptor values.

outputFile

The output file for the similarity/distance values. This will be a matrix of size T*C.

identifier

The molecule identifier of the molecules in the comparison file which will be the first entry in every line.

Descriptor calculation

Example application for calculating descriptors: basic ones, atom property based transformations and user defined substructure counts.

Options

sh calculateDescriptors.sh [+jcc] [-jcc] [+ap] [-ap] [+SSKey] [-SSKey] [+binarySMARTS] [-binarySMARTS] [+countSMARTS] [-countSMARTS] [+normalize] [-normalize] [-i*inputFormat*] [-o*outputFormat*] [-t*TRUE_VALUE*] *inputFile outputFile*

[-i*inputFormat*]

Format of the input file.

[-o*outputFormat*]

Format of the output file.

[-t*TRUE_VALUE*]

Defines TRUE VALUE descriptor (will NOT be normalized).

[+jcc]

Calculate all available native descriptors in JCompChem (default)

[-jcc]

Does not calculate all available native descriptors in JCompChem

[+ap]

Calculate all available atom property descriptors in JCompChem (default)

[-ap]

Does not calculate all available atom property descriptors in JCompChem

[+SSKey]

Calculate SSkey fingerprint (default)

[-SSKey]

Does not calculate SSkey fingerprint

[+binarySMARTS]

Calculate binary SMARTS (default)

[-binarySMARTS]

Does not calculate binary SMARTS

[+countSMARTS]

Calculate counting SMARTS

[-countSMARTS]

Does not calculate counting SMARTS (default)

[+normalize]

Normalize descriptor values. SMARTS and fingerprints are ignored

[-normalize]

Do not normalize the descriptor values (default)

Warning: If you use binary AND counting SMARTS you should NOT use your own names, or only the binary SMARTS results will be stored. That's obvious, ONE name, ONE result.

EuklidianComparison

Example application for comparing molecules. Molecules can be compared using descriptor values (euklidian distance).

Options

sh euklComparison.sh *typeTarget targetFile typeToCompare toCompareFile descriptorFile outputFile identifier*

typeTarget

Input/Output type of the target file.

targetFile

Target file (size T). This file will be loaded into the memory.

typeToCompare

Input/Output type of the file to which the target file will be compared.

toCompareFile

Comparison file (size C). This file will be loaded from file and can be huge.

descriptorFile

These descriptors will be used for comparing molecules.This must be a list of native value descriptors.

outputFile

The output file for the euklidian distances. This will be a matrix of size T*C.

identifier

The molecule identifier of the molecules in the comparison file which will be the first entry in every line.

Statistic

Example application for calculating the statistic of the descriptor entries in a multiple molecule file.

Options

sh statistic.sh *inputType inputFile outputFile*

inputType

Input/Output type of the molecule file with descriptor values.

inputFile

The input file.

outputFile

The output file.

Note that for a filname e.g. `test.sdf` the statistic and the descriptor binning will be always stored in the files `test.sdf.statistic` and `test.sdf.binning` independently from the given output file in the command line.

Point Group Symmetry

Example application for calculating the symmetry in a molecule file. All specific structures should have corresponding elements in the same position generic structure does. Planes are characterized by the surface normal direction (taken in the direction from the coordinate origin) and distance from the coordinate origin to the plane in the direction of the surface normal. Inversion is characterized by location of the inversion center. Rotation is characterized by a vector (distance+direction) from the origin to the rotation axis, axis direction and rotation order. Rotations are in the clockwise direction looking opposite to the direction of the axis. Note that this definition of the rotation axis is not unique, since an arbitrary multiple of the axis direction can be added to the position vector without changing actual operation. Mirror rotation is defined by the same parameters as normal rotation, but the origin is now unambiguous since it defines the position of the plane associated with the axis.

Options

sh symmetry.sh [maxaxisorder *value*] [maxoptcycles *value*] [same *value*] [primary *value*] [final *value*] [minoptstep *value*] [maxoptstep *value*] [gradstep *value*] [minchange *value*] [minchgcycles *value*] *inputFile*

[maxaxisorder *value*]

Maximum order of rotation axis to look for. Default value: 20

[maxoptcycles *value*]

Maximum allowed number of cycles in symmetry element optimization. Default value: 200

[same *value*]

Atoms are colliding if distance falls below this value. Default value: 0.0010

[primary *value*]

Initial loose criterion for atom equivalence. Default value: 0.05

[final *value*]

Final criterion for atom equivalence. Default value: 0.05

[minoptstep *value*]

Termination criterion in symmetry element optimization. Default value: 1.0E-7

[maxoptstep *value*]

Largest step allowed in symmetry element optimization. Default value: 0.5

[gradstep *value*]

Finite step used in numeric gradient evaluation. Default value: 1.0E-7

[minchange *value*]

Minimum allowed change in target methodName. Default value: 1.0E-10

[minchgcycles *value*]

Number of minchange cycles before optimization stops. Default value: 5

inputFile

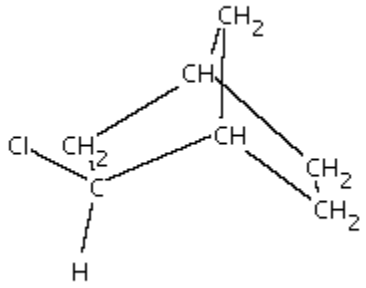
Molecul input file.

Chapter 11. Support

For further question there are archived mailing lists and tracking systems available:

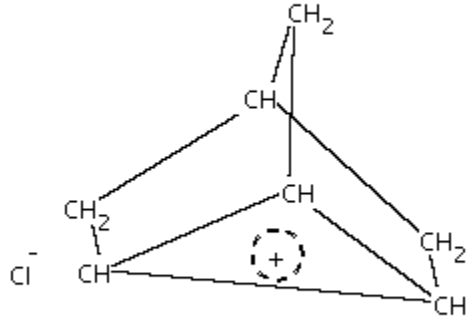
- Bug tracking system (http://sourceforge.net/tracker/?atid=425969&group_id=39708&func=browse): Bugs should be send to the bug tracking system.
- Support request tracking system (http://sourceforge.net/tracker/?atid=425970&group_id=39708&func=browse): Support requests should be send to the support request tracking system.
- Help mailing list (http://lists.sourceforge.net/lists/listinfo/joelib-help) and Help archive (http://sourceforge.net/mailarchive/forum.php?forum_id=26962): This list can be used for general questions and problems.
- Developer mailing list (http://sourceforge.net/mailarchive/forum.php?forum_id=10209) and Developer archive (http://sourceforge.net/mailarchive/forum.php?forum_id=10209): Developers and people with algorithm, feature, and design request should mail to this list. We recommend to have a at first a look at the source code, so we can discuss things with guessing or concrete wishes what can be technically really improved.

Appendix A. Summary of molecular structures



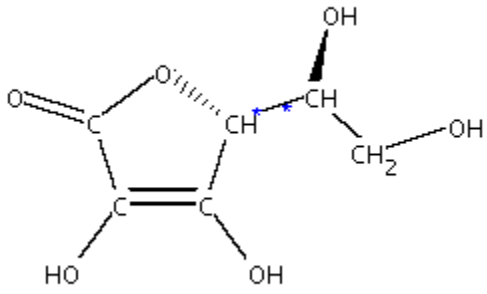
exo-norcamphan

structures/exo_norcamphan: width=300 height=200 rotate=270



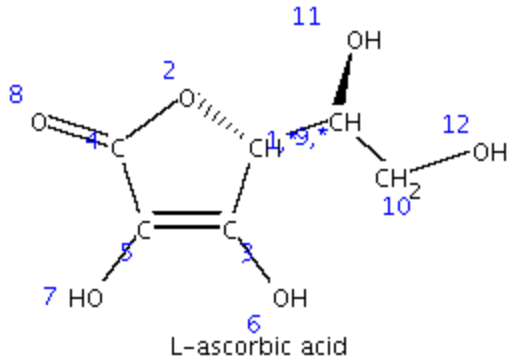
norbornyl cation (SN,intermediate product)

structures/exo_norcamphan_norbonyl_cation: width=300 height=250 rotate=270 conjRing=2,3,4,c+



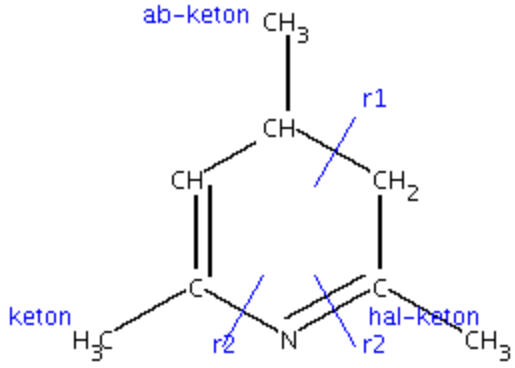
L-ascorbic acid

structures/l_ascorbic_acid: width=300 height=200 rotate=270 labels=9=*;1=*

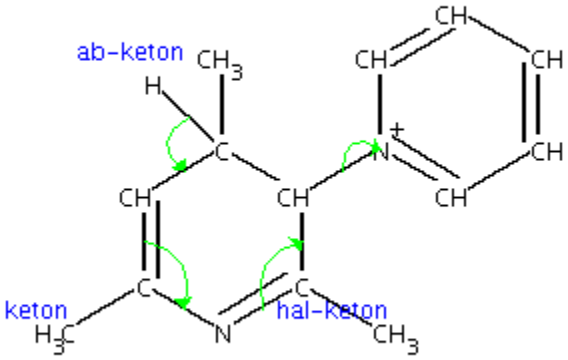


L-ascorbic acid

structures/l_ascorbic_acid_numbers: width=300 height=200 shownumbers rotate=270 labels=9=*;1=*



structures/pyridin_kroehnke: width=300 height=200 rotate=270 hideEndCarbon orthoLines=1,l,sr2,6;5,r,sr2,6;2,r,sr1,3 labels=8=hal-keton;7=ab-keton;9=keton



structures/pyridin_kroehnke_arom: width=300 height=270 rotate=270 hideEndCarbon arrows=3,16,l,3,4;4,5,r,5,6;1,6,r,1,2;7,2,r,7 labels=13=ab-keton;14=hal-keton;15=keton

Bibliography

[bk02] M. Böhm and G. Klebe, *Development of New Hydrogen-Bond Descriptors and Their Application to Comparative Molecular Field Analyses*, *J. Med. Chem.*, 45, 1585-1597, 2002.

[bmv84] P. Broto, G. Moreau, and C. Vandycke, *Molecular Structures: Perception, Autocorrelation Descriptor and SAR Studies*, *Eur. J. Med. Chem.*, 19, 66-70, 1984.

[bs93] B. L. Bush and R. P. Sheridan, *PATTY: A Programmable Atom Typer and Language for Automatic Classification of Atoms in Molecular Databases*, *J. Chem. Inf. Comput. Sci.*, 33, 756-762, 1993.

[clr98] T. Cormen, C. Leiserson, and R. L. Rivest, 0-262-03141-8, MIT-Press, *Introduction to Algorithms*, 1998.

[dl93] A. N. Davies and P. Lampen, *JCAMP-DX for NMR*, *Appl. Spec.*, 47, 1093-1099, 1993.

[dw88] R. S. Mc Donald and P. A. Wilks, *JCAMP-DX: A Standard Form for Exchange of Infrared Spectra in Computer Readable Form*, *Appl. Spec.*, 42, 151-162, 1988.

[ers00] P. Ertl, B. Rohde, and P. Selzer, *Fast calculation of molecular polar surface area as a sum of fragment-based contributions and its application to the prediction of drug transport properties*, *J. Med. Chem.*, 43, 3714-3717, 2000.

[fig96] J. Figueras, *Ring Perception Using Breadth-First Search*, *J. Chem. Inf. Comput. Sci.*, 36, 986-991, 1996.

[fwz04] H. Fröhlich, J. K. Wegner, and A. Zell, *Towards Optimal Descriptor Subset Selection with Support Vector Machines in Classification and Regression*, *QSAR Comb. Sci.*, 23, 311-318, 2004.

[gas88] J. Gasteiger, 0-387-503676, Springer Verlag, *Empirical Methods for the Calculation of Physicochemical Data of Organic Compounds*.

[gas95] J. Gasteiger, *Keyword Reference Manual for Gasteiger Clear Text Files*.

[gbt02] A. Golbraikh, D. Bonchev, and A. Tropsha, *Novel Z/E-Isomerism Descriptors Derived from Molecular Topology and Their Application to QSAR Analysis*, *J. Chem. Inf. Comput. Sci.*, 42, 769-787, 2002.

[ghhjs91] J. Gasteiger, B. M. Hendriks, P. Hoever, C. Jochum, and H. Somberg, *JCAMP-CS: A Standard Format for Chemical Structure Information in Computer Readable Form*, *Appl. Spec.*, 45, 4-11, 1991.

[gm78] J. Gasteiger and M. Marsili, *A New Model for Calculating Atomic Charges in Molecules*, *Tetrahedron Lett.*, ?, 3181-3184, 1978.

[gt03] A. Golbraikh and A. Tropsha, *QSAR Modeling Using Chirality Descriptors Derived from Molecular Topology*, *J. Chem. Inf. Comput. Sci.*, 42, 144-154, 2003.

[gxsb00] L. Xue, F. L. Stahura, J. W. Godden, and J. Bajorath, *Searching for molecules with similar biological activity: analysis by fingerprint profiling*, *Pac. Symp. Biocomput.*, 8, 566-575, 2000.

[lhdl94] P. Lampen, H. Hillig, A. N. Davies, and M. Linscheid, *JCAMP-DX for Mass Spectrometry*, *Appl. Spec.*, 48, 1545-1552, 1994.

[ml91] E. C. Meng and R. A. Lewis, *{D}etermination of {M}olecular {T}opology and {A}tomic {H}ybridisation {S}tates from {H}eavy {A}tom {C}oordinates*, *J. Comp. Chem.*, 12, 891-898, 1991.

[mor65] H. L. Morgan, *The Generation of a Unique Machine Description for Chemical Structures - A Technique Developed at Chemical Abstracts Service*, *J. Chem. Doc.*, 5, 107-113, 1965.

[mr99] P. Murray-Rust and H. S. Rzepa, *Chemical Markup, XML, and the Worldwide Web. 1. Basic Principles*, *J. Chem. Inf. Comput. Sci.*, 39, 928-942, 1999.

[mr01a] P. Murray-Rust and H. S. Rzepa, *Chemical Markup, XML and the World-Wide Web. 2. Information Objects and the CMLDOM*, *J. Chem. Inf. Comput. Sci.*, 41, ?-?, 2001.

[mr01b] G. V. Gkoutos, P. Murray-Rust, H. S. Rzepa, and M. Wright, *Chemical markup, XML, and the world-wide web. 3. toward a signed semantic chemical web of trust*, *J. Chem. Inf. Comput. Sci.*, 41, 1295-1300, 2001.

[msg99] M. C. Hemmer, V. Steinhauer, and J. Gasteiger, *Deriving the 3D Structure of Organic Molecules from their Infrared Spectra*, *Vibrat. Spect.*, 19, 63-67, 1999.

[sdf] Inc. MDL Information Systems, *Structured Data File format (http://www.mdli.com/downloads/literature/ctfile.pdf)*.

[smarts] Inc. Daylight Chemical Information Systems, *Smiles ARbitrary Target Specification (SMARTS) (http://www.daylight.com/dayhtml/doc/theory/theory.smarts.html)*.

[smiles] Inc. Daylight Chemical Information Systems, *Simplified Molecular Input Line Entry System (SMILES) (http://www.daylight.com/dayhtml/smiles/smiles-intro.html)*.

[tc00] R. Todeschini and V. Consonni, 3-52-29913-0, Wiley-VCH, *Handbook of Molecular Descriptors*.

[wc99] S. A. Wildman and G. M. Crippen, *Prediction of Physicochemical Parameters by Atomic Contributions*, *J. Chem. Inf. Comput. Sci.*, 39, 868-873, 1999.

[wei88] D. Weinenger, *SMILES: a Chemical Language for Information Systems. 1. Introduction to Methodology and Encoding Rules*, *J. Chem. Inf. Comput. Sci.*, 28, 31-36, 1988.

[wei89] D. Weinenger, *SMILES 2: Algorithm for Generation of Unique SMILES Notation*, *J. Chem. Inf. Comput. Sci.*, 29, 97-101, 1989.

[wil01] E. L. Willighagen, *Processing CML Conventions in Java (http://www.ijc.com/abstracts/abstract4n4.html)*, *Internet Journal of Chemistry*, 4, 4, 2001.

[wy96] W. P. Walters and S. H. Yalkowsky, *ESCHER-A Computer Program for the Determination of External Rotational Symmetry Numbers from Molecular Topology*, *J. Chem. Inf. Comput. Sci.*, 36, 1015-1017, 1996.

[wz03] J. K. Wegner and A. Zell, *Prediction of Aqueous Solubility and Partition Coefficient Optimized by a Genetic Algorithm Based Descriptor Selection Method*, *J. Chem. Inf. Comput. Sci.*, 43, 1077-1084, 2003.

[wfz04a] J. K. Wegner, H. Froehlich, and A. Zell, *Feature Selection for Descriptor based Classification Models. 1. Theory and GA-SEC Algorithm*, *J. Chem. Inf. Comput. Sci.*, 44, 921-930, 2004.

[wfz04b] J. K. Wegner, H. Froehlich, and A. Zell, *Feature Selection for Descriptor based Classification Models. 2. Human Intestinal Absorption*, *J. Chem. Inf. Comput. Sci.*, 44, 931-939, 2004.

[zup89] J. Zupan, 0-471-92173-4, Wiley-VCH, *Algorithms for Chemists*.

Glossary

Here are some usefull abbreviations.

Chemoinformatics terms

Chemical Markup Language

XML standard with special definitions for molecular structures and molecular data like spectroscopic data. The advantage of CML is the expliclite data type definition and verbosity of this format. Additionally there exists a lot of XML parsers out there which faciliate to process these data type.

Comparative Field Analysis

QSAR based on grid around a molecule or a set of aligned molecules which is obtained by calculating the energy of a atom probe (often a lennard-jones potential). The with problem for this approach is the required superposition of the molecules. Topological approaches (e.g. HQSAR) can avoid this problem if we do not use 3D structures. On the otherside is it helpfull to use 3D informations make 'closer' predictions to a potentially abstract and virtual pharmacophore.

Hologram Quantitative Structure Activity Relationship

QSAR based on counted subtructure patterns. It was already shown that this approach obtains analogue results as ComFA.

Joint Commitee on Atomic and Molecular Physical Data

Format for storing chemical and physical chemical related data. Mainly used for spectroscopical data.

Maximum Common Substructure

This is a NP complete problem, also known as graph isomorphism. It can be efficiently solved for two molecules using clique detection algorithm.

Quantitative Structure Activity Relationship

General approach to calculate a relationship between molecular structures and an experimental value, e.g. solubility. It was already shown that 2D approaches based on molecular descriptors obtains analogue results as ComFA.

Structure Based Drug Design

With the structure of the target protein--ligand complex, one can better understand the structure activity relationships of existing compounds, suggest new analogs to synthesize in current series, and develop novel concepts and ideas for completely new ligand moieties. This methodology is now known as structure--based drug design.

SMiles ARbitrary Target Specification

A line notation for defining a substructure.

Simplified Molecular Input Line Entry System

A line notation for defining a molecular structure.

Index

Atom
  access, ?
    (see also Bond, access)
  add, ?
Bond
  access, ?
  add, ?
Descriptor, ?
  Atom properties, ?
  Fingerprints, ?
  Primitive/Native, 24
  Transformations, ?
    Radial Distribution function, ?
Examples, ?
Molecule, ?
  access atoms, ?
  add atoms, ?
  examples, 45