chem-bla-ics

cdk2024 #3: an unexpected downstream project

Egon Willighagen

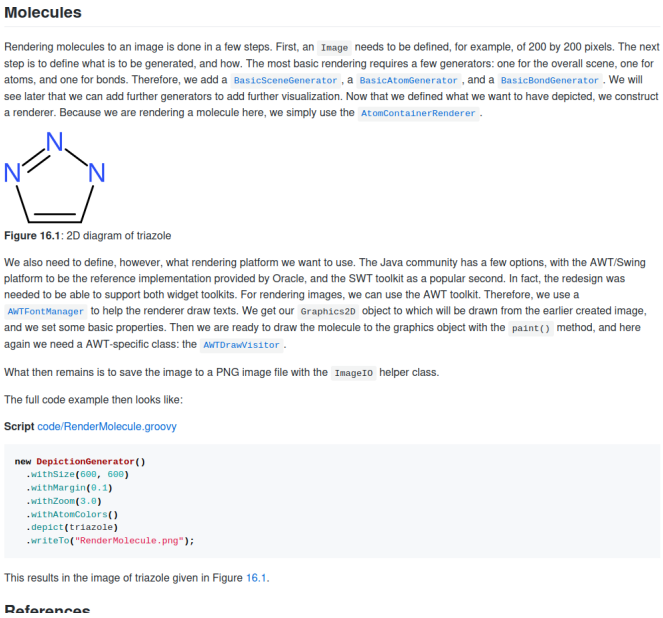
Published June 16, 2024

Citation

Willighagen, E. (2024). cdk2024 #3: an unexpected downstream project. *Chem-bla-ics*. https://doi.org/10.59350/dtfq8-5x011

Keywords

Cdk, Grant, Cdk2024



Molecules

Rendering molecules to an image is done in a few steps. First, an `Image` needs to be defined, for example, of 200 by 200 pixels. The next step is to define what is to be generated, and how. The most basic rendering requires a few generators: one for the overall scene, one for atoms, and one for bonds. Therefore, we add a `BasicSceneGenerator`, a `BasicAtomGenerator`, and a `BasicBondGenerator`. We will see later that we can add further generators to add further visualization. Now that we defined what we want to have depicted, we construct a renderer. Because we are rendering a molecule here, we simply use the `AtomContainerRenderer`.

Figure 16.1: 2D diagram of triazole

We also need to define, however, what rendering platform we want to use. The Java community has a few options, with the AWT/Swing platform to be the reference implementation provided by Oracle, and the SWT toolkit as a popular second. In fact, the redesign was needed to be able to support both widget toolkits. For rendering images, we can use the AWT toolkit. Therefore, we use a `AWTFontManager` to help the renderer draw texts. We get our `Graphics2D` object to which will be drawn from the earlier created image, and we set some basic properties. Then we are ready to draw the molecule to the graphics object with the `paint()` method, and here again we need a AWT-specific class: the `AWTDrawVisitor`.

What then remains is to save the image to a PNG image file with the `ImageIO` helper class.

The full code example then looks like:

Script code/RenderMolecule.groovy

```
new DepictionGenerator()
    .withSize(600, 600)
    .withMargin(0.1)
    .withZoom(3.0)
    .withAtomColors()
    .depict(triazole)
    .writeTo("RenderMolecule.png");
```

This results in the image of triazole given in Figure 16.1.

References

Copyright

chem-bla-ics

In the CDK2024 grant we wrote about updating various software projects using the Chemistry Development Kit. We even wrote that "[r]equired API changes will be publicly shared and disseminated with the Groovy Cheminformatics with the Chemistry Development Kit book (egonw.github.io/cdkbook/)". The *Groovy Cheminformatics with the Chemistry Development Kit* book is a project that has run since 2009.

```
commit c5cbf9b5dd49baf582afc595c9cbafc714c5199f
Author: Egon Willighagen
Date:   Fri Apr 10 12:34:42 2009 +0200

    Initial copy of the current draft; converted into separate project for
easier branching
    for tunes of the book for workshops and sorts
```

The original version was in LaTeX and sold online via Lulu.com. Because all code examples were run (the first public edition had 72 pages with 75 code examples), like RMarkdown of Jupyter Notebooks by design, I was able to make many releases. The big advantage of this was that when API changes happened, this would be visible by code not compiling or by output changing.

At some point I open sourced the book (doi:10.6084/M9.FIGSHARE.2057790.V1) and then realized that I can convert the book to Markdown:

```
commit 2630699aa280200188f2ae9ef3f0698964926752
Author: Egon Willighagen
Date:   Mon Dec 24 16:59:14 2018 +0100

    Create chapter3.md
```

This is the version available at egonw.github.io/cdkbook/ for some time now. So, now that for SMARTCyp I need to update the visualization, I went book to my book of code examples (I have a collection of more than 200 examples), but then found that the chapter on Depiction was missing. I was not looking forward to this, because I know that the code examples predate a massive improvement by John Mayfield of the rendering stack and I never got around to see if the examples from the book work well enough with that new API (one is actually updated).

That is when I realized that the *Groovy Cheminformatics* book actually also is a downstream project that needs updating. I have been doing this already and it's fairly smooth so that I did not think of including it in the grant, other than updating the Migration chapter. I now had enough time to dive into this project. I need that, because the goal of the project is also to learn about all the meta science aspects of project maintenance, roles, communication, etc. Therefore also this blog post: we need a track record, to collect data.

Anyway, porting the first script went fairly easy, but I am now running into a stacktrace:

chem-bla-ics

```
Processing  RenderSelection.groovyin
doing RenderSelection.out ...
org.codehaus.groovy.control.MultipleCompilationErrorsException: startup
failed:
/home/egonw/var/Projects/hub/cdkbook-source/code/RenderSelection.groovy: 39:
unable to resolve class ExternalHighlightGenerator
 @ line 39, column 16.
   generators.add(new ExternalHighlightGenerator());
                  ^
org.codehaus.groovy.syntax.SyntaxException: unable to resolve class
ExternalHighlightGenerator
 @ line 39, column 16.
```

That brings us to the task of how to find where that class is coming from, which happens to be something I already had to write up for up for `RingSearch`. Dependency galore.

References

- 10.6084/M9.FIGSHARE.2057790.V1