24. Migration

Going from one CDK release to another brings in API changes. While the project tries to keep the number of changes minimal, these are inevitible. This chapter discusses some API changes, and shows code examples on how to change your code. Section 24.1 discusses changes from 1.4 to 2.0, Section 24.2 discusses changes from 1.2 to 1.4, Section 24.3 discusses changes from 1.0 to 1.2, and Section 24.4 discusses changes from 1.0 to 1.4.
The set of changes include changed class names. For example, the CDK 1.2 class MDLWriter is now called MDLV2000Writer to reflect the V2000 version of the MDL formats.

24.1. CDK 1.4 to 2.0

This section highlights the important API changes between the CDK 1.4 and 2.0 series.

24.1.1. Removed classes

Several classes have been removed in this version, for example, because they are superceeded by other code or were considered redundant.

Removal of the NoNotify interface implementation

The NoNotificationChemObjectBuilder and the matching implementation classes are removed. Please use the SilentChemObjectBuilder instead. The same, of course, applies to all implementation classes. For example, NNMolecule is removed.

Removal of IMolecule

The IMolecule interface and all implementing classes have been removed. They were practically identical in functionality to the IAtomContainer interface, except the implication that the IMolecule was for fully connected structures only. This separation was found to be complicated, and was therefore removed. Please use the IAtomContainer interface instead.

24.1.2. Renamed classes and methods

Rename of IteratingMDLReader to IteratingSDFReader

Strictly speaking the MDL files span a set of files and a SD file is different from a molfile. This is reflected in the reader name change: IteratingMDLReader is now called IteratingSDFReader.

CDKAtomTypeMatcher.findMatchingAtomTypes()

The method findMatchingAtomTypes of the CDKAtomTypeMatcher gained a 's' and was previously called findMatchingAtomType. The new name is more consistent, reflecting the fact that it perceives atom types for all atoms.

24.1.3. Changed behavior

Some classes and methods have the same API, but have slightly different behavior as before. For example, the SmilesGenerator now requires that all atoms have implicit hydrogen counts set. This can be done with the CDKHydrogenAdder as explained in Section 14.4.

24.1.4. Constructors that now require a builder

The advantage of the builders in the CDK is that code can be independent of data class implementations (and we have three of them in CDK 1.6, at this moment). Over the past years more and more code started using the approach, but that does involve that more and more class constructors take a IChemObjectBuilder. CDK 1.6 has two more constructors that now take a builder.

DescriptorEngine

The DescriptorEngine constructor is changed to now take a IChemObjectBuilder which is needed to initialize descriptor instances.

SMARTSQueryTool

The second constructor that now needs a IChemObjectBuilder is that of the SMARTSQueryTool. Here it is passed on to the SMARTSParser which needs it for its data structure for the matching.

ModelBuilder3D

The getInstance() method of the ModelBuilder3D class now also requires a IChemObjectBuilder. See Section 14.6.

CDKAtomTypeMatcher

A significant change in the CDKAtomTypeMatcher behavior is that it now returns a special 'X' atom type when no atom type could be perceived. See Section 12.2.4.

24.1.5. Static methods that are no longer

Some previously static methods are no longer, and now require the instantiation of the class.

UniversalIsomorphismTester

The UniversalIsomorphismTester is an example class that now needs to be instantiated. However, the class is easy to instantiate. For example:

Script 24-1: code/Isomorphism.groovy

```
butane = MoleculeFactory.makeAlkane(4);
isomorphismTester = new UniversalIsomorphismTester()
println "Is isomorphic: " +
  isomorphismTester.isIsomorph(
    butane, butane
  )
```

24.1.6. IsotopeFactory

A major API change happened around the IsotopeFactory. Previously, this class was used to get isotope information, which it gets from an configurable XML file. This functionality is now available from the XMLIsotopeFactory class. However, to improve the speed of getting basic isotope information as well as to reduce the size of the core modules, CDK 1.6 introduces a Isotopes class, which contains information extracted from the XML file, but is available as a pure Java class. The APIs for getting isotope information is mostly the same, but the instantiation is much simpler, and also no longer requires an IChemObjectBuilder:

Script 24-2: code/IsotopesDemo.groovy

```
isofac = Isotopes.getInstance();
uranium = 92;
for (atomicNumber in 1..uranium) {
  element = isofac.getElement(atomicNumber)
}
```

24.1.7. IFingerPrinter

The IFingerprinter API was changed to accomodate for two types of fingerprints: the bit fingerprint, outlined by the IBitFingerprint interfaces, and the count fingerprint, defined in the ICountFingerprint interface. The IFingerprinter interface now defines getRawFingerprint(IAtomContainer), getCountFingerprint(IAtomContainer), and getBitFingerprint(IAtomContainer). These methods returns various kind of fingerprints. For example, getRawFingerprint(IAtomContainer) returns a Map with Strings representing the various parts of the fingerprint as well as the matching count, and it is this map that is used as input to the getCountFingerprint(IAtomContainer) method, which returns this information as a ICountFingerprint implementation. If the count for each bit is not important, the getBitFingerprint(IAtomContainer) method can be used, which returns a IBitSetFingerprint implementation.

Because the previous Fingerprinter interface did not include the counting of how often a bit was set, implementing the new getRawFingerprint(IAtomContainer) method will likely take some effort, but the other two methods can in many cases just wrap other methods in the class, as shown in this example code:

Script 24-3: code/FingerprinterMigration.java

```
  public ICountFingerprint getCountFingerprint(
    IAtomContainer molecule
  ) throws CDKException {
    return new IntArrayCountFingerprint(
      getRawFingerprint(molecule)
    );
  }
  public IBitFingerprint getBitFingerprint(
    IAtomContainer molecule
  ) throws CDKException {
    return new BitSetFingerprint(
      getFingerprint(molecule)
    );
  }
}
```

When implementing the IFingerprinter interface, it is easiest to also extend the abstract AbstractFingerprinter class.

24.1.8. SMILESGenerator

The SMILES stack is replaced in this CDK version. This introduces a few API changes, outlined here. The new code base is much faster and more functional that what the CDK had before. Below are typical new SmilesGenerator API usage.
Generating unique SMILES is done slightly differently, but elegantly:

Script 24-4: code/UniqueSMILES.groovy

```
generator = SmilesGenerator.unique()
smiles = generator.createSMILES(mol)
println "$smiles"
```

Because SMILES with lower case element symbols reflecting aromaticity has less explicit information, it is not my suggestion to use. Still, I know that some of you are keen on using it, for various sometimes logical reasons, so here goes. Previously, you would use the setUseAromaticityFlag(true) method for this, but you can now use instead:

Script 24-5: code/AromaticSMILES.groovy

```
generator = SmilesGenerator.generic().aromatic()
smiles = generator.createSMILES(mol)
println "$smiles"
```

24.2. CDK 1.2 to 1.4

This section highlights the important API changes between the CDK 1.2 and 1.4 series.

24.2.1. Creating objects with an IChemObjectBuilder

One very prominent changes is how the IChemObjectBuilder works (see Section 10).
The CDK 1.2 code:

```
IChemObjectBuilder builder =
  DefaultChemObjectBuilder.getInstance();
IMolecule molecule = builder.newMolecule();
molecule.addAtom(builder.newAtom("C"));
```

looks now like:

Script 24-6: code/MigrationNewBuilder.groovy

```
IChemObjectBuilder builder =
  DefaultChemObjectBuilder.getInstance();
IAtomContainer molecule = builder.newInstance(
  IAtomContainer.class
);
molecule.addAtom(
  builder.newInstance(IAtom.class, "C")
);
```

Please note that the builder.newInstance() method may actually return null. This is not the case for the DefaultChemObjectBuilder, or the alternative SilentChemObjectBuilder builder, but future releases may have dedicated builders that do have such functionality. However, these builder would not supposed to be used for building molecules anyway.
The general patterns of newInstance() calls is that the first argument is the interface for which you want an instance. All further parameters are passed as parameters for the object's constructor. The builder maps the input to appropriate class constructors. To know what parameters you can pass when instantiating an IAtom with the DefaultChemObject-Builder, you would look at the constructor of Atom. Therefore, we can also call:

Script 24-7: code/MigrationNewBuilder2.groovy

```
IAtom atom = builder.newInstance(
  IAtom.class, "C", new Point2d(0,0)
);
```

A full overview of the available IChemObjectBuilders is found in Chapter 10.

24.2.2. Implicit hydrogens

A second API change lies deep in the IAtom interface. To reflect more accurately the meaning of the method, the IAtomType.getHydrogenCount() has been renamed to IAtomType.getImplicitHydrogenCount(), and likewise the setter methods. The 1.2 code:

```
carbon.setHydrogenCount(4);
```

has to be updated to:

Script 24-8: code/MigrationImplicitHydrogens.groovy

```
carbon.setImplicitHydrogenCount(4);
```

Yeah, that is a simple one. Just to make clear, in both versions the count reflected the number of implicit hydrogens. The getHydrogenCount() suggested, however, to return the number of all hydrogens attached to that atom, that is, the sum of implicit and explicit hydrogens. See also Section 3.4.

24.2.3. Iterating readers

Up to CDK 1.4.7 the IIteratingChemObjectReader implementations had a next() method that returned a IChemObject class. Depending on the file format, this could be a IChemModel or an IAtomContainer. The interface now uses generics, however, and the next() method now returns an IAtomContainer or IChemModel, making casting in the user code obsolete.

24.3. CDK 1.0 to 1.2

This section highlights the important API changes between the CDK 1.0 and 1.2 series.

24.3.1. MFAnalyser

Version 1.2 removed the MFAnalyser class in favor of a more elaborate framework to handle molecular formulas. Please refer to Sections 3.3.3 and 14.7 for more detail on the new framework.

24.4. CDK 1.0 to 1.4

As discussed elsewhere, the CDK 1.2 series was released without rendering support (see Chapter 15), because the rendering engine was undergoing a complete rewrite. A good part of this has been finished, and a new rendering API is now available. The changes are too extensive to discuss here, and the reader is refered to new API discussed in Chapter 15 and start from scratch.