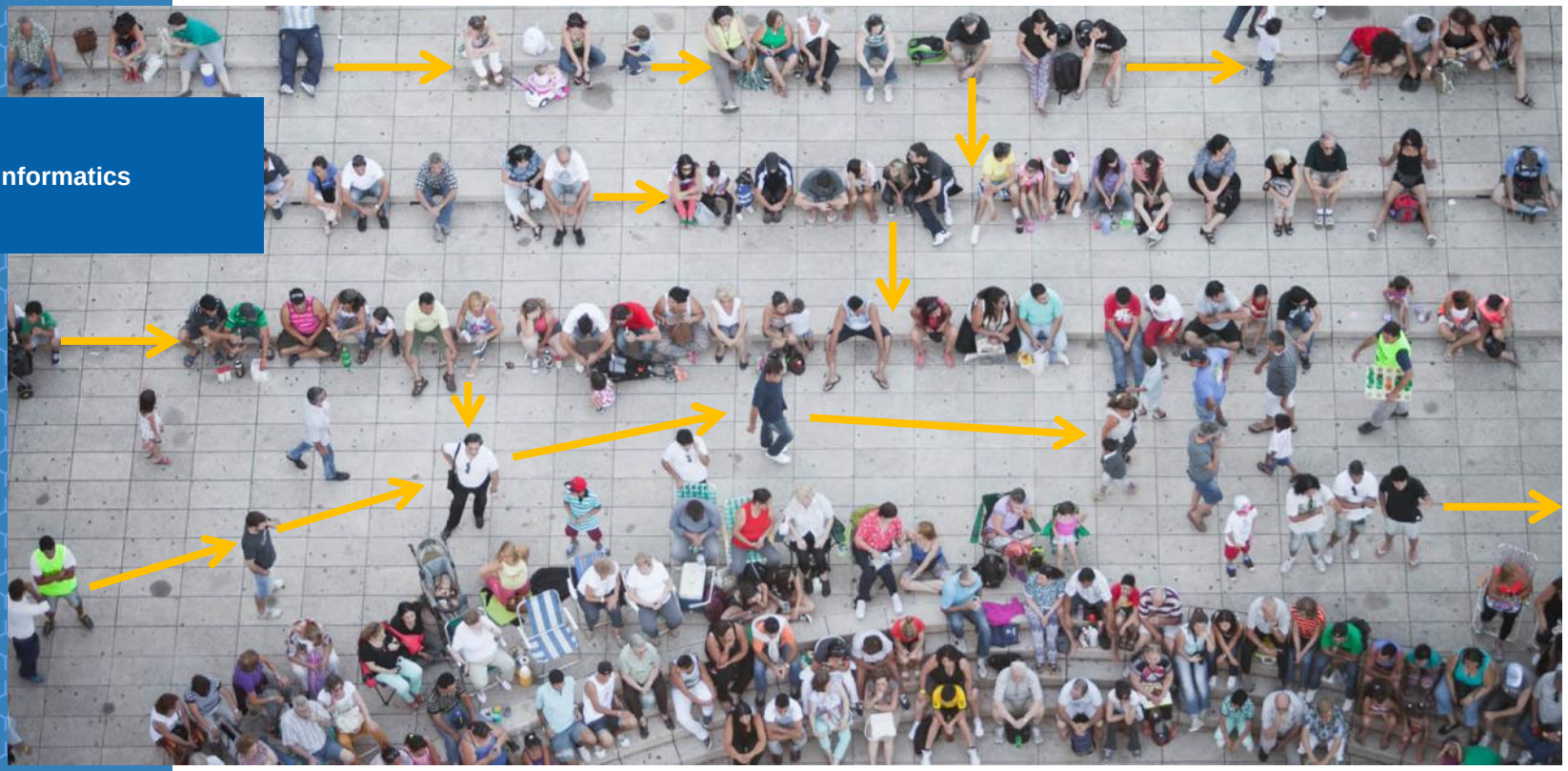
NIBR Informatics



How to Develop New RDKit Nodes for KNIME Workflows

Manuel Schwarze, Senior Principal Software Engineer
Basel, Switzerland
October 26-28, 2016

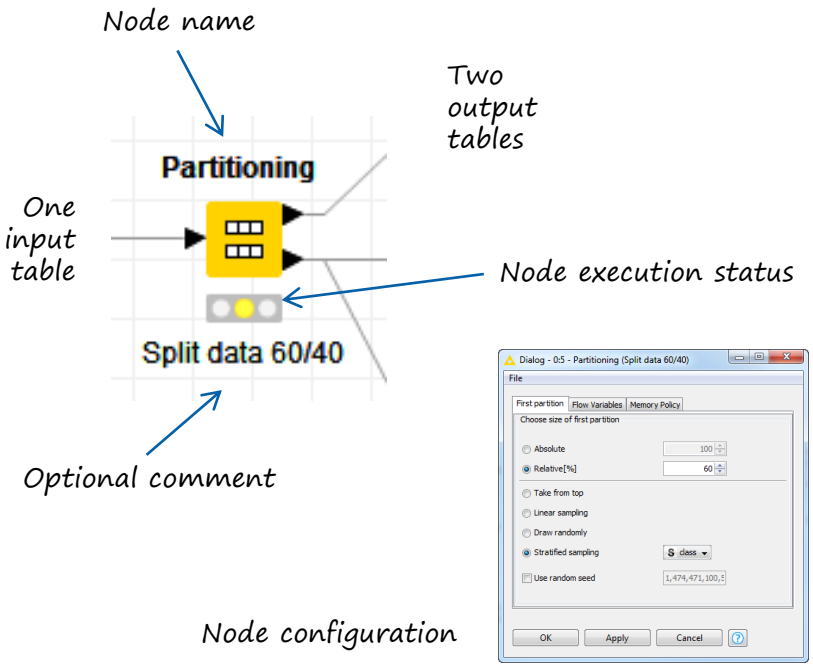
NOVARTIS

Agenda

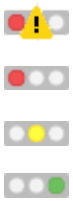
- What is KNIME and what can I do with it?
- What has RDKit to do with KNIME?
- How is software developed in the KNIME Community?
- How easy it would be if I had a wizard ...
- ... Ok, I got a wizard. Now what?
- What can I do to go beyond the limit?

What is KNIME and what can I do with it?

- One of the top 3 leading advanced analytics platforms (Gartner Magic Quadrant 2016)
- Free, open-source, large developer community
- Large set of workflow nodes
- Open architecture based on Eclipse Java platform:
  - KNIME Core and KNIME Labs Extensions
  - KNIME Community Extensions
  - KNIME Partner Extensions
  - KNIME Commercial Extensions
- Used widely in life science, government and service sectors



A KNIME node – The smallest «piece of software»



Node has a wrong configuration or failed execution

Node needs to be configured or no input data

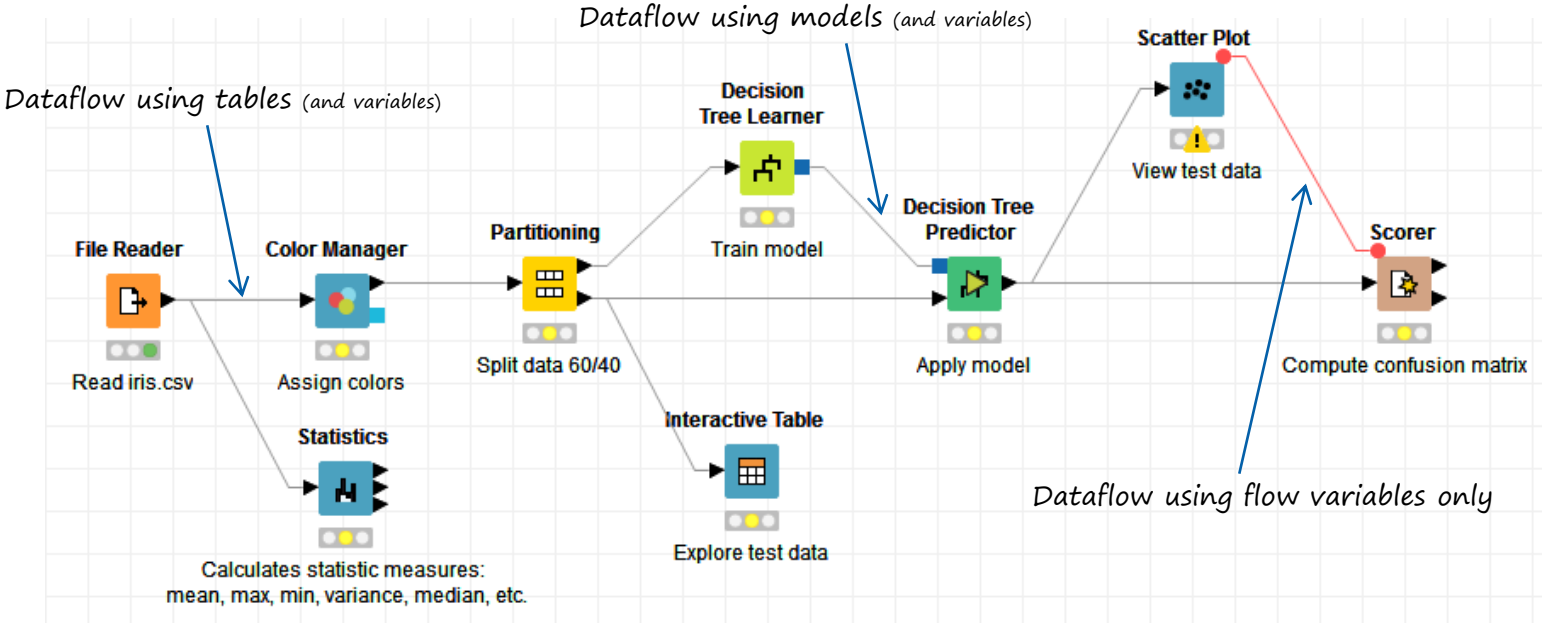
Node is ready to be executed

Node has been successfully executed

Possible node execution statuses

What is KNIME and what can I do with it?

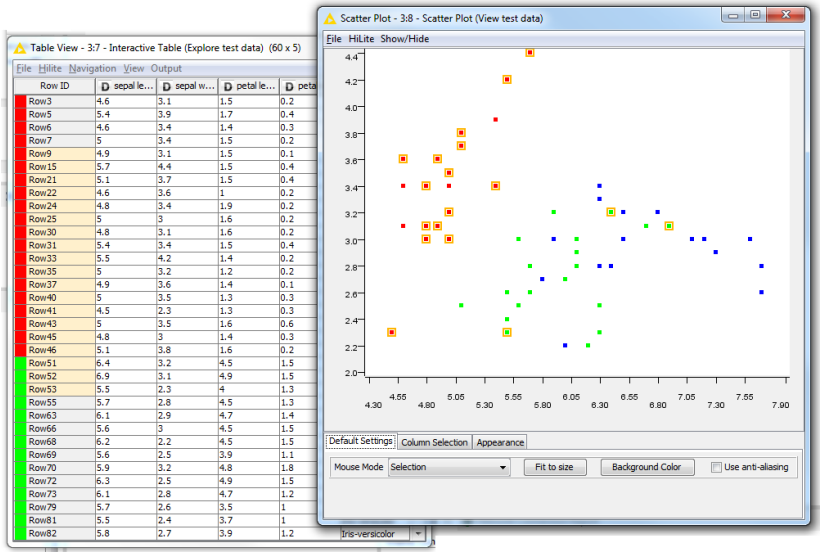
- KNIME workflows:
  - Read your data from anywhere
  - Analyse and change your data
  - Use loops, conditions, error handling
  - Visualize your data
  - Document what you do while you are doing it
  - Write data anywhere you want to



An example workflow – A collection of KNIME nodes

What is KNIME and what can I do with it?

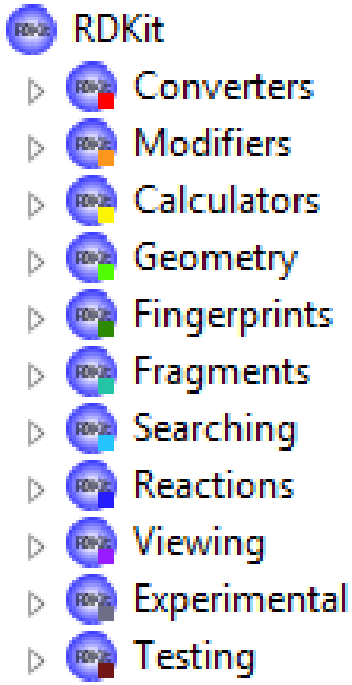
- Main advantages over «just scripting»:
  - Step-by-step execution
  - Reviewing of input and output for each step
  - Resetting just a few last steps to try another configuration
  - Combination of multitude of data sources and tools
  - Easy visualization of data
  - Documentation of logic behind it
  - Mechanisms to hide complexity (experts prepare, users run it):
    - KNIME Meta Nodes
    - KNIME Web Portal (Commercial)



Highlighting examples across KNIME nodes

What has RDKit to do with KNIME?

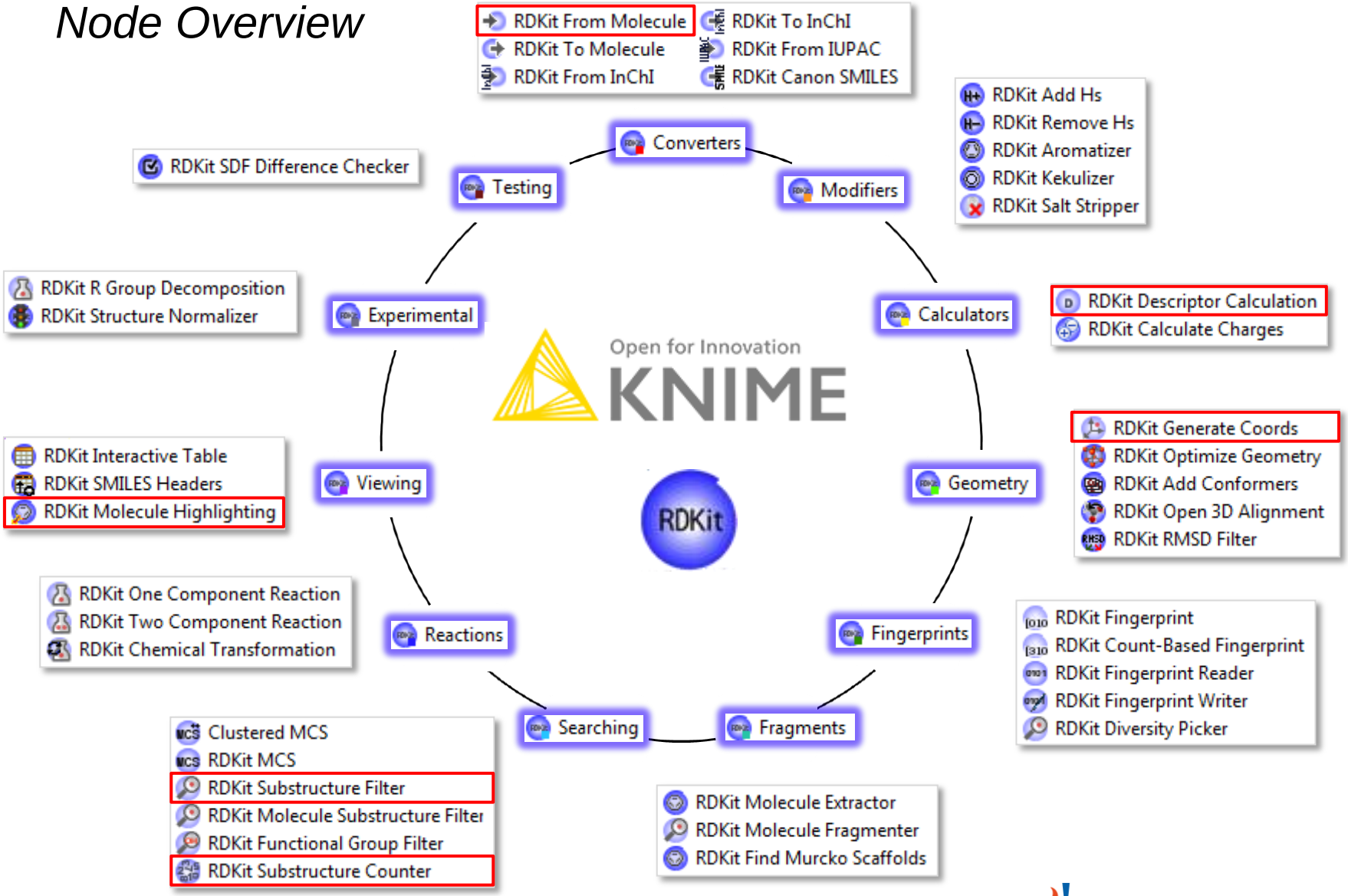
- RDKit brings chem-informatics functionalities into KNIME:
  - Structure rendering
  - RDKit molecule and chemical reaction as internal data models
  - Functionalities based on molecules, reactions and fingerprints
- RDKit nodes are developed as part of the KNIME Community
- KNIME is based on Java
- RDKit is based on C/C++ and Python
- Many C/C++ APIs made available as Java API (via SWIG automatisms)
- Only C/C++ functionalities available through Java API can be used in KNIME RDKit nodes
- Currently 11 categories of RDKit nodes with 41 nodes, 16 of them capable of «streaming» (big data)



Categories of RDKit nodes

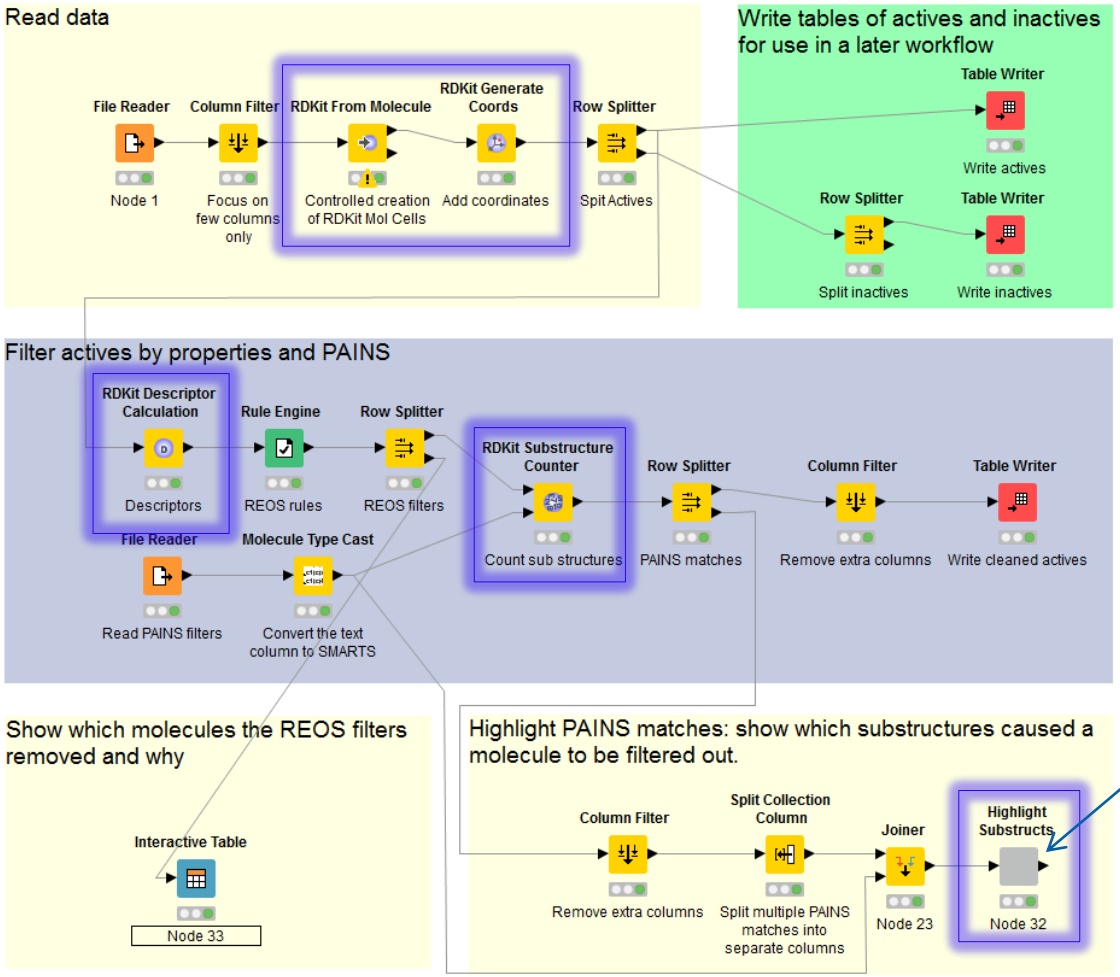
What has RDKit to do with KNIME?

Node Overview



What has RDKit to do with KNIME?

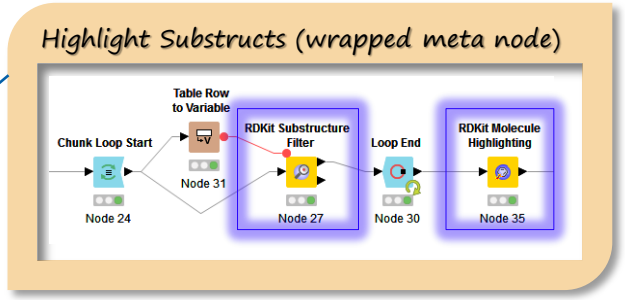
Demo workflow: High-throughput screening hitlist triaging



Demonstration: Removing molecules that may have interfered with the assay (using the PAINS filters) or that have undesirable physicochemical properties (using the REOS filters).

Uses the RDKit descriptor calculator, substructure filters, and structural highlighting.

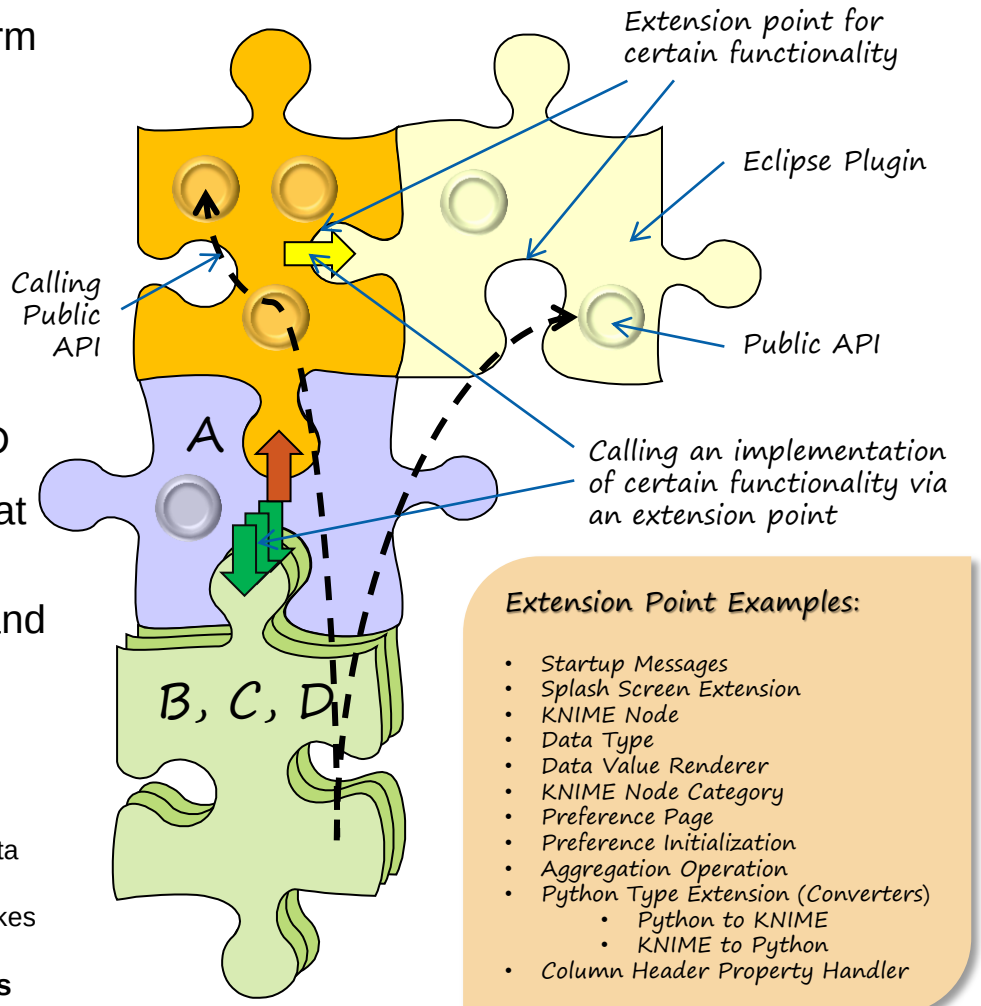
This is a KNIME translation of the first step of the TDT2014 tutorial created by Sereina Riniker and Greg Landrum (http://submit.teach-discover-treat.org/download/2014/1_Sereina_Riniker and https://github.com/sriniker/TDT-tutorial-2014).



Publicly available: knime://EXAMPLES/_Old%20Examples%20(2015%20and%20before)/099_Community/01_RDKit/09901004_Filtering%20A%20Hitlist

How is software developed in the KNIME Community?

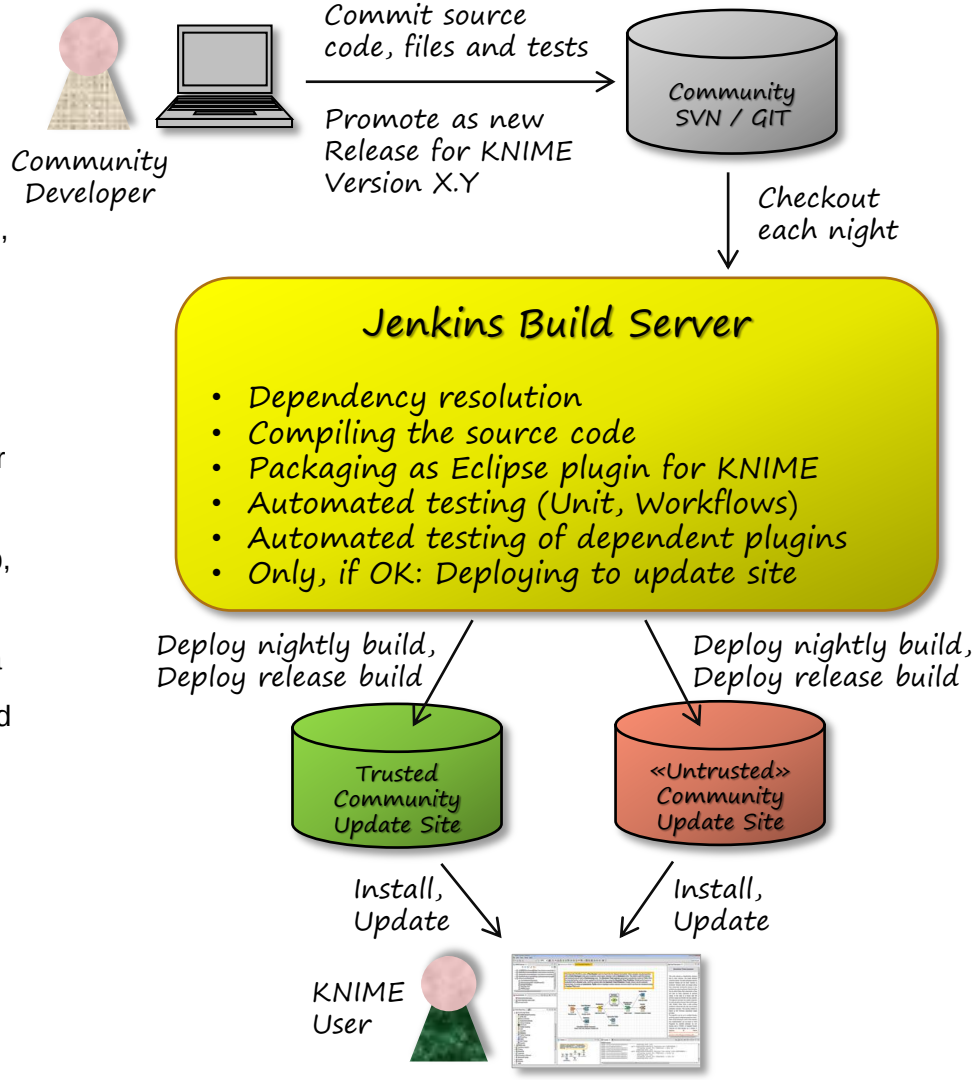
- KNIME is based on Eclipse Java platform which is designed for extensions
- Extension points can be defined by a plugin A
- Other plugins B, C, D can implement these extension points and register themselves with Eclipse
- Plugin A can now make use of functionality provided by plugins B, C, D
- KNIME offers many extension points that other plugins may implement
- Plugins can declare APIs to be visible and usable by other plugins
- Example:
  - **RDKit Feature for KNIME combines multiple plugins:**
    - **RDKit Types plugin:** Access to RDKit binaries, data type and renderer implementations, API is public
    - **RDKit Nodes plugin:** Nodes implementations, makes use of data types and RDKit functionality
  - **Erlwood Knime Open Source Core and Vernalis PMI Plugin:** Node implementations with a dependency to RDKit Types to use RDKit functionality



Extension points in Eclipse Java platform

How is software developed in the KNIME Community?

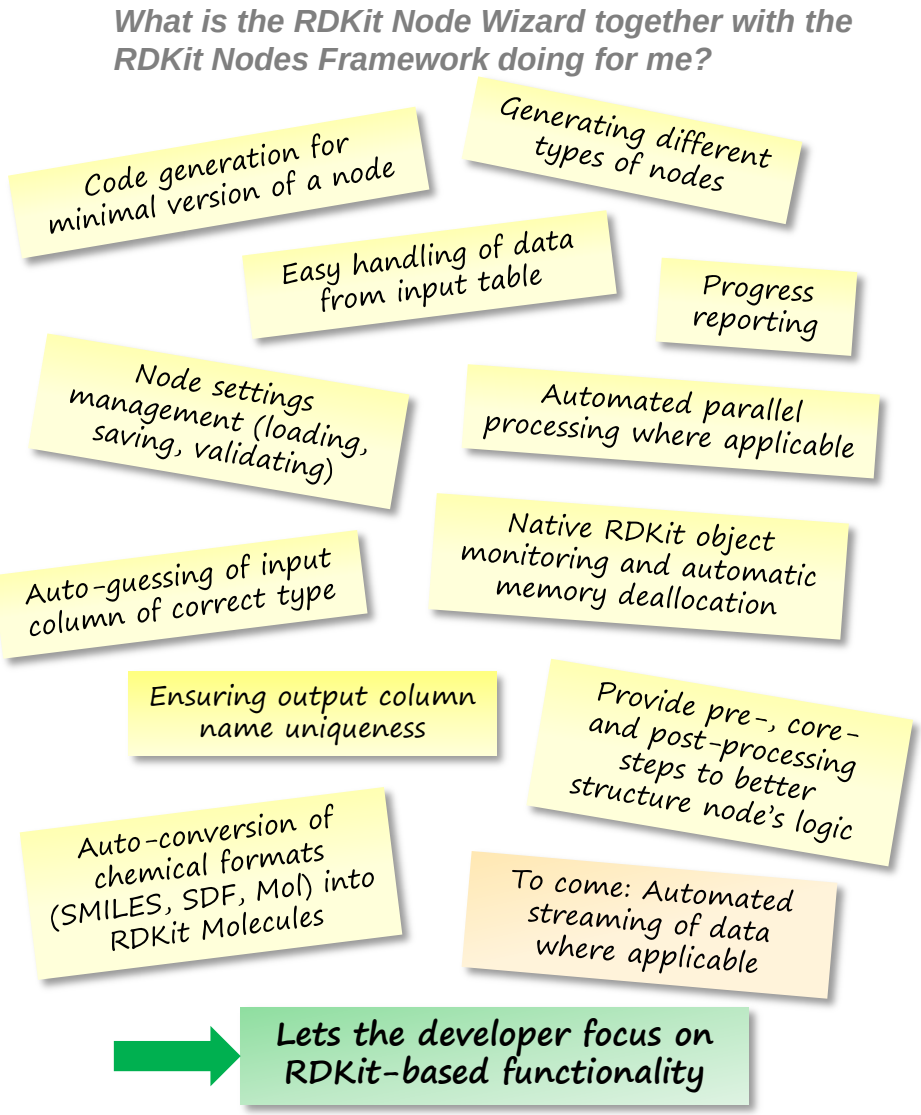
- KNIME users and companies may contribute their KNIME extensions to the public as open-source
- Wide range of topics:
  - **Cheminformatics** (RDKit, CheS-Mapper, EMBL-EBI, Applicability Domain, Erlwood, EU-OPENSCREEN, Indido, CDK, Lhasa Metabolism Feature, Lhasa Model Building, OCHEM (QSAR modelling), Vernalis PDB)
  - **Bioinformatics** (HCS, NGS, openBIS, OpenMS, Protein Inference Algorithms)
  - **Image processing** (Integrations of CellProfiler, Clear Volumn, Ilastik, ImageJ, OMERO, Python and Tess4J)
  - **Integration of scripting languages** (Groovy, Matlab, Python, R, Octave)
  - **Diverse** (DYNAMTRIX Customer Intelligence, JMS Messaging Connector, ShapeFile Support, MMI Data Analysis, Palladin Internet Information Mining, RapidMiner Integration, REST Web Service Calls and Handling, Self-Tuning Association Rules)
- Code is developed, maintained and supported by community developers
- User communication via forums
- Trusted and «Untrusted» extensions to differentiate between quality levels



KNIME Community Development Process

How easy it would be if I had a wizard …

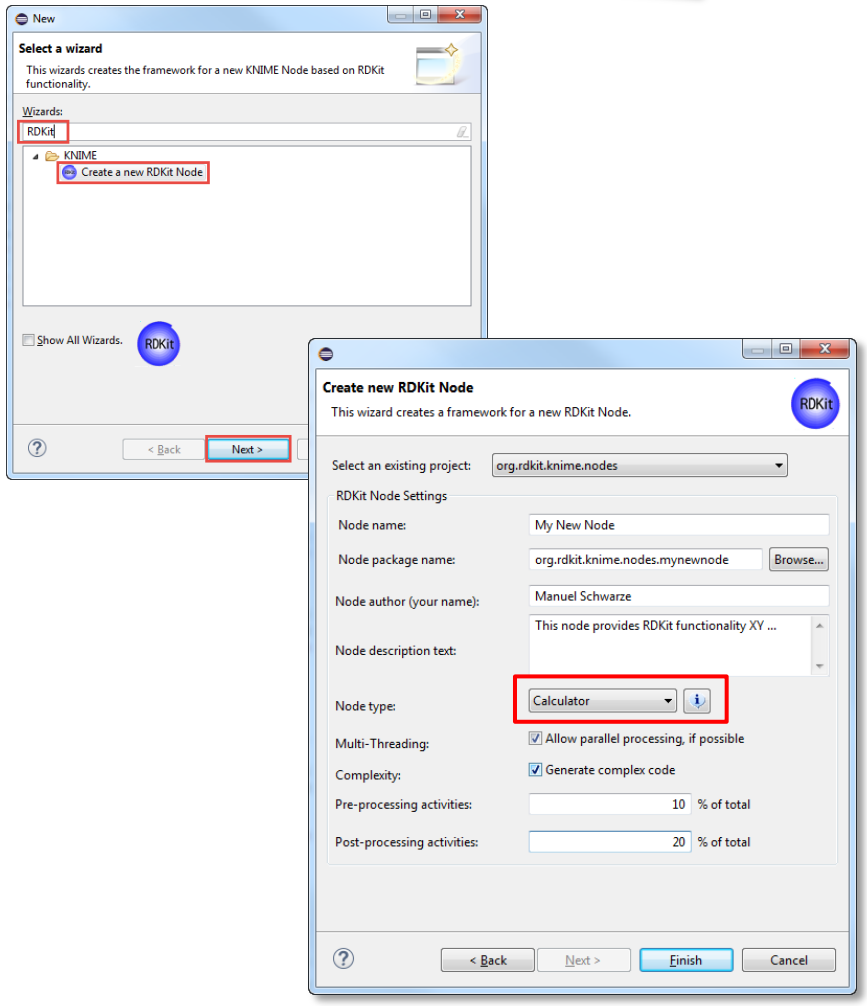
- KNIME Node Wizard available to write new KNIME nodes, but developer has still plenty to do
- RDKit Node Wizard available to write new RDKit based KNIME nodes, which are based on a little framework spezialized on KNIME and RDKit
- Multiple dynamic and flexible code templates
- Used in **KNIME SDK** (Software Development Kit), which contains the full-fledged Eclipse IDE (integrated development environment) when developing RDKit nodes source code



… Ok, I got a wizard. Now what?

Use it! ☺

- Install the KNIME SDK
- Create a workspace
- Setup update sites to plugins we are interested in
- Get the RDKit Nodes source code from Github and import these Eclipse projects into the KNIME SDK
- Install other plugins required by RDKit Nodes
- Install the RDKit Node Wizard
- Start the RDKit Node Wizard:
  - Navigate in the package explorer to project org.rdkit.knime.nodes
  - Select an existing node package, e.g. org.rdkit.knime.nodes.addhs
  - Select menu File – New – Other ... and enter «RDKit» to filter wizards
  - Click on «Create a new RDKit Node» and the «Next>» button
- More details in backup slides

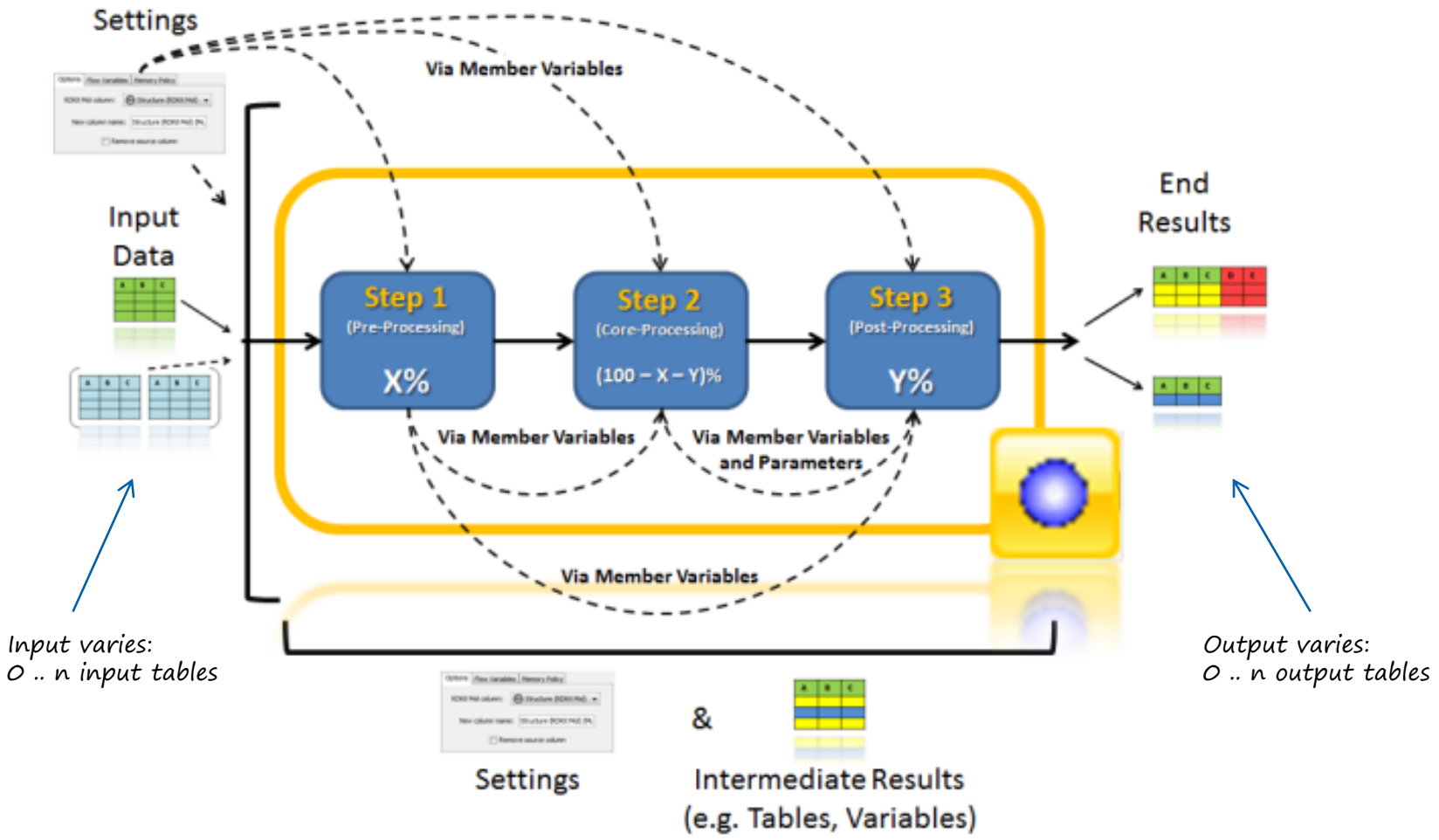


The RDKit Node Wizard

… Ok, I got a wizard. Now what?

Typical RDKit node execution scheme

Use it! ☺

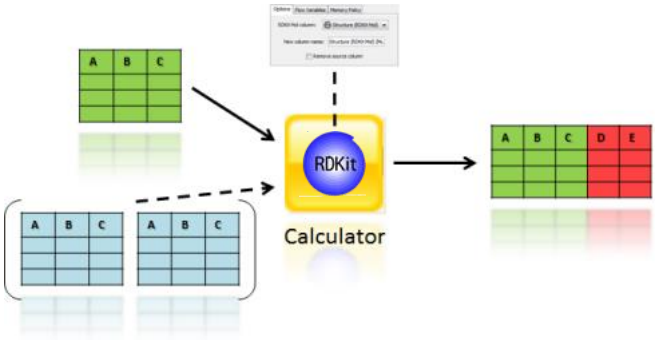


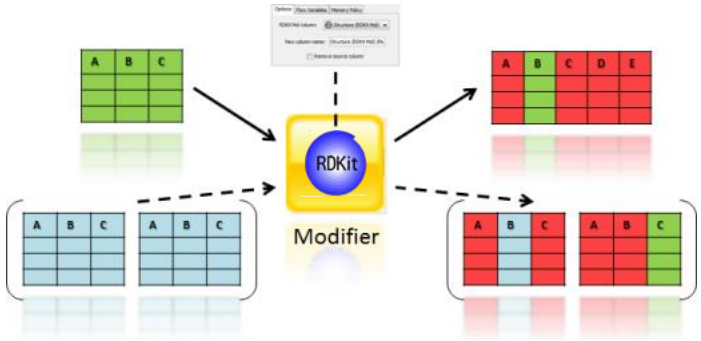
NOVARTIS

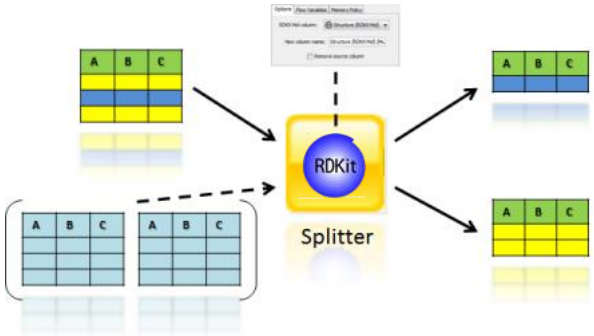
… Ok, I got a wizard. Now what?

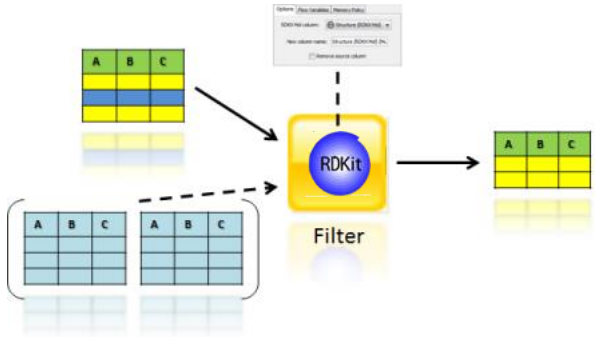
RDKit node types supported by the wizard

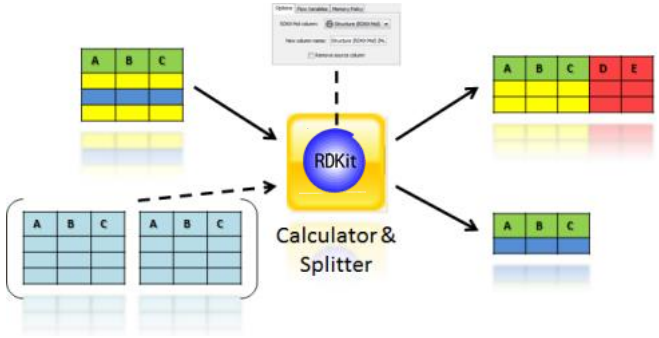
Use it! ☺

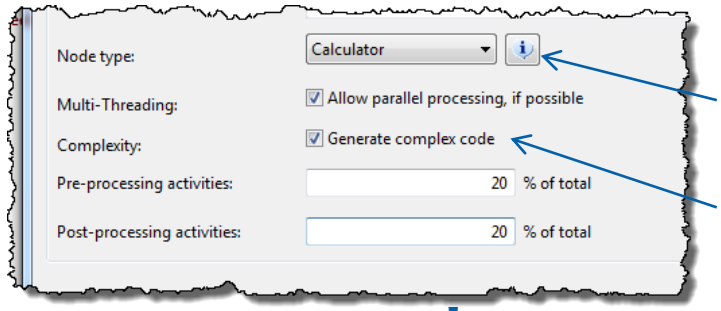












Detailed descriptions of different node types

Enables pre- and post-processing

… Ok, I got a wizard. Now what?

Generate source code and plugin changes

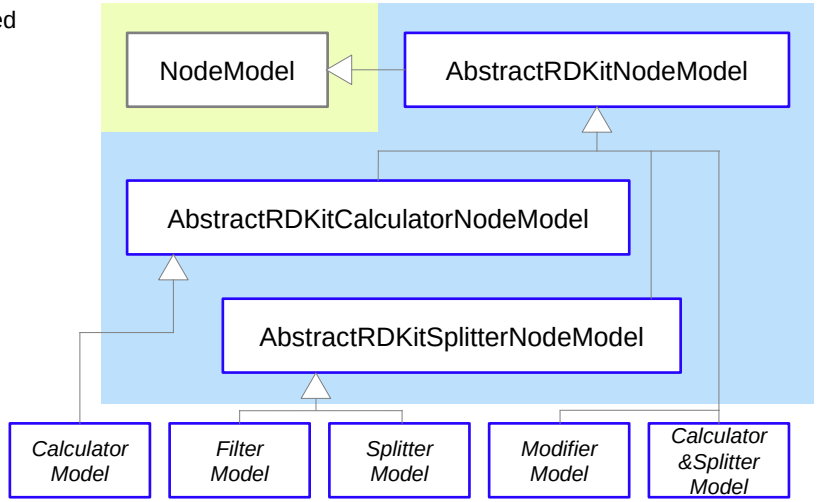
Use it! ☺

- Creating new Java package `org.rdkit.knime.nodes.mynewnode`
  - **`RDKitMyNewNodeDialog.java`**
    - Is derived from super class of RDKit Nodes framework
    - Defines parameters for the node (setting models)
    - Defines graphical user interface to enter parameters (dialog components)
  - **`RDKitMyNewNodeFactory.java`**
    - Defines if there is a dialog to provide node parameters (normally there is one)
    - Defines number of views (normally there are no views)
      Note: The normal output table is not a view
  - **`RDKitMyNewNodeModel.java`**
    - Is derived from a super class of the RDKit Nodes framework that is based on the selected node type
    - Defines number of input and output tables (normally 1:1 or 1:2)
    - Defines input columns that shall be processed for each input table and how they are handled (e.g. empty cells, errors handling)
    - Defines output table specification (columns with their names and data types)
    - Uses parameters defined in the dialog class
    - Defines configuration method used to validate node parameters and provide some dynamic default settings
    - Defines execution logic in output factory - used to calculate new data, usually based on RDKit functionality
  - Look for «TODO»s in the code and follow the instructions

```
/**
 * Creates the settings model to be used for the input column.
 *
 * @return Settings model for input column selection.
 */
static final SettingsModelString createInputColumnNameModel() {
    return new SettingsModelString("input_column", null);
}
```

```
...
super.addDialogComponent(new DialogComponentColumnNameSelection(
    createInputColumnNameModel(), "RDKit Mol column: ", 0,
    RDKitMolValue.class));
...
```

Tipp: Auto-completion is your friend, e.g. try to write «DialogComponent», then hit Ctrl+Space to learn what dialog components are available in KNIME and RDKit Nodes



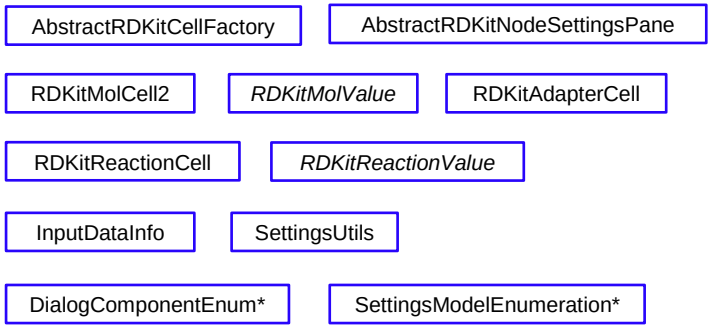
… Ok, I got a wizard. Now what?

Generate source code and plugin changes

Use it! ☺

- Further things in Java package `org.rdkit.knime.nodes.mynewnode`
  - **`default.png`**
    - This icon will show up on the node – change it to reflect what the node does
  - **`RDKitMyNewNodeFactory.xml`**
    - Defines node name and help content (description, parameters)
    - Defines input and output ports descriptions of the node (seen as tooltips)
- Adding new node package to **`/META-INFO/MANIFEST.MF`**
- Adding new node extension point registration to **`/plugin.xml`**
  - Defines that the new node appears in the RDKit category
  - Defines the sub-category and order of the node relative to others
- Let's explore some generated source code for a calculator node!

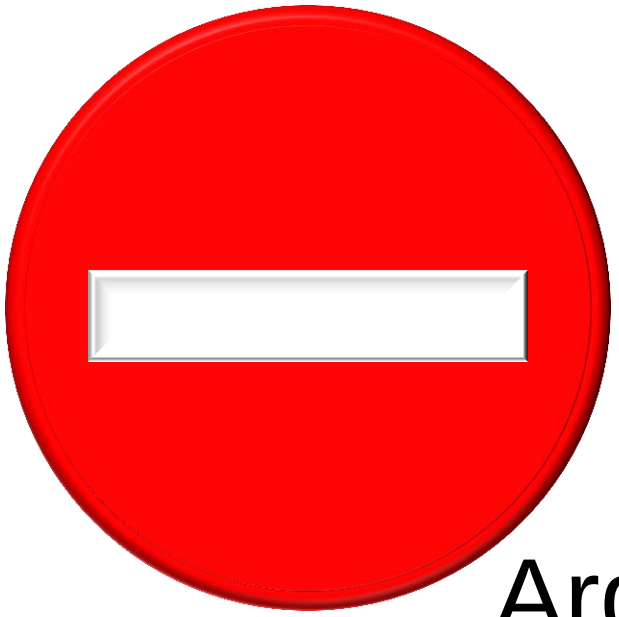
```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE knimeNode PUBLIC "-//UNIKN//DTD KNIME Node 2.0//EN"
"http://www.knime.org/Node.dtd">
<knimeNode icon="./default.png" type="Manipulator">
    <name>RDKit Add Hs</name>

    <shortDescription>
        Adds hydrogens to an RDKit molecule.
    </shortDescription>

    <fullDescription>
        <intro>Adds hydrogens to an RDKit molecule.
        </intro>
        <option name="RDKit Mol column">The input column with RDKit
         Molecules.</option>
        <option name="New column name">The name of the new column,
         which will contain the calculation results.</option>
        <option name="Remove source column">Set to true to remove
         the specified source column from the result table.</option>
    </fullDescription>

    <ports>
        <inPort index="0" name="Input table with RDKit Molecules">
          Input table with RDKit Molecules</inPort>
        <outPort index="0" name="Output table with RDKit Molecules
          with added Hs">Output table with RDKit Molecules with
          added Hs</outPort>
    </ports>
</knimeNode>
```



Other useful classes of RDKit Nodes Framework

What can I do to go beyond the limit?

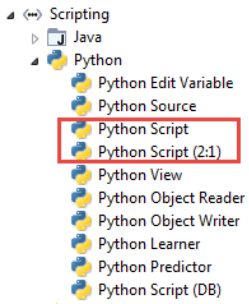
- What limit? Why is there a limit?
  - Desired RDKit functionality must be available via Java API
  - If not available, maybe ...
    - it is only available in Python
      (talk to contributor to migrate it to C++)
    - it was forgotten to make it visible
      (talk to Greg)



Argh!

What can I do to go beyond the limit?

- Everything(?) of RDKit is available through Python
- KNIME offers Python nodes for scripting (under Scripting/Python!)
- RDKit extension offers tight integration between KNIME's RDKit data types (molecule, reaction, fingerprints) and Python scripts
- Use Python nodes with little scripts to go beyond the limit!

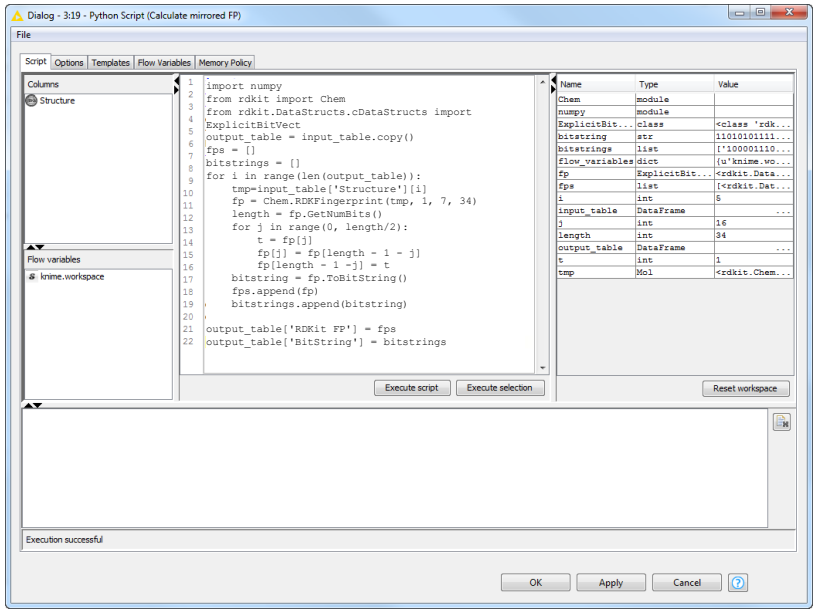


KNIME Python Nodes in KNIME 3.2

Prerequisites:

- Python installation
- Installation of modules:
  - Numpy
  - Google protobuf
  - Pandas
  - Jedi
  - RDKit
- KNIME Python Integration extension installation
- Correct KNIME Preference settings for Python Executable Path in «KNIME/Python»

See also: Install instructions



Configuration dialog of Python Script node

Acknowledgements

- T5 Informatics:
  - Greg Landrum
- knime.com
  - Greg Landrum
  - Thorsten Meinl
  - Bernd Wiswedel
- Novartis:
  - Mark Duffield (NIBR NX)
  - Nadine Schneider (NIBR NX)
  - Nikolas Fechner (NIBR NX)
- ETH Zurich, Switzerland
  - Sereina Riniker
- RDKit and KNIME open-source community

Thank you –

Any Questions?

NOVARTIS

Backup Slides

NOVARTIS

How To: Setup

- Downloaded OS-specific SDK (64-bit) from https://www.knime.org/downloads/overview
- Run the installer and install in C:\RDKitKnimeWorkshop\KNIME_SDK_3.2
- Use as workspace a new folder C:\RDKitKnimeWorkshop\Workspace
- Create a subfolder org.rdkit in the workspace folder
- Close Welcome Screen
- Install other plugins: Menu Help – Install New Software ...
  - Click on «Available Update Sites» link and add the following sites:
    - For KNIME Core / Labs plugins: http://update.knime.org/analytics-platform/3.2/
    - For KNIME Community Nightly Builds: http://update.knime.org/community-contributions/trunk
    - For Subclipse SVN integration: http://subclipse.tigris.org/update_1.8.x

How To: Setup

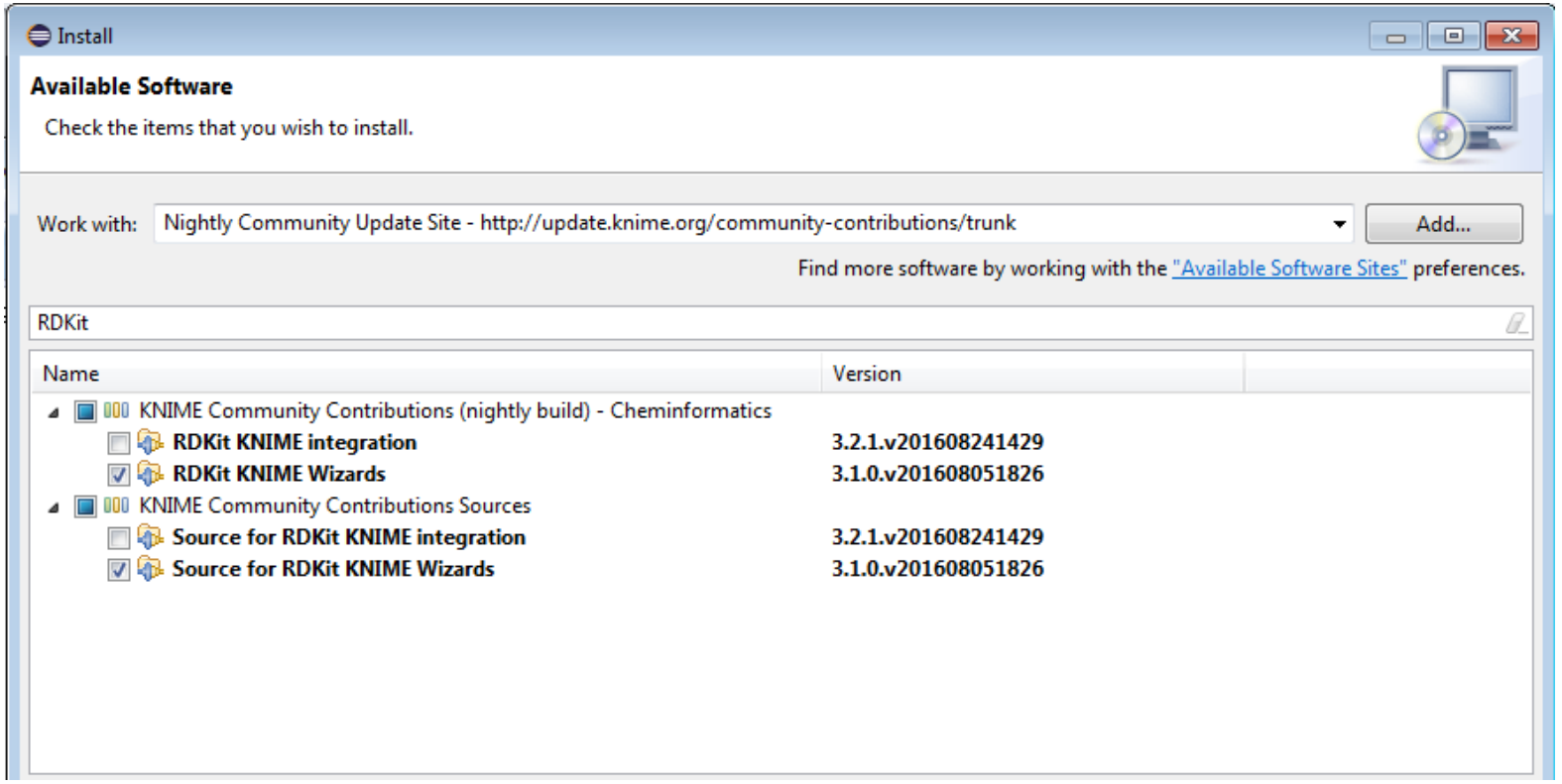
- Install plugins that the RDKit Types and Nodes plugin **depends on** and for test workflows
- Install RDKit plugins from Nightly Community Update Site to get the «RDKit KNIME Wizards» plugin (http://update.knime.org/community-contributions/trunk/)
- Install SVN or Github client with internet access
- Checkout the RDKit community extension source code from SVN or Github into the workspace folder org.rdkit (https://community.knime.org/svn/nodes4knime/trunk/org.rdkit/, https://github.com/rdkit/knime-rdkit)
- The org.rdkit folder contains now multiple projects to be imported to the KNIME SDK as projects:
  - org.rdkit.knime.bin.<your OS flavor>
  - org.rdkit.knime.feature
  - **org.rdkit.knime.nodes** ← This is the project we will work on
  - org.rdkit.knime.testing
  - org.rdkit.knime.testing.feature
  - **org.rdkit.knime.types**
  - org.rdkit.knime.update
  - org.rdkit.knime.wizard
  - org.rdkit.knime.wizard.feature

Dependencies to install:

- *KNIME Base Chemistry Types and Nodes*
- *KNIME Chemistry Add-Ons*
- *KNIME Python Integration*
- *KNIME SVG Support*
- *KNIME Streaming Execution (Beta)*
- *ChemAxon/InfoCom Marvin Extensions Feature*
- *Recommended:*
  - *KNIME Virtual Nodes*
  - *KNIME JavaScript Views*
  - *KNIME REST Client Extension*
  - *KNIME Distance Matrix*
  - *KNIME File Handling Nodes*
- *Associated sources*

How To: Setup

- RDKit Wizard – Getting it from the KNIME update site:



How To: Setup

- KNIME plugins that RDKit nodes depend on and which are useful for testing workflows
- Include source plugins, which are good for debugging
- You can always uninstall or update installed plugins and of course any time add new plugins

Tips and Tricks

KNIME & Python

- Scripts to start up Python out of KNIME with controlled RDKit environment:
  - Windows: rdkit_python.bat (Example)

```
@ECHO OFF
SET RDBASE=C:\Applications\Anaconda\library-downloads\RDKit_YYYY_MM_D
SET PYTHONPATH=%RDBASE%;%PYTHONPATH%
SET PATH=C:\Applications\Anaconda;C:\Applications\Anaconda\Scripts;%RDBASE%\lib;%PATH%
python.exe %*
```

  - Linux: rdkit_python.sh (Example)

```
#!/bin/bash
export RDBASE=/Users/ManuelSchwarze/RDKit_git
export PYTHONPATH="$RDBASE:/usr/local/lib/python2.7/site-packages:."
export DYLD_LIBRARY_PATH="$RDBASE/lib:$DYLD_LIBRARY_PATH"
/usr/bin/python "$@" 1>&1 2>&2
```

  - Mac: rdkit_python.sh (Example)

```
#!/bin/bash
export RDBASE=/Users/ManuelSchwarze/RDKit_git
export PYTHONPATH="$RDBASE:/usr/local/lib/python2.7/site-packages:."
export LD_LIBRARY_PATH="$RDBASE/lib:$LD_LIBRARY_PATH"
/usr/bin/python "$@" 1>&1 2>&2
```

- Configure the script in KNIME Preferences – KNIME – Python