Helmholtz Zentrum München

KNIME4NGS: a comprehensive toolbox for high-throughput Next Generation Sequencing analysis - Technical supplement

Maximilian Hastreiter, Tim Jeske, Jonathan Hoser, Michael Kluge, Kaarin Ahomaa, Marie-Sophie Friedl, Sebastian J. Kopetzky, Jan-Dominik Quell, H.-Werner Mewes and Robert Küffner

November 10, 2016

Contents

1 Technical Documentation

In this technical documentation, we will briefly outline the structure of KNIME4NGS and explain possibilities how to extend our node package and gain access to our additional features like the HTE Executor. We also describe how to integrate KNIME4NGS in existing pipelines.

1.1 Node Structure

All KNIME4NGS nodes are based on the standard KNIME NodeModel which provides all default node functionalities. In order to improve this original model, we have developed several additional NodeModels that extend the default NodeModel and come with distinct functionalities and characteristics (Figure 1.1).

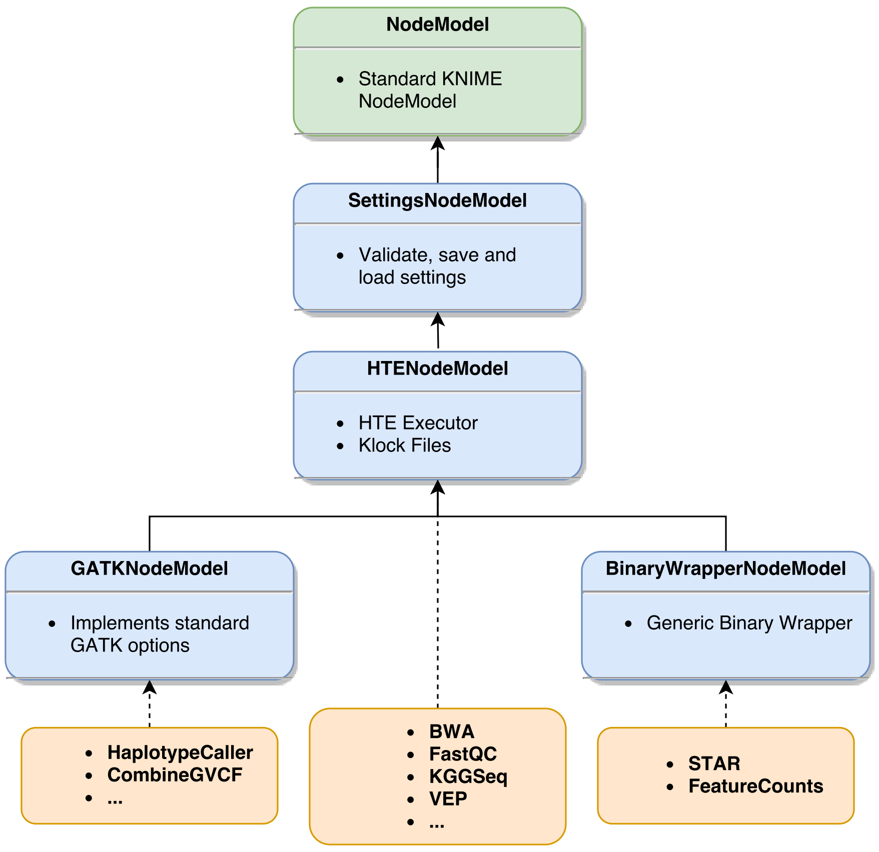

Figure 1.1: The NodeModel structure of the KNIME4NGS nodes. The newly developed NodeModels are highlighted in blue, nodes that extend the respective classes are listed in orange boxes.

NodeModels

- **NodeModel** - The standard KNIME NodeModel which is used for all KNIME nodes.
- **SettingsNodeModel** - The abstract SettingsNodeModel extends the original NodeModel and simplifies the storage of node settings by implementing a general method for saving, loading and validation of node settings.
- **HTENodeModel** - The abstract HTENodeModel extends the SettingsNodeModel and implements the HTE Executor. This executor is capable of controlling the external tool processes and keeps track of termination status and logging (See section about HTE and Klock for further information).

- **GATKNodeModel** - The GATKNodeModel further extends the HTENodeModel and implements several standard GATK parameters that are required for all GATK tools.
- **BinaryWrapperNodeModel** - The BinaryWrapperNodeModel extends the HTENode-Model and implements a generic wrapper that can be used to easily include further binaries/tools into the set of existing KNIME nodes.

1.2 Extending KNIME4NGS

As the source code is freely available under `https://github.com/ibisngs/knime4ngs-src`, you can easily extend the set of existing nodes by using the provided abstract NodeModels. Depending on the requirements, it is possible to use any of the previously mentioned classes. Thereby, also the functionalities of the HTE Executor or the klock files can be included (provided by the HTENodeModel).

1.3 HTE database schema

The details of the execution of any KNIME4NGS node are logged in the HTE database if the preference page is configured accordingly. The HTExecution table contains an entry for each started node execution (see Table 1.1). Whenever a node fails, details are stored in the HTError table (see Table 1.2).

column_name	type	notes
exec_id	INTEGER	unique execution ID of a node
klock_string	BLOB	execution command
node_name	VARCHAR(255)	name of the executed node
host_name	VARCHAR(255)	name of execution host
node_time	VARCHAR(255)	time stamp of execution start
node_successful_finished	INTEGER	0 if node is running or has failed, 1 if node was successfully executed
node_threshold	INTEGER	maximum number of automated restarts
node_count	INTEGER	number of restarts carried out

Table 1.1: **HTExecution table.**

column_name	type	notes
err_id	INTEGER	unique ID for erroneous termination of a node
exec_id	INTEGER	ID of the corresponding node execution database entry
time	VARCHAR(255)	time stamp when error occurred
err_msg	BLOB	error message

Table 1.2: **HTError table.**

1.4 Combining KNIME4NGS with other nodes/workflows

The KNIME4NGS nodes can be combined with existing nodes or workflows. Since storing raw NGS data within KNIME tables is not reasonable, all nodes return file paths of their respective output files (Note: Typical file types like .fastq, .bam and.vcf are stored in type-specific file cells). These file paths can be then be used in the following nodes to either import or read/process the data. Correspondingly, file paths can also be used to provide an input for the nodes, although in

this case, we recommend using the FileLoader node.
If those requirements are fulfilled, the KNIME4NGS nodes can be combined with any other node/workflow.

1.5 In-house tools

1.5.1 FastQCmod

FastQCmod is an extension of FastQC (currently build on release v0.10.1). The extension includes a logic to extract the analysis results of the individual FastQC analysis methods and write a more computer readable output, to allow for a simplified reusal by the RawReadManipulator. Furtheron, in the light of the fact that CPU power is currently more widely available than I/O-Bandwidth, FastQCmod contains code for parallelization of the analysis of single fastq-files, compared to the per-sample/per-file parallelization of the original FastQC. A backport of recent FastQC versions to FastQCmod is in preparation.

1.5.2 RawReadManipulator

The RawReadManipulator (RRM) is an in-house development of a modular, highly parallelized filtering pipeline for raw reads. The Java application is easily extensible with custom filtering-modules and makes use of the data-independence that reads (or read-pairs) have towards each other. All available filters have many available configuration options, such that the most common filtering needs should be met when using RRM. Tests have shown the parallelization to be scaling well with the number of CPUs given to the task, with the bottleneck of available data-I/O-bandwith as the limiting factor in terms of filtering-speed.

1.5.3 Availability

Both tools are available under the GPLv3 license at https://github.com/ibisngs/knime4ngs-src/tree/v1-6-5/libs. Source code is included within the respective jar files.