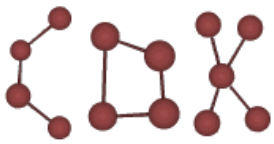

News

The Newsletter of the CDK Project — Volume 1/2, November 2004

Editorial

by Egon Willighagen and Christoph Steinbeck

Editorial

This is the second issue of CDK News, but it is the first issue with a ISSN number: 1614-7553. We would also like to announce that Uli Fechner joined the editorial board. He is contributing the recurrent *Frequently Asked Question* series.

We are also very happy that the size of CDK News has grown since the first issue. There are five articles submitted, of which only one is contributed by an editor. The number of pages has grown from eight to twelve in this issue. We hope that this issue gets downloaded and read as much as the first. That issue was downloaded over 150 times in the first four months.

It's noteworthy that several articles in this issue have CDK source fragments. Examples are given on how to customize file IO, predict ^{13}C-NMR chemical shifts, and how to use the **IteratingMDLReader**. We appreciate very much these code examples, as they highlight the development kit feature of the CDK.

Also we like to recommend to cite articles in this newsletter like one cites articles in journals too. Two examples can be found in articles in this issue. The BibTeX source code would look like:

```
@ARTICLE{WILLIGHAGEN2004,
  AUTHOR = "Willighagen, E.L.",
  TITLE = "{What's 2004 going to bring?}",
  VOLUME = "1",
  NUMBER = "1",
  JOURNAL = "CDK News",
  YEAR = "2004",
  PAGES = "3--4"
}
```

We plan to put BibTeX files online with items for all articles published in CDK News.

Last issue we announced that we are going to publish a CDK News at least twice a year, but our current goal is every four months. This means that the next issue is published in March 2005. So keep in mind that the deadline for next issue is 14 February of next year.

Finally, we would like to remark that we still have one opening in the editorial board. So if interested, please let us know.

Egon Willighagen
Radboud University Nijmegen, The Netherlands
e.willighagen@science.ru.nl
Christoph Steinbeck
Cologne University Bioinformatics Center CUBIC, *Germany*
c.steinbeck@uni-koeln.de

Contents of this issue:

Editorial 1
Customizing file IO 2
First steps in the implementation of a force field for the CDK package 4
Spok - The Spectrum Organisation Kit 5
Frequently asked Questions 6
Predictor 7
Konqueror web shortcuts to the CDK API . . . 8
Literature 9
CDK ChangeLog 10
Errata 12

Customizing file IO

This article describes how file IO in CDK can be customized to do or do not certain things when files are read or written. This is very convenient for certain kinds of file formats. The possibilities will be exemplified using the command line utility `cdk-fileconvert` and Java source code.

by Egon Willighagen

An interesting feature of file IO in CDK is that it is customizable. Before I will give all the details, let's start with a simple example: creating a Gaussian input file for optimizing the structure of methane, and let's start with an XYZ file, that is, with 'methane.xyz':

```
5
methane
C  0.25700 -0.36300  0.00000
H  0.25700  0.72700  0.00000
H  0.77100 -0.72700  0.89000
H  0.77100 -0.72700 -0.89000
H -0.77100 -0.72700  0.00000
```

Later I'll explain how IO customization is done on a source code level, but I'll use the CDK **FileConvertor** first now. In all command line options I will use the `cdk-fileconvert` shell wrapper which is equivalent to

```
java -cp cdk-all.jar \
  org.openscience.cdk.applications.FileConvertor
```

The following command will convert the XYZ file into a Gaussian input file:

```
cdk-fileconvert -o gin methane.xyz
```

The three letter code *gin* indicates the **io.program.GaussianInputWriter** which writes the Gaussian input format. Use `cdk-fileconvert --help` to see what other output formats are supported. The output will look something like

```
# b3lyp/6-31g

Created with CDK (http://cdk.sf.net/)

0 1
C 0 0.257 -0.363 0.0
H 0 0.257 0.727 0.0
H 0 0.771 -0.727 0.89
H 0 0.771 -0.727 -0.89
H 0 -0.771 -0.727 0.0
```

The writer used the default IO options in the above example. So, the next step is to see which options the writer allows.

IO options

To get a list of options for a certain IO class in CDK, use the `--listoptions`, or `-l`, argument. For the **GaussianInputWriter** this would be:

```
cdk-fileconvert -l \
  program.GaussianInputWriter
```

The output is a list of options supported by the IO class. At the time of writing, the class supported eight IO options: Basis, Method, Command, Comment, OpenShell, ProcessorCount, UseCheckPointFile and Memory. All options have defaults, and the output given earlier is created using these.

There are two mechanism that allow overwriting the default IO settings. One uses a text interface and is started with:

```
cdk-fileconvert -q all -o gin methane.xyz
```

Alternatively, you can make a Java properties file, for example called 'custom.props', of which the content may look like:

```
Basis=6-31g*
Command=geometry optimization
Comment=Job started on Linux cluster \
  on 20041010.
ProcessorCount=5
```

You can use these setting with the command:

```
cdk-fileconvert -p custom.props -o gin \
  methane.xyz
```

The nice thing about the **FileConvertor** is that it can convert a large set of files in one command. For example, try: `cdk-fileconvert -o gin *.xyz`.

io.listener.ChemObjectIOListener

At the Java source code level the above is also possible. All CDK IO classes implement the **io.ChemObjectIO** interface, which has three methods of interest (in version 1.6 of the interface):

```
public IOSetting[] getIOSettings();
public void addChemObjectIOListener\
  (ChemObjectIOListener listener);
public void removeChemObjectIOListener\
  (ChemObjectIOListener listener);
```

The method `getIOSettings()` returns the settings available for the IO class. In the **GaussianInputWriter** there are eight settings, so it will return an array of length eight. Later I will discuss the **io.setting.IOSetting** class and will first discuss the **io.listener.ChemObjectIOListener**. Classes implementing this interface have to implement only one method, and how they do that it's up to the class:

```
public void processIOSettingQuestion\
  (IOSetting setting);
```

 ISSN 1614-7553

CDK contains three implementation of this interface: **TextGUIListener**, **SwingGUIListener** and **PropertiesListener**, all in the `io.listener` package. The first was used when we used the command `cdk-fileconvert -q`. The second is a Swing implementation with the same function and is used in JChemPaint (`http://jchempaint.sf.net`), see Fig. 1. The third implementation is the one which I will use in the following source code example.

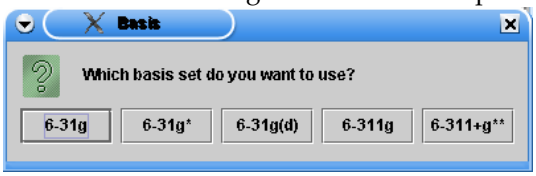

Figure 1: Customization question asked by the SwingGUIListener.

PropertiesListener

If we want to automize file conversion, it is not possible to use the interactive **TextGUIListener** and **SwingGUIListener** classes, but we need to use the **PropertiesListener** instead. This class is also used in the **FileConvertor** when the user uses a properties file with the -p option.

Consider the following source code:

```
 1 import java.io.*;
 2 import java.util.*;
 3
 4 import org.openscience.cdk.*;
 5 import org.openscience.cdk.io.*;
 6 import org.openscience.cdk.io.listener.*;
 7 import org.openscience.cdk.io.program.*;
 8 import org.openscience.cdk.io.setting.*;
 9
10 public class Example {
11
12   public static void main(String[] args) {
13     // the custom settings
14     Properties customSettings =
15       new Properties();
16     customSettings.setProperty(
17       "Basis", "6-31g*");
18     customSettings.setProperty(
19       "Command",
20       "geometry optimization");
21     customSettings.setProperty(
22       "Comment",
23       "Job started on Linux cluster " +
24       "on 20041010.");
25     customSettings.setProperty(
26       "ProcessorCount", "5");
27
28     PropertiesListener listener =
29       new PropertiesListener(
30         customSettings
31       );
32
33     try {
34       // create the writer
35       GaussianInputWriter writer =
36         new GaussianInputWriter(
37           new FileWriter(
38             new File("methane.gin")
39           )
40         );
41       writer.addChemObjectIOListener(
42         listener
43       );
44
45       XYZReader reader = new XYZReader(
46         new FileReader(
47           new File("methane.xyz")
48         )
49       );
50
51       // convert the file
52       ChemFile content =
53         (ChemFile)reader.read(
54           new ChemFile()
55         );
56       Molecule molecule =
57         content.getChemSequence(0).
58           getChemModel(0).
59           getSetOfMolecules().
60           getMolecule(0);
61       writer.write(molecule);
62       writer.close();
63     } catch (Exception e) {
64       e.printStackTrace();
65     }
66   }
67
68 }
```

The first eight lines import the packages required for this example: some classes from the default Java libraries; the core CDK classes in the `org.openscience.cdk` package; and, of course, the classes in CDK's IO packages.

The **PropertiesListener** takes a **Properties** class as parameter in its constructor. Therefore, the properties for writing the Gaussian input file are defined first on lines 14 to 26. Then, on lines 28 to 31, the **PropertiesListener** can be instantiated.

On lines 33 to 65 the file conversion happens. First the output writer is created on lines 35 to 40, specified to write to the 'methane.gin' file.

The most important lines in this example are lines 41 to 43: on those lines the **PropertiesListener** is registered with the output writer. Only then will the output be customized with the settings given earlier in the method.

On lines 45 to 49 the input reader is constructed, and the lines 51 to 62 perform the format conversion. On line 63 all **Exception**s are caught that might be thrown during the conversion process.

Customizable IO classes

I promised to get back on the **IOSetting** class. If you check again the **ChemObjectIO** and **ChemOb-**

jectIOListener interfaces, this class has the function of containing information about the IO setting, and store the active value.

CDK currently has four **IOSetting** implementations: **BooleanIOSetting**, **IntegerIOSetting**, **StringIOSetting** and **OptionIOSetting**. The first three classes can only take specific settings: yes or no, integers, and **String**s respectively.

The **OptionIOSetting** is a special class; besides one default, it contains a list of suggestions. The **ChemObjectIOListener**s, i.e. all current implementations, will offer this list of options to the user, as depicted in Fig. 1.

If you plan to add **IOSetting**s to existing or new IO classes, please have a look at the source code of the **GaussianInputWriter** or the **CMLWriter**.

I hope you appreciate the simplicity of the whole architecture.

Egon Willighagen
Radboud University Nijmegen, The Netherlands
`e.willighagen@science.ru.nl`

First steps in the implementation of a force field for the CDK package

Development of a 3D-structure model builder tool.

by Christian Hoppe

Introduction

As mentioned in the CDK News 1.1[1] we recently started to work on a CDK implementation of a force field. Force fields are widely used in the area of chemoinformatics, e.g. to optimize the 3D geometry of a molecule, in conformational search or studies in molecular dynamics[2]. A force field of a molecule describes with a set of equations the potential energy surface of that molecule. Since those equations are not based on first principles empiricaly or quantenmechanicaly precalculated data is needed. This parameterisation limits the application of force fields and over the years several different sets of equations with corresponding data have been developed[2, 3]. In general one can differentiate these force fields in two classes, one for macromolecules and one for small organic molecules (usually less than 200 heavy atoms)[2]. We are aiminig at the latter and plan to provide at least a couple of minimization alogorithms which can be used in combination with a force field to optimize the 3D-structure of a molecule.

3D-Model builder

In the process of implementing a force field method for the CDK package, we started with the development of a tool for the generation of reasonable 3D starting structures. One of the most severe problems in the generation of 3D coordinates is the layout of rings and ring systems. Therefore we followed a knowledge-based approach in collecting and storing unique ring systems (ignoring different conformations) to use them as templates in the 3D structure generation process[4]. This procedure differs from real 3D structure generating programs as e.g. CORINA[5], which stores for small ring systems conformational templates. We did not consider conformations at this step, because we only want to generate a reasonable starting structure for the force field. We downloaded a collection of small molecules as molfile from the nci databank (`http://cactus.nci.nih.gov/ncidb2/download.html`)[6]. To extract the molecule data stored in this file (249,071 3D-structures) the **IteratingMDLReader** from the CDK software package was used.

```
import org.openscience.cdk.io.iterator
  .IteratingMDLReader;

iteratingMdlReader = new IteratingMDLReader(
  new BufferedReader(new FileReader(nci_molecules))
);

Molecule molecule = null;
while (iteratingMdlReader.hasNext()) {
  molecule = (Molecule)imdl.next();
  // ...
}
```

To identify ring systems in a molecule one can use e.g. the **SSSRFinder** class. This class identifies the smallest set of smallest rings of a molecule. The **RingPartioner** class can partition this sssRingSet into RingSets of connected rings which share at least an atom, a bond or three or more atoms with another ring in the RingSet. The container class **RingSet** is able to store, handle and manipulate rings and ring systems.

```
import org.openscience.cdk.RingSet;
import org.openscience.cdk.ringsearch.SSSRFinder;
import org.openscience.cdk.ringsearch
  .RingPartitioner;

SSSRFinder sssrf = new SSSRFinder();
```

 ISSN 1614-7553

```
Vector ringSets = RingPartitioner.partitionRings(
  sssrf.findSSSR(molecule)
);
```

After a scan of all 249,071 NCI molecules, we collected 11610 unique ring systems. In order to build a 3D structure for a new modeling candidate, we first examined its molecular structure for the existence of one of the template ring systems. To map the template ring systems onto a query structure the **UniversalIsomorphismTester** class should be used. With the **QueryAtomContainer** class one is now able to configure a substructure or isomorphism search for your own requirements (e.g. search only for equivalent bond systems). If a template ring system could be mapped, its coordinates were assigned to the modeling candidate and aliphatic chains were layed out thereafter.

Currently, molecules with unknown ring systems cannot be handled by this approach.

Dr. Christian Hoppe
Universität zu Köln, Germany
christian.hoppe@uni-koeln.de

Bibliography

[1] E.L. Willighagen. What's 2004 going to bring? *CDK News*, 1(1):3–4, 2004.

[2] N.L. Allinger. *Force Fields: A Brief Introduction*, volume 2 of *Encyclopedia of Computational Chemistry*, pages 1013–1015. 1998.

[3] J.R. Maple. *Force Fields: A General Discussion*, volume 2 of *Encyclopedia of Computational Chemistry*, pages 1015–1024. 1998.

[4] J. Sadowski. *3D Structure Generation*, volume 1 of *Handbook of Chemoinformatics*, pages 231–261. 2003.

[5] Rudolph C. Sadowski J. Gasteiger, J. *Automatic Generation of 3D-Atomic Coordinates for Organic Molecules.*, volume 3 of *Tetrahedron Comput. Methodol.*, pages 537–547. 1990.

[6] Takahasi Y. Abe H. Sasaki S. Ihlenfeldt, W.D. *CACTVS: A Chemistry Algorithm Development Environment*, pages 102–105. Daijuukagakutouronkai Dainijuukai Kouzoukasseisoukan Shinpojiumu Kouenyoushishuu. 1992.

Spok - The Spectrum Organisation Kit

An introductional overview on the current state of the Spectrum Organisation Kit.

by Tobias Helmus

The Spectrum Organisation Kit is a program written in Java for the organisation and visualisation of spectral data. It is now in an alpha state and is intended to be used by experimental scientists for the analysis of spectral data, the assignment of these data to structural information and the management of larger amounts of spectral/structural data. It will be published under an open source license.

At the moment the program is able to display continuous and peak data in form of charts by using the JFreeChart library [1]. A comparable view of the peak and the continuous chart is provided, and zooming into all charts is already implemented. For giving the possibility to enter a structure for every spectrum the JChemPaint editor for 2D molecular structures [2] is embedded into the application.

Persistence of the spectral and structural data is realised by serialisation of the spectrum objects to XML files via the XStream library [3], whereas the structural information is written to CML files using the CDK **CMLWriter**. The code for the latter looks like this:

```
FileWriter out = new FileWriter("Filename");
CMLWriter cmlwriter = new CMLWriter(out);
cmlwriter.write((SetOfMolecules) structure);
cmlwriter.close();
```

where structure is the **SetOfMolecules** constructed by painting a structure in JChemPaint [2].

For the spectrum object, the `toXML()` method of the XStream package returns the java object in one XMLString which then can be written to a file with a standard **FileWriter**:

```
File file = new File("Filename");
FileWriter out;
XStream xstream = new XStream();
String xml = xstream.toXML((Spectrum) spectrum);
out = new FileWriter(file);
out.write(xml);
out.close();
```

At the time of writing the program was capable of reading spectral data in the jcamp-dx format. For this purpose the jcamp-dx library hosted on SourceForge [4] was used, which will be the reference implementation of the IUPAC JCAMP-DX spectroscopy data standard [5]. This library offers routines for reading and writing all spectrum types and data formats defined in this standard. Using the **JCampReader** is very straightforward and can easiest be done by creating a spectrum from a string containing the whole jcamp-dx file:

```
JCAMPReader jcamp = JCAMPReader.getInstance();
jcampSpectrum = jcamp.createSpectrum(jcampString);
```

If a file without a peak table is imported, a peak extraction can be performed by one of the two peak picking algorithms provided by the jcamp-dx package. Spok will shortly be downloadable in a first version at SourceForge and on our group website at `http://almost.cubic.uni-koeln.de/jrg`.

Tobias Helmus
Junior Research Group for Applied Bioinformatics
University of Cologne, Germany
`tobias.helmus@uni-koeln.de`

Bibliography

[1] The JFreeChart Homepage. `http://www.jfree.org/jfreechart/`.

[2] C. Steinbeck S. Krause, E. Willighagen. JChemPaint - Using the Collaborative Forces of the Internet to Develop a Free Editor for 2D Chemical Structures. *Molecules*, 5:93–98, 2000.

[3] The XStream Homepage. `http://xstream.codehaus.org/`.

[4] The JCAMP-DX Library Homepage. `http://sourceforge.net/projects/jcamp-dx/`.

[5] R.S. McDonald P.S. McIntyre D.N. Rutledge A.N. Davies P. Lampen, R.J. Lancashire. A Generic JCAMP-DX Standard File Format, JCAMP-DX V.6.00. 2002.

Frequently asked Questions

"Frequently asked Questions" is the first article of a series in the CDK newsletter. It is compiled of selected questions and answers that are taken from the CDK user mailing list (`cdk-user@lists.sourceforge.net`). Additionally, the author considers questions that are of general interest for CDK users. All credits for the expert answers go to the helpful developers and users who are contributing to the user mailing list.

by Uli Fechner

What exactly is the difference between the SaturationChecker, the ValencyChecker, and the ValencyHybridChecker?

All three classes provide methods for checking whether the valences of an atom are saturated with regard to a particular atom type. The oldest class is **SaturationChecker**. It is exclusively able to deal with neutral atoms. To overcome this limitation the **ValencyChecker** has been written. It not only adds the facility to handle charged atoms but also implements a slightly different algorithm. During the development of the SMILES parser another problem occured: the SMILES 'c1ccccc1' was not parsable into the aromatic 'benzene', unless an atom type list was used that considers hybridization states. Such a list, together with an enhanced algorithm compared to the **ValencyChecker**, is employed by **ValencyHybridChecker** so that hybridization states can be dealt with. The atom type lists of **SaturationChecker**, **ValencyChecker** and **ValencyHybridChecker** are accessible over the URL `http://cdk.sourceforge.net/atlists.html` or in the directory 'cdk/src/org/openscience/cdk/config' of an installed distribution.

Where can I find the API documentation in the CDK source package?

The API HTML documentation is created when the command `ant -f javadoc.xml` is executed in the CDK root directory. This creates a directory '/doc/api' in the CDK root directory that contains the complete API documentation.

Do canonical SMILES created by the CDK match the Daylight canonical SMILES?

No, they do not match each other. The CDK method implements the algorithm described by Weininger *et al.* in 1989 [1]. After several years of this publication Daylight changed their rules for generating canonical SMILES. To date their current method is not published. This lack of information does not allow the implemention of the new Daylight rules for canonical SMILES.

According to the API the class Molecule extends the class AtomContainer but it does not add anything at all to what is provided in AtomContainer. What is the purpose of Molecule?

The class **Molecule** exists to give a meaning to the content of an **AtomContainer**. Strictly speaking, a **Molecule** should not allow fragmented moieties. But this is not enforced at the moment.

How is an atom deleted from an AtomContainer?

You can remove a single atom with the method `AtomContainer.removeAtom(Atom)`. In contrast, the method `AtomContainer.removeAtomAndConnectedElectronContainers(Atom)` deletes an atom, its bonds and the lone pairs that are attached to it.

How does the fingerprint algorithm work?

The hashed fingerprint algorithm implemented by CDK closely follows the approach taken by Daylight [2]. The algorithm generates all atom sequences up to 6 atoms that are present in a molecule by a depth-first serch. Such a sequence might be a single atom (path length 1) or a longer chain of atoms like C-C-O-C=O. Then it generates a hash code for each sequence using the build-in hash function of the Java programming language. These hash codes initialize a pseudo-random number generator (RNG). The first random number between zero and 1023 generated by the RNG is taken to set the respective bit in the fingerprint. Due to the hashing function it is not guaranteed that different molecules map to different fingerprints. This is unavoidable with the myriads of possible molecular structures that are mapped on a fingerprint with only 1024 bits. Nevertheless, the algorithms' ability to distinguish between different structures is known to be quite good [3].

How is chirality handled in CDK?

Bond-based sterochemistry information can be read, e.g. from a MDL input file, and stored in the class **Bond**. **AtomParity** implements a storage container for atom-based stereochemistry. Unfortunately, at the time of this writing the atom-based stereochemistry information is not yet regarded by the SMILES parser.

Uli Fechner
Goethe-University Frankfurt, Germany
`u.fechner@chemie.uni-frankfurt.de`

Bibliography

[1] David Weininger, Arthur Weininger, and Joseph L. Weininger. SMILES 2. Algorithm for Generation of Unique SMILES Notation. *J. Chem. Inf. Comput. Sci.*, 29:97–101, 1989.

[2] Daylight theory manual. http://www.daylight.-com/dayhtml/doc/theory/theory.toc.html, January 2004.

[3] Robert D. Brown and Yvonne C. Martin. Use of structure-activity data to compare structure-based clustering methods and descriptors for use in compound selection. *J. Chem. Inf. Comput. Sci.*, 36:572 – 584, 1996.

Predictor

Predictor is a stand-alone tool for predicting ^{13}C-NMR shifts, based on NMRShiftDB data.

by Stefan Kuhn

Introduction

NMRSHIFTDB (`www.nmrshiftdb.org`, [1]) is a CDK-based project. It is an open-content web-database of nuclear magnetic resonance (NMR) data. One of its features is HOSE-code based spectrum prediction, which is available via the online interface. In order to be able to integrate this into offline applications, we now offer a stand-alone predictor, which can be downloaded at `http://www.nmrshiftdb.org/download/NmrshiftdbServlet/nmrshiftdb.sdf.zip?nmrshiftdbaction=predictor`.

Usage

The downloaded JAR file contains a java class for performing the prediction and a dump of the current data. Additionally you need the following jars to run a prediction: cdk-core.jar, cdk-extra.jar, JNL.jar, all available from nmrshiftdb cvs at `http://cvs.sourceforge.net/viewcvs.py/nmrshiftdb/nmrshiftdb/lib/`. The prediction tool is easy to use: There is only one method, taking a molecule and one of its atoms as input and giving back the predicted values. Code could look like this:

```
MDLReader mdlreader = new MDLReader(
  new FileReader(args[0])
);
Molecule mol = (Molecule)mdlreader.
  read(new Molecule());
PredictionTool predictor =
  new PredictionTool();
double[] result = predictor.
  predict(mol, mol.getAtomAt(0));
System.err.println(result[0] + " A "
  + result[1] + " A " + result[2]);
```

The result is an array of doubles, the meanings are: 0=lower limit, 1=mean, 2=upper limit calculated via confidence limits, 3=median, 4=used spheres, 5=number of values, 6=standard deviation, 7=min value, 8=max value. Please note that the java vm should be started with at least 128 MB of mem-

ory, since data are read to memory in the constructor. This enables fast predictions once the object is created, but means you should make sure PredictionTool can be garbage collected when no longer needed.

An example of using Predictor would be to predict the ^{13}C-spectrum of Aspirin (Fig. 1). With the data as of October 11, we get the following values:

Atom No.	Prediction	Spheres	SDBS
1	132.13	4	132.51
2	121.13	5	126.17
3	131.56	4	134.9
4	112.78	3	124.01
5	156.98	2	151.28
6	115.01	3	122.26
7	165.93	4	170.2
8	169.64	4	169.76
9	20.74	5	20.99

Reference values are from SDBS (`http://www.aist.go.jp/RIODB/SDBS/sdbs/owa/sdbs_sea.cre_frame_disp?sdbsno=532`). Aspirin is not yet in NMRShiftDB, as one can see from the sphere numbers; prediction would be the contained spectrum in case there was one.

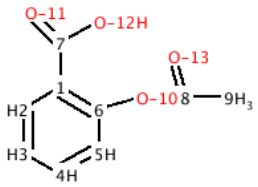

Figure 1: Structure of Aspirin.

Stefan Kuhn
Universität zu Köln, Germany
`stefan.kuhn@uni-koeln.de`

Bibliography

[1] C. Steinbeck. NMRShiftDB. *CDK News*, 1(1):2–3, 2004.

Konqueror web shortcuts to the CDK API

This short article shows how a web shortcut can be defined in Konqueror that allows quick access to the CDK Application Programming Interface (API).

by Egon Willighagen

Web shortcuts

Web shortcuts are a technology used in KDE (`http://www.kde.org`) that allow quick access to certain resources. For example, KDE 3.3 has the shortcut `dict:` defined that will look up a word in the Merriam-Webster Online (`www.m-w.com`). Some examples:

dict:atom Looks up *atom* in the Merriam-Webster Online

gg:acetic acid site:woc.sci.kun.nl Looks up *acetic acid* on `http://www.woc.sci.kun.nl/`

db:orderedlist Looks up the <orderedlist> element in the DocBook documentation

doi:10.1021/ci025584y Looks up the CDK article in the Journal of Chemical Information and Computer Sciences.

Such shortcuts can be ideal for web sites on which you often look up information. For example, I have created a web shortcut to ChemFinder (`http://chemfinder.camsoft.com/`) using the CAS registry number to retrieve information about chemicals. To search for formaldehyde I type `chemfinder:50-00-0` (see Figure 1).

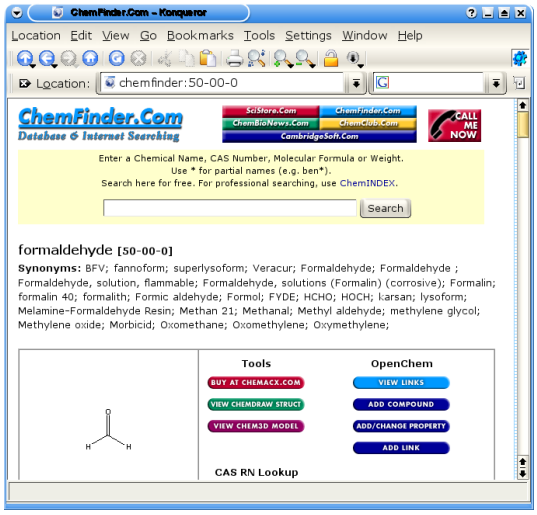

Figure 1: Web shortcut for ChemFinder.

CDK API

The CDK API is created from the Java source code using the JavaDoc utility. The API for the latest release is available on the CDK web site: `http://cdk.sf.net/api/`. You can then find the API for a spe-

cific class by browsing to the package to which the class belongs and then clicking the class name in the list of classes for that package. It would be quicker to simply type `cdk:Atom` instead.

To set up a new web shortcut in Konqueror you open the *Configure Konqueror* window accessible from the *Settings* menu (see Figure 2).

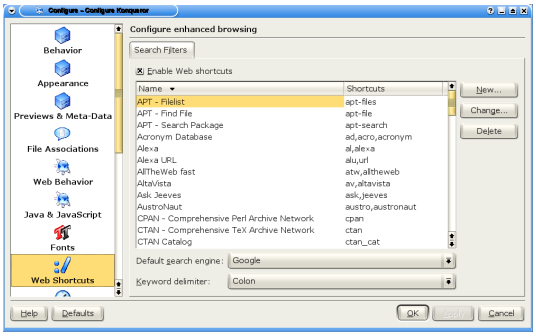

Figure 2: The settings for the CDK web shortcut.

Select the *Web Shortcuts* icon in the menu on the left, and press the `New...` button that appears on the right. Fill in the fields as depicted in Figure 3, and press `OK`.

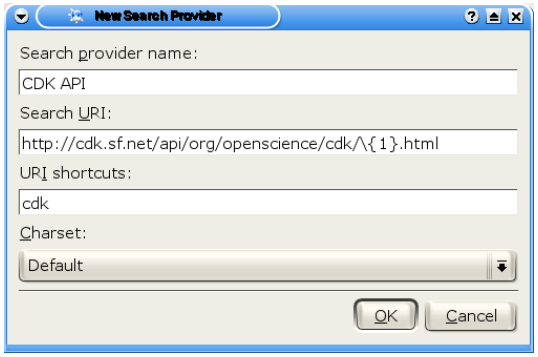

Figure 3: The *Configure Konqueror* window.

The Search URI we just defined (http://cdk.sf.net/api/org/openscience/cdk/\{1}.html) uses a simple replace to create the URL with which to look up the information; it replaces *\{1}* with whatever we type. For example, when we now type the shortcut `cdk:Atom`, the page `http://cdk.sf.net/api/org/openscience/cdk/Atom.html`. Thus, when you want to access the JavaDoc for **CMLWriter** in the `org.openscience.cdk.io` package you have to type `cdk:io/CMLWriter`. In other words, one has to use a forward slash instead of the more intuitive dot.

Your newly created Konqueror web shortcut facilitates quick access to the CDK API. I hope this example also gave ideas on how to define other web shortcuts.

Egon Willighagen
Radboud University Nijmegen, The Netherlands
`e.willighagen@science.ru.nl`

Literature

"Literature" is a recurrent column describing recently published articles that have in some way to do with CDK.

by Egon Willighagen

Since last issue four articles have been published which have some relation to CDK: they describe algorithms implemented in CDK, use CDK for research, or describe software that was compared with CDK.

Structure Diagram Generation

An article by Fricker et al.[1] discusses the process of automated 2D structure diagram generation. They propose an algorithm where diagrams are created while applying directional constraints, with the goal to clearly display the similarities in a set of molecules. It does this by orienting the common substructure in the same directions.

The **StructureDiagramGenerator** does not have this feature, but it does allow using templates which can be used to achieve a similar result: the above algorithm could be implemented in CDK by first applying a maximal common substructure search (see **UniversalIsomorphismTester**) on the whole set of molecules, and then adding this common substructure as template to the layout engine (see **StructureDiagramGenerator**).

Smallest Set of Smallest Rings

Berger et al.[2] discuss literature on the *smallest set of smallest rings* (SSSR) and related sets. These sets are properties of the molecular graph. The article gives examples where the SSSR algorithm by Figueras fails[3], which CDK implements in the **FiguerasSSSRFinder** class. CDK recently was extended with a new SSSR algorithm that addresses the problems in

Berger's article. The new algorithm can be found in **SSSRFinder**. Both CDK classes can be found in the `cdk.ringsearch` package.

Alternative development platform

The article by Vistoli et al. [4] discusses the VEGA platform which they propose as an alternative toolkit for developing chemo- and bioinformatical programs. The toolkit has a plugin and scripting features, the latter not yet available in CDK. The toolkit is written in C++ and available for many platforms. The toolkit is marketed as freeware, and uses a special VEGA customized open source license.

Pathway Analysis

Wittig et al.[5] discuss data quality of biological databases and look at the classification of small molecular compounds with the aim of getting better insight in biochemical reaction databases. The CDK is used to classify the compounds which are represented by SMILES strings on which substructure searches are applied to identify functional groups.

Egon Willighagen

Radboud University Nijmegen, The Netherlands
`e.willighagen@science.ru.nl`

Bibliography

[1] P.C. Fricker, M. Gastreich, and M. Rarey. Automated drawing of structural molecular formulas under constraints. *J.Chem.Inf.Comput.Sci.*, 44(3):1065–1078, 2004.

[2] F. Berger, C. Flamm, P.M. Gleiss, J. Leydold, and P.F. Stadler. Counterexamples in chemical ring perception . *J.Chem.Inf.Comput.Sci.*, 44(2):323–331, 2004.

[3] J. Figueras. Ring perception using breadth-first search. *J. Chem. Inf. Comput Sci.*, 36:986–991, 1996.

[4] A. Pedretti, L. Villa, and G. Vistoli. VEGA - An open platform to develop chemo-bioinformatics applications, using plug-in architecture and script programming. *J.Comput.-Aided Mol.Design*, 18:167–173, 2004.

[5] U. Wittig, A. Weidemann, R. Kania, C. Peiss, and I. Rojas. Classification of chemical compounds to support complex queries in a pathway database. *Comp. Funct. Genom.*, 5(2):156–162, 2004.

CDK ChangeLog

"CDK ChangeLog" is a series in the newsletter summarizing the changes in the CDK library since the previous newsletter.

by Egon Willighagen

A new version of CDK has been released five times since the last CDK News issue. In these four months a lot of new features have been added. The full list of changes can, as always, be reviewed on the CDK website [1].

The 20040513 Release

- An important change in this release is the addition of an hybridization field to the **AtomType** class. This new field was required for parsing hybridization information with the **SMILESParser** found in SMILES strings [2]. Consider, for example, the string "c1ccccc1" for benzene. This SMILES defines six sp^2 carbons in a ring structure. (See also the paragraph for the 20040622 release.)

- CDK contains a few atom type lists, which used to be in a custom XML format, but have now been transformed into CML2 format [3]. In addition, these lists are now also accessible from the CDK website [4].

- Furthermore, a few classes have been relocated to other packages which was proposed some time ago. This was done to clean up the CDK API. Among these are the following relocations: **Primes** and **RandomNumbersTool** have been moved to the `org.openscience.cdk.math` package; **ConnectivityChecker** to `org.openscience.cdk.graph`, and **Projector** to `org.openscience.cdk.geometry`. The latter class projects 3D coordinates in a two-dimensional plane.

- In the **core** module the `getPoint2D()` and methods named similarly have been renamed to `getPoint2d()` (with a lower case *d*) to match the **Point2d** class from the *vecmath* library.

- Finally, this release fixed many bugs in, amongst others, the following classes: **Renderer2D**, **Controller2D** and **SMILESParser**.

The 20040622 Release

- This release has many changes. One of them is in the **Renderer2D** which can be customized in more detail. For example, the font used for displaying text can be defined, and anti-aliasing can be turned on and off, but is now turned on by default, resulting in much nicer graphics. The latter, however, required a API change. The **Renderer2D** now takes a **Graphics2D** object as parameter, but in recent virtual machines only those are used.

- The `cdk.libio.jmol` was updated with a Jmol to CDK conversion method that matches the architecture in Jmol 10 [5]. As a consequence, Jmol IO classes can now be used, as is exemplified by the new **JMEReader** and **MOPACReader** which wrap around Jmol IO classes.

- An important new class is **AtomParity** which can hold atom based stereochemistry information. Instances can be stored in the **AtomContainer** class. It defines an atom parity with a number and a ordered sequence of four atoms.

- Other new things in CDK include an INChI reader [6], an improved command line interaction for FileConvertor, and the new methods **MFAnalyser** `.removeHydrogensPreserveMultiplyBonded()` and **CDKEditBus** `.setChemModel(Reader)`.

- An important change happened to the API of CDK: the **IsotopeFactory** and **AtomTypeFactory** have been moved from the `cdk.tools` to the `cdk.config` package.

- Furthermore, this release has a large set of bug fixes. The most important one is a redesign of the **SMILESParser**. It now correctly parses SMILES strings with hybridization information, e.g. 'c1ccccc1': first it parses the string into a six-ring of sp^2 carbons. Then it adjusts the bond orders, and in a last step it detects aromaticity.

- Other bug fixes were applied to, amongst other, these classes: **Mol2Reader**, **SmilesGenerator**, **Renderer2D**, **FileConvertor** and **CMLWriter**.

The 20040626 Release

- This version was released shortly after the 20040622 release to solve the problem with the broken build file; some configuration files were missing from the distribution. It also fixes a problem with writing and reading partial atom charges in CML2.

The 20040917 Release

- Likely the most important addition to this release is the new algorithm for detection of the smallest set of smallest rings (SSSR). For small molecules it is slower than the **FiguerasSSSRFinder**, but it addresses a number of situations where the Figueras algorithm fails.

- Other additions in this release include an INChI reader for its plain text format (a reader for the XML based INChI format was added in the 20040622 release) and the `.getAllIDs()` methods to the `tools.manipulator` classes.

- A set of code clean ups and JavaDoc fixes have been applied which were found using PMD [7] and DocCheck [8]. Also bugs have been fixed in **IDCreator**, **MDLWriter** and **ChemObject**`.getProperties()`.

- A very important API change occurred in the **core** module classes. The `clone()` methods now, consistently, do a deep clone, while the new `shallowCopy()` methods do a shallow copy of the **core** classes.

- Another high impact API change is a full rewrite of the **Crystal** class which now uses **Vector3d** for representation of the unit cell axes.

The 20041015 Release

- The October release only contains bug fixes. Some important ones are in the **UniversalIsomorphismTester** and deal with using `isomorphism.matchers` classes with the isomorphism tester.

- Another important fix is in the **core** classes, and deals with the propagation of **ChangeEvents** in the core classes.

- A final note is that this release successfully compiles with Java 5.0 from Sun, but that writing CML with that virtual machine is not working as expected; the CML JUnit tests fail with that Java version. The problem is the incompatibility with the XML functionality in Java 5.0 and the CMLDOM library.

Egon Willighagen
Radboud University Nijmegen, The Netherlands
`e.willighagen@science.ru.nl`

Bibliography

[1] CDK ChangeLog. http://cdk.sourceforge.net/changelog.html.

[2] David Weininger, Arthur Weininger, and Joseph L. Weininger. SMILES 2. Algorithm for Generation of Unique SMILES Notation. *J. Chem. Inf. Comput. Sci.*, 29:97–101, 1989.

[3] P. Murray-Rust and H.S. Rzepa. Chemical markup, xml, and the world wide web. 4. cml schema. *Journal of Chemical Information and Computer Sciences*, 43(3):757–772, 2003.

[4] AtomType Lists. http://cdk.sourceforge.net/atlists.html.

[5] Jmol. http://www.jmol.org/.

[6] IUPAC-NIST Chemical Identifier. http://www.iupac.org/projects/2000/2000-025-1-800.html.

[7] PMD. http://pmd.sourceforge.net/.

[8] Doc Check Doclet. http://java.sun.com/j2se/javadoc/doccheck/.

Errata

The previous issue's "What's 2004 going to bring?" had an incorrect webreference for the Ghemical project. The correct link should be http://www.uku.fi/~thassine/ghemical/.