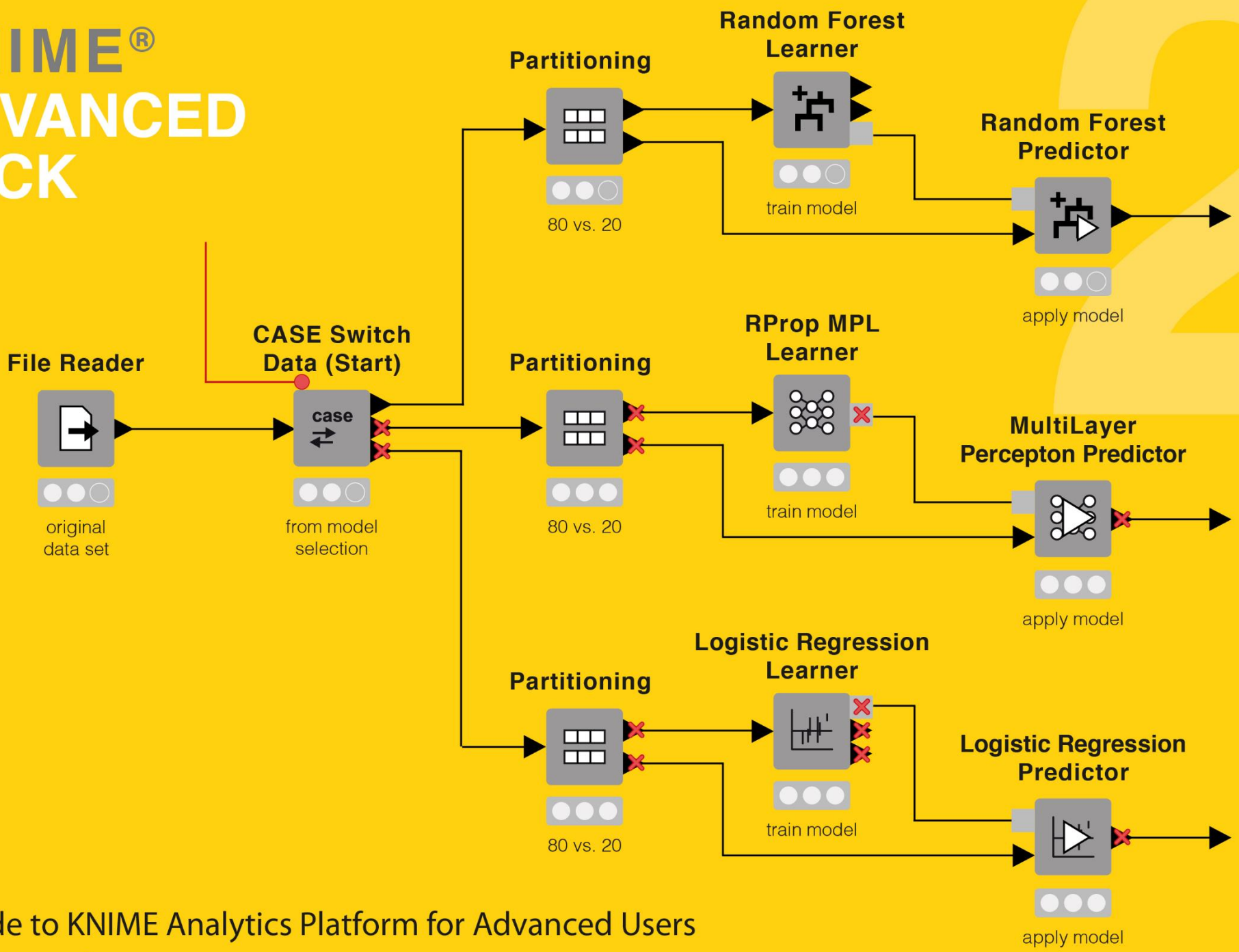
# KNIME®
# ADVANCED LUCK




A Guide to KNIME Analytics Platform for Advanced Users

Authors: Rosaria Silipo and Jeanette Prinz

This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

Copyright©2019 by KNIME Press

All rights reserved. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording or likewise.

This book has been updated for KNIME **3.7**.

For information regarding permissions and sales, write to:

KNIME Press
Technoparkstr. 1
8005 Zurich
Switzerland

knimepress@knime.com

ISBN: 978-3-9523926-0-7



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

# Table of Contents





This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

# Acknowledgements

We would like to thank a number of people for their help and encouragement in writing this book.

In particular, we would like to thank Bernd Wiswedel for answering our endless questions about calling external REST services from inside a workflow, and Iris Adae for explaining the most advanced features of the Date&Time nodes.

Special thanks go to Peter Ohl for reviewing the book contents and making sure that they comply with KNIME intended usage and to Heather Fyson for reviewing the book's English written style.

Finally, we would like to thank the whole KNIME Team for their support in publishing and advertising this book.



This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

# Chapter 1. Introduction

## 1.1. Purpose and Structure of this Book

KNIME Analytics Platform is a powerful tool for data analytics and data visualization. It provides a complete environment for data analysis which is fairly simple and intuitive to use. This, coupled with the fact that KNIME Analytics Platform is open source, has led a large number of professionals to use it. In addition, third-party software vendors develop KNIME extensions in order to integrate their tools into KNIME Analytics Platform. KNIME nodes are now available that reach beyond customer relationship management and business intelligence, extending into the field of finance, life sciences, biotechnology, pharmaceutical, and chemical industries. Thus, the archetypal KNIME user is no longer necessarily a data mining expert, although his/her goal is still the same: to understand data and to extract useful information.

This book was written with the intention of building upon the reader's first experience with KNIME. It expands on the topics that were covered in the first KNIME user guide ("KNIME Beginner´s Luck" [1]) and introduces more advanced functionalities. In the first KNIME user guide [1], we described the basic principles of KNIME Analytics Platform and showed how to use it. We demonstrated how to build a basic workflow to manipulate, visualize, and model data, and how to build reports. Here, we complete these descriptions by introducing the reader to more advanced concepts. A summary of the chapters provides you with a short overview of the contents to follow.

Chapter 2 describes the nodes needed to connect to a database, import data, build an appropriate SQL query to select a subset of the data or for some required processing, and finally to write data back into the database. Accessing a database, importing data, and building SQL queries are the basic operations necessary for any, even very simple, data warehousing strategy.

Of course, the largest source of data is nowadays the Internet. Chapter 3 is dedicated to alternative ways of getting data besides files and databases, i.e. web data sources. Chapter 3 starts with the connectors to Google Sheets, continues with access to REST services, and concludes with a web crawling example workflow. Those are definitely powerful tools to search for data elsewhere.

Chapter 4 introduces the Date&Time object and the nodes to turn a String column into a Date&Time column, to format it, to extract a time difference, and in general to perform date and time based operations. The Date&Time object provides the basis for working with time series. The last section of chapter 4 briefly describes a few nodes dedicated to time series analysis.

A very important concept for the KNIME workflows is the concept of "flow variables". Flow variables enable external parameters to be introduced into a workflow to control its execution. Chapter 5 describes what a flow variable is, how to create it, and how to edit it inside the workflow, if needed.



This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

Most data operations in KNIME Analytics Platform are executed on a data matrix, named data table. This means that an operation is executed on all data rows. This is a big advantage in terms of speed and programming compactness. However, from time to time, a workflow also needs to run its rows, one after the other, through an operation. That is, sometimes it needs a real loop. Chapter 5 introduces a few nodes that implement loops: from a simple “for” cycle to more complex loops, such as looping on a list of values or feeding the current iteration results into the next iteration.

Chapter 7 illustrates the use of logical switches to change the workflow path upon compliance with some predefined condition.

Chapter 8 is an extension of chapter 6 in “KNIME Beginner´s Luck” [1]: it describes a number of advanced features of KNIME reporting tool. First of all, it explains how to introduce parameters into a report and how flow variables and report parameters are connected. Later on, in the chapter, a few more reporting functions are discussed which can be used to create a more dynamic report.

In this introductory chapter, we list the data and the example workflows that have been built for this book and note the KNIME Extensions required to run some of the example workflows.

## 1.2. Data and Workflows for this Book

In the course of this book we will put together a few workflows to show how KNIME Analytics Platform works. In each chapter we will build one or more workflows and we will expect the reader to build a few more in the exercises.

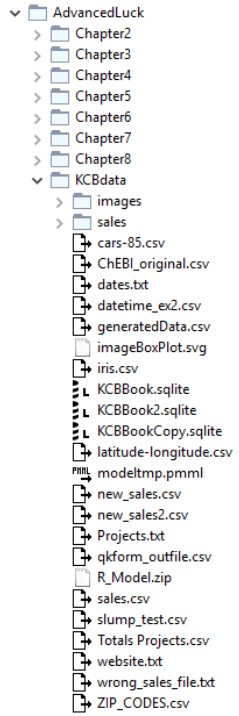
When you purchased this book, in the purchase email containing the link to the pdf file, there should also have been a link to the Download Zone file. The Download Zone file is a .knar file and contains the data and workflows used and implemented in this book.

- Download the Download Zone knar file onto your machine
- Import it into KNIME Explorer:
  - Select “File” -> “Import KNIME Workflow … “
  - In the “Import Workflow” window, select option “Select File” and navigate to the Download Zone knar file.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

### 1.1. Workflows and data used in this book, as imported from the Download Zone.knar file




At the end of the import operation, in your "KNIME Explorer" panel you should find a folder named "Advanced Luck" and containing Chapter2, Chapter3, Chapter4, etc … subfolders, each one with workflows and exercises to be implemented in the corresponding chapters of this book. In addition, under the main folder "Advanced Luck", you should find a KCBdata subfolder containing all necessary data.

The data used for the exercises and for the demonstrative workflows of this book were either generated by the author or downloaded from the UCI Machine Learning Repository [2], a public data repository (http://archive.ics.uci.edu/ml/datasets). If the data set belongs to the UCI Repository, a full link is provided here to download it. Data generated by the author, that is not public data, are located only in the KCBdata folder.

Data sets from the UCI Machine Learning Repository [2]:

- Automobile: http://archive.ics.uci.edu/ml/datasets/Automobile
- Slump_test: http://archive.ics.uci.edu/ml/datasets/Concrete+Slump+Test

This book is not meant as an exhaustive reference for KNIME Analytics Platform, although many useful workflows and aspects of it are demonstrated through worked examples. This text is intended to give you the confidence to use the advanced functions in KNIME Analytics Platform to manage and mine your own data.



This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

## 1.3. Memory Usage in KNIME Analytics Platform

Sometimes some workflows require exceptional memory usage. The amount of memory available to KNIME Analytics Platform is stored in the knime.ini file. The knime.ini file is located in the directory in which KNIME has been installed, together with the knime.exe file. The knime.ini file contains a number of settings required by the KNIME software.

1.2. The "knime.ini" file

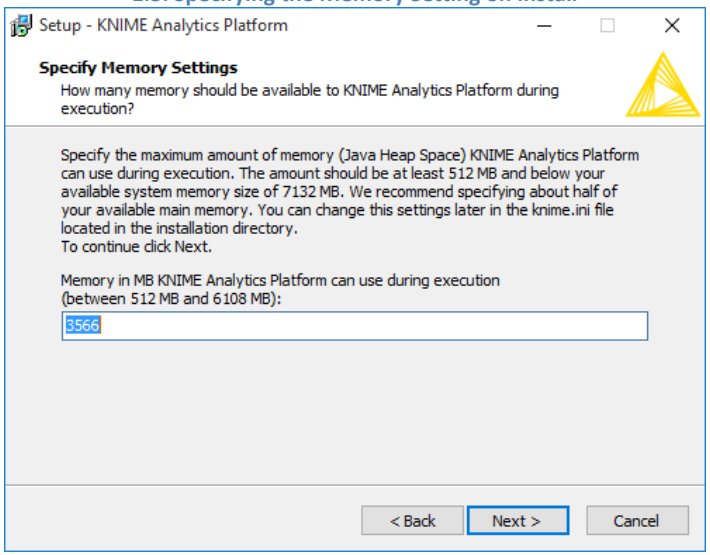
```
-startup
plugins/org.eclipse.equinox.launcher_1.3.100.v20150511-1540.jar
--launcher.library
plugins/org.eclipse.equinox.launcher.win32.win32.x86_64_1.1.300.v20150602-1417
-vmargs
-server
-Dsun.java2d.d3d=false
-Dosgi.classloader.lock=classname
-XX:+UnlockDiagnosticVMOptions
-XX:+UnsyncloadClass
-Dknime.enable.fastload=true
-XX:CompileCommand=exclude,javax/swing/text/GlyphView,getBreakSpot
-Xmx3566m
-Dorg.eclipse.swt.browser.IEVersion=10001
-Dsun.awt.noerasebackground=true
```

**-Xmx<size>** is the setting that defines the maximum heap size available to run the workflows. You can define this value by editing the knime.ini file or at installation time. During installation, in fact, you are required to insert the maximum amount of memory available to KNIME Analytics Platform. After that, if you run into memory problems, you probably need to manually increase the heap space (-Xmx option) directly in the knime.ini file to a size compatible with the memory you have on your machine.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

### 1.3. Specifying the Memory Setting on Install




Now, let's suppose that you are having a memory problem while executing a workflow. You have already changed the -Xmx value in the knime.ini file to a higher value, but that was not enough. You are still running into memory problems when your workflow is executed. Even though this kind of problem occurs very rarely in KNIME, it can still happen. In this case, you need to know whether the problem is due to your workflow or to some other program running on your machine at the same time.

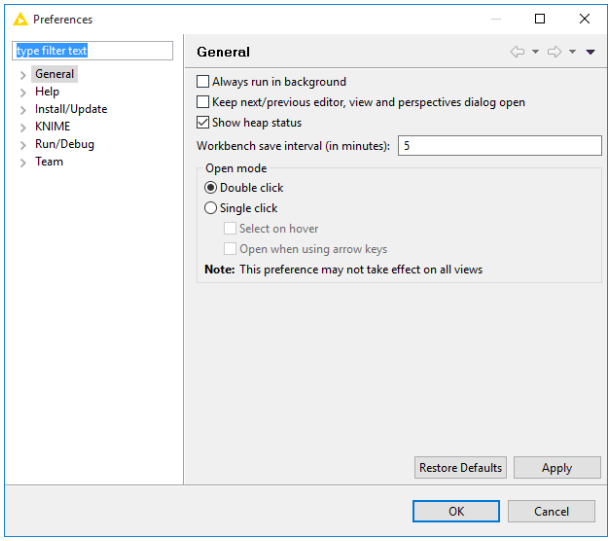
There is an easy way to monitor how much heap space is being used by a workflow and if this reaches the maximum limit assigned by the –Xmx option. In the Top Menu in the KNIME workbench:

- Click "File"
- Select "Preferences"
- "Preferences" window opens
- In "Preferences" window
    - Select "General"
    - In the frame on the right named "General", enable the option "Show heap status"
    - Click "OK"
    - Now, in the lower right corner you can see a small number showing the heap status.

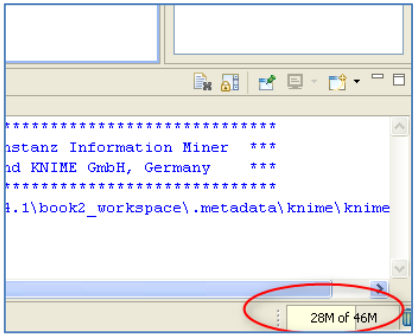


This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

1.4. The "Preferences" window with the "Show heap status" option




1.5. The bottom right corner shows the heap status




To run the example workflows and the exercises provided in this book, you will need to install the following KNIME Extensions:

- The whole “KNIME & Extensions” group
- “Palladian for KNIME” under “KNIME Community Contributions – Other”

In order to install a KNIME Extension:

- In the top menu of the KNIME workbench, select “File” -> “Install KNIME Extensions …”
- In the “Install” window:
  - Open the group containing your extension, like for example “KNIME & Extensions” group
  - If you do not know where your extension package is located, just run a search by inserting a few related keywords in the top textbox
  - Select  your extension
  - Click “Next” and follow installation instructions



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

# Chapter 2. Database Operations

## 2.1. Database Nodes: Modular vs. All-in-One Approach

We proceed with the exploration of the advanced features of KNIME Analytics Platform by having a look into the database operations. A first glance was already provided in the first book of this series, "KNIME Beginner's Luck" [1, http://www.knime.org/knimepress/beginners-luck]. Here, though, we investigate the multiple possibilities for connecting, reading, writing, and selecting data from and to a database in much greater detail.

First of all, we want to create the workflow group "Chapter2", to host all workflows for this chapter. Then, in this new workflow group, we want to create an empty workflow with name "Database_Operations". The goal of this workflow is to show how to connect to a database, retrieve data from the database, and write data into the database.

In the newly created workflow named "Database_Operations", we read the data from the "sales" table in the "KCBBook.sqlite" database. SQLite is a file based database software. Since it requires neither server nor authentication, it is suitable to show the KNIME database nodes without the hassle of setting up a full-blown database. In SQLite the only thing you need is the file, "KCBBook.sqlite" in this case, containing the database available in folder KCBdata. The "sales" table inside the database contains the sale records for a fictitious company. Such records consist of:

- The name of the product (product)
- The sale country (country)
- The sale date (date)
- The quantity of products sold (quantity)
- The amount of money (amount) generated by the sale
- A flag to indicate whether the purchase was paid by cash or credit card (card)

The sales for four products are recorded: "prod_1", "prod_2", "prod_3", and "prod_4". However, "prod_4" has been discontinued and is not significant for the upcoming analysis. In addition, the last field called "card" contains only sparse values and we would like to exclude it from the final data set.

There are three ways to read data from a database into a KNIME workflow.

1. **The very modular approach, using one node for each step**. We start with one node to just establish the connection to the database; then a node to select the table to work on; a few nodes, one after the other, to build the SQL query to extract the required data set; and finally one



This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

last node to run the SQL query on the database and import the results into the KNIME workflow. There are many ways to build a SQL query with KNIME, each one fitting a given level of SQL expertise.

2. **The less modular approach, combining the database connection and the table selection into one node**. Here, the first step includes connecting to the database and selecting the table to work on in the same node; next we build the appropriate SQL query; and finally we run the SQL query on the database and we import the results into the KNIME workflow. With this approach we still have the freedom to build one or many modular SQL queries. However, we are tied to the only database table selected in the first connection node.

3. **The all-in-one node approach**. Here, we connect to the database, select the table, build the SQL query, and import the resulting data into KNIME, all in the same node.

The first approach allows us for more freedom to deal with special databases, including big data platforms. The last approach is more compact, but requires some SQL knowledge and does not allow for multiple SQL queries. The second approach is a compromise between the first and last, allowing for modular and multiple SQL queries, but being tied to a given database table. Whichever data retrieval approach is chosen, all the required nodes are located in the “Database” category.

## 2.2. Connect to a Database: Database Connector Nodes

Let’s start with the first approach. The first step is to just establish a connection to a database, nothing more, and to do that you just need to know the specs of your database: *server URL, credentials, JDBC driver*. In the Database/Connector category we find all connector nodes; that is those nodes that just connect to a database. In particular, we find dedicated nodes for selected databases and one generic node to connect to any database.

The generic connector node is named “Database Connector” and produces a database connection at the output port, indicated with a red square port. It can connect to any database, as long as you provide the correct JDBC driver. The JDBC driver is a file, usually provided by the database software distributor, interfacing the SQL script with the database software.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

## (Generic) Database Connector

The “Database Connector” node just establishes a connection to a database. To do that, it requires the following values:

- **Database driver file**. Most commonly used database drivers have been preloaded in KNIME and are available from the “Database Driver” menu. If your database driver is not in the list, check below how to upload a new database driver file.

- **Database URL**. The database URL has to be built as follows:
  jdbc: <DB-software-name>://<server-name>:<port nr>/<database-name>

  The port number in the database URL has to be provided by the database vendor.

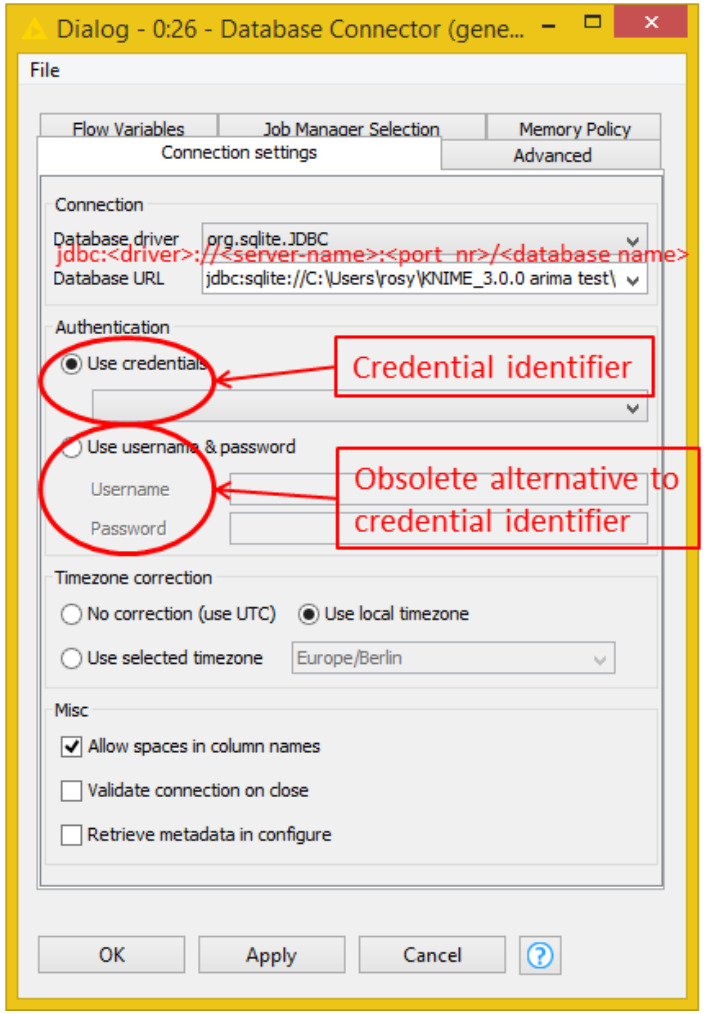
- **Time Zone** for date and time values.

- The **credentials** to access the database, as:
  - One of the credential identifiers defined in “Workflow Credentials” at the workflow level

  OR alternatively (but deprecated)
  - username and password

- A few more useful miscellaneous utility settings.

2.1. Configuration window of the “Database Connector” node




Most databases require credentials (i.e. username and password). KNIME keeps each credential set in a credential identifier parameter, named “Workflow Credential”. One or more credential identifiers can be set through the option “Workflow Credentials” in the context menu of the workflow.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

For security reasons, credential identifiers are encrypted during the whole time a workflow is open or executed, are neither visible nor accessible by other workflows, and passwords are not saved. At every new opening of the workflow you will be prompted to re-enter the password for each credential identifier.

The “Workflow Credentials” option in the context menu of a workflow is only active if the workflow is open in the workflow editor.

**Note**. The alternative to a workflow credential is to enter the username and password directly in the database node dialog. However, this authentication mode does not encrypt passwords automatically and it is therefore not as secure as a workflow credential. For this reason, the username/password authentication mode has been deprecated since the early versions of KNIME and, even if still present in the node dialog for backward compatibility, it is not recommended.

## Workflow Credentials

In the “KNIME Explorer” panel:

- Right-click the desired workflow
- Select “Workflow Credentials”

In the “Workflow Credentials …” window:

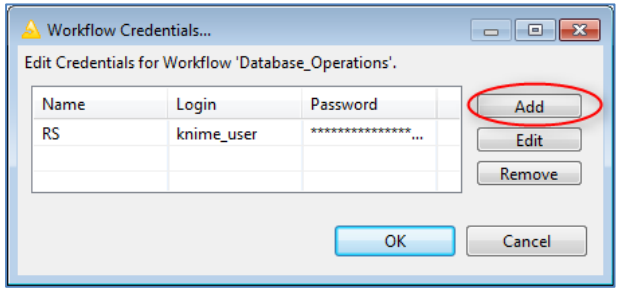
- Click the “Add” button

In the “Add/Edit Credentials” window:

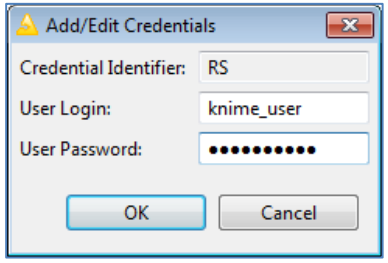
- Define an ID to identify the pair (username, password).
- Enter the username in the “User Login” box
- Enter the password in the “User Password” box
- Click “OK”

Back in the “Workflow Credentials …” window, click “OK”.

2.2. Define multiple database credentials




2.3. „Add/Edit Credentials“ window




If you decide to bypass the workflow identifier and go with explicit username and password instead, remember to provide a master key value for password encryption and decryption, to enhance security when accessing the database. The master key can be entered in the “Preferences” page under “File” -> “Preferences” -> “KNIME” -> “Master Key”.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

We configured the workflow “Database_Operations” to connect to the SQLite database file KCBBook.sqlite with a generic “Database Connector” node. We selected the SQLite driver in the configuration window. For a SQLite database, the database URL takes the shape of a file path (Fig. 2.1).

**Note**. In a generic “Database Connector” node, the URL for the sqlite file cannot use the knime:// protocol and the relative path. It needs the absolute URL path of the database file, which might make things complicated when moving the workflow into another environment.

In case you are using an older version or a not so common database, you will need to resolve to the generic “Database Connector” node. Because of your uncommon choice of database, it is possible that the JDBC driver you need is not part of the pre-loaded JDBC driver set. In this case, you need to upload your own JDBC driver file onto KNIME Analytics Platform via the “Preferences” window.

## Upload a new JDBC Driver

In the top menu:

- Select “File” -> “Preferences”

In the “Preferences” window

In the left frame:

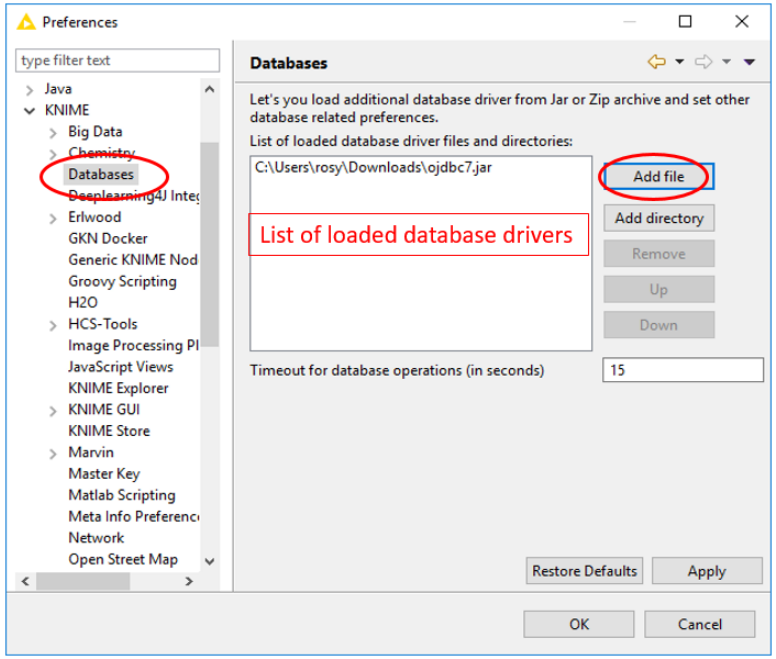
- Open “KNIME”
- Select “Databases”

In the right frame:

- Click button “Add File” or “Add directory” depending on whether your driver consists of a single file or of a complete folder
- Load the database driver file
- If the file is accepted, it appears in the list of loaded database drivers in any database node dialog.

- Click “OK”

2.4. Load a database driver file into KNIME “Preferences”






This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

**Note.** The database driver file is usually provided by the database vendor together with the database software.

After loading the database driver file in the "Preferences" window, the database driver becomes a general KNIME feature and is available for all workflows in all workspaces.

The other connector nodes available in the Database/Connector category are dedicated connectors. This means that their task is to establish a connection with only one specific database. "Hive Connector" for example is dedicated to connect to Apache Hive, "MySQL Connector" to connect to MySQL, "PostgreSQL Connector" to PostgreSQL, and so on. Since these nodes are dedicated for specific databases, some settings, like for example the JDBC driver, are hard-coded into the node, making their configuration window simpler and therefore less error prone.

## (Dedicated) SQLite Connector

The "SQLite Connector" node establishes the connection to a SQLite database (file) and requires the following values:

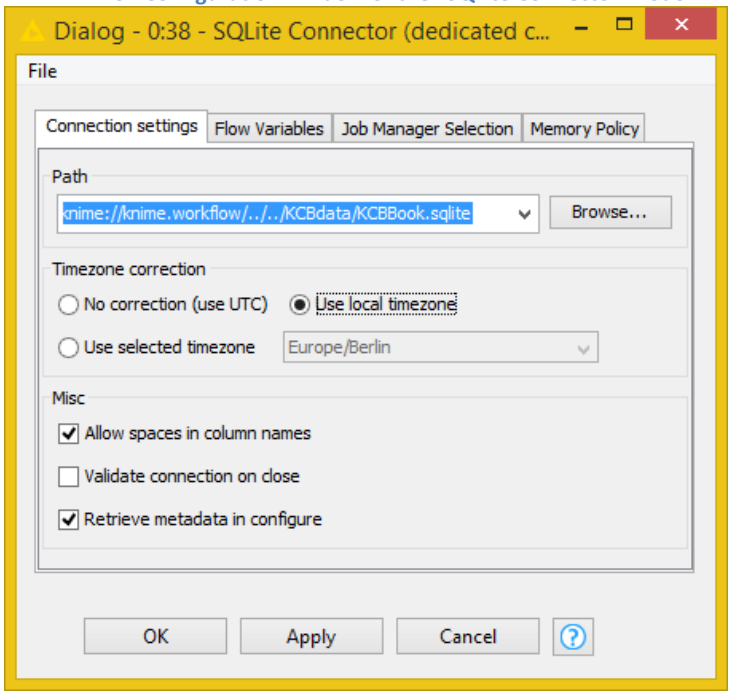
- **Database URL**. In this case, the database URL is just the path of the sqlite file containing the database.
- **Time Zone** for date and time values.
- A few useful miscellaneous utility settings.

This node outputs a pure database connection port.

Note that no selection of JDBC driver is required, because the SQLite JDBC driver is hard-coded into the node. Also no credentials are required, because SQLite databases do not require authentication.

Also note that the database URL accepts the knime:// protocol and therefore the relative path to the workflow location, which enhances the workflow portability.

2.5. Configuration window of the "SQLite Connector" node






This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

In this book we adopted the simple SQLite database, as described in section 2.1. As an example for all dedicated connector nodes, we show the “SQLite Connector” node only. Other dedicated database connector nodes differ from this one in the specific settings required by their database master. The database driver setting is usually missing in the configuration window of a dedicated connector node, since it has been pre-loaded as part of the node.

**Note**. In a dedicated “SQLite Connector” node, the URL for the sqlite file can use the knime:// protocol and the relative path, which makes the workflow more portable.

## 2.3. Select the Table to work on: the Database Table Selector Node

Once we have a connection to the database, we need to start working on the data. The next step then is to select the table to work on. That is the task for the “Database Table Selector” node. This node transforms a database connection (red square input port) into a database SQL query (brown square output port) to be executed on the input database connection.

### Database Table Selector

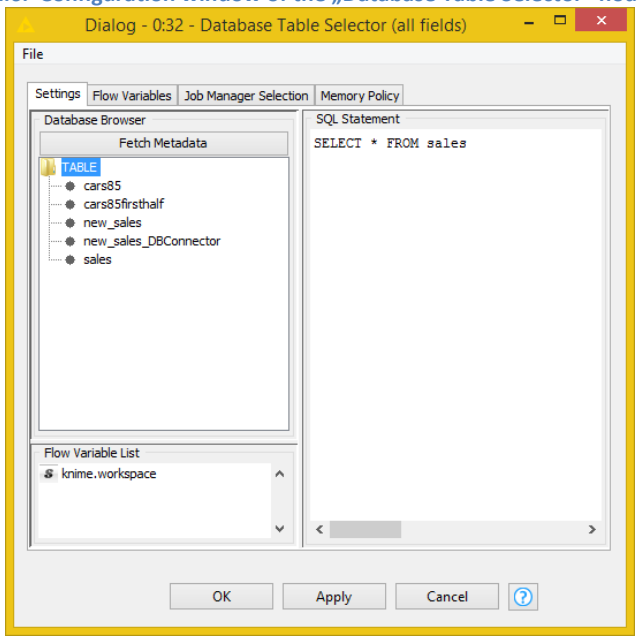
The “Database Table Selector” node takes a database connection at the input port and exports a SQL statement to be executed on the same database connection.

You can construct the SQL statement manually in the “SQL Statement” panel of the node configuration window.

Invoking the database structure with the “Fetch Metadata” button on the left, you can double-click field and table names from the “Database Browser” panel to insert them automatically and with the right syntax in the edited SQL query.

This node works pretty much like the “Database Table Connector” node (see next section), besides referring to a specific database connection at the input port.

2.6. Configuration window of the „Database Table Selector“ node

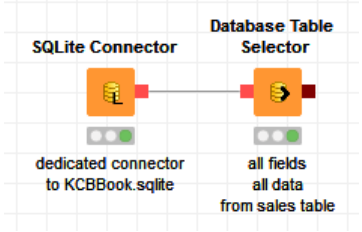





This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

This modular approach separating database connection from table selection allows to process different database tables on different workflow branches. It also allows to deal with special table selection statements for the many different databases, big data platforms, SQL dialects, and NoSQL scripts.

2.7. Sequence of "SQLite Connector" node and "Database Table Selector" node




## 2.4. Connect to a Database AND Select the Table: the Database Table Connector Node

The second approach combines the two steps – connection to database and table selection – into one single node: the "Database Table Connector" node. The "Database Table Connector" node resides in Database/"Read/Write" category. This node connects to a specific table inside a database and leaves the connection open to retrieve the data. In addition, it accepts a customized SQL query to pre-process (and possibly reduce) the data before importing them into the KNIME workflow. The SQL query is then made available at the output port (square brown port).

**Note**. The "Database Table Connector" node and the "Database Connector" node show two different output ports: brown square one and pale red square the other. Different output ports identify different compatible following nodes with the same type of input ports.

If you are not an SQL expert or if you do not know the database structure, some help comes from the "Database Browser" panel available on the left side of the configuration window of most database nodes. After clicking the "Fetch Metadata" button, the "Database Browser" panel rescues and shows the structure of the selected database. Double-clicking a table or a table field in the "Database Browser" panel automatically inserts the table name or the table field name with the right SQL syntax in the SQL editor on the right.

**Note**. A "Database Table Connector" node is the equivalent to a "Database Connector" node followed by a "Database Table Selector" node.

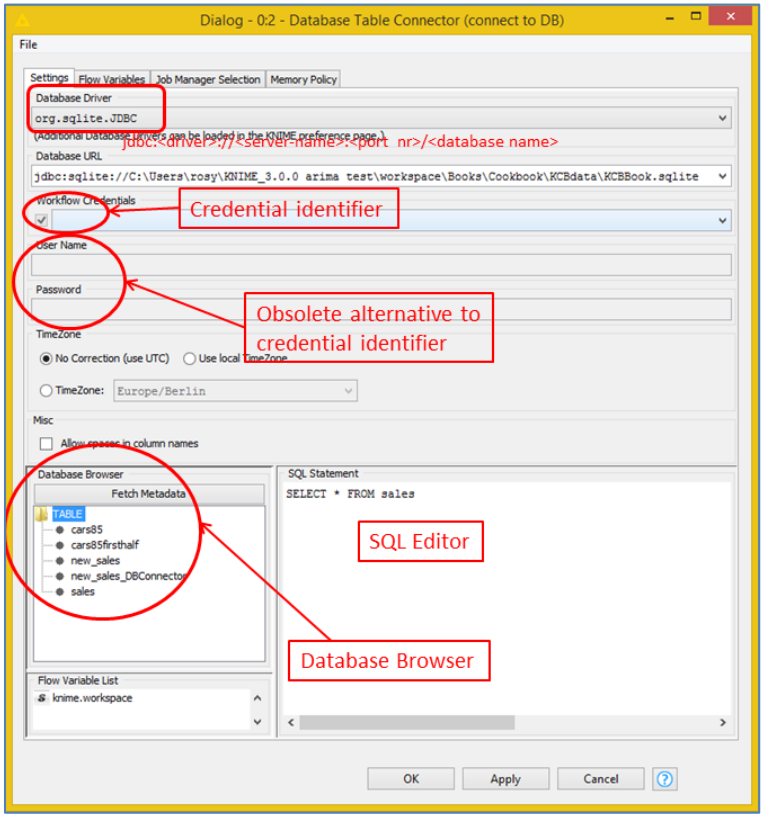
The "Database Table Connector" node binds your connection to a given database table. To deal with multiple tables and special table selection procedures you need to revert to the "Database Connector" + "Database Table Selector" strategy. On the other side, connecting to a database is shorter and more compact when using just this node.



This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

## Database Table Connector

2.8. Configuration window of the "Database Table Connector" node




The "Database Table Connector" node requires the following settings:

- **Database driver**. Most commonly used database drivers have been preloaded in KNIME Analytics Platform and are available from the "Database Driver" menu of database nodes. If your database driver is not in the list, check section 2.2 for how to upload a new database driver.

- **Database URL**. The database URL follows the database required structure. For example, for most databases, the structure required is:

  jdbc: <DB-software-name>://<server-name>:<port nr>/<database-name>

  The port number in the database URL has to be provided by the database vendor. The URL structure corresponding to the selected database software is uploaded in the configuration window as soon as the database driver has been selected.

  For SQLite this becomes: jdbc: sqlite:// <path of database file>.

- **Time Zone** for date and time values.

- The credentials to access the database:
  - One of the credential identifiers available in the "Workflow Credentials" menu

  OR alternatively (but deprecated)
  - username and password

- The **SQL SELECT statement** to retrieve the data from the database The SQL statement can be built with the help of the "Database Browser" on the left. The "Database Browser" panel shows the database structure and, if opened, the tables structure as well. Double-clicking a table or a table field in the "Database Browser" panel, inserts the object name with the right syntax in the SQL editor on the right.



This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

## 2.5. In-Database Processing

In the “Database Table Connector” node and in the “Database Table Selector” node, we used a very simple SQL query: “SELECT * from sales”. This query simply downloads the whole data set stored in table “sales”. However, what we really want to do is to get rid of the rows that pertain to product “prod_4” and to get rid of the “card” column, before we pull in the data set from the database.

One option is to customize the SQL query in the “Database Table Connector” node, something like:

```
SELECT product, country, date, quantity, amount from sales WHERE product != 'prod_4'
```

For the non-SQL savvy, the KNIME Analytics Platform offers a few database manipulation nodes. These nodes implement SQL queries through a graphical user interface bypassing the whole SQL script. They take a SQL query as input (brown square port) and produce a SQL query as output, which consists of the input SQL query augmented with the SQL query implemented in the node itself.

For example, in order to implement the SQL query above, we just need a “Database Row Filter”, to filter out records with ‘prod_4’, and a “database Column Filter” node to remove the field named “card”.

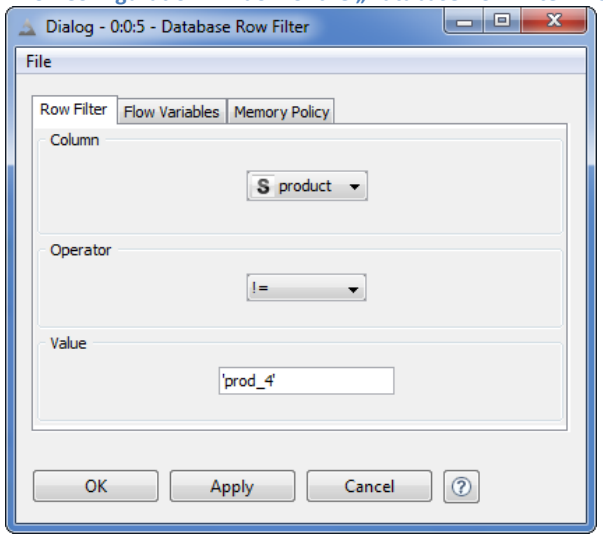
### Database Row Filter

The “Database Row Filter” node customizes a SQL query, according to a filtering criterion, to keep only the matching data records.

In order to define the filtering criterion, the node requires the following settings:

- Table field (“Column”) on which to operate
- Operator (=, >, <, !=, etc…)
- Matching pattern (“Value”)

2.9. Configuration window of the „Database Row Filter“ node





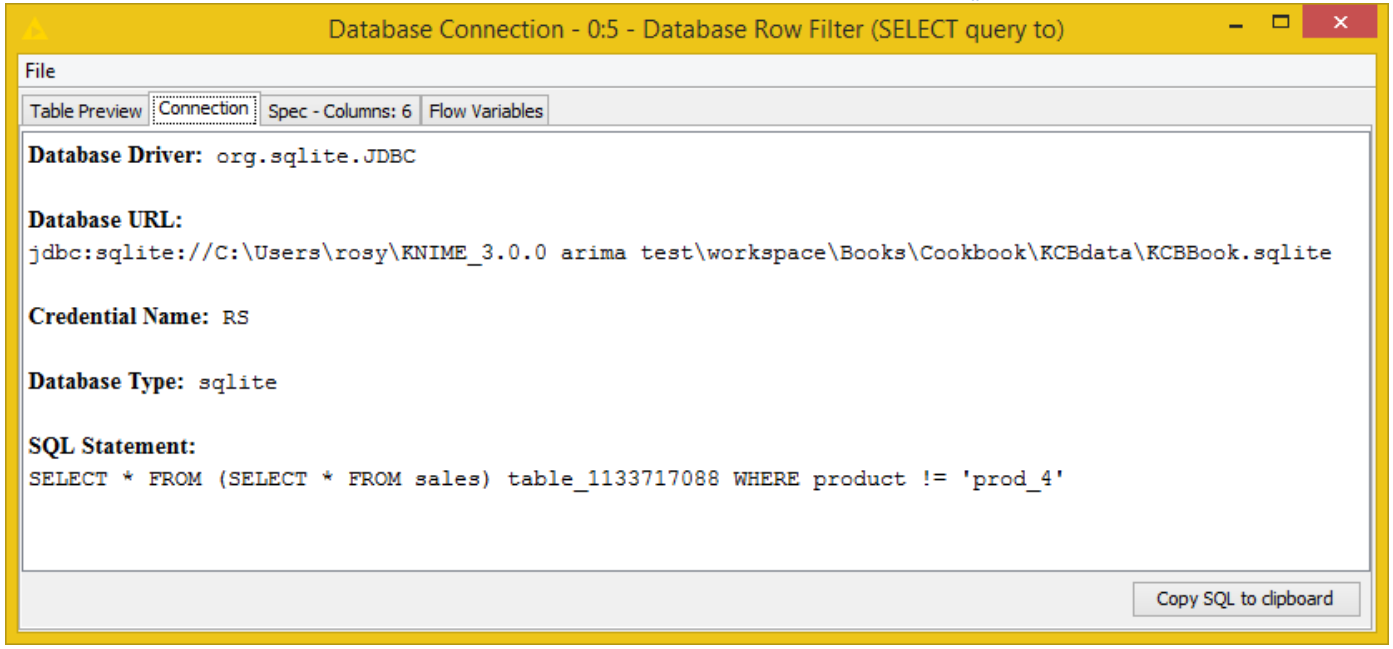

This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

**Note**. In the configuration window of the “Database Row Filter” node, for most database software tools, the matching pattern value has to be included in single quotation marks (‘…’) for string matching. Conversely, no software tools require extra delimitations for number matching.

We connected a “Database Row Filter” node to the “Database Table Connector” node. The “Database Row Filter” node was set to filter in all rows where the column “product” was different (operator “!=”) from the value “prod_4”, i.e. it was supposed to filter out all rows with value “prod_4” in the column “product”.

After execution, if we select the last item of the node context menu, named “Database Connection”, we obviously see no data table, since the node produces a SQL query and no data. However, we can see a few new tabs in the “Database Connection” window. In the “Connection” tab we can find a summary of the database connection including the final SQL query as built through the sequence of all previous database manipulation nodes. The button in the low right corner, named “Copy SQL to Clipboard”, copies this final SQL query onto the clipboard to make it available for further usage.

2.10. “Database Connection” window from the contenxt menu of the „Database Row Filter“ node




In the “Database_Operations” workflow, a “Database Column Filter” node was also introduced to follow the “Database Row Filter” node and to remove column “card” from the data set.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

## Database Column Filter

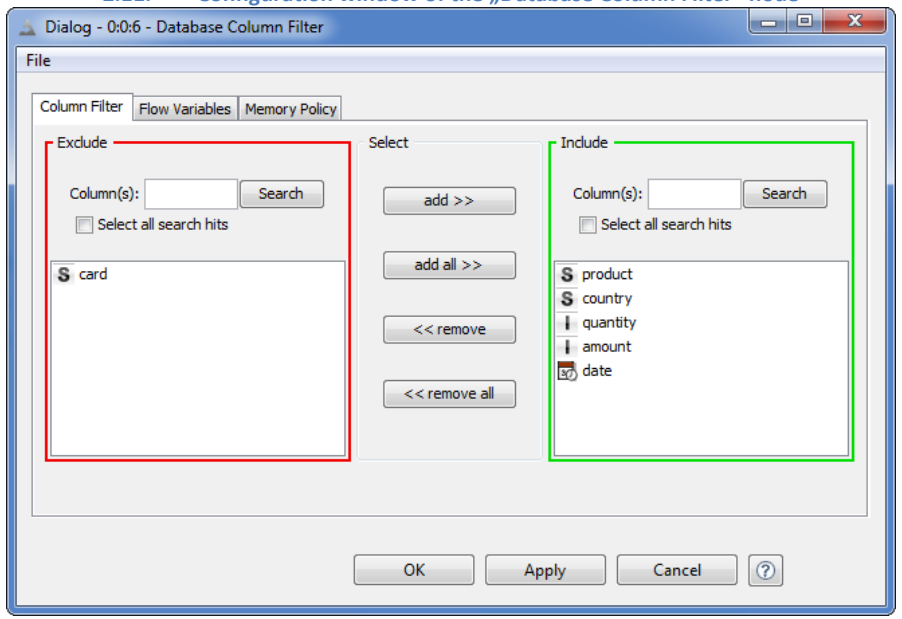
The “Database Column Filter” node customizes a SQL query to exclude or include some of the fields in the original data table.

Its configuration window is designed like the configuration window of a “Column Filter” node. That is, it is based on an “Exclude/Include” framework.

- The columns to be kept are listed in the “Include” frame on the right
- The columns to be removed are listed in the “Exclude” frame on the left

To move single columns from the “Include” frame to the “Exclude” frame and vice versa, use the “add” and “remove” buttons. To move all columns to one frame or the other use the “add all” and “remove all” buttons.

2.11. Configuration window of the „Database Column Filter“ node




A “Search” box in each frame allows searching for specific columns, in the event that an excessive number of columns impedes the data column overview.

With an appropriate knowledge of the SQL syntax, it is possible to add some SQL free code to any existing SQL query, with the node “Database Query”. This node takes a SQL statement as input, adds the SQL query written in its configuration window, and exports the total SQL query at the output port.

If you know your way around SQL, but you are not a SQL wizard, you can write smaller SQL queries and pile them up using a sequence of “Database Query” nodes.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

## Database Query

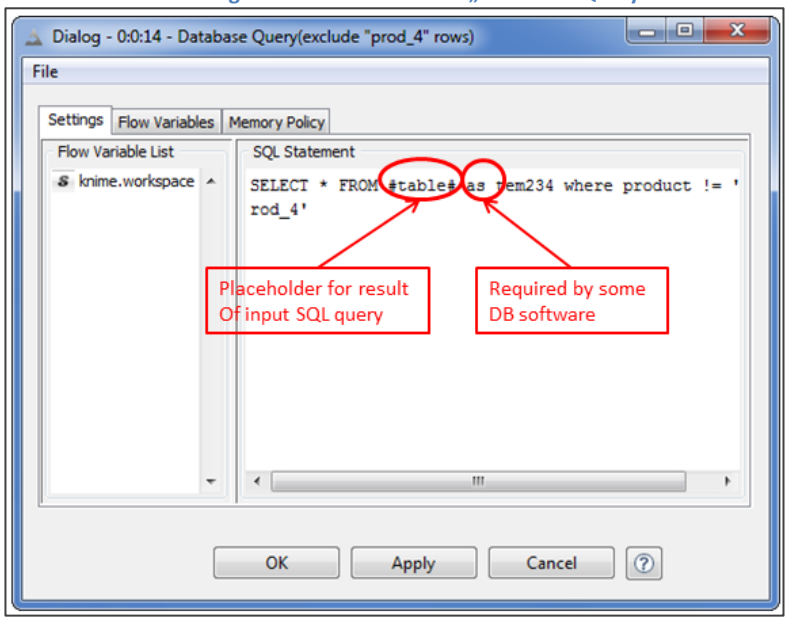
The "Database Query" node has the task of adding SQL instructions to the SQL query at its input port.

The only setting required in its configuration window is the additional SQL query.

**Notes**.

- The notation #table# is a placeholder for the result table of the input SQL Query. **Do not remove it!**
- Some database software also requires the statement "`as <new-table-name>`" to work.

2.12. Configuration window of the „Database Query" node




In the "Database_Operations" workflow, we introduced a "Database Query" node to implement our target query, including a row filter and a column filter, as described above.

```
SELECT * FROM #table# as tem234 where product != 'prod_4'
```

**Note**. The "Database Row Filter" node, as the "Database Column Filter" node and the "Database Query" node, do not operate directly on the data, they simply customize the input SQL query without executing it. In fact, all database processing nodes do not have a data output port (black triangle), but instead a database output port (brown square). This is because they do not output a data table, but just a SQL query.

Sometimes, for very large database tables, it can be useful to create a much targeted SQL query before pulling in the data. In fact, the download of very large database tables might consume all available memory and slow down the database node execution.

If you need to execute a SQL query on the database before pulling out the data, then the "Database SQL Executor" is your node. This node implements and executes a SQL query on the database connection available at its input port. It then re-presents the same database connection at the output port



This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

for further database operations. Note that the output port only has the database connection and not the SQL query that was executed. This one remains within the node and is not transmitted to the subsequent nodes.

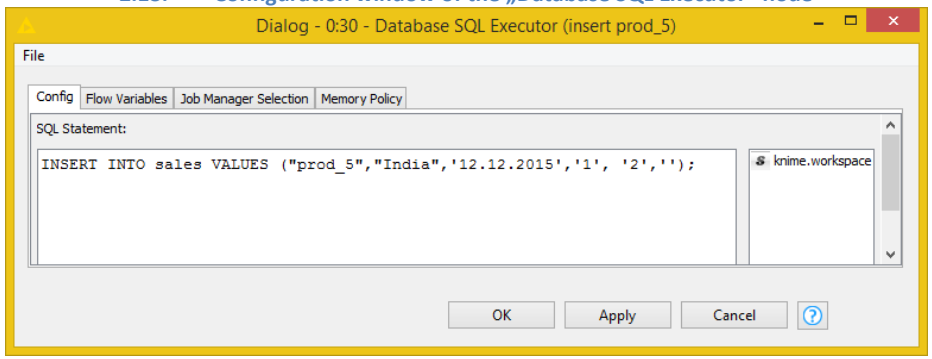
The task of the “Database SQL Executor” node is similar to the task of the “Database Query” node, besides the fact that the SQL query is not only built, but also physically executed on the database during the node execution. While the “Database Query” node produces a SQL statement that gets appended to input SQL statements after the node execution, the “Database SQL Executor” node creates the SQL statement, runs it against the selected database during the node execution, and does not export any SQL statement.

## Database SQL Executor

The “Database SQL Executor” node executes a SQL query on the database connection available at its input port.

The only configuration setting required is the SQL query.

2.13. Configuration window of the „Database SQL Executor“ node




Since later on in the “Database_Operations” workflow, we would like to use the “Database DELETE” and “Database UPDATE” node, that physically alter the content of the database, we introduced here a “Database SQL Executor” node to insert a spurious record with product ‘prod_5’ in the database, that will be removed later. As we have already said, execution of this node physically executes the SQL query on the connected database, inserting the spurious record in the database table before proceeding with the next workflow operations.

Finally, two last nodes complete the landscape of the database processing nodes: “SQL Inject” and “SQL Extract”.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

## SQL Inject

The “SQL Inject” node takes a database connection and a flow variable as input and produces a SQL query at the output port. The flow variable contains the SQL query that will be produced at the output port.

The goal of this node is similar to the goal of the “Database Table Selector” node. The only difference is in the way the new SQL query is defined: the “Database Table Selector” node builds the SQL query in an SQL editor in the configuration window, while the “SQL Inject” node takes the SQL query from the input flow variable of type String.

Since the SQL query comes from the input port and this is all the settings needed, the node requires no configuration besides the name of the input flow variable containing the SQL query.

## SQL Extract

The inverse path of the “SQL Inject” node is implemented by the “SQL Extract” node.

The “SQL Extract” node connects to a SQL query running on a database connection (brown square) and extract the SQL statement, which is output as a flow variable and as a data table. The SQL query in the flow variable port can feed a “SQL Inject” node, therefore allowing for extraction and re-execution of complex SQL statements.

The “SQL Extract” node also requires no configuration besides the name of the output flow variable.

In workflow “Database_Operations”, we used the “Database SQL Extractor” node to extract the SQL query resulting from the cascade of the “Database Row Filter” node and the “Database Column Filter” node. The final result was stored in a flow variable named “sql” and also presented at the output port of the node. The extracted SQL query is the following:



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

```
SELECT product,country,date,quantity,amount FROM (SELECT * FROM (SELECT * FROM sales) table_1133717088 WHERE product != 'prod_4') table_765435881
```

Where the outer SELECT statement is the result of the “Database Column Filter” node and the inner SELECT statement is the result of the “Database Row Filter” node.

## 2.6. Looping on Database Data

The SQL SELECT statement extracted in the previous section is equivalent to the following SQL statement:

```
SELECT product, country, date, quantity, amount from sales
WHERE product = 'prod_1' OR product = 'prod_2' OR product = 'prod_3'
```

This is a very commonly used type of SELECT query: a query looping over a number of distinct values. As there are only three values involved in the WHERE condition, this query is still manageable manually. Sometimes, though, the number of values involved in the WHERE condition can be much higher or even unknown. For example, the condition values can be the result of a previously executed SELECT query, like:

```
SELECT product, country, date, quantity, amount from sales WHERE product IN (<list-of-values-from-column>)
```

In cases like this, it can be necessary to use the values of another column as the matching patterns for the WHERE condition in the SELECT query. In our workflow, for example, we could create a data table with the values “prod_1”, “prod_2” and “prod_3” in one column and use this column’s values as the matching patterns for the WHERE condition in the SELECT query.

To create a data table from inside a workflow we can use the “Table Creator” node. The “Table Creator” node simulates an Excel Sheet and is often used to create temporary small data sets. We have introduced it into the workflow “Database_Operations” to create two data columns: one named “include” containing “prod_1”, “prod_2”, and “prod_3” and one named “exclude” containing “prod_5”. The idea would be to loop through all values in column “include” and extract the matching records from the database.

Now that we have a data column containing all values we want to match in the database, we need a node to loop on all those values and search for possible matches: the “Database Looping” node. This node has two input ports and one output port. At one input port the node expects a data table and optionally at the other input port a database connection. A data table is also produced at the output port.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

## Table Creator

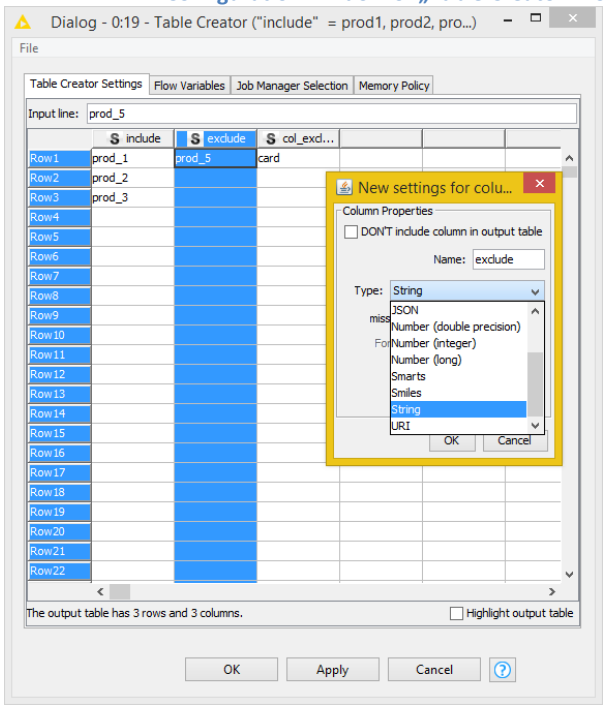
The “Table Creator” node provides a small editor to manually generate data from inside a workflow.

The “Table Creator” node does not belong to the database category and it is actually located in the category “IO” -> “Other” in the “Node Repository” panel.

The configuration window of the “Table Creator” node contains the data editor with the following properties:

- The cell content is editable
- Selecting a column and right-clicking its column header displays a menu to change the column’s properties (name, type, missing values, and format) and allows to insert or delete columns
- Selecting a row and right-clicking its RowID allows to change the RowID’s properties and to insert or remove rows
- “Copy and paste” of cells from Excel sheets is also enabled

2.14. Configuration window of „Table Creator“ node




The “Database Looping” node connects to a database, implements and executes a SQL query like:

```
SELECT * FROM <table-name> WHERE <column-name> IN ('#PLACE_HOLDER_DO_NOT_EDIT#')
```

Where, during execution, `#PLACE_HOLDER_DO_NOT_EDIT#` is substituted with the values from the selected column of the input data table, and finally imports the resulting data from the database into the KNIME workflow.

How can the database connection be optional? Indeed, this is the first node of many more to come that has a dynamic configuration window. We have already seen something similar in the configuration window of the “Database Table Connector” node, where the workflow credentials field was disabled if the selected JDBC driver was the one for the SQLite database. Here this dynamic effect is even more evident. Depending on whether a database connection is present at the input port, the configuration window gets reduced and simplified.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

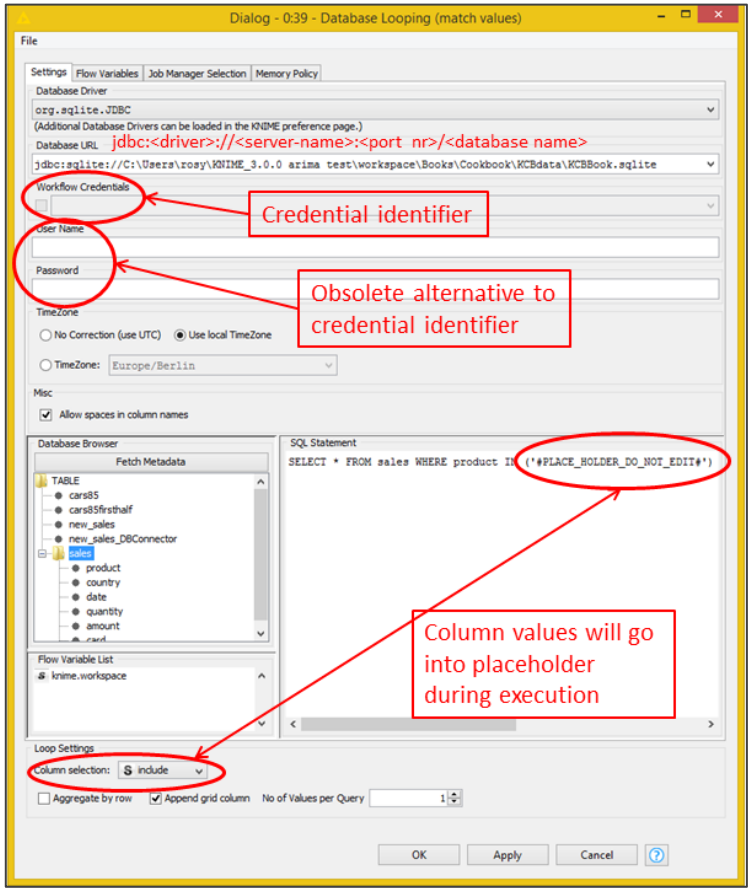
## Database Looping without Database Connection

In case a database connection is not provided at the input port, the configuration window requires :

- **Database driver**, which can be chosen from the list of databases available in the "Database Driver" menu.
- **Database URL**, which has to be built as follows:
  jdbc: <DB-software-name>://<server-name>:<port nr>/<DB-name>
- **Credentials** to access the database, if required:
  - One of the credential identifiers

  OR alternatively
  - username and password to access the database
- The **column** from the input data table whose distinct values are to replace the "`#PLACE_HOLDER_DO_NOT_EDIT#`" in the SELECT query
- The *option "aggregate by row"* that aggregates the results for each distinct value of the "`#PLACE_HOLDER_DO_NOT_EDIT#`" into a comma separated list
- The *option "append grid column"* that appends a column with the values for "`#PLACE_HOLDER_DO_NOT_EDIT#`"
- The *option "Append Grid Column"* to collect the looping values in an output data column
- The *option "No of Values per Query"* that specifies the number of distinct values from the selected column in `#PLACE_HOLDER_DO_NOT_EDIT#`.

2.15. Configuration window of the „Database Looping" node without a database connection at the input port






This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

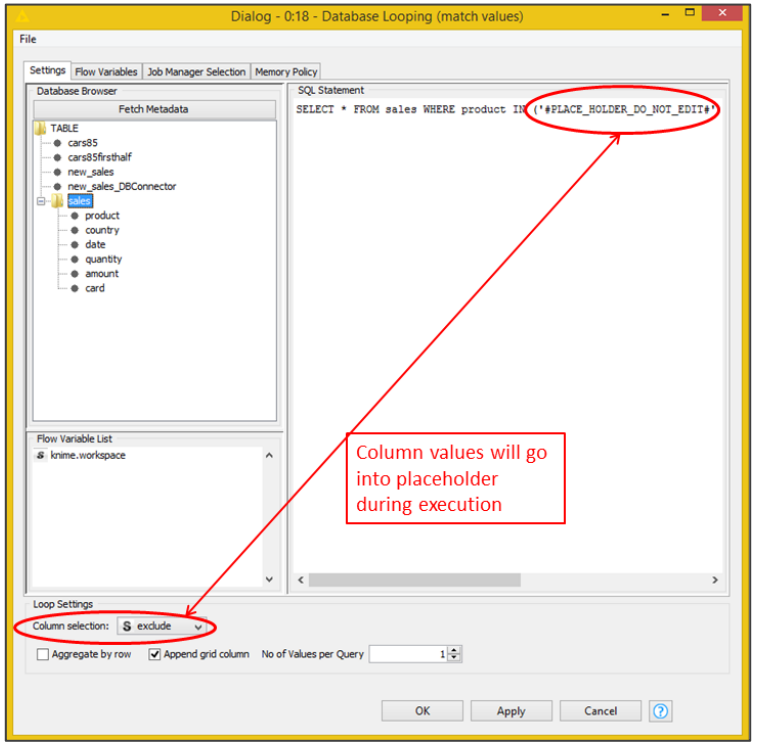
## Database Looping with Database Connection

If the database connection is available, all settings about the database are not necessary anymore.

In this case, the configuration window requires only:

- The **SQL statement** with the WHERE … IN … clause for the loop on the list of values
- The **column** from the input data table whose distinct values are to replace the "`#PLACE_HOLDER_DO_NOT_EDIT#`" in the SELECT query
- The *option "aggregate by row"* that aggregates the results for each distinct value of the "`#PLACE_HOLDER_DO_NOT_EDIT#`" into a comma separated list
- The *option "append grid column"* that appends a column with the values for "`#PLACE_HOLDER_DO_NOT_EDIT#`"
- The *option "No of Values per Query"* that specifies the number of distinct values from the selected column in `#PLACE_HOLDER_DO_NOT_EDIT#`.

2.16. Configuration window of the „Database Looping" node with a database connection at the input port




The option "No of Values per Query" limits the number of values used per query in the #PLACE_HOLDER_DO_NOT_EDIT# field. This means that each query replaces the field #PLACE_HOLDER_DO_NOT_EDIT# with only a maximum number of values from the selected column, as specified in the option "No of Values per Query". This is useful in case the selected column contains an excessive number of different values and leads to a too large query statement.

The flag "Append Grid Column", when enabled, appends a column to the input data table that includes the matching values from the #PLACE_HOLDER_DO_NOT_EDIT# field.



This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

**Note.** `#PLACE_HOLDER_DO_NOT_EDIT#` cannot be moved around in the SELECT query. The SELECT query of the "Database Looping" node can only be modified in terms of column selection and database table from which to extract the data.

## 2.7. Read and Write Data resulting from a SQL Query

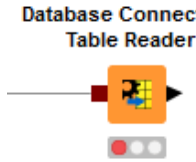
So far we have connected to the database and built a SQL query that fits our purposes. How do we run the SQL query on the database and get the results into the KNIME workflow? We need a node that executes the SQL query at the input port on the database and produces the resulting data set at the output port. We need a "Database Connection Table Reader" node.

### Database Connection Table Reader

The "Database Connection Table Reader" node executes the SQL query at its input port on the database and produces the resulting data table at its output port.

This node does not need any configuration settings, besides some rowID default specs. Everything needed, such as the database connection and the SQL query to execute, is contained in the input SQL query.

2.17. The "Database Connection Table Reader" node




**Note.** The "Database Connection Table Reader" node has a brown square as input port and a black triangle as output port; i.e. it takes a database connection with a SQL query as input and produces a data table as output.

We introduced a few "Database Connection Table Reader" nodes into the "Database_Operations" workflow, each one connected to a different branch. Since all branches though are implementing the same SQL query, either as free code, or created with the help of database processing nodes, or injected from a flow variable, you can compare the SQL results by comparing the output data tables from all these "Database Connection Table Reader" nodes.

Sometimes we do not even need to pass through the KNIME Analytics Platform. We connect to the database, we select the table, we build the SQL query, we execute it against the selected database, and we just write the results into a table in the same or in another database. In order to execute the SQL query and write the resulting data directly into a database table, without ever passing through a KNIME data table, we can use the "Database Connection Table Writer" node.



This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

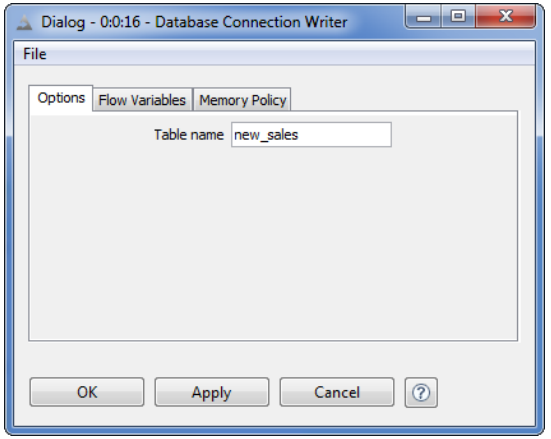
### Database Connection Table Writer

The “Database Connection Table Writer” node reads and executes a SQL query and writes the resulting data into a database table. At the input port (brown square) we find the database connection with the SQL query. No output port is present, since the data gets written into a database table and does not move up into the workflow.

The only setting required in the configuration window is the name of the database table to which the data is written. The database name is already known to the node since it is part of the database connection parameters.

If the specified table already exists, it will be dropped and recreated (i.e. overwritten).

2.18. Configuration Window of the „Database Connection Table Writer“ node




We introduced one “Database Connection Table Writer” node to write the data resulting from the sequence “Database Row Filter” + “Database Column Filter” into a new database table named “new_sales”.

## 2.8. All in one Node: Database Connection, Table Selection, SQL Query, and Exporting Data

We have seen that, if we can write little SQL, we do not need the extra nodes to build a SELECT query. We can write the SELECT query directly into a “Database Table Connector” node and then read the data into the workflow by means of a “Database Connection Table Reader” node. Actually, we can reduce the number of nodes even further by using a “Database Reader” node.

The “Database Reader” node has an optional input port for database connection and an output port for a data table. Indeed, the “Database Reader” node is one of those database nodes with a dynamic configuration window. If a database connection is provided at its input port, all settings necessary for the database connection are taken from there and disappear from the configuration window. However, if the database connection is not available at the input port, such information has to be provided inside the configuration window.

The “Database Reader” node performs all database operations:



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

- Connects to the database
- Reads and executes the SQL statement in its configuration window, including the database table selection
- Pulls the resulting data from the database into a KNIME data table

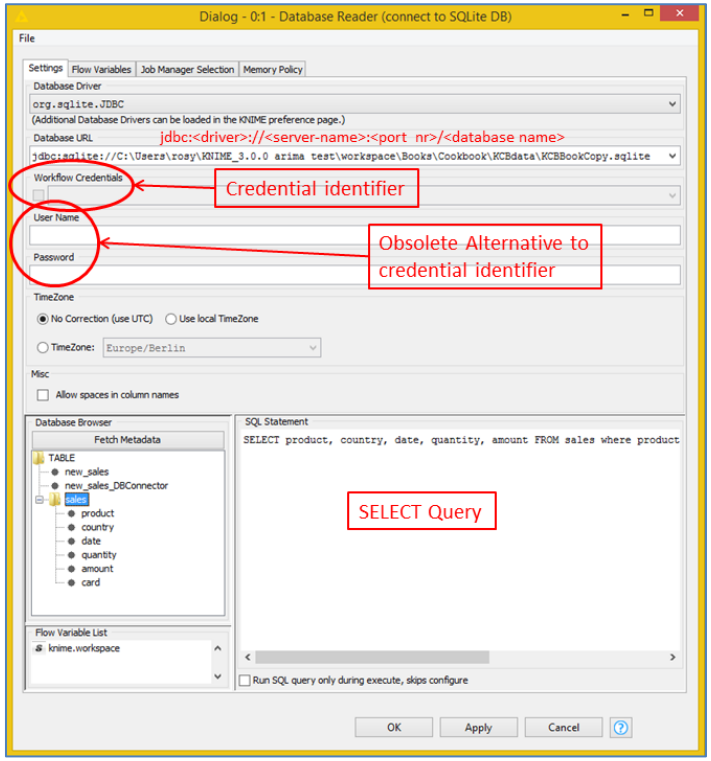
## Database Reader without Database Connection

The configuration window of the "Database Reader" node, when no database connection is provided at its input port, is similar to the configuration window of the "Database Connector" node and requires the following values:

- **Database driver**. Most commonly used database drivers have been preloaded in KNIME and are available from the "Database Driver" menu. If your database driver is not in the list, check above how to upload a new database driver into KNIME.
- **Database URL**. The database URL has to be built as follows:
  jdbc: <DB-software-name>://<server-name>:<port nr>/<database-name>
- **Time Zone** for date and time values.
- The **credentials** to access the database:
  - One of the credential identifiers available in "Workflow Credentials"

  OR alternatively (but deprecated):
  - Username and password
- The **SQL SELECT statement** to retrieve the data from the database.
  Here the "Database Browser" panel can help with building the SQL query. Just hit "Fetch Metadata" to see the database structure and double-click table and field names to make them appear automatically in the SQL query with the right syntax.

2.19. The configuration window of the "Database Reader" node when NO database connection is available at its input port




With some database software, it is possible to use multiple SQL statements in the SQL editor. In this case, the SQL statements have to be separated with a semicolon (";").



This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

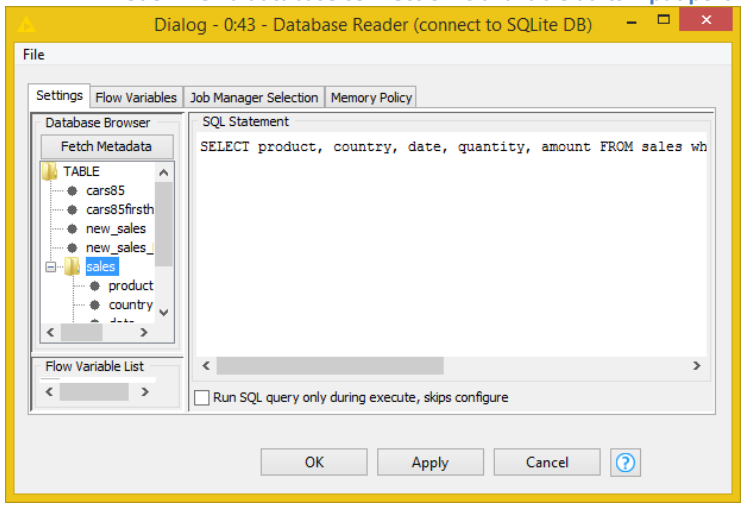
**Note**. The disadvantage of using a standalone "Database Reader" node consists of the impossibility to generate more than one customized data table from the same database connection.

## Database Reader with Database Connection

The configuration window of the "Database Reader" node, when a database connection is provided at its input port, only requires the SQL SELECT statement to retrieve the data from the database table.

Here the "Database Browser" panel can help with building the SQL query. Just hit "Fetch Metadata" to see the database structure and double-click table and field names to make them appear automatically in the SQL query with the right syntax.

2.20. The configuration window of the "Database Reader" node when a database connection is available at its input port




Similarly to the "Database Reader" node, KNIME offers a "Database Writer" node. The "Database Writer" node writes data from a data table input port into a database table. The "Database Writer" node has of course no output port, since it does not output any data.

The "Database Writer" node configuration window consists of three tabs: "Settings", "SQL Types", and "Advanced".

The "Advanced" tab only contains the number of rows to be written in each batch job. For higher performance you should choose a higher number. However, a too high number might need more memory.

The "Database Writer" node also has a dynamic configuration window for the "Settings" tab, depending on whether a database connection is available at the input port. The configuration window switching works similarly to that one of the "Database Reader" node.



This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

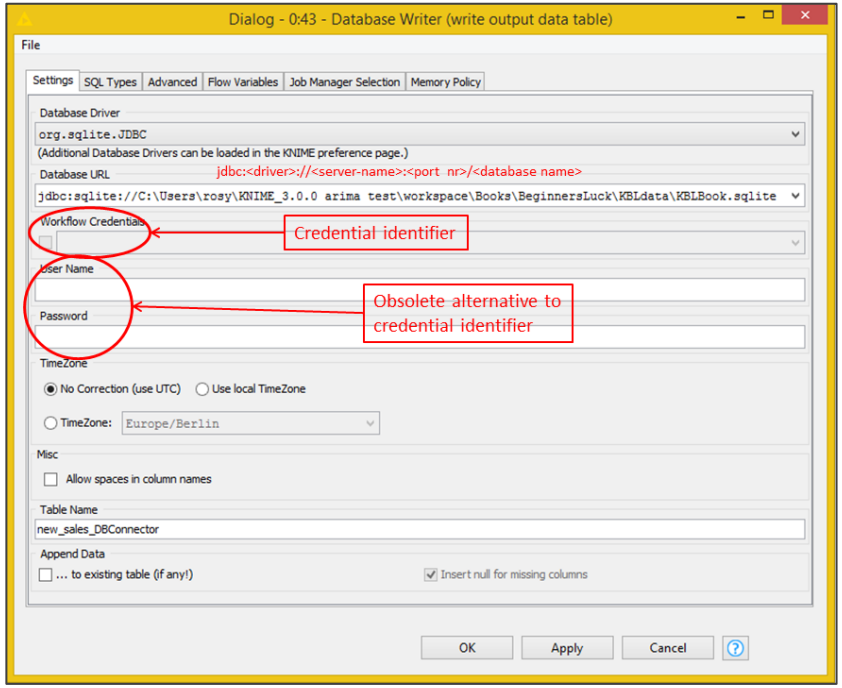
## Database Writer: Settings tab with no Database Connection

If no database connection is available at its input port, the configuration window requires the following values:

- **Database driver**. The database driver can be chosen from the list of all database drivers available in the “Database Driver” menu.
- **Database URL**. The database URL has to be built as follows:
  jdbc: <DB-software-name>://<server-name>:<port nr>/<database-name>
- The **credentials** to access the database:
  - One of the credential identifiers available in the “Workflow Credentials” menu

  OR alternatively (but deprecated)
  - Username and password
- **Time Zone** for date and time values
- The name of the **table** to write into
- The *append* or *overwrite* flag in case the table already exists in the database
- The option on whether to *allow spaces* in column names

2.21. Configuration window of the „Database Writer“ node: the “Settings” tab






This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

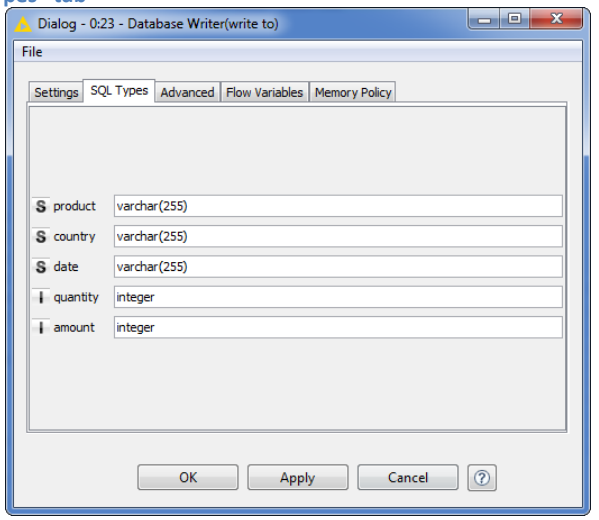
## Database Writer: SQL Types tab

The “Database Writer” node has a second tab in the configuration window: the “SQL Types” tab.

This tab contains the types of the data columns to be used to write into the database.

The type textboxes are editable and you can change a column type by typing a new type in the textbox. Of course the new type has to be compatible with the actual data column type.

2.22. Configuration window of the „Database Writer“ node: the “SQL Types” tab




In the lower part of the “Database_Operations” workflow we introduced a “Database Reader” node, without providing it with a database connection at its input port and using:

- The SQLite JDBC driver
- The “KBLBook.sqlite” database located in the “KBLdata” folder in the “KNIME Explorer” panel
- No credentials, since SQLite database does not require authentication
- The SQL statement `SELECT product, country, date, quantity, amount FROM sales where product != 'prod_4'`

In the upper right corner we introduced a “Database Writer” node to write into a new table named “new_sales_DBConnector” in the database provided by the input database connection.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

## 2.9. Database UPDATE and DELETE Commands

Two more “all-in-one” nodes are available in the “Database” category: the “Database Update” and the “Database Delete” node. Both nodes connect to a database and perform a specific query (update or delete) on a specific subset of data as defined by the “Select WHERE Columns” panel in their configuration window. The “Select WHERE Columns” panel uses the values in the selected column(s) to match the values in the corresponding database table field(s) for the WHERE clause.

These nodes also have a dynamic configuration window, bypassing the database connection settings if a database connection is provided at their optional database connection input port.

Both nodes have an input port for a data table, an optional input port for a database connection, and an output port to export a data table. The data table at the input port is used for the WHERE and SET conditions in the DELETE and UPDATE statements.

### Database Delete

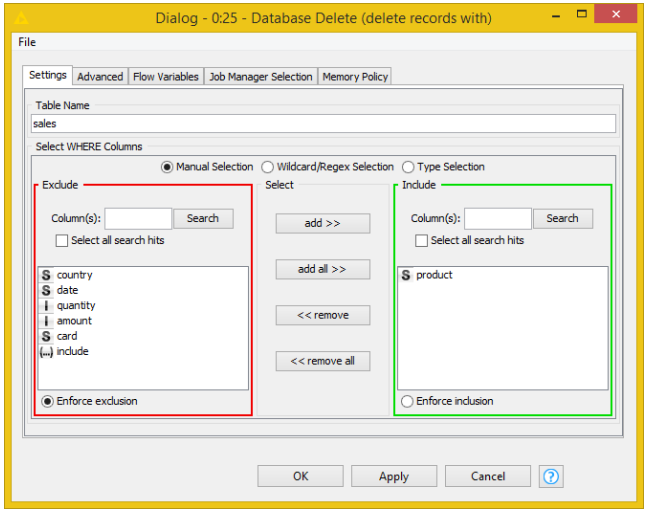
The “Database Delete” node deletes all records in the selected table that match the WHERE clause.

The WHERE clause uses the selected column(s) in the Include/Exclude frame.

The WHERE clause identifies the table field (<field>) with the same name as the selected column.

All records in the table, with <field> value matching one of the values in the selected column, are then deleted during execution through a DELETE statement.

2.23. Configuration window of „Database Delete“ node with database connection






This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

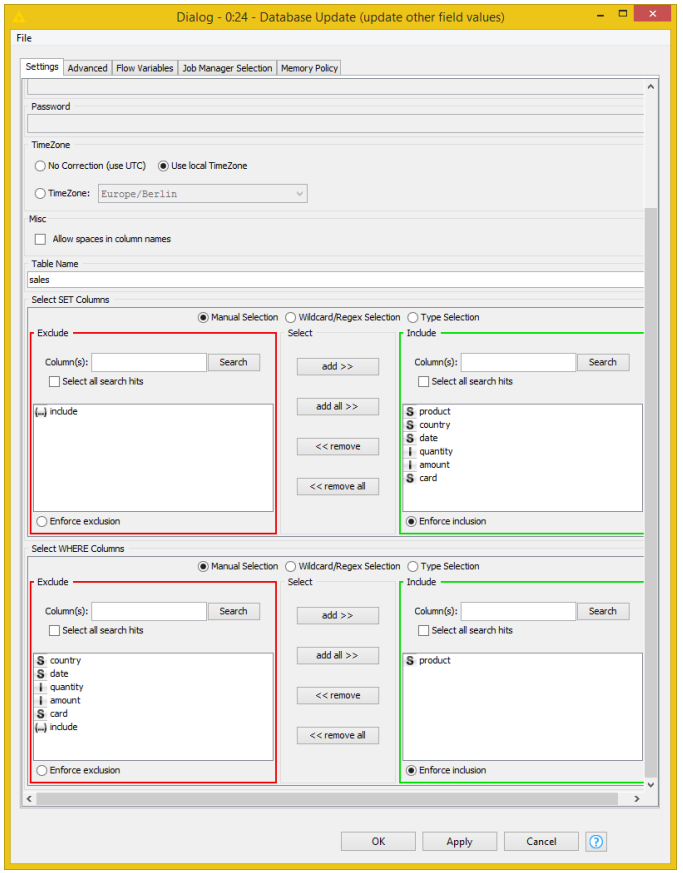
## Database Update

The “Database Update” node updates the values of some table fields through a SET WHERE statement.

The WHERE clause is built like for the “Database Delete” node. That is, the WHERE clause identifies the table field (<field>) with the same name as the selected column in the “Select WHERE Columns” frame and then finds all those records with <field> value matching one of the values in the selected column. Let’s call this the WHERE set.

Record values in the WHERE set are then changed, according to the SET condition. All field(s) with the same name as the selected column(s) in the “Select SET Columns” frame, take on the value(s) of the SET columns, through an UPDATE statement.

2.24. Configuration window of „Database Update“ node without database connection

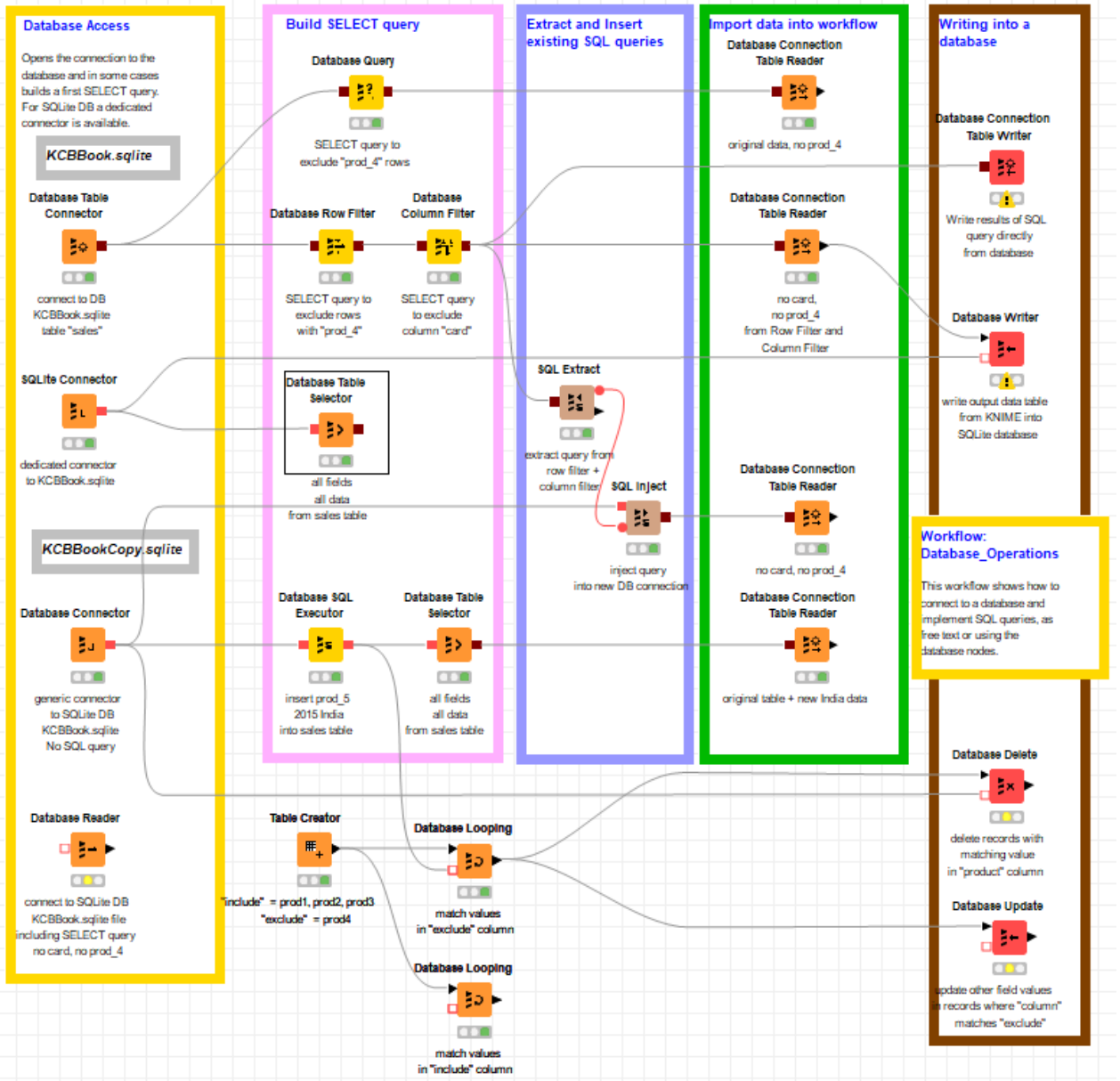



The final workflow, named “Database_Operations”, is shown in the figure below.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

2.25. The “Database_Operations” workflow in folder “Chapter2” shows a number of nodes for database operations. From right to left: Database Connectors (dedicated and generic), database manipulation nodes, utility nodes, readers and writers.






This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

## 2.10. Big Data Platforms and MongoDB

Amongst the dedicated connectors, there are a few dedicated to big data platforms, such as Apache Hive, Impala Cloudera, HP Vertica, and more. These nodes can be obtained by installing the KNIME Big Data Extension.

As for all databases, you can connect to a big data platform using either the dedicated connector or the generic “Database Connector” node. If you use the generic “Database Connector” node you will need to supply the JDBC driver file, usually provided by the big data platform vendor.

As for all databases, after the connection has been established, you can use a “Database Table Selector” node for table selection and a sequence of database manipulation nodes and “Database Query” nodes to build the desired SELECT query.

However, writing to a big data platform is not possible through a database writer, database update, or database delete node. For big data platforms we need to use one of the following nodes.

- An “HDFS Connection” node or one of its related nodes (“WebHDFS Connection” and “SecureWebHDFS Connection”) followed by an “Upload” node to upload data onto the HDFS platform; followed by a “Download” node to download the data from HDFS. All these nodes are available in “File Handling”/”Remote” category.

- An “HttpFS Connection” node followed by a “Hive Loader” or an “Impala Loader” node, to load data on a Hive or Impala database. These nodes are available in the “Database” category, either under “Connectors” or under “Reade/Write”.

- An alternative way to load data onto a big data platform is to go through Spark and use the Parquet format. These nodes are available in the “Apache Spark” category.

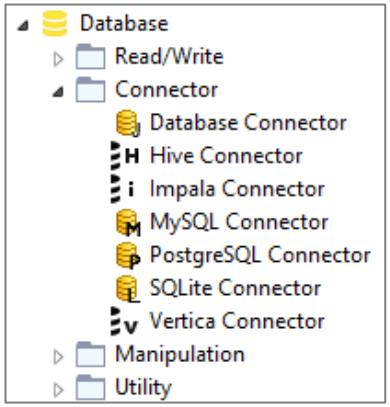
If you have opted for NoSQL databases, such as MongoDB, CouchDB, or NewSQL, in general you need to rely on REST service nodes in the REST Web Services category (see chapter 3) to extract information.

Indeed, for now only nodes to read, write, update, save, and remove records in a MongoDB database are available in the KNIME Analytics Platform under the “KNIME Labs”/”MongoDB” category.

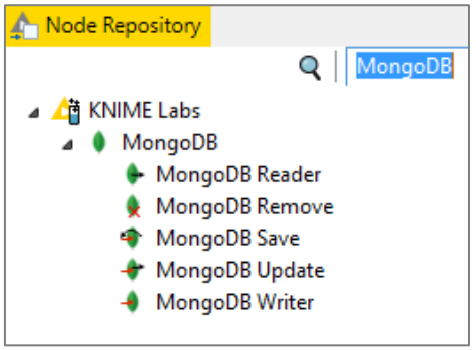


This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

2.26. Dedicated connector nodes to big data platforms available through the KNIME Big Data extension




2.27. The MongoDB nodes in "KNIME Labs"




## 2.11. Exercises

### Exercise 1

Create an empty workflow, called "Exercise1", in an "Exercises" workflow group under the existing "Chapter2" workflow group.

The "Exercise1" workflow should prepare the database tables for the next 2 exercises. It should therefore:

- Read the file "cars-85.csv" (from the "KCBdata" folder);
- Write the data to an SQLite database table named "cars85" in KCBBook.sqlite database;
- Write only the first 20 rows of the data into a table called "cars85firsthalf" in the same KCBBook.sqlite database.

**Solution to Exercise 1**

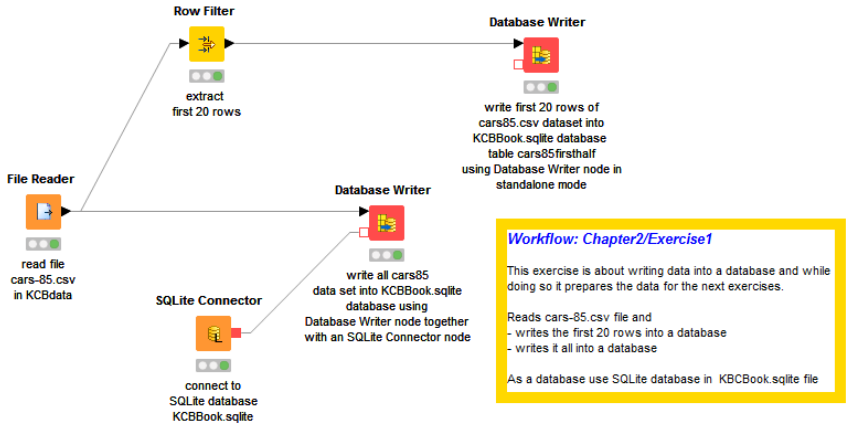


This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

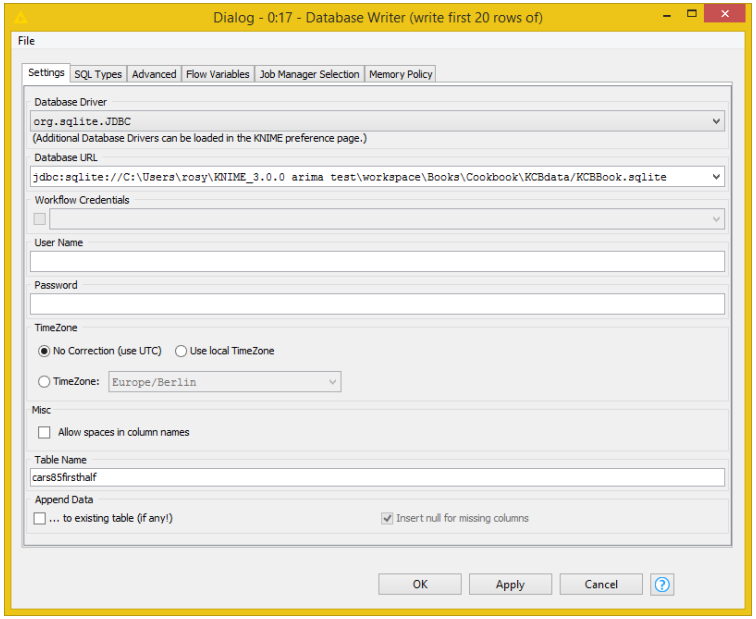
The solution workflow to Exercise 1 is shown in figure 2.28. The "Database Writer" node is used once in standalone mode, therefore requiring JDBC driver and full path to the database, and once in conjunction with a "SQLite Connector" node.

Figure 2.29 and 2.30 show the different configuration windows for the "Database Writer" node for the two cases.

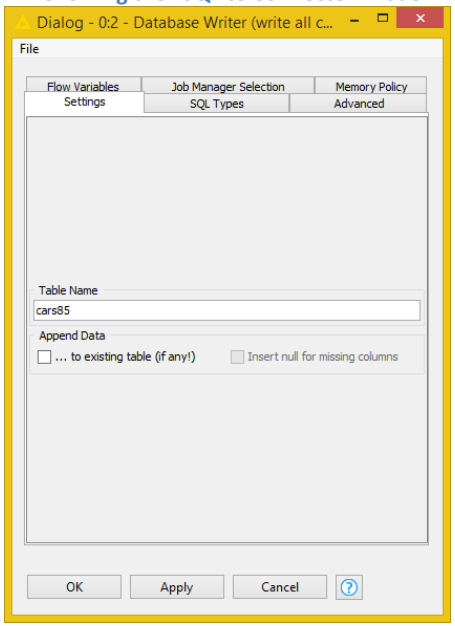
### 2.28. Exercise 1: The workflow




### 2.29. Exercise 1: Configuration window of the "Database Writer" node in standalone mode




### 2.30. Exercise 1: Configuration window of the "Database Writer" node following the "SQLite Connector" node






This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

**Note.** The use of username and password instead of a credential identifier is discouraged in database nodes. In fact the Workflow Credentials are automatically encrypted, while the username and password are only encrypted if a "Master Key" is activated in the "Preferences" page. In general, usage of "Workflow Credentials" is more secure than using the username and password for each database node.

## Exercise 2

In the workflow group named "Chapter2\Exercises" create a workflow called "Exercise2" to perform the following operations:

- Connect to "KBCBook.sqlite" database and read "cars85" table
- Remove the first two columns: "symboling" and "normalized_losses"
- Only keep rows where "make" is "bmw" or "audi"

**Solution to Exercise 2**

There are many ways to implement this exercise. We propose four of them:

- "Database Table Connector" + in-database processing nodes
- "Database Connector" + "Database Table Selector" + in-database processing nodes for column selection + SQL query for row filter (WHERE)
- "SQLIte Connector" + "Database Reader" + full SQL query
- All in-one "Database Reader" + full SQL SELECT query

The full SQL SELECT query for the last two options is:

```
SELECT make, fuel_type, aspiration, nr_doors, body_style, drive_wheels, engine_location, wheel_base,
length, width, height, curb_weight, engine_type, cylinders_nr, engine_size, fuel_system, bore, stroke,
compression_ratio, horse_power, peak_rpm, city_mpg, highway_mpg, price
FROM cars85 where body_style = 'wagon' OR body_style = 'sedan'
```

The SELECT query is a bit tedious to write mainly because of all the columns of the table we want to keep. This same SQL query can be implemented with a combination of "Database Column Filter" and "Database Row Filter" nodes (first approach from the list).

Alternatively we can use a simple SQL query for the row filter and a "Database Column Filter" node (second approach in the list). The simple SQL query for the row filtering part then takes the shape:

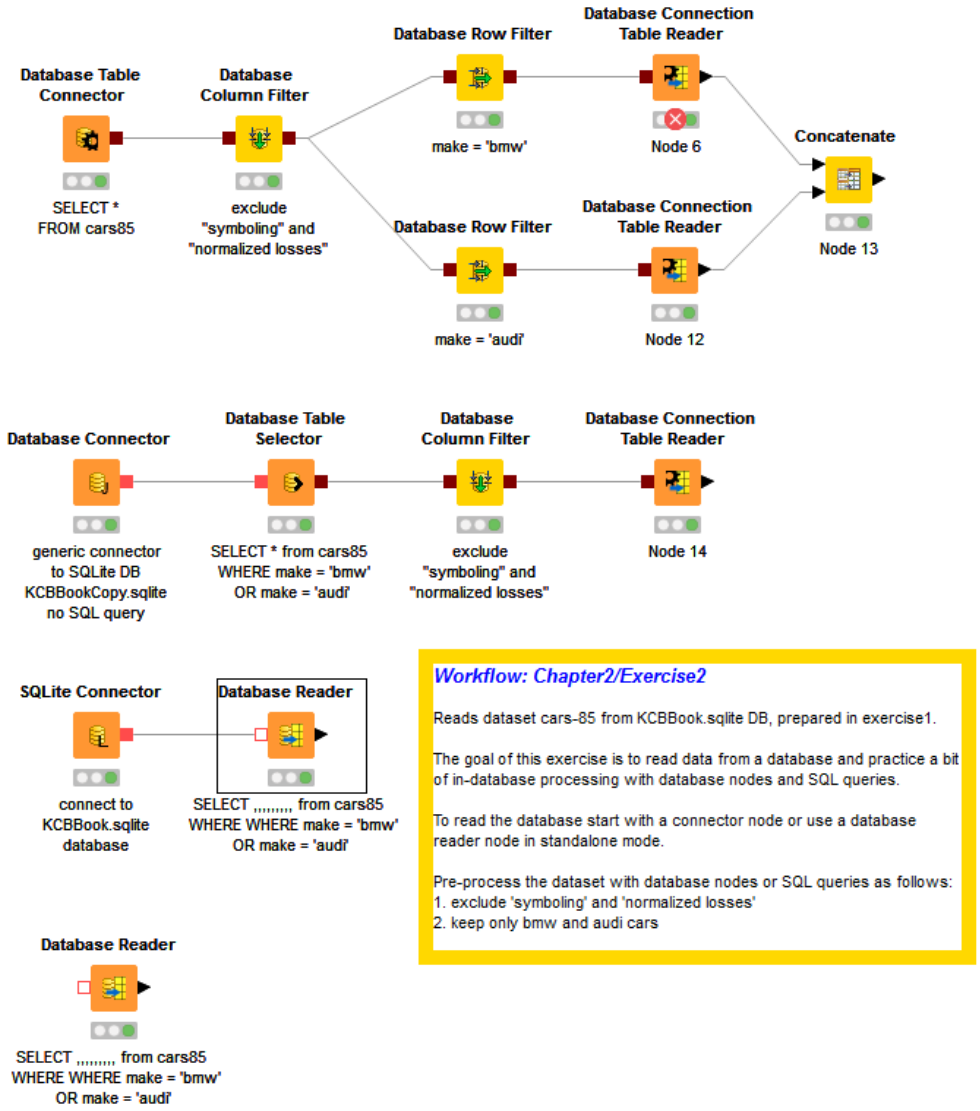


This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

```
SELECT * FROM cars85 where body_style = 'wagon' OR body_style = 'sedan'
```

All described approaches are shown in the figure below.

### 2.31. Exercise 2: Accessing and filtering data with SQL queries and/or in-database processing nodes






This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

## Exercise 3

In the “Chapter2\Exercises” workflow group create a workflow called “Exercise3”. The goal of this exercise is to practice with the Database Looping node.

Extract “make”, “nr_doors”, “length”, “width”, and “engine_type” from table “cars-85” in KCBBook.sqlite database, as prepared in exercise1. Like in exercise 2, pre-process the dataset to keep only “bmw” and “audi” cars. This time though use a “Database Looping” node instead of SQL queries and database filter nodes.

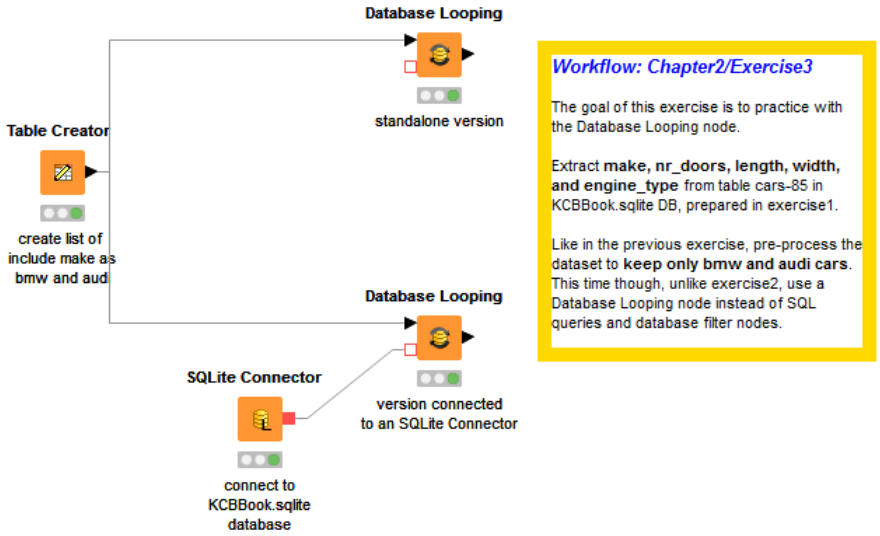
**Solution to Exercise 3**

For the solution of this exercise we need the list of the values to keep for attribute “make”: “audi” and “bmw”. We build this with a “Table Creator” node in a column named “include-make”.

Afterwards a “Database Looping” node loops on all the distinct values in column “include-make” and keeps the rows in the database table where “make” matches any of these values.

The “Database Looping” node is used in standalone mode and in conjunction with a dedicated connector node.

2.32. Exercise 3: Workflow






This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

# Chapter 3. Accessing Information from the Web

We are all familiar with the World Wide Web, the Web: a huge data pool whose resources can be accessed through Universal Resource Locators (URLs). Accessing information on the web through KNIME Analytics Platform allows us to retrieve and manipulate data in real-time.

An important repository of information for public and private documents resides in the whole Google environment: docs, sheets, drives, and so on. Similarly to Excel sheets, KNIME Analytics Platform can equally access Google Sheets.

## 3.1. Accessing Google Sheets

One powerful and simple way to create and edit data online while collaborating with other users in real-time is via Google Sheets. KNIME Analytics Platform offers a full set of Google Sheets dedicated nodes. A number of such nodes are available to connect, read, write, update, and append cells, rows, and columns into private or public Google Sheets. They can be found in the Node Repository under:

- "KNIME Labs"/"Google API Connectors (Labs)"/"Google Sheets Connectors"
- "Social Media"/"Google API Connectors"

Let's start with connecting generically to the Google API via the Google Authentication node. This node provides a generic connection to the Google API, which can then be used to connect to a Google Sheet or other Google services.

Notice that user credentials to access Google are never seen by KNIME. All KNIME keeps in memory is the authentication derived from the authentication operation on the Google access page. Even the authentication key is not necessarily saved anywhere on your hard-disk.

**Note**. KNIME Analytics Platform does not store your Google credentials, just the authentication key (if at all). Your Google credentials are entered in the Google sign-in page and not in KNIME.

If you do not want to share the Google authentication key, remember to make the workflow forget about it by clicking the button "Clear Selected Credentials" in the configuration window of the Google Authentication node.



This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

## Google Authentication

This node connects to the Google API and authenticates the user via the Google authentication page.

One must authenticate using the Authenticate button. The user is then redirected to the Google authentication page on the user's default web browser, where he/she is asked to grant access to a number of Google services.

By default, the authentication key is kept in memory and **not stored** in the node or anywhere else on the machine, unless differently specified in the configuration window. If the authentication key is kept in memory, user must authenticate again at each new KNIME session.

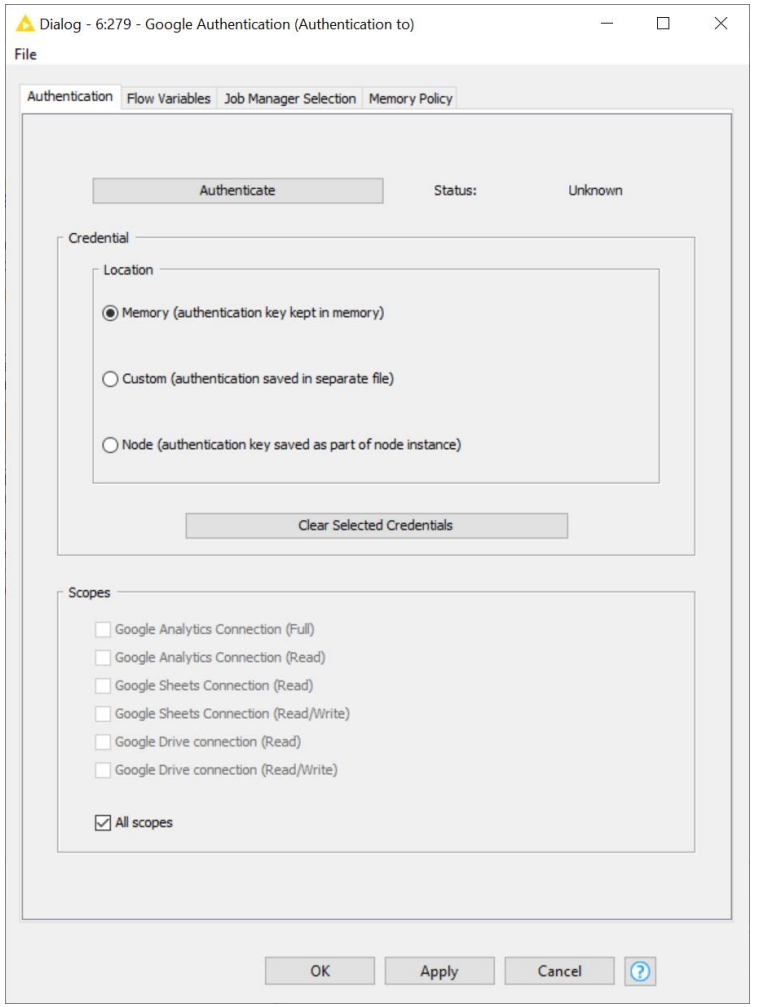
The authentication key can be stored in a file via the "Custom" option. In this case, a folder must be specified where to store the key. This option is useful in case a shared authentication key is used.

The authentication key can also be stored together with the node. In this way, the user does not need to re-authenticate at the next instance of KNIME Analytics Platform.

If the same session of KNIME Analytics Platform is kept alive and the user does not want to share his/her authentication key with other users, the button "Clear Selected Credentials" allows the workflow to forget about the authentication key that has been stored in memory.

The extent of the number of Google services accessible via the authentication key is defined in the lower part of the configuration window. The default option is that all Google services will be accessible via the authentication key. However, a more restricted access can be set.

3.1. Configuration window of „Google Authentication" node






This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

Among all available Google services, after authentication, we now want to access the Google Sheets service. The node to do that is the “Google Sheets Connection” node.

### Google Sheets Connection

The Google Sheets Connection node creates a connection to Google Sheets, given an existing Google API connection at its input port.

Since the whole required information is already contained in the authentication key at its input port, no other configuration setting is required and no configuration window is provided.

It is clear that the input authentication key must be enabled to access the Google Sheets service even in Read/Write mode, if required.

Now, let’s read a tab of a Google sheet. To read a tab ("sheet") of a Google spreadsheet, we use the “Google Sheets Reader” node.

In the “Google Sheets Reader” node, the spreadsheet can be selected among all spreadsheets with access permisison.

Inside the selected sheet, you can specify a specific range of cells to be read. The range must be entered in A1 notation (E.g. "A1:G10"). For more information about A1 notation visit: https://developers.google.com/sheets/api/guides/concepts#a1_notation .



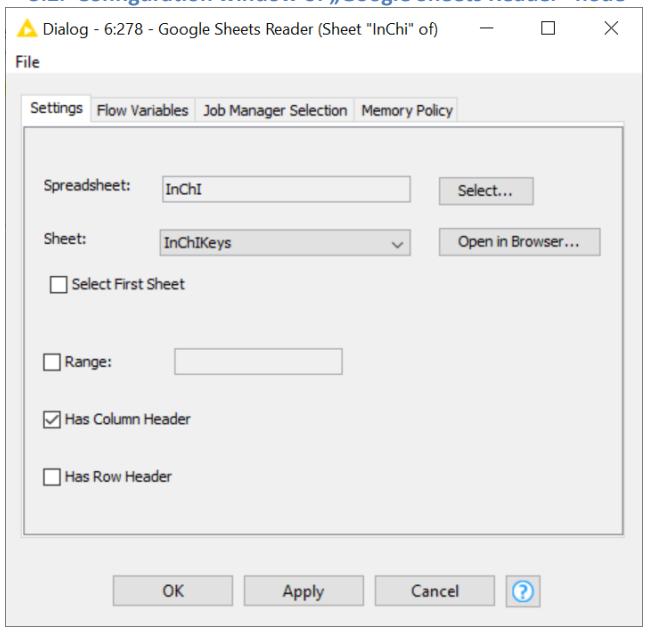
This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

## Google Sheets Reader

The Google Sheets Reader accesses data from one tab “sheet” of one Google spreadheet. The following settings can be configured:

- **Spreadsheet**: One can simply select a spreadsheet from the list of spreadsheets available on Google Drive. Clicking on button *“Select…”* opens a dialog with the list of available spreadsheets from Google drive. If a document doesn’t appear in this list, make sure that you have permissions to access it and that you have opened it once within a browser to associate it with your Google account.
- **Sheet:** Select the sheet from the spreadsheet that should be read. Available sheets can be selected from the drop-down menu. The button “Open in Browser…” opens the selected spreadsheet in the browser. This is useful to ascertain whether that is the sheet of interest.
- **Select First Sheet**: When selected, the first sheet of the spreadsheet will be read instead of the one selected from the drop-down menu.
- **Range**: The range of cells that should be read from the sheet can be specified here in A1 notation. (E.g. "A1:G20").
- Two checkboxes that specify if the sheet includes column names and row ids, respectively.

3.2. Configuration window of „Google Sheets Reader“ node

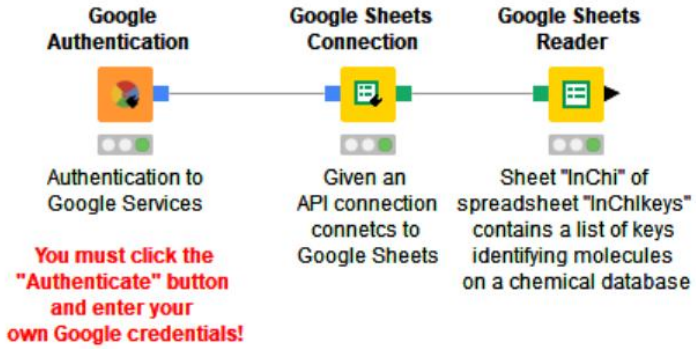



To translate these notions into a practical workflow, we built a workflow in “Chapter3” named “Access_GoogleSheets”. This workflow accesses the sheet named “InChIKeys” from a public Google spreadsheet named “InChI”. This sheet contains a list of InChI keys to access molecules on a chemical database. We first connected to the Google Spreadsheet using the “Google Sheets Interactive Service Provider” and retrieved the data from the Spreadsheet “InChI” and the Sheet “InChIKeys” using the “Google Sheets Reader” node.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

### 3.3. Workflow “Access_GoogleSheets” accessing an reading content from a Google Spreadsheet




There are more nodes covering different functionalities on Google Sheets. We have not introduced these nodes in the example workflow, but we will cover them briefly in the rest of this section. For example, the node “Google Sheets Appender” adds a new sheet to an existing spreadsheet in Google Sheets.

The configuration settings are very similar to the settings of the “Google Sheets Reader” node, plus a few additional writing settings.

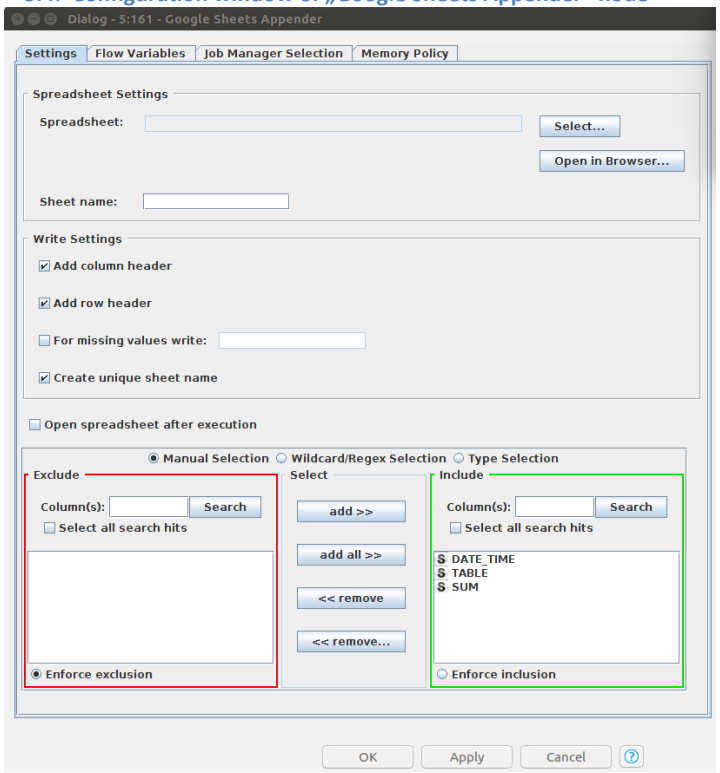


This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

## Google Sheets Appender

- **Spreadsheet** and **Sheet name** require the selection of the spreadsheet among one of the available spreadsheets on Google drive and the selection of the sheet inside the spreadsheet.
- **Add column/Add row  header:** Specifies, whether the column names should be written in the first row and whether the row ID's should be written in the first column of the spreadsheet.
- **For missing values write:** By selecting this option, you can specify a string you want to substitute for missing values. If the option is left unchecked, the cells with missing values remain empty.
- **Create unique sheet name:** The node will create a unique sheet name based on the given sheet name. (Example: Should 'SheetOne' already exist, the unique sheet name will be 'SheetOne (#1)')
- **Open spreadsheet after execution:** Opens the spreadsheet after it has been written successfully. The spreadsheet will be opened in the systems' default browser.
- **Exclude/Include columns:** Uses a classic Include/Exclude frame to select the columns that will be written to the sheet file.

3.4. Configuration window of „Google Sheets Appender“ node




Instead of appending a new sheet, we might want to update an existing one.  For that, we use the “**Google Sheets Updater”** node.



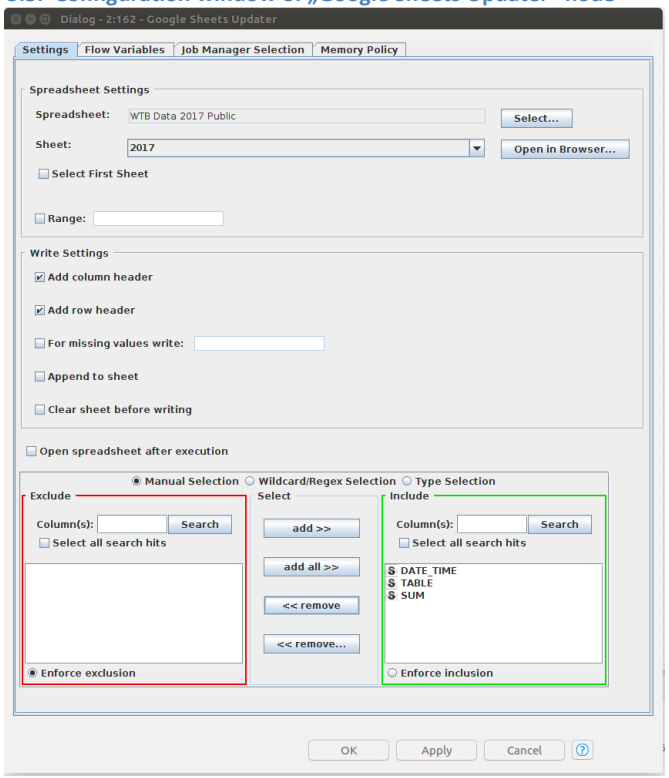
This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

## Google Sheets Updater

The “Google Sheet Updater” node enables us to write an input data table to an existing Google sheet. It can overwrite some or all of the content of a sheet or append to the content of a sheet. The dialog settings are the same as in the “Google Sheets Appender” node, with a few additional options:

- **Range**: If only part of the content of a sheet is to be overwritten, the relevant range can be specified in A1 notation. The size of the input table must not exceed the size of the selected range, otherwise execution will fail.
- **Append to sheet:** When this option is selected, the data table content will be appended to the selected sheet. This means we append data to an existing sheet and do not add a new sheet into an existing spreadsheet as we did with the “Google Sheets Appender” node.
- **Clear sheet before writing:** When this option is selected, the sheet or the selected range of the sheet will be cleared before writing. This deletes the content in the specified sheet/range.

3.5. Configuration window of „Google Sheets Updater“ node




If you want to write the input data table to a new Google Sheets spreadsheet instead of overwriting or appending data to an existing sheet, you can use the “Google Sheets Writer” node.

Taken together, these nodes enable us to access and manipulate data in Google Sheets through a KNIME workflow.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

### Google Sheets Writer

The “Google Sheet Writer” node enables us to write a Google sheet.

The configuration settings are the same as for the “Google Sheets Updater” node.

However, **Spreadsheet** and **Sheet name** cannot be selected from a dropdown menu of existing spreadsheets but must be entered manually.

## 3.2. Accessing REST Services

Application to application communication is becoming more widespread. These methods of communication between applications over a network are referred to as Web Services. Web Services have been standardized as a means of interoperating between different operating systems running different software programs in different programming languages.

Nowadays a number of web services are available, such as financial estimations, weather reports, chemistry related data. This means that in many data related fields, tools and data have been made directly available in the form of web services. Often it is enough to connect to such web services and send the appropriate message request with the appropriate input data to get the required information. Following the knowledge recycling principle, meaning that it is better to use a tool available on the web rather than to re-implement it ourselves, we would like to be able to connect to and run a web service inside any of our workflows. One very powerful way to do this is using the REST Web Services nodes.

Web services that follow the **REpresentational State Transfer** (**REST**) architectural principles [3] offer a robust and flexible way to access and manipulate textual representations of web resources. Since version 3.2, KNIME Analytics Platform provides a category, named “REST Web Services”, which contains nodes that enable the user to interact with REST services.

The “REST Web Services” category contains 4 nodes - “GET Request”, “POST Request”, “PUT Request”, and “DELETE Request” – to deal with REST operations. The “REST Web Services” category gets installed onto the “Node Repository” panel through the REST Client Extension. As to install any KNIME Extension Package, go to the Top Menu:

- Select “File”
- Select “Install KNIME Extensions…”
- From the “Available Software” window, in the top box containing “type filter text”, type “web”



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

- Inside “KNIME & Extensions” select the package called “KNIME REST Client Extension”
- Click “Next”
- Follow the installation instructions

After the package has been successfully installed, you should find a category named “REST Web Services” in the “Tools & Services” category in the “Node Repository” panel.

In order to show the potentialities of this “REST Web Services” nodes, we created two empty workflows in “Chapter3” folder and we named them “GETRequest” and “POSTRequest”.

The “GETRequest” workflow accesses the JSONPLaceholder API to retrieve fake post content from fake user IDs. JSONPlaceholder API is a development test utility for developers to test their requests to REST APIs. The request takes the form:

```
https://jsonplaceholder.typicode.com/posts?userId=<userID>
```

The value of parameter userID has to be passed in the request message and the list of posts with their content is returned with the response message in JSON format.

The list of required userID is created in a Table Creator node. Then the REST request is built using a “String Manipulation” node as:

```
join("https://jsonplaceholder.typicode.com/posts?userId=", $userID$)
```

where $userID$ comes from the userID input column and contains the userID values.

The "GET Request" node is introduced to submit the GET request to the REST server. The "GET Request" node has 4 tabs in its configuration window:

- "Connection Settings" contains all REST service settings, including the URL source;
- "Authentication" sets the authentication credentials, if required;
- “Request Header” allows for customization of the request headers;
- “Response Header” allows to interpret customized response headers.

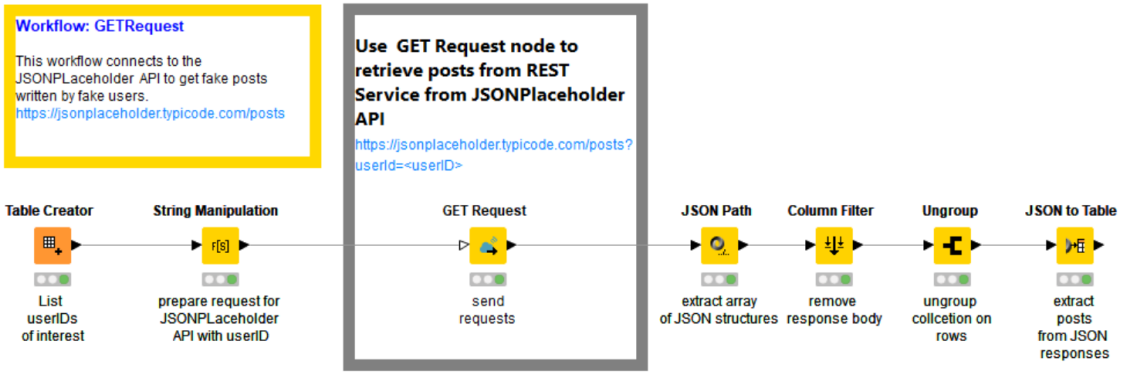
When executed, the “GET Request” node sends all requests in batches of N, as defined in the “Concurrency” setting, to the REST service specified in the URL parameter and presents the subsequent response at its output port. The size of N can be specified in the Connection Settings under Concurrency (Number of concurrent requests). The JSONPlaceholder API REST service does not require any authentication and therefore we selected option “None” in the “Authentication” tab in the configuration window of the GET Request node.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

The response is usually encapsulated in a XML or JSON structure, which needs to be interpreted to extract the value(s) we were looking for, in our case the body and title values. If the response is returned in XML format, you probably need to use the “XPath” node from the “XML” category in the “Node Repository” to retrieve the values you are interested in. If the response is returned in JSON format, you need to resource to one of the nodes in the “JSON” category, including “JSON Path”, “JSON To XML”, and especially “JSON to Table”. In our example, we used “JSON Path” and “JSON to Table” nodes. JSON Path is used to extract the array of posts from the response JSON structure; while the JSON To Table node does all the remaining parsing work for us, exposing at the output port all values contained in the input JSON structure.

### 3.6. Workflow “GETRequest”






This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

## GET Request: "Configuration Settings" Tab

GET request URL(s) that are sent to a REST service is (are) identified either through one single manually inserted fixed URL or through a list of dynamic URLs in an input data column.

Two options in the "**Connection Settings**" tab respectively enable these two modes: "URL" and "URL column".

"**Delay (ms)**" and "**Concurrency**" both deal with multiple requests. "Delay (ms)" specifies a delay between two consecutive requests, e.g. in order to avoid overloading the web service. "Concurrency" allows for N parallel GET requests to be sent, if the REST service allows it.

The flags in the **SSL** section push for a higher tolerance in security when checking the REST host SSL certificates. When enabled, even if some SSL certificates are not perfect the returned response is accepted.

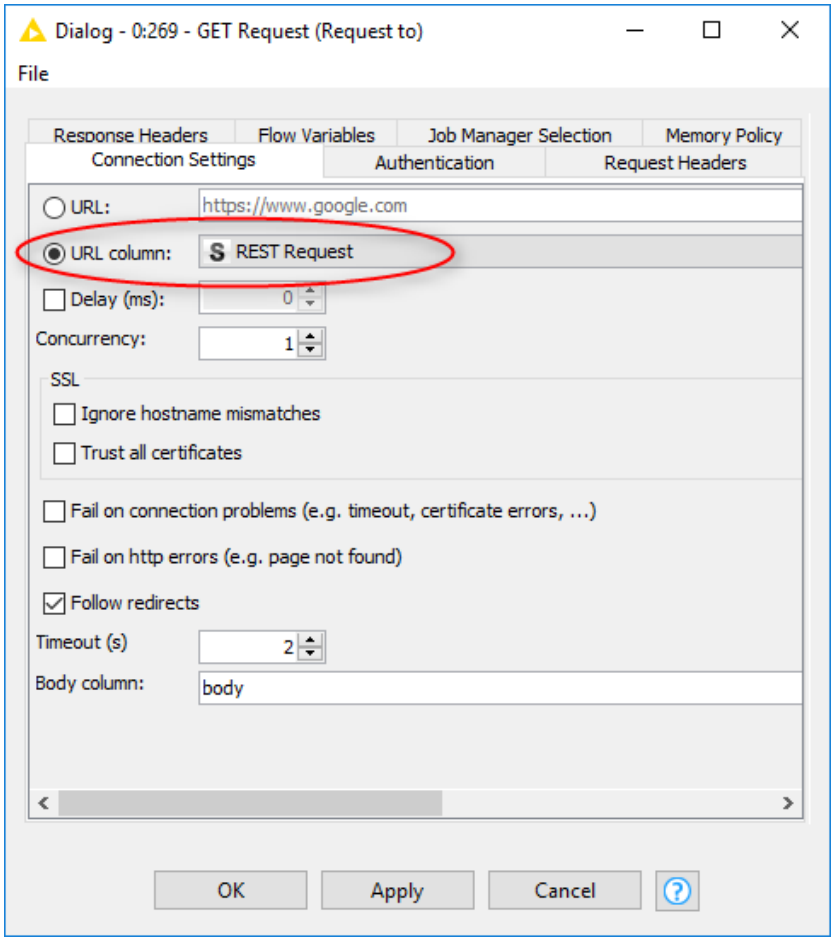
When something fails in the REST service, the output data table will contain the rows with the status of the requests. In some cases, we might desire the workflow to just stop and signal the REST error. In this case, the two options "**Fail on Connection problems**" and "**Fail on HTTP errors**" should be enabled.

The flag "**Follow Redirects**" forces the GET request to be redirected, if so specified in the REST service.

"**Timeout**" sets the number of seconds to wait before declaring a connection timed out.

"**Body column**" contains the name of the response column in the output data table.

3.7. "GET Request" node configuration: "Configuration Settings" tab

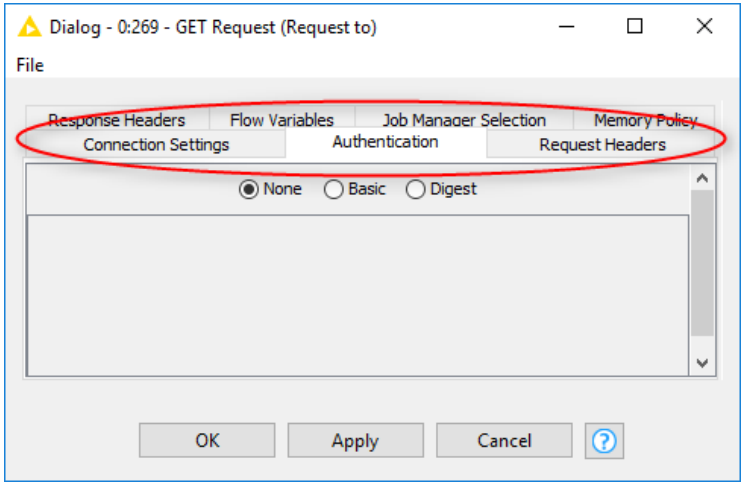





This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

## GET Request: the other Tabs

3.8. "GET Request" node configuration: the other tabs




The "**Authentication**" tab sets the authentication credentials, if required by the REST service.

The node supports a number of authentication methods, e.g. None, BASIC, and DIGEST. Username and password can be provided manually or via workflow credentials. As for the database nodes, workflow credentials are automatically encrypted, while manual typing of username and password requires the definition of a Master Key for the encryption process. The Master Key can be set in the "Preferences" page.

Every request being shipped off to a REST service may contain a header. By default in the "GET Request" node, requests are shipped with no header. However, custom request headers can be defined in the "**Request Headers**" tab. A request header consists of many parameters and every parameter consists of 3 fields: key, value, and kind. Three request headers have been pre-loaded as templates: none, generic REST headers, and web page related headers.

The response object that comes back can also contain headers, at least the status of the request and the content-type. Other headers can be imported if the flag named "Extract all Headers" at the very top of the "**Response Headers**" tab is enabled. If you prefer not to extract all headers from the response, but just some, you can set the key names one by one in the tab table. The value associated with the keys will be extracted from the response object and placed in a data column of the output table. The name of this data column is also set in the "Response Headers" tab.



This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

## JSON Path

The "JSON PAth" node parses a JSON structure via a custom query.

In order to do that, it needs the following settings in the configuration window:

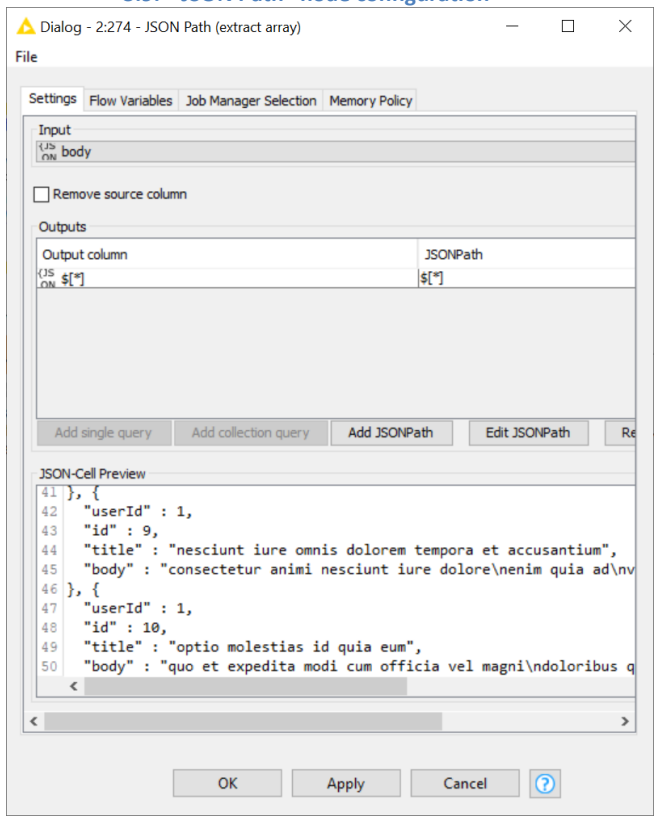
- The input data column containing the JSON structures to parse
- The editor for the custom query.
- A flag to optionally remove the original JSON column from the output table

In the lower part of the configuration window a preview frame is available.

It is possible to interactively click JSON items in the preview frame and automatically define the JSON custom parsing query.

The buttons above the preview frame allow to add/edit/remove the resulting custom JSON parsing query into the editor.

3.9. "JSON Path" node configuration






This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

## JSON to Table

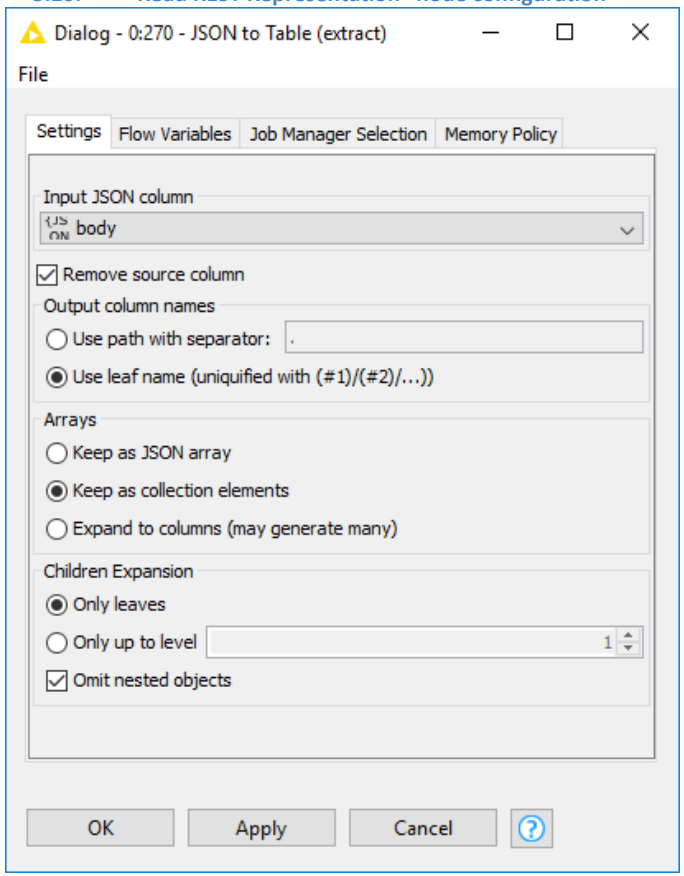
The "JSON to Table" node parses a JSON structure, extracts all values, and puts them in a KNIME data table at the output port.

In order to do that, it needs the following settings in the configuration window:

- The input data column containing the JSON structures to parse
- Customized or default names for the data columns resulting from the parsing operation.
- The structure of the resulting columns (it is good practice to keep the result as a collection column, since we do not know a priori how many columns will result from this blind operation).
- The expansion level of the JSON structure (again, it is good practice to expand as little as possible the original structure, since we do not know a priori how many levels it will contain).

The result at the output port is a KNIME data table, containing the values from the REST response, including the REST URL Request and the Request Status.

3.10. "Read REST Representation" node configuration



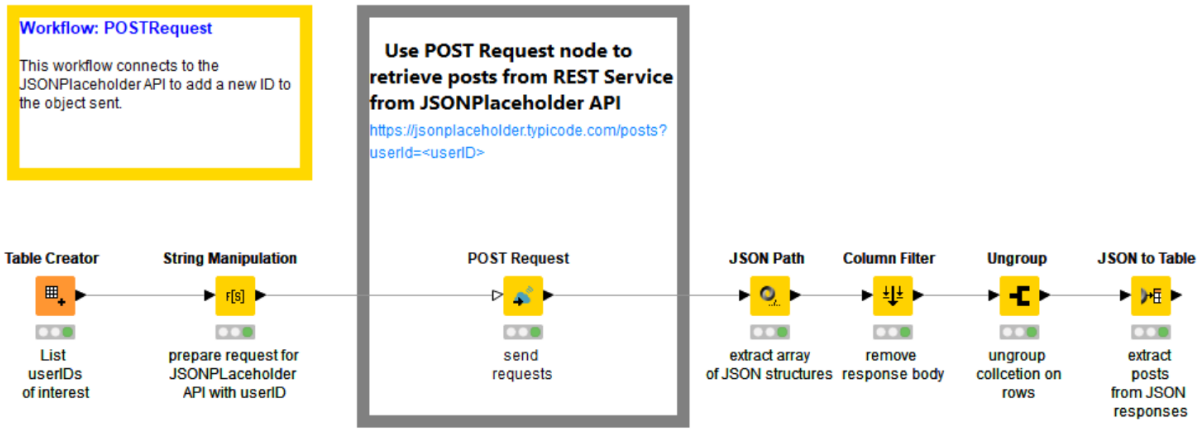

The same post request to the JSONPlaceholder API REST service could be sent via POST Request. "REST Web Services" category offers the "POST Request" node to send POST requests to a service.

As a practical example to illustrate the functionality of the "POST Request" node, we altered the previous workflow to send a POST request to add a new ID to the object in the request. The POST request is the same as the previous GET Request. The associated functionality is different. Thus, instead of a having a "GET Request" node, we introduced a "POST Request" node.



This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

### 3.11. Workflow “POSTRequest”




The “POST Request” node submits a POST request to the REST server. The "POST Request" node has 5 tabs in its configuration window:

- "Connection Settings" contains all REST service settings, including the URL source;
- "Authentication" sets the authentication credentials, if required;
- “Request Header” allows for customization of the request headers;
- “Request Body” to send the data to be transmitted through the POST request;
- “Response Header” allows to interpret customized response headers.

Similarly to the “GET Request” node, when executed, the “POST Request” node sends all requests in batches of N, as defined in the “Concurrency” setting, to the REST service specified in the URL parameter and presents the subsequent response at its output port. Depending on the response format, XML or JSON, you can use either an “XPath” node or a “JSON to Table” node. Here we used again a “JSON Path” and a “JSON to Table” node.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

## POST Request: "Configuration Settings" Tab

POST request URL(s) that are sent to a REST service is (are) identified either through one single manually inserted fixed URL or through a list of dynamic URLs in an input data column.

Two options in the "**Connection Settings**" tab respectively enable these two modes: "URL" and "URL column".

"**Delay (ms)**" and "**Concurrency**" both deal with multiple requests. "Delay (ms)" specifies a delay between two consecutive requests, e.g. in order to avoid overloading the web service. "Concurrency" allows for N parallel GET requests to be sent, if the REST service allows it.

The flags in the **SSL** section push for a higher tolerance in security when checking the REST host SSL certificates. When enabled, even if some SSL certificates are not perfect the returned response is accepted.

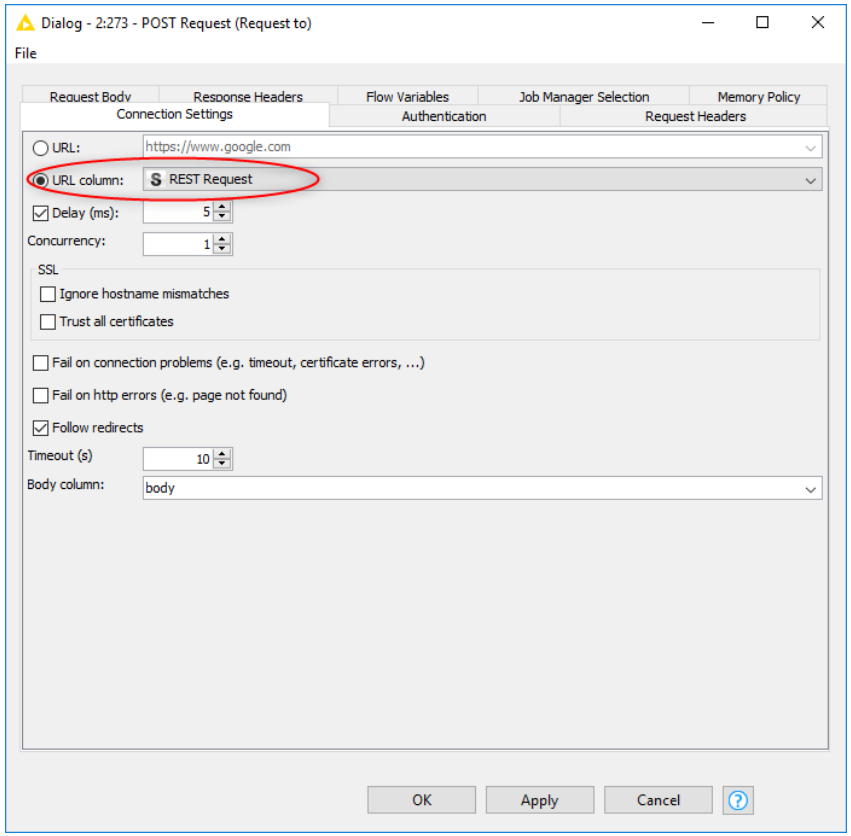
When something fails in the REST service, the output data table will contain the rows with the status of the requests. In some cases, we might desire the workflow to just stop and signal the REST error. In this case, the two options "**Fail on Connection problems**" and "**Fail on HTTP errors**" should be enabled.

The flag "**Follow Redirects**" forces the GET request to be redirected, if so specified in the REST service.

"**Timeout**" sets the number of seconds to wait before declaring a connection timed out.

"**Body column**" contains the name of the response column in the output data table.

3.12. "POST Request" node configuration: "Configuration Settings" tab





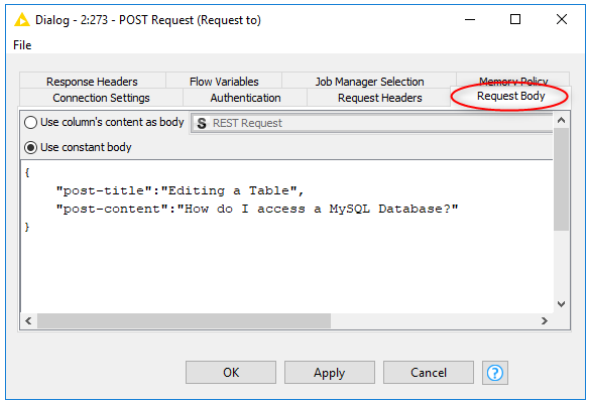

This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

## POST Request: "Request Body" Tab

POST request(s) sometimes need a body of data. The data for the request body is passed via the "Request Body" tab in the configuration window of the "POST Request" node.

The request body can either be a manually inserted fixed text or the content of a data cell from the input table. This is decided in the radio button options "Use column's content as body" or "Use constant body".

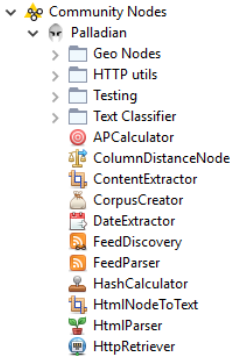
3.13. "POST Request" node configuration: "Request Body" tab




If you need to connect to a SOAP based web service via WSDL file, this is also possible from within a KNIME workflow. Here you would need to use the "Generic Web Service Client" node.

## 3.3. Web Crawling

3.14. Palladian category in Node Repository




Another way to retrieve text data from the web is to run a web crawler. No worries! You do not need to leave the KNIME Analytics Platform to run a web crawler.

Indeed, one of the KNIME Community Extensions, provided by Palladian, offers a large number of nodes for web search, web crawling, geo-location, RSS feed, and other web related functions. In this section we explore only two of these nodes: the "HttpRetriever" and the "HtmlParser" node. Those are the only two nodes we need to implement a web crawler. They are located in *"Palladian"/"HTTP utils"* sub-category in the Node Repository.

As an example of a workflow performing a web crawling, we have created a new workflow named "WebCrawling" in "Chapter3" folder. This workflow is supposed to take a URL String as input, download the content of the URL page, parse it and transform it into an XML data cell, and finally return a Document type cell containing, author, title, and page body.



This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

As to install any KNIME Extension Package (in this case the “Palladian for KNIME” extension), go to the Top Menu:

- Select “File”
- Select “Install KNIME Extensions…”
- From the “Available Software” window, in the top box containing “type filter text”, type “web”
    - Inside “KNIME & Extensions” select the package called “KNIME REST Client Extension”
    - Click “Next”
    - Follow the installation instructions

After the package has been successfully installed, you should find a category named “Palladian” in the “Community Nodes” category in the “Node Repository” panel.

We set the URL to explore in a “Table Creator” node as https://www.knime.com/knime-for-data-scientists - the summary of the KNIME Analytics Platform tool.

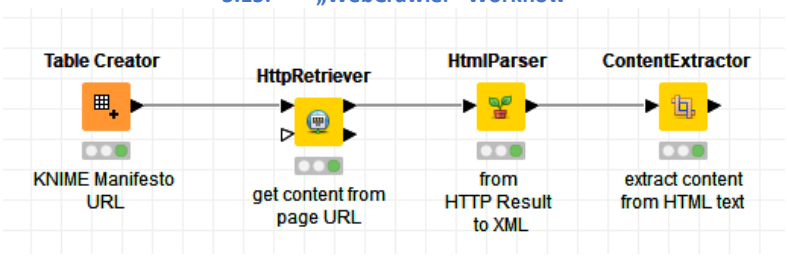
The “HttpRetriever” node then takes the URL as input and downloads all web content available at the selected URL. The “HttpRetriever” node has two output ports. The first output is an HTTP-type column, named “Result” containing the download status code and the downloaded content. The second output port contains the cookies set during the node execution, if any.

The “HtmlParser” node imports and parses the HTTP-type object generated by an “HttpRetriever” node. At its output port the content of the HTTP column is presented in the form of an XML object. After executing an “HttpRetriever” node and an “HtmlParser” node, the content of the web page has been downloaded and imported into KNIME as an XML cell.

Now, we need to create a Document type column including: author, title, and full content from the original web page. This is done by the “ContentExtractor” node using the “Readability” extraction algorithm.

Et voilà! The web crawling operation was implemented using only four nodes.

3.15. „WebCrawler“ Workflow






This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

## HttpRetriever

This node allows to download content from a web page via a number of different HTTP methods: GET, POST, HEAD, PUT, and DELETE. The node then imports the web content and the cookies as binary data and allows to specify arbitrary HTTP headers.

The configuration window is organized on three tabs.

**Tab "Options"** requires:

- The data column with the URLs of the pages to connect to
- Each URL can require a different download method. If this is the case, you can set the value of an input column as download method for each URL. If no such column is provided, all contents are downloaded via GET
- The HTTP entity column, and optionally type, can be specified, if an HTTP entity is required for the download to start
- An emergency stop criterion on the download file size
- Whether to append an output column with the download status

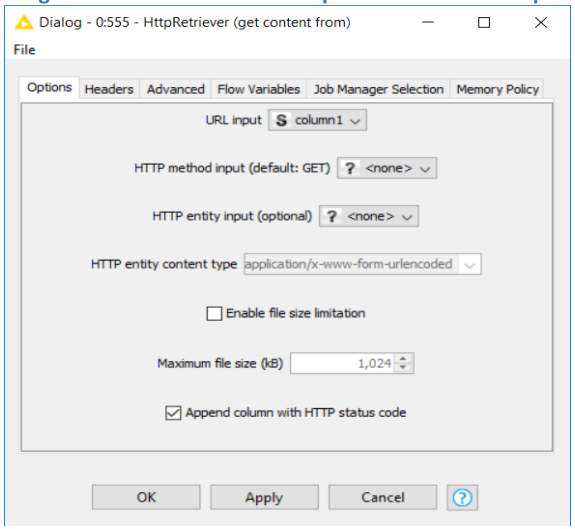
**Tab "Headers"**

If HTTP headers are required for the transfer, the column(s) containing HTTP Headers are set in this tab.

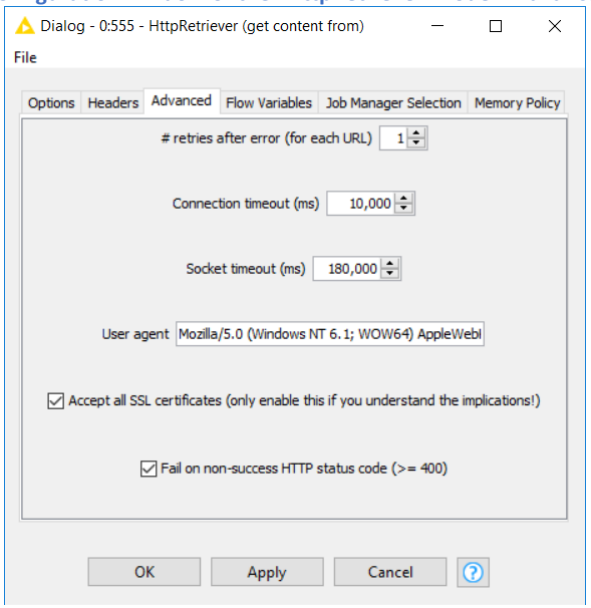
**Tab "Advanced"** requires:

- The settings for stopping the execution in case of errors or connection problems (# retries and timeouts)
- The HTTP user agent for the data transfer
- A tolerance flag about accepting SSL certificates (this is of course a sensitive security option)
- Force an error on the node execution, if the HTTP status code for the data transfer returns some kind of failure

3.16. Configuration window of the "HttpRetriever" node: "Options" tab




3.17. Configuration window of the "HttpRetriever" node: "Advanced" tab






This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

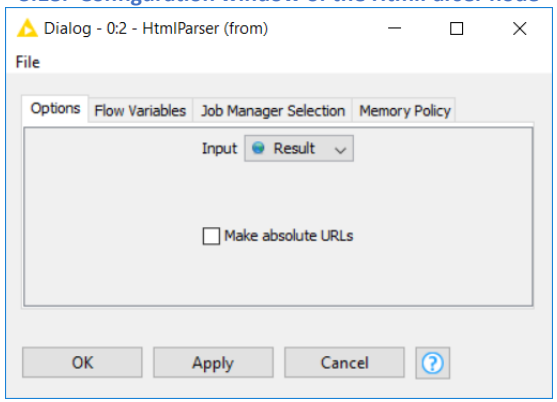
## HtmlParser

The HtmlParser node transforms an HTTP object into an XML object. To perform this task, it relies on the HTML Parser available in the Validator.nu Java library for HTML5.

The node does not need much in terms of configuration settings, just the name of the HTTP column at its input port to be transformed into an XML type column.

The option “Make absolute URLs”, when enabled, converts all relative URLs in the document to absolute URLs.

3.18. Configuration window of the HtmlParser node




## Content Extractor

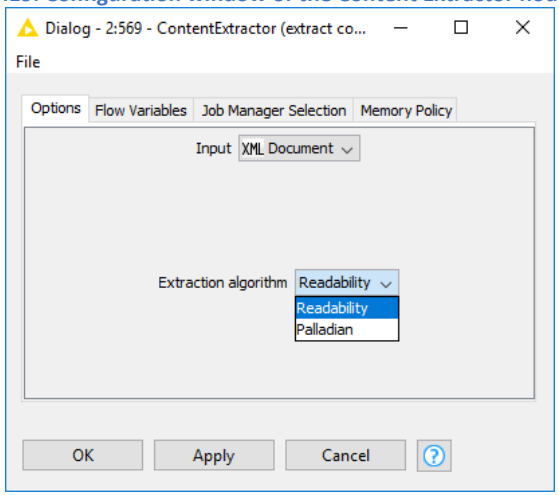
Also this node belongs to the Palladian extension. This node helps with the extraction of text from HTML and, more generally, XML documents, by removing all HTML tags, footers and headers, ads, navigation etc.

There two possible extraction algorithm: “Readability” imported from Arc90 and the native “Palladian”. Readability performs best for general content extraction, while Palladian performs best when parsing comment sections.

The other configuration setting consists of the XML-type input data column.

This node produces a Document-type cell at the output port with the content of the downloaded web page.

3.19. Configuration window of the Content Extractor node






This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

## 3.4. Exercises

### Exercise 1

Retrieve information from a chemical database, specifically ChemSpider, from the Royal Society of Chemistry, using a GET Request. This resource is a freely available dictionary of small chemical compounds. ChemSpider provides access to its database via REST services.

Before using the service, obtain a list InChI Keys from Google Sheets as described in section 3.1.

Using this list, then use the "InChIKeyToMol" operation to convert InChIKey to MOL by searching InChIKeys against the ChemSpider database, as:

```
https://www.chemspider.com/InChI.asmx/InChIKeyToMol?inchi_key=<key>
```

**Solution to Exercise 1**

We first connect to Google Sheets using the "Google Sheets Interactive Service Provider" node and retrieve the data from the Spreadsheet "InChI" and the Sheet "InChIKeys" using the "Google Sheets Reader" node.

We create the GET Request in the "String Manipulation" node as:

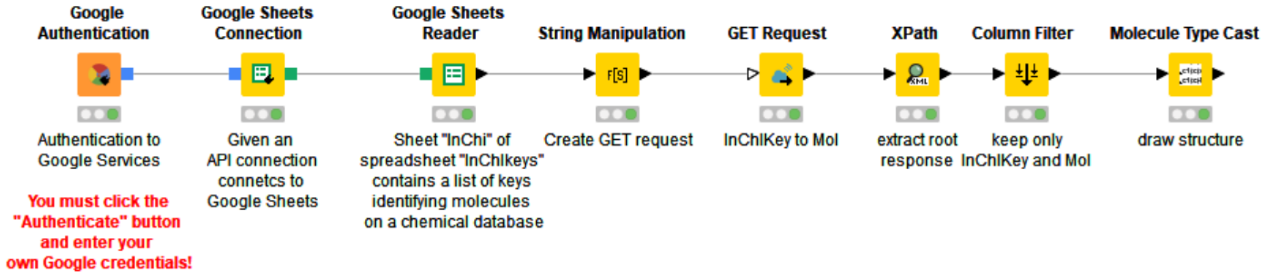
```
join("https://www.chemspider.com/InChI.asmx/InChIKeyToMol?inchi_key=",$InChIKey$)
```

and execute it using the "GET Request" node.

The "XPath" node enables us to extract the result.

The final result should be a table containing chemical structures written in Mol format of type String. These can be viewed using the "Molecule Type Cast", which convert the field type String to Mol.

3.20. Solution workflow for Exercise 1



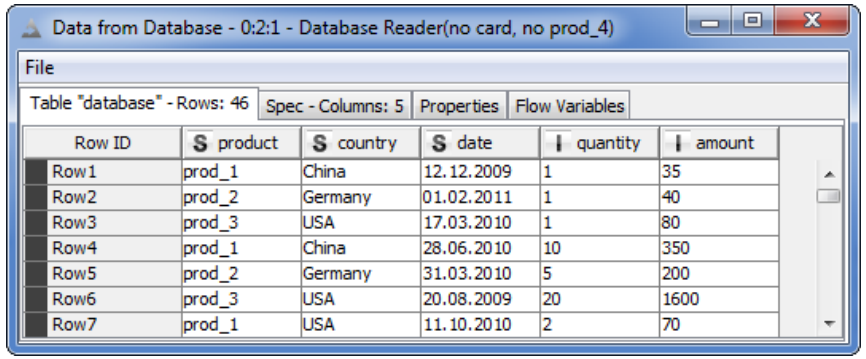



This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

# Chapter 4. Date&Time Manipulation

## 4.1. The Date&Time Type

4.1. The data table produced by the "Database_Operations" workflow implemented in chapter 2




| Row ID | S product | S country | S date | I quantity | I amount |
|---|---|---|---|---|---|
| Row1 | prod_1 | China | 12.12.2009 | 1 | 35 |
| Row2 | prod_2 | Germany | 01.02.2011 | 1 | 40 |
| Row3 | prod_3 | USA | 17.03.2010 | 1 | 80 |
| Row4 | prod_1 | China | 28.06.2010 | 10 | 350 |
| Row5 | prod_2 | Germany | 31.03.2010 | 5 | 200 |
| Row6 | prod_3 | USA | 20.08.2009 | 20 | 1600 |
| Row7 | prod_1 | USA | 11.10.2010 | 2 | 70 |

Let's now have a brief look at the data table resulting from any of the branches of the "Database_Operations" workflow built in chapter 2. We have three columns of String type ("product", "date", and "country") and two columns of Integer type ("quantity" and "amount").

However, the data column "date" contains the contract date for each record and should be treated as a date type variable. KNIME Analytics Platform has indeed the following dedicated data types for date and time data:

- Date
- Time
- Date & Time
- Date & Time with zone

For the KNIME Analytics Platform 3.4 release a full rewrite of the Date and Time support was performed. In earlier versions, a single date/time type was available, which could be used for date, time, as well as date and time objects. This date/time type is still available as a legacy type ("Legacy Date&Time") in legacy nodes. In the new releases, however, we have one dedicated type for date, one for time, one for date & time, and one for date and time with a time zone. Using a dedicated type for each object makes the code less error prone.

**Note.** New Date&Time support is available since the release of KNIME Analytics Platform 3.4! Now, four dedicated types are available for date and/or time, with or without an attached time zone.

If you import an old workflow into a recent KNIME Analytics Platform release, your date/time nodes will be converted in to a legacy node available in "Other Data Types"/"Time Series"/"Time Series (legacy)" category in the Node Repository. Such nodes rely on the "Legacy Date&Time" type.

In order to convert from a "Legacy Date&Time" type to the new Date&Time type, you will need to use the "Legacy Date&Time"to Date&Time" node. The "Date&Time to legacy Date&Time" node moves the data cell into the opposite format direction.



This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

Date&Time formats in KNIME are expressed by means of:

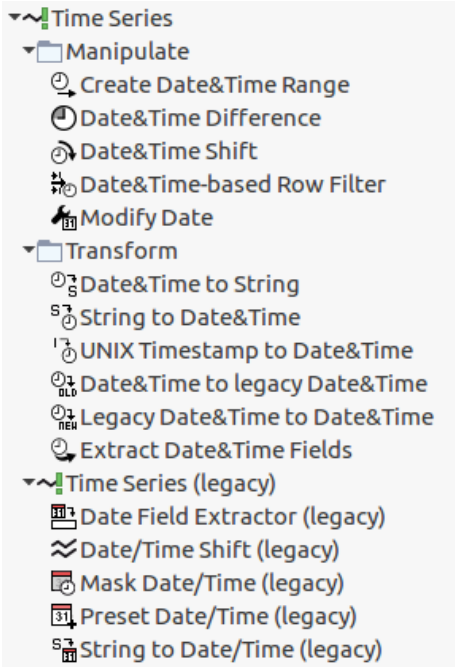
- A number of “d” digits for the day
- A number of “M” digits/characters for the month
- A number of “y” digits for the year
- A number of “h” digits for the hour
- A number of “m” digits for the minutes
- A number of “s” digits for the seconds
- A number of “S” digits for the milliseconds

These dedicated digits combined together produce a string representation of the date and/or time (with or without time zone). The table below shows a few examples for 3:34pm on the 21st of March 2011 in Berlin.

| Type: format | String representation |
|---|---|
| Date: dd-MM-yyyy | 21-03-2011 |
| Date: MMM/dd/yyyy | Mar/21/2011 |
| Time: hh:mm | 15:34 |
| Time: ss.SSS | 00.000 |
| Date&Time:dd.MM.yyyy hh.mm.ss.SSS | 21.03.2011 15:34:00.000 |
| Date&Time: dd.MMM.yy:hh.mm | 21.Mar.11:15.34 |
| Date & Time with zone: ss.SSSyyyy-MM-dd'T'HH:mmSVV'['zzzz'] | 00.0002011-03-21T03:34+02:00[Europe/Berlin] |

The new Date&Time type carries a number of dedicated nodes implementing a large variety of operations. A full category “Other Data Types”/“Time Series” contains a large number of nodes implementing Date&Time manipulation functionalities.

4.2. Manipulation functionalities for Date&Time objects




## 4.2. How to produce a Date&Time Column

How can we generate a Date (or Time, Date & Time, Date & Time with zone) type data column? There are many possibilities to do that. A “Database Reader” node, for example, automatically reads a SQL timestamp field into a Date&Time column. Another possibility is to read in a String data column, for example with a “File Reader” node, and then convert it into a Date&Time type. This section explores the “String to Date&Time” and “Date&Time to String” conversion nodes.



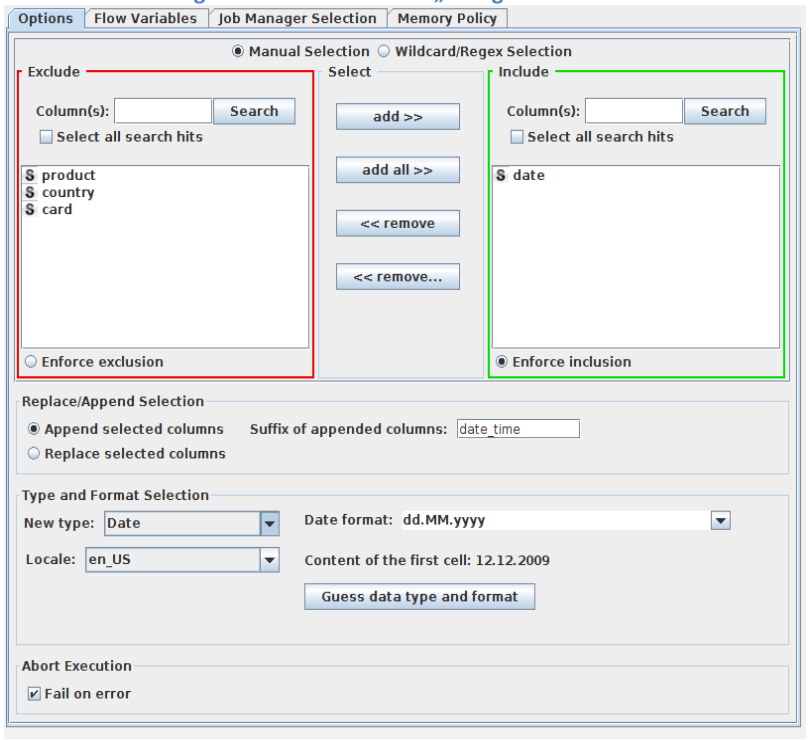
This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

## String to Date&Time

The "String to Date&Time" node converts a String cell into a Date&Time cell according to a given format. The configuration window requires :

- The String column(s) containing the date/time objects to be converted. The column selection is performed through an include/exclude column selection framework
- The option for the resulting column to replace the original String column ( "Replace selected column" option) or be appended to the input data table ("Replace selected column" option and suffix for appended new column)
- The new column type (Date, Time, Date&Time, or Date&Time with zone) and locale to express it
- The Date format to be used. Here you can choose from a number of pre-defined Date&Time formats available in the "Date format" menu; or you can edit the Date&Time format manually; or you can let the node guess the Date&Time format through the button "Guess data type and format")
- A check box that indicates whether reading errors are tolerated or not. If checked, the node will fail on errors.

4.3. Configuration window of the „String to Date&Time" node




**Note**: All Date&Time nodes support multiple columns. Columns can be easily included or excluded in the options using manual selection or wildcard/regex selection.

In order to show the tasks implemented by these and more nodes in the "Time Series" sub-category, let's create a new workflow, to be named "DateTime_Manipulation". In the newly created workflow, we use a "File Reader" node to read the "sales.csv" file from the KCBData folder. The "File Reader" node does not automatically assign the Date&Time type to date and time values. It just reads them in as String. Before proceeding with more complex manipulations, let's convert the "date" column from the default String to the Date&Time type by using a "String to Date&Time" node. For that, after the "File Reader" node, a "String to Date&Time" node has been introduced to convert the "date" column from the String type to the Date type according to "dd.MM.yyyy" format. In fact, the dates contained in the data column "date" are formatted as "dd.MM.yyyy" and should then be read with that format.



This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

In the configuration window of the “String to Date&Time” node, we also opted to replace the original String column with the new Date&Time column.

After executing the node, the resulting data table should be as in figure 4.4, where the data column “date” is now of Date type. The little icon showing a calendar and a clock indicates a Date&Time type.

4.4. Column “date” after the conversion to DateTime with the „String to Date/Time“ node

Output table - 0:28 - String to Date&Time (convert String)

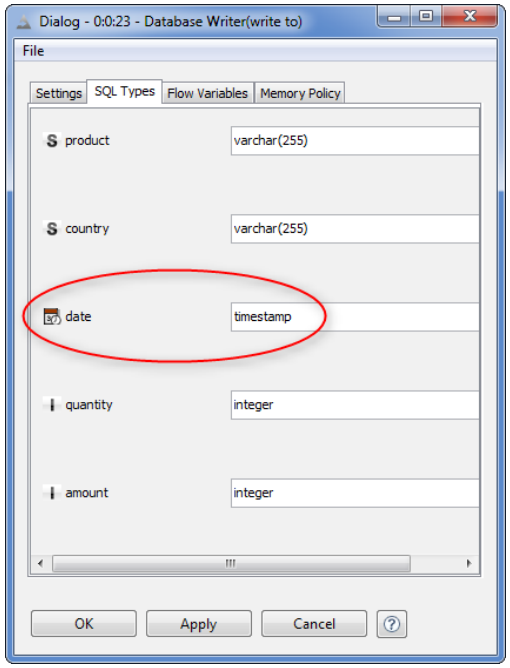
File Hilite Navigation View

Table "default" - Rows: 47 | Spec - Columns: 6 | Properties | Flow Variables

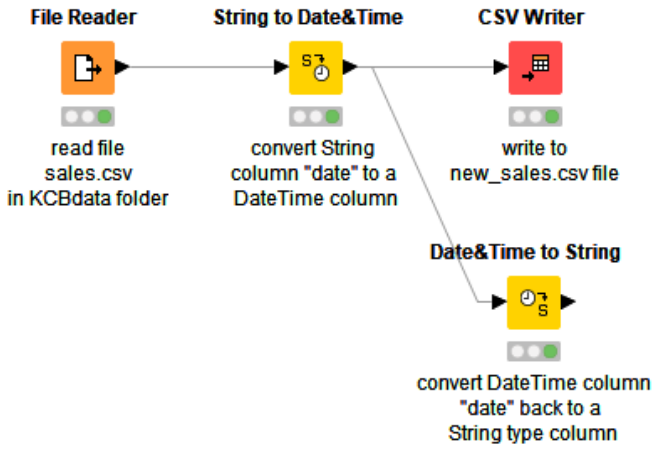
| Row ID | product | country | date | quantity | amount | card |
|---|---|---|---|---|---|---|
| Row0 | prod_1 | China | 2009-12-12 | 1 | 35 | Y |
| Row1 | prod_2 | Germany | 2011-02-01 | 1 | 40 | Y |
| Row2 | prod_3 | USA | 2010-03-17 | 1 | 80 | Y |
| Row3 | prod_1 | China | 2010-06-28 | 10 | 350 | Y |
| Row4 | prod_2 | Germany | 2010-03-31 | 5 | 200 | Y |

Finally, we wrote the new data table with the Date Type column into a CSV file, with a “CSV Writer” node. If instead of writing to a CSV file we want to save the new data table into a database with a “Database Writer” node, we might need to manually set the SQL Type of the final field to “timestamp” in the “SQL Types” tab of the node configuration window (depending on the database).

4.5. "SQL Types" tab in the configuration window of a "Database Writer" node to explicitly export the Date&Time type into the database table




4.6. The upper part of the "DateTime_Manipulation" workflow






This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

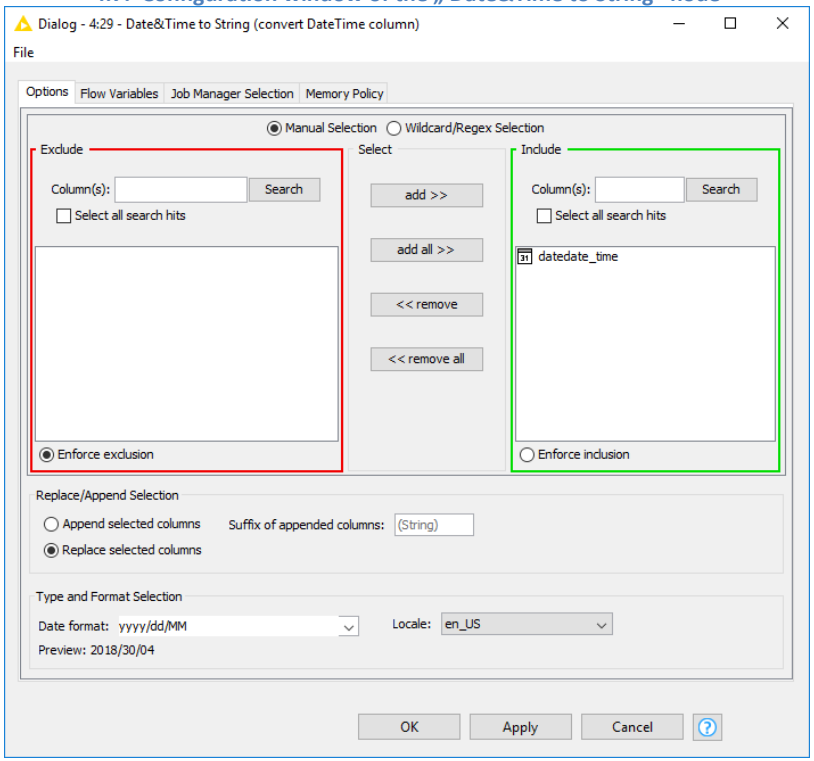
Sometimes, in contrast, it might be necessary to convert a Date&Time column into a String column. The “Date&Time to String” node walks the data in the opposite direction to the “String to Date&Time” node. In the “DateTime_Manipulation” workflow, we introduced a “Date&Time to String” node to convert the Date&Time column “date” back to a String type with format “yyyy/MM/dd”.

## Date&Time to String

The “Date&Time to String” node converts a Date&Time cell into a String object according to a given Date&Time format. The configuration window requires :

- The Date&Time type column(s) to be converted into String type
- The option for the resulting column to replace the original String column ( “Replace selected column” option) or be appended to the input data table (“Replace selected column” option and suffix for appended new column)
- The format to build the String pattern
    - A number of pre-defined Date&Time formats are available in the “Date format” menu
    - The “Date format” box can be edited manually in order to create the required Date&Time format, if the one you want is not available in the menu
    - The locale to express the Date&Time object

4.7. Configuration window of the „ Date&Time to String“ node




Another way to create a Date&Time type column is to use the “Create Date&Time Range” node. This node generates a number of equally spaced date, time, or date and time values. You can:

- set the number of values, the starting point in time, and the ending point in time, and calculate the corresponding interval between values
- set the number of values, the starting point, and the interval size, and calculate the corresponding end point in time
- set the interval, the starting point, and the end point and calculate how many values to generate.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

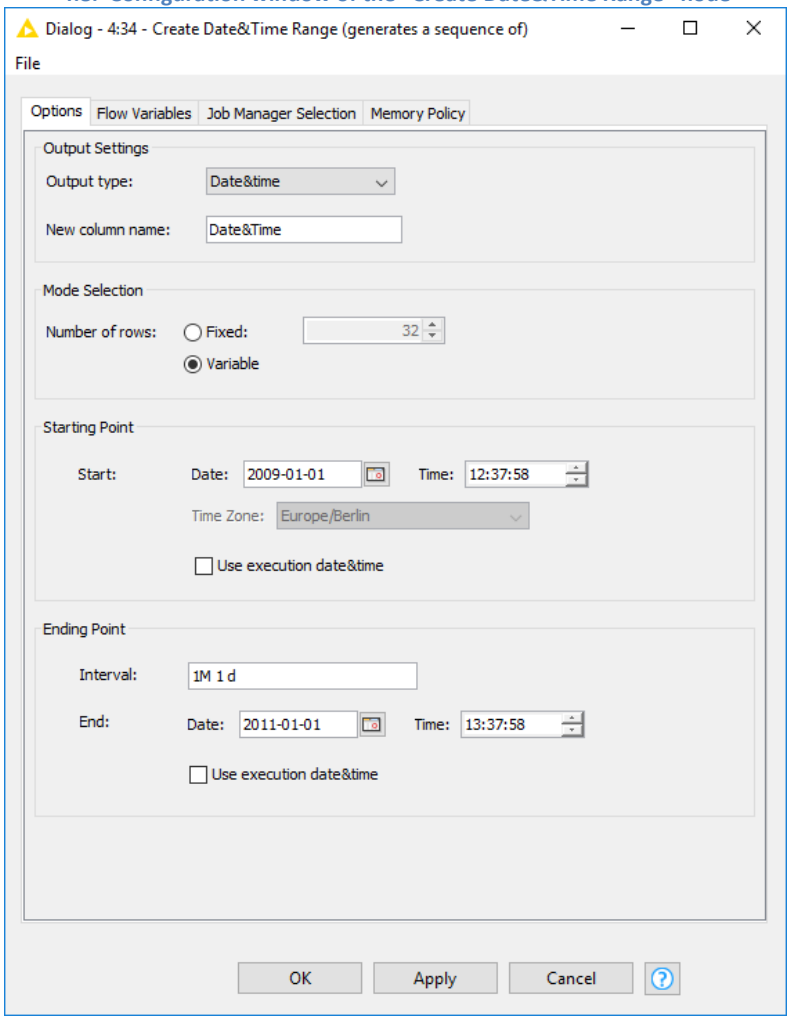
## Create Date&Time Range

The “Create Date&Time Range” node creates a number of Date&Time values from a start point in time to an end point in time. Values are equally spaced. This means that you set the number of values, the start, and the end and determine the interval; or set the number of values, the interval, and the start, and determine the end; or set the interval, the start, and the end, and determine the number of points.

The configuration window requires :

- The output column containing the new values: name and type
- The starting and end points in time. These two points will have the same format as defined for the new output column: just Date, just Time, Date&Time, or Date&Time with zone
- One of the three strategies to generate the values: a fixed number of rows or a variable number of rows.
  - If you have selected a fixed number of rows, also select the starting point and then the end point or the interval
  - If you have selected a variable number of rows, then select the starting point, the end point, and the interval size. Interval size can be either:
    - time or date ISO-8601 based representation
    - short letter representation (e.g. '2y 3M 1d')
    - long word representation (e.g. '2 years 3 months 1 day')
  - As a special for an end date, there is the current execution time. This can turn out to be useful to generate timestamps.

4.8. Configuration window of the “Create Date&Time Range” node




ISO-8601 uses H for hours, m for minutes, s for seconds, d for days, M for months, y for years, e.g. '2y 3M 1d' means an interval of 2 years 3 months and 1 day.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

In the same workflow named “DateTime_Manipulation”, we introduced a “Create Date&Time Range” node to generate n Date&Time type values between Jan 1 2009 12:37 and Jan 1 2011 13:37 spaced equally spaced as 1 month and 1 day. The total number of rows generated by this node was then 24.

## 4.3. Refine Date&Time Values

To each one of these Date&Time values we decided to change time to 15:00. The node that generally changes time is the “Modify Time” node. This node works only on time. It can append a preset time value if none is present, change the current time value to a preset time value, or remove the current time value in the data cell.

### Modify Time

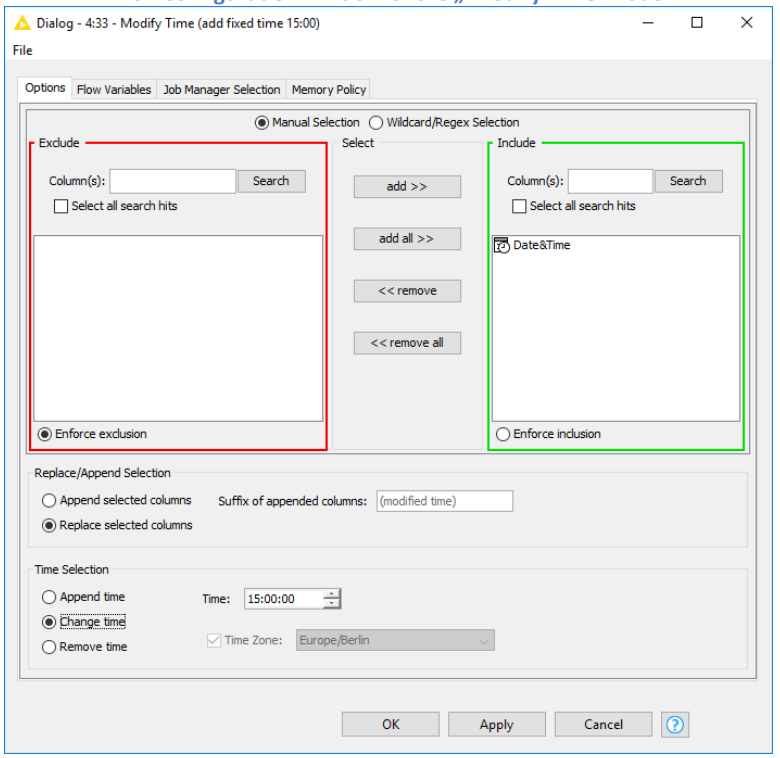
The “Modify Time” node modifies the time value in a data cell by:

- Appending a preset time value
- Changing current time value with a preset time value
- Removing current time value.

The configuration window requires :

- The column(s) on which to modify the time. Columns are selected via an Include/Exclude column selection framework.
- Whether to replace the selected input column or to create a new one to append to the input data table
- The modification strategy: “Append”, “Change”, “Remove”
- In case of “Append” or “Change”, the preset time value
- In case of “Append” also the time zone to associate to the new time value.

4.9. Configuration window of the „ Modify Time” node






This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

Now we could move all Date&Time values one day ahead. That is we would like to add +1 day to all Date&Time values we have created. The node that adds and subtracts a duration from a Date&Time value is the “Date&Time Shift” node. The duration can be expressed as a number for a given granularity, like n days or k months, or as a duration value according to the ISO-8601 date and time duration standards.

## Date&Time Shift

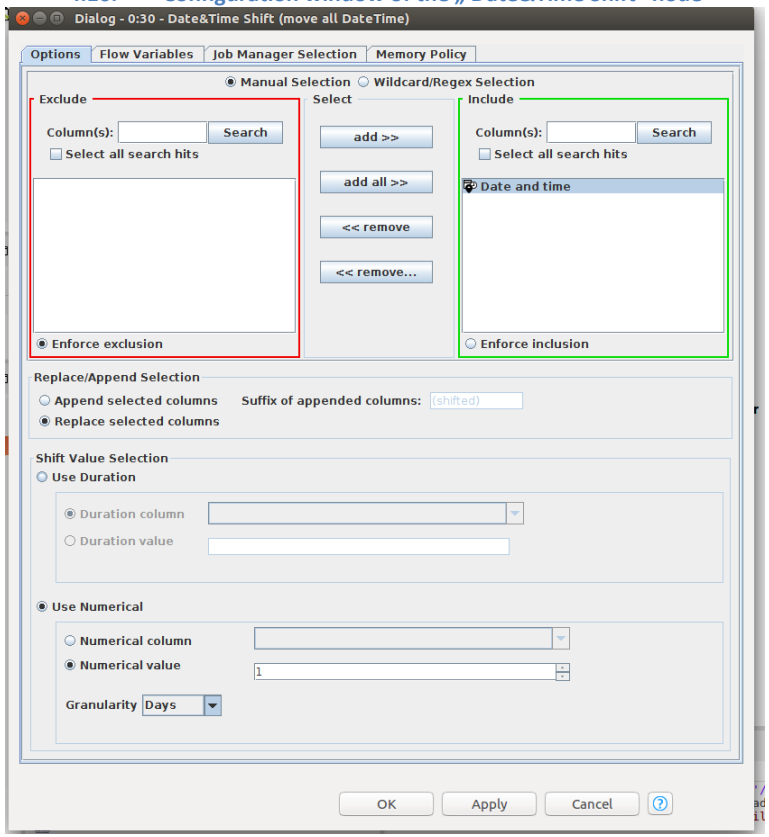
The “Date&Time Shift” node shifts a Date&Time value of a defined amount. The configuration window requires :

- The input Date&Time column(s) to shift. Those are selected via an Include/Exclude framework.
- Whether to create a new Date&Time column and append it to the input data table or to replace the selected input column. The suffix for the appended columns can be provided in the text field to the right.
- The shift value expressed as duration or as number:
  - **Use Duration**
    *Duration column*. Takes the shift value from a String input columns.
    *Duration value*. Adds/substracts this constant shift value.
    The String duration value can be either:
    - An ISO-8601 representation of date and time
    - A short letter representation (e.g. '2y 3M 1d')
    - A long word representation (e.g. '2 years 3 months 1 day')
  - **Use Duration**

    *Numerical column*. Takes the shift value from a numerical input column. A positive value will be added to the reference date or time, a negative one subtracted from it.

    *Numerical value*. Adds/subtracts this constant shift value. In case “Numerical Value” option is selected, *Granularity* defines the shift value granularity (day, hour, month, week, etc…).

4.10. Configuration window of the „ Date&Time Shift” node




Positive and negative shift values are possible to move forward and backward in time.

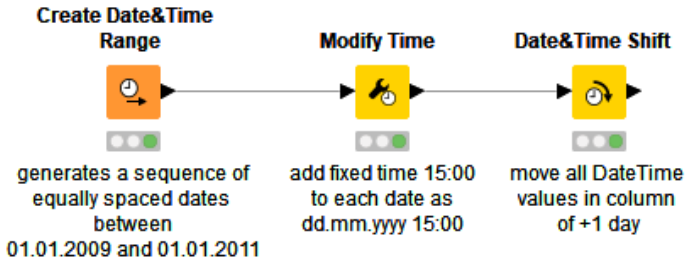


This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

In this section we have shown only two Date&Time nodes to change Date&Time values. There are many more. For example, similarly to the “Modify Time” node, a “Modify Date” and Modify Time Zone” node are available.

Figure 4.11 shows the lower part of the “DateTime_Manipulation” workflow as developed in this section.

4.11. The lower part of the "DateTime_Manipulation" workflow




## 4.4. Row Filtering based on Date&Time Criteria

Let’s dive deeper now into Date&Time manipulation. Very often a data analysis algorithm needs to work on the most recent data or on data inside a specified time window. For example, balance sheets usually cover only one year at a time; the results of an experiment can be observed inside a limited time window; fraud analysis runs on a daily basis; and so on. This section shows a number of row filtering operations based on date/time criteria.

The most common Date&Time based data selection is the one extracting a time window. This kind of data row filtering requires setting explicit initial and final date/time objects. Only the data rows falling inside this time window are kept, while the remaining data rows are filtered out. The “Date&Time-based Row Filter” node performs exactly this kind of row filtering based on the explicit definition of a time window.

In order to show practically how such time based filtering criteria can be implemented, we used the “File Reader” node to read the “sales.csv” file from the KCBData fdolder in a new workflow, named “DateTime_Manipulation_2”. After the “File Reader” node, a “String to Date&Time” node converted all String type dates to Date&Time objects. Then a “Date&Time-based Row Filter” node was introduced to select all sales that happened in a pre-defined time range. For the time range we used “2009-01-01” and “2011-01-01”. We set these dates as starting and end points, respectively, in the configuration window of the “Date&Time-based Row Filter” node and we obtained a data set with 37 sales data rows at the output port, covering the selected 2-year time span.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

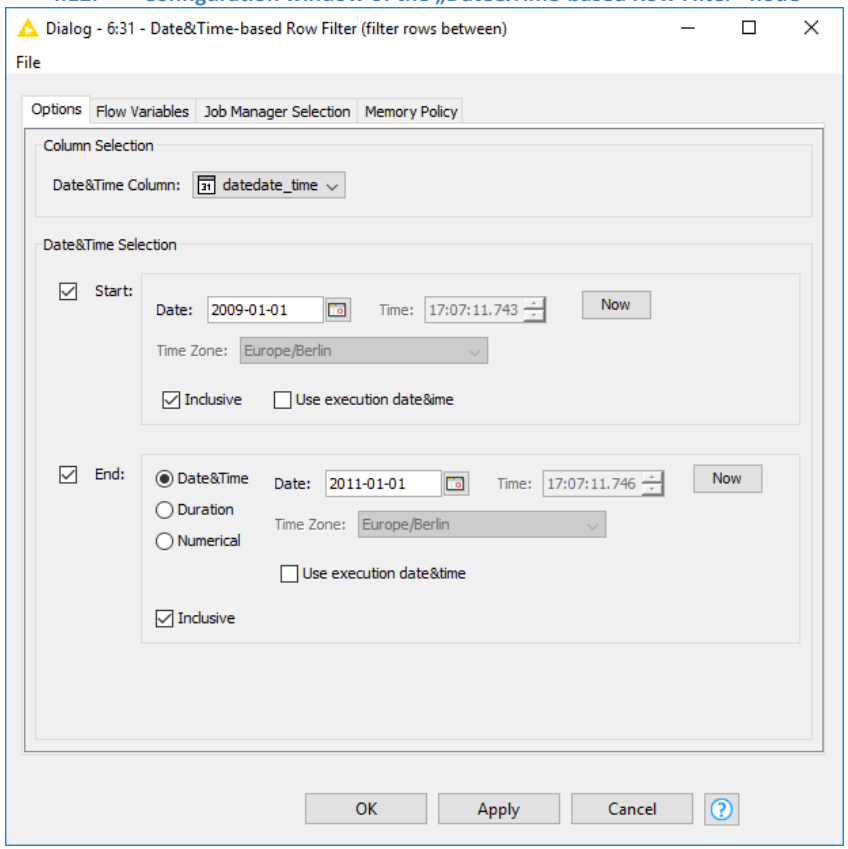
## Date&Time-based Row Filter

The "Date&Time based Row Filter" node implements row filtering based on a time window criteria. In fact, it keeps all data rows from the input data table inside a pre-defined time window.

The configuration window requires :

- The input Date&Time column to which the filtering criterion should apply
- The time window, i.e.:
    - The time window starting point, in terms of date and/or time
    - The time window end point, in terms of date and/or time. It is possible to assign the end of the time window using a duration formatted String (according to ISO-8601 date and time duration standards) or a numerical value with its granularity (days, months, …)
    - Execution time can be used as a starting point and as an end point for the time window
    - The "Inclusive" flag includes the extremes or the time window in the filter criterion.

4.12. Configuration window of the „Date&Time-based Row Filter" node




Let's suppose now that the year 2010 was a troubled year and that we want to analyze the data of this year in more detail. How can we isolate the data rows with sales during 2010? We could convert the "date" column from the Date&Time type to the String type and work on it with the string manipulation nodes offered in the "Data Manipulation"-> "Column" category. There is, of course, a much faster way with the "Extract Date&Time Fields" node.

The "Extract Date&Time Fields" node decomposes a Date&Time object into its components. A date can be decomposed into year, month, day number in month, day of the week, quarter to which the date belongs, and day number in year. A time can be decomposed in hours, minutes, seconds, and milliseconds. Here are some examples:



This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

| date | year | Month (no) | Month (text) | Day | Week day (no) | Week day (text) | quarter | No of day in year |
|---|---|---|---|---|---|---|---|---|
| 26.Jun.2009 | 2009 | 6 | June | 26 | 6 | Friday | 2 | 177 |
| 22.Sep.2010 | 2010 | 9 | September | 22 | 4 | Wednesday | 3 | 265 |

| time | hours | minutes | seconds | milliseconds |
|---|---|---|---|---|
| 15:23:10.123 | 15 | 23 | 10 | 123 |
| 04:02:56.987 | 4 | 2 | 56 | 987 |

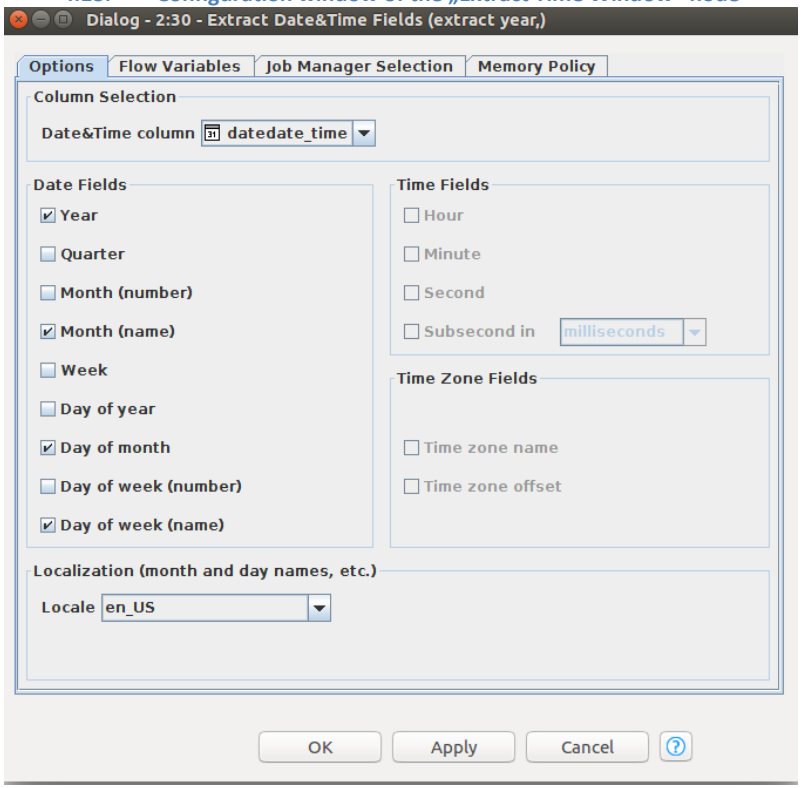
## Extract Date&Time Fields

The "Extract Date&Time Fields" node extracts the components (fields) of a Date&Time object.

The configuration window requires :

- Column Selection. A Date, a Time, a Date Time, or a Zoned Date Time column whose fields to extract.
- Date Fields. Pick and choose which fields to extract (year, week, hour, time zone name ...)
- The locale. This sets the locale to express the output Strings

4.13. Configuration window of the „Extract Time Window" node






This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

**Note.** The “Extract Date&Time Fields” node is particularly useful when combined with data aggregation nodes (“GroupBy”, “Pivot”, etc.)

In the “DateTime_Manipulation_2” workflow, we introduced an “Extract Date&Time Fields” node after the “Date&Time-based Row Filter” node. The configuration settings were set to extract year, month (name), day of week (name), and day number in the month from each Date&Time cell in the column named “datedate_time”.

The month and the day of week can be represented numerically, from 1 for January to 12 for December and from 1 for Sunday to 7 for Saturday, or as text strings with the full month and weekday name. We selected a text result for both the month and the day of week component. The data table at the output port, thus, had 11 columns, 4 more than the original 7 columns. The 4 additional columns are: “Year”, “Month”, Day of month”, and “Day of week (name)”.

In order to isolate the 2010 sales, we added a “Row Filter” node after the “Extract Date&Time Fields” node in the “DateTime_Manipulation_2” workflow to keep all rows with “Year” = 2010. The resulting data table had all rows referring to sales performed in 2010. Similarly, we could have summed up the sale amounts by month with a “GroupBy” node to investigate which months were more profitable and which months showed a sale reduction across the 2 years, 2010 and 2011. Or we could have counted all sales by weekday to see if more sales were made on Fridays compared to Mondays. The decomposition of date and time opens the door to a number of pattern analysis and exploratory investigations over time.

Let’s suppose now that we only want to work on the most recent data, for example on all sales that are not older than one year. We need to calculate the time difference between today and the sale date to exclude all rows with a sale date older than one year. The “Date&Time Difference” node calculates the time difference between two Date&Time values. The two Date&Time values can be:

- two Date&Time cells on the same row,
- one Date&Time cell and the current execution date/time,
- one Date&Time cell and a fixed Date&Time value,
- or finally one Date&Time cell and the corresponding Date&Time cell in the previous row.

The time difference can be calculated in terms of days, months, years, etc…, even milliseconds.



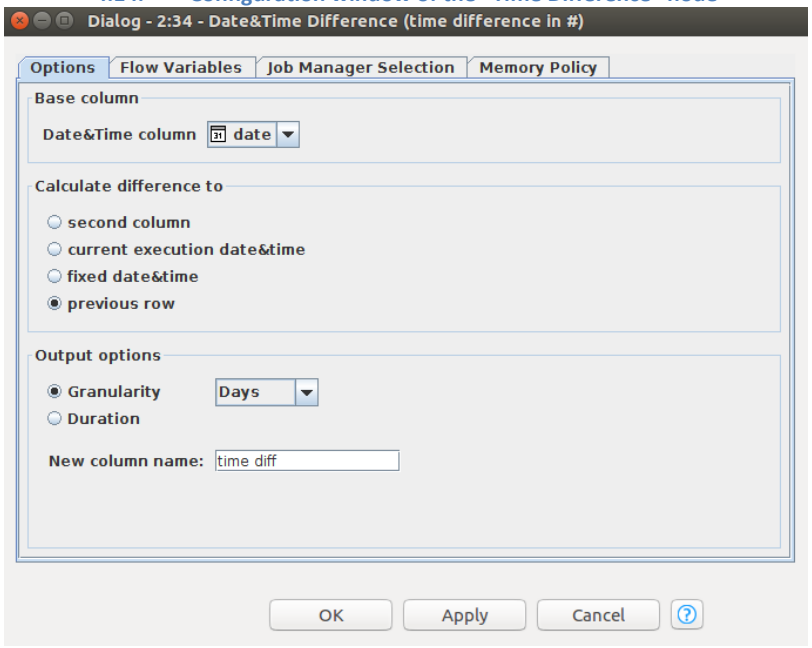
This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

## Date&Time Difference

The "Date&Time Difference" node calculates the time elapsed between two Date&Time values. The configuration window requires :

- The input column to use for the first Date&Time values
- The second Date&Time value, that is:
  - another Date&Time value in the same data row (Use "second column" option). In this case the name of the second column is required.
  - the current date/time at execution (Use "current execution date&time" option)
  - a fixed date/time (Use "fixed date&time" option). In this case the fix Date&Time value is required.
  - the Date&Time cell in the previous row in the same column (Use "previous row" option)
- The output options:
  - Granularity to output the difference in number of days, months, years, etc …
  - Duration to express the difference as date-based or time-based duration, according to ISO-8601 date and time duration standards
  - Name of the output column

4.14. Configuration window of the "Time Difference" node




In the "DateTime_Manipulation_2" workflow, we connected a "Date&Time Difference" node to the "File Reader" node to calculate the time elapsed between today ("Current execution date&time" option) and the Date&Time values in the "date" column, i.e. to calculate how long ago the sale contract was made. We selected "month" for the granularity of the time difference, i.e. the measure of how old a sale is, is expressed in number of months. The resulting differences were appended to a column named "time_diff". Sales older than one year had a "time_diff" value larger than 12 months. We then used a "Row Filter" to filter out all rows where "time_diff" > 12 months

Another interesting application of the "Date&Time Difference" node is to calculate the time intervals between one sale and the next. This is helpful to see if the sale process improves with time or after some special event, for example some marketing initiative. In this case we need to sort the sale records by sale date with a "Sorter" node. The "Sorter" node can manage Date&Time types and can sort them appropriately. After that, we use a "Date&Time Difference" node and we configure it to calculate the time difference between the sale date in the current row and the sale date in the



This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

previous row in terms of days. The resulting column “time_diff” contains the number of days between one sale and the previous one. A simple line plot of the “time_diff” column can give us interesting insights into the dynamics of the sale process.

## 4.5. Moving Average and Aggregation

In the “Other Data Types”/“Time Series”/”Smoothing” sub-category you also find a node that is more oriented towards time series analysis rather than just Date&Time manipulation: this is the “Moving Average” node.

The “Moving Average” node calculates the moving average [4] on a time series stored in a column of the input data table. The moving average operates on a k-sample moving window. The k-sample moving window is placed around the n-th sample of the time series. An average measure is calculated across all values of the moving window and replaces the original value of the n-th sample of the time series. Then the moving window moves over to the next (n+1)-th sample of the time series. And so on…

A number of slightly different algorithms can produce slightly different moving averages. The differences consist of how the moving window is placed around the n-th sample and how the average value is calculated. Thus, two parameters are particularly important for a moving average algorithm:

- The position of sample n inside the k-sample moving window
- The formula to calculate the average value of the moving window

The moving average algorithm is called:

- Backward, when the *n-th* sample is the last one in the moving window
- Center, when the *n-th* sample is in the center of the moving window (in this case size *k* must be an odd number)
- Forward, when the *n-th* sample is at the beginning of the moving window
- Cumulative, when the whole past represents the moving window; in this case is *n=k*
- Recursive, when the new value of the *n-th* sample is calculated on the basis of the *(n-1)-th* sample

If *v(i)* is the value of the original sample at position *i* inside the k-sample moving window, the algorithm to calculate the average value for the n-th sample can be one of the following:



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

| Algorithm | Formula | Notes |
|---|---|---|
| simple average measure | $avg(n) = \frac{1}{\mathrm{k}} \cdot \sum_{i=0}^{k} \mathrm{v(i)}$ | |
| Gaussian weighted average measure | $avg(n) = \frac{1}{\mathrm{k}} \cdot \sum_{i=0}^{k} \mathrm{w(i)} \cdot \mathrm{v(i)}$ | Where w(i) is a Gaussian centered around the nth sample, whose standard deviation is $\frac{k-1}{4}$ |
| harmonic mean | $avg(n) = \frac{n}{\sum_{i=0}^{k-1} \frac{1}{\mathrm{v}\left(\mathrm{n} + \mathrm{i} - \frac{\mathrm{k}-1}{2}\right)}}$ | The harmonic mean can only be used for strictly positive *v(i)* values and for a center window. |
| simple exponential | $avg(n) = EMA(v,n) = \alpha \cdot \mathrm{v(n)} + (1-\alpha) \cdot simple\_exp(\mathrm{n}-1)$<br>$simple\_\exp(0) = v(0)$ | Where: $\alpha = \frac{2}{k+1}$ and *v = v(n)* |
| double exponential | $avg(n) = 2 \cdot EMA(v,n) - EMA(EMA(v,n),n)$ | |
| triple exponential | $avg(n) = 3EMA(v,n) - 3EMA(EMA(v,n),n) + EMA(EMA(EMA(v,n),n),n)$ | |
| | | Where: $\alpha = \frac{2}{k+1}$ and |



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

| | | |
|---|---|---|
| old exponential | $avg(n) = EMA\_backward(v,n)$ $= \alpha \cdot v(n) + (1-\alpha) \cdot backward\_simple(n-1)$ | *backward_simple(n)* is the simple average of the moving window where the *n-th* sample is at the end. |

Based on the previous definitions, a *backward simple moving average* replaces the last sample of the moving window with the simple average; a *simple cumulative moving average* takes a moving window as big as the whole past of the n-th sample and replaces the last sample (*n-th*) of the window with the simple average; a center Gaussian moving average replaces the center sample of the moving window with the average value calculated across the moving window and weighted by a Gaussian centered around its center sample; and so on. The most commonly used moving average algorithm is the center simple moving average.

## Moving Average

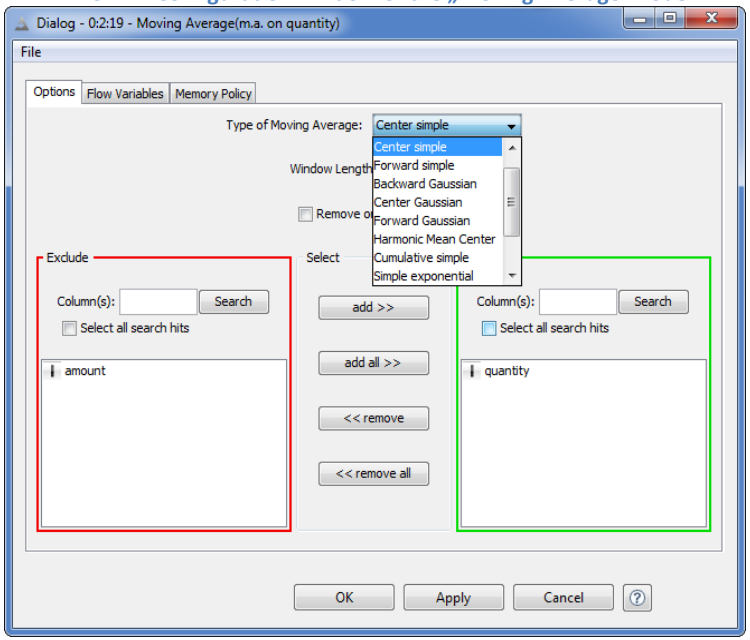
The "Moving Average" node calculates the moving average of one or more columns of the data table. The configuration window requires :

- The moving average algorithm
- The length of the moving window in number of samples
- The flag to enable the removal of the original columns from the output data table
- The input data column(s) to calculate the moving average

The selection of the data column(s), to which the moving average should be applied, is based on an "Exclude/Include" framework.

- The columns to be used for the calculation are listed in the "Include" frame on the right
- The columns to be excluded from the calculation are listed in the "Exclude" frame on the left

4.15. Configuration window of the „Moving Average" node




To move single columns from the "Include" frame to the "Exclude" frame and vice versa, use the "add" and "remove" buttons. To move all columns to one frame or the other use the "add all" and "remove all" buttons.



This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

A “Search” box in each frame allows searching for specific columns, in case an excessive number of columns impedes an easy overview.

**Note.** If a center moving average is used, the length of the moving window must be an odd number. The first (n-1)/2 values are replaced with missing values.

In the “DateTime_Manipulation_2” workflow we applied a “Moving Average” node to the output data table of the “File Reader” node. The center simple moving average was applied to the “quantity” column, with a moving window length of 11 samples. A “Line Plot” node placed after the “Moving Average” node showed the smoothing effect of the moving average operation on the “quantity” time series (Fig. 4.17).

The “Moving Aggregation” node extends the “Moving Average” node. Indeed, it calculates a number of additional statistical measures, besides average, on a moving window. In the configuration window you need to select the data column, the statistical measure to apply, the size and type of the moving window, and a few preferences about the output data table structure. Many statistical and aggregation measures are available in the “Moving Aggregation” node. They are all described in the tab “Description” of the configuration window.

## Moving Aggregation

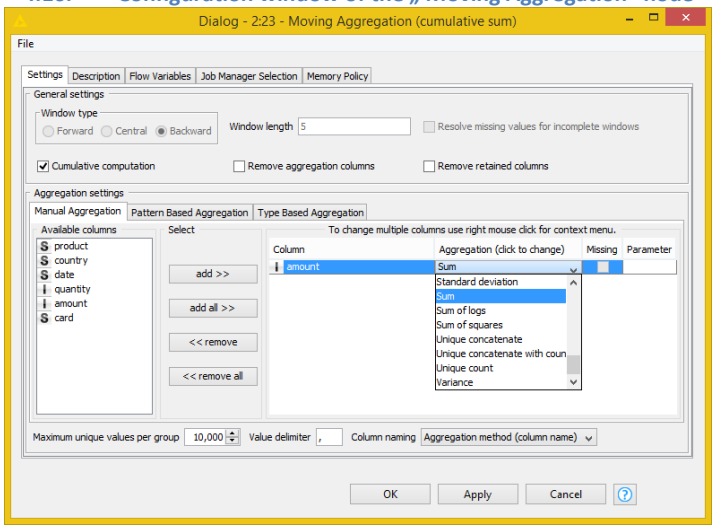
The “Moving Aggregation” node calculates statistical and aggregation measures on a moving window.

The “Settings” tab in the configuration window requires:

- The statistical or aggregation measure to use
- The input data column for the calculation
- The type and size of the moving window
- Checkboxes for the output data table format
- The checkbox for cumulative aggregation

A second tab, named “Description”, includes a description of all statistical and aggregation measures available for this node.

4.16. Configuration window of the „ Moving Aggregation“ node






This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

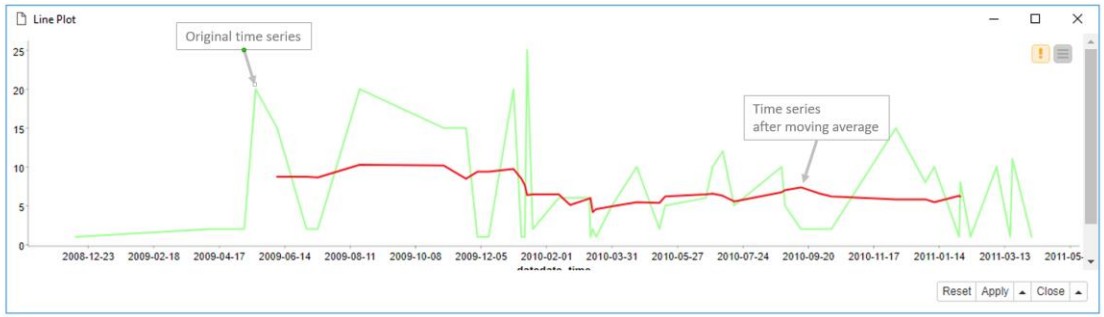
**Note.** Selecting data column “amount”, aggregation measure “mean”, window type “Central”, and window size 11, the same output time series is generated as by the “Moving Average” node as configured above.

Finally, when checking the “Cumulative computation” checkbox, the “Moving Aggregation” node uses the whole time series as a time window and performs a cumulative calculation. The most common cumulative calculation is the cumulative sum used in financial accounting for year to date measures.

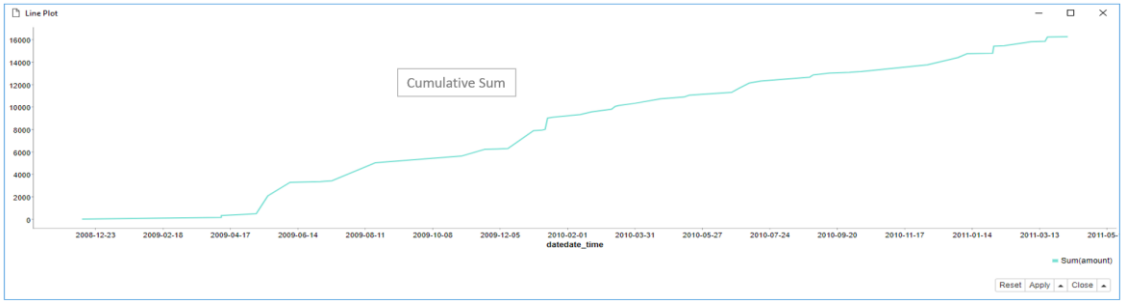
In the “DateTime_Manipulation_2” workflow, we added a “Moving Aggregation” node to the output data table of the “File Reader” node. The cumulative sum was calculated for the “amount” data column, over the whole time series. The plot of the resulting time series is obtained through a “Line Plot (JavaScript)” node.

Note. The “Fast Fourier Transform (FFT)” node implements the Fast Fourier Transform of a signal. The node is part of the “AI.Associates Signal Processing Nodes” extension under “KNIME Community Contributions – Other”.

### 4.17. Moving Average effect on time series “quantity”




### 4.18. Cumulative Aggregation of time series “amount”






This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

4.19. The “DateTime_Manipulation_2” workflow


## 4.6. Time Series Analysis

The “Time Series” category offers also some meta-nodes for time series analysis. Three meta-nodes are available: to correct seasonality, to train an auto-prediction model, to apply the auto-prediction model to predict new value(s). All these nodes rely on the “Lag Column” node.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

## Lag Column

The "Lag Column" node copies a data column x(t) and shifts it: *n* times 1, 2, …, n step each time or one time only *p* steps down. It can work in three different ways, producing:

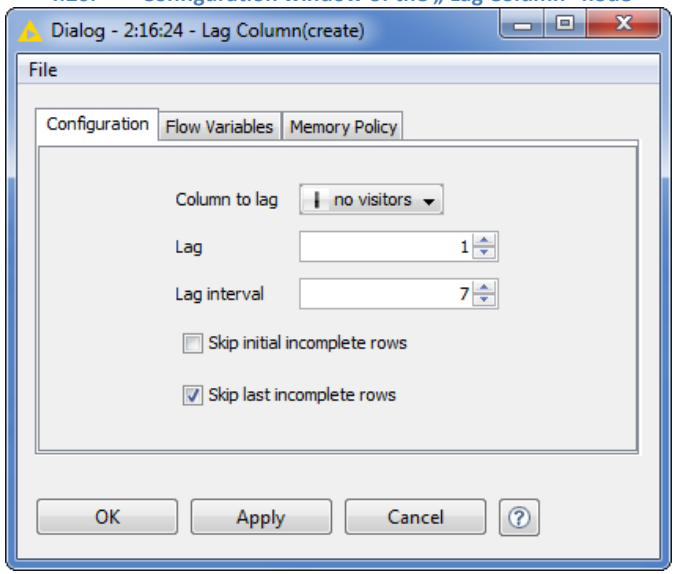
- one copy shifted *p* steps x(t), x(t-p)
- *n* copies, each shifted *1, .. n* steps resp. x(t), x(t-1), …, x(t-n)
- *n*p* copies, each shifted *p*(1, …n )* steps x(t), x(t-p), x(t-p*2), … x(t-p*n)

Where *p* is the "Lag Interval" and *n* the "Lag" value.

The data column to shift, the Lag, and the Lag Interval are then the only important configuration settings required.

Two more settings state whether the first and last incomplete rows generated by the shifting process have to be included or removed from the output data set.

4.20. Configuration window of the „ Lag Column" node




**Note.** If the data column values are sorted in time ascending order, shifting the values up or down means shifting the values in the past or in the future. Vice versa if sorted in time descending order.

The metanode "Seasonality Correction" defines a seasonality index in a "Lag Column" node, for example Lag Interval=7 for a daily time series with weekly seasonality. The "Lag Column" node then pairs each time series value with its correspondent 7 days earlier. Note that this temporal alignment is only true if the time series had been previously sorted by time in ascending order. Following the "Lag Column" node, a "Java Snippet (simple)" node subtracts the past sample (one week earlier in our case) from the current sample, de facto subtracting last week values from current week values and removing weekly seasonality. The output time series is just the series of differences of this week with respect to past week values.

The metanode "Time-Series Auto-Prediction Training" builds an auto-predictive model for time series. First, a "Lag Column" node aligns past and present, that is current value with previous *n* values (Lag=*n*), putting in a row x(t-n), x(t-n-1), … x(t). The idea is to use the past x(t-n), x(t-n-1), …, x(t-1) to predict the future x(t). After the "Lag Column" node, a "Partitioning" node partitions the data sequentially from the top. Sequential partition is important for time series analysis. Since some data mining algorithm can learn the underlying dynamic of a time series, randomly partitioning data would offer bits of future to the learning algorithm, which would never happen in real life applications. Finally, a learner node handling numeric

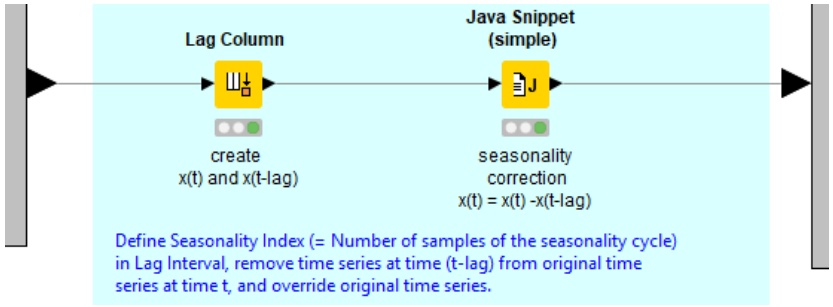


This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

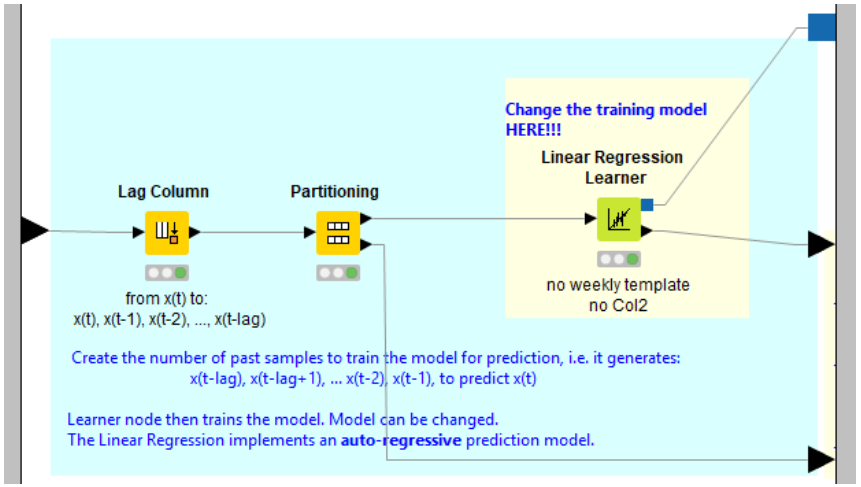
predictions predicts x(t) based on the past values in the same row. The default learner node is the “Linear Regression Learner” node, but it can be changed with any other learner node able to handle numeric predictions.

The “Time-Series Auto-Prediction Predictor” metanode uses the model built by the “Time-Series Auto-Prediction Training” metanode, to predict new values based on the existing ones. It contains the predictor node associated with the learner node in the “Time-Series Auto-Prediction Training” metanode and a “Numeric Scorer” node to quantify the prediction error.

#### 4.21. “Seasonality Correction” metanode




#### 4.22. “Time-Series Auto-Prediction Training” metanode

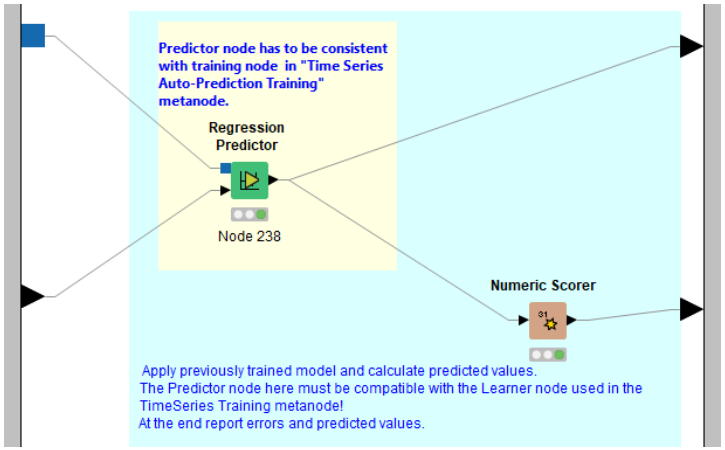



In the workflow named “time_series” we import the time series consisting of number of daily visitors to a web site over time from 2012 till 2014. After removing missing values and sorting the time series by ascending time, we remove the weekly seasonality (Lag Interval = 7) pattern with a “Seasonality Correction” metanode, we train an auto-regressive model using a “Linear Regression Learner” node in a “Time-Series Auto-Prediction Training” metanode, we calculate the predicted values and the numerical prediction error with a “Time-Series Auto-Prediction Predictor” metanode.

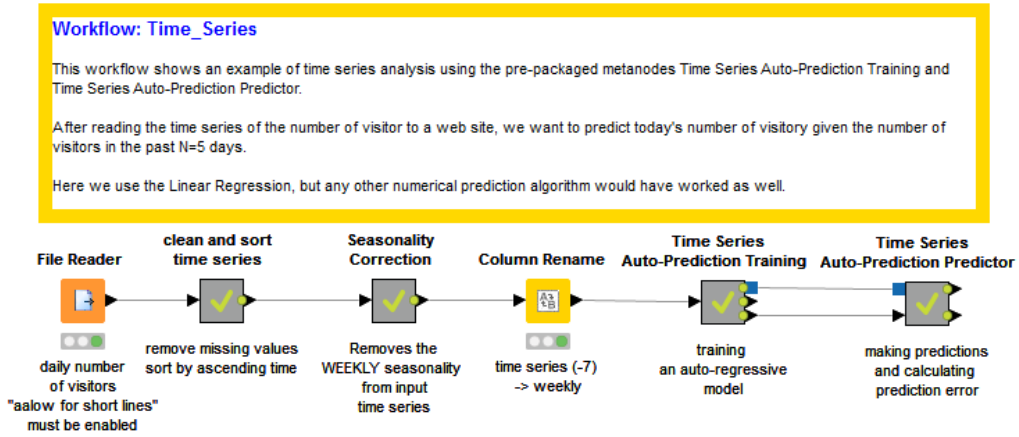


This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

4.23. "Time-Series Auto-Prediction Predictor" metanode




4.24. The "time_series" workflow




## 4.7. Exercises

Create a workflow group "Exercises" under the existing "Chapter4" workflow group to host the workflows for the exercises of this chapter.

### Exercise 1

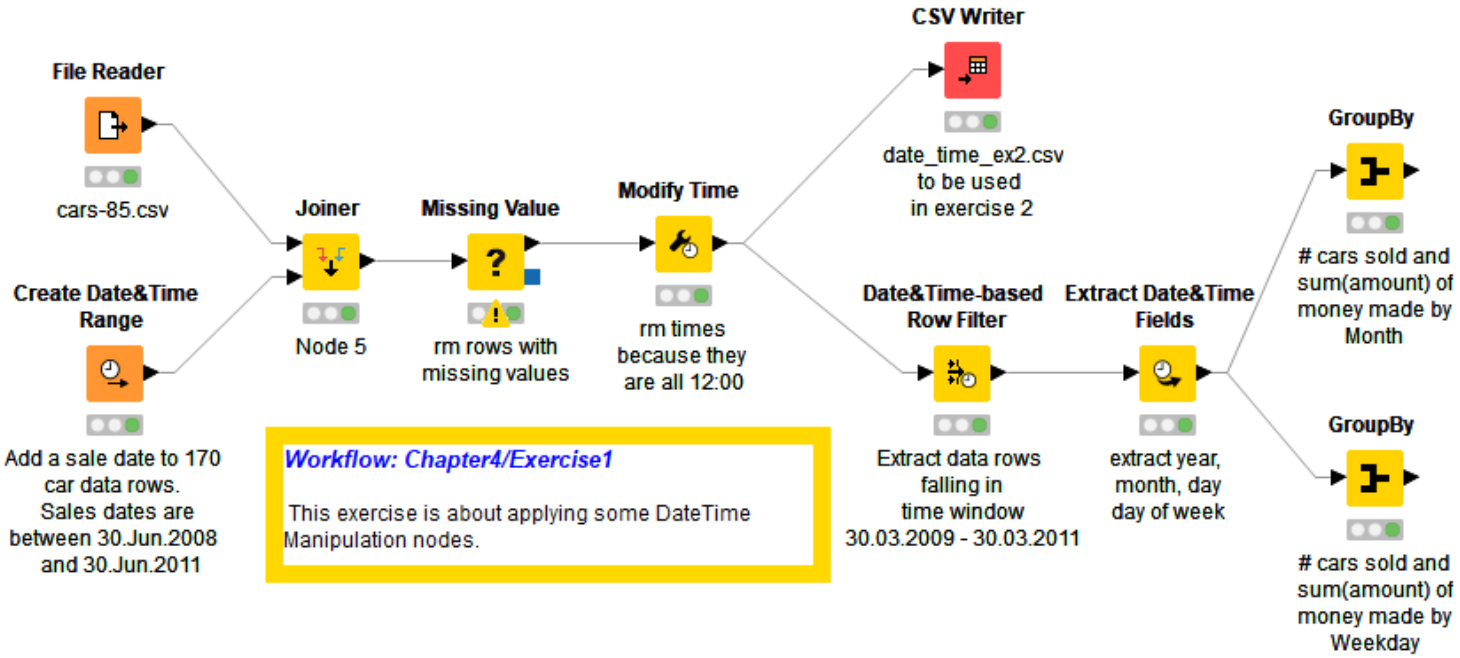
- Add a random date between 30.Jun.2008 and 30.Jun.2011 to the first 170 rows of the "cars-85.csv" file;
- Remove the time;
- Write the new data table to a CSV file;
- If you consider the newly added dates as sale dates and the values in the "price" column as sale amounts, find out which month and which weekday collects the highest number of sales and the highest amount of money.



This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

**Solution to Exercise 1**

4.25. Exercise 1: The workflow




The rows generated by the "Time Generator" node are only 170. The "Joiner" node is then set with a "left outer Join" to keep all rows of the "cars-85.csv" file. The "CSV Writer" node writes a file named "date_time_ex2.csv".

## Exercise 2

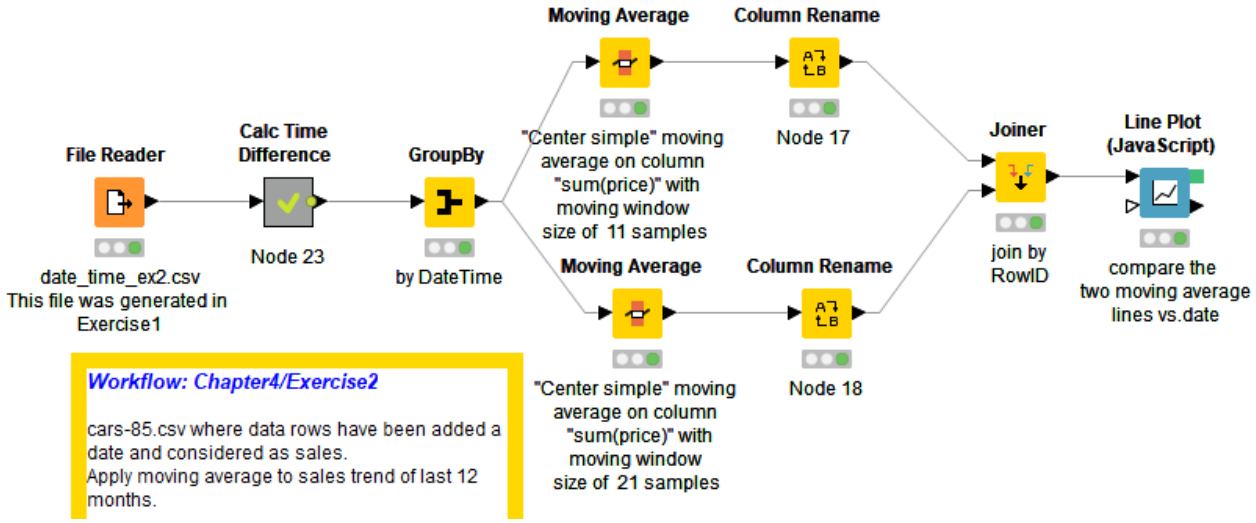
- Import the "date_time_ex2.csv" file produced in Exercise 1;
- Isolate the data rows with "date and time" older than one year from today (in the workflow solution today's date was "06.Apr.2011");
- Calculate the amount of money ("price" column) made in each sale date ( "date and time" column);
- Apply a moving average algorithm to the time series defined in the previous exercise and observe the effects of different sizes of the moving window.



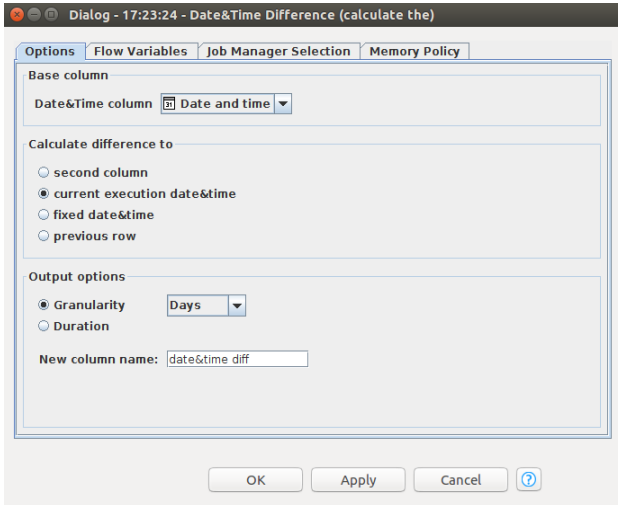
This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

**Solution to Exercise 2**

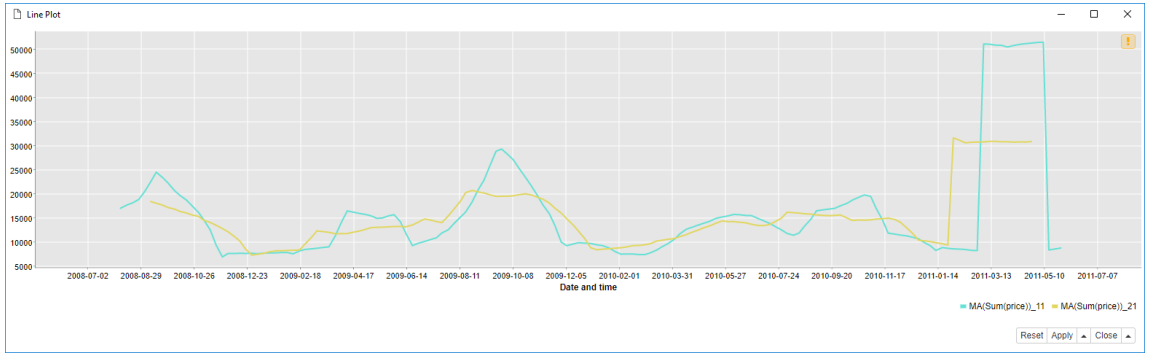
4.26. Exercise 2: The workflow




4.27. Settings for the "Time Difference" node in "Calc Time Difference" metanode




4.28. Plots of the original time series, after a center simple moving average on 11 samples, and after a center simple moving average on 21 samples






This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

# Chapter 5. Flow Variables

## 5.1. What is a Flow Variable?

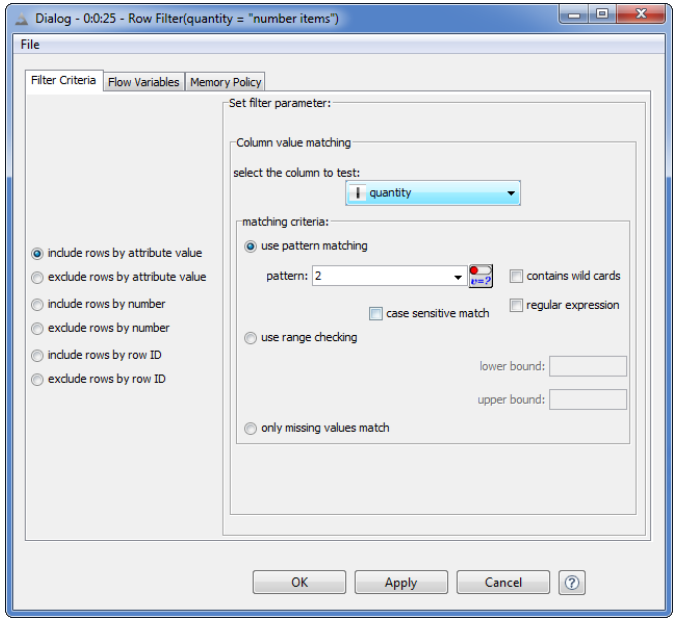
In KNIME Analytics Platform it is possible to define external parameters to be used throughout the entire workflow: these parameters are called "Workflow Variables" or shortly "Flow Variables". Flow variable values can be updated before or during each workflow execution through dedicated nodes and features; the new values can then be used to parametrically change any node configuration settings.

Let's create a workflow group called "Chapter5" to host the workflows implemented in the course of this chapter. Let's also create an empty workflow named "Flow_Vars" as its first example workflow. First thing, we read the "sales.csv" file from the KCBData folder; then we convert column "date" from type "String" to type "DateTime"; finally we apply the "Date&Time-based Row Filter" node to filter the rows with "date" between "01.Jan.2009" and "01.Jan.2011".

On the remaining rows, we want to find all those sales where a given number of items ("quantity" column) has been sold. For example, if we look for all sales, where 2 items have been sold, we just use a "Row Filter" node using "quantity = 2" as a filtering criterion.

Let's suppose now that the number of sold items we are looking for is not always the same. Sometimes we might want to know which sales sold 3 items, sometimes which sales sold 10 items, sometimes just which sales sold more than *n* items, and so on. In theory, at each run I should open the "Row Filter" node and update the filtering criterion. But this is a very time consuming approach, especially if more than one Row Filter node is involved, and it is not well suited to a remote workflow execution on a KNIME server.

5.1. "Row Filter" node's settings to find the sales records where exactly 2 items have been sold




We can parameterize the pattern matching in the row filtering criterion by using a flow variable. In this example, we could define a flow variable as an integer with name "number items" and initial (default) value 2. We could then change the matching pattern in the filtering criterion in the "Row Filter"



This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

node to take on the flow variable value rather than the fixed number set in the configuration window. That is, we would like to have a filtering criterion like `quantity = "number items"` rather than `quantity = 2`. At each workflow execution, the value of the flow variable can be changed to retrieve only those sale records with the newly specified number of sold items.

There are two ways to create a flow variable in a workflow:

- Globally at the workflow level and available to all nodes in the workflow;
- Locally inside the workflow by means of dedicated nodes and available only to the subsequent nodes in the workflow.

The following sections explore these two options. That is, how to create and update a flow variable at the workflow level and make it available to the whole workflow and how to create and update a flow variable from inside the workflow and make it available only to subsequent nodes.

## 5.2. Creating a Flow Variable for all Nodes in the Workflow

In this section we create a global flow variable visible by all nodes in the "Flow_Vars" workflow. In order to implement the parameterized row filtering described in the previous section, the flow variable is created of type Integer, with name "number items", and a default value of 2.



This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

In the “KNIME Explorer” panel:

- Select the workflow that needs the flow variable
- Open the workflow’s context menu (right-click)
- Select the “Workflow Variables …” option
- A small window for the administration of global flow variables opens. This window contains the list of global flow variables already available to the workflow, if any. Each flow variable is described by three parameters: Name, Type, and Value.

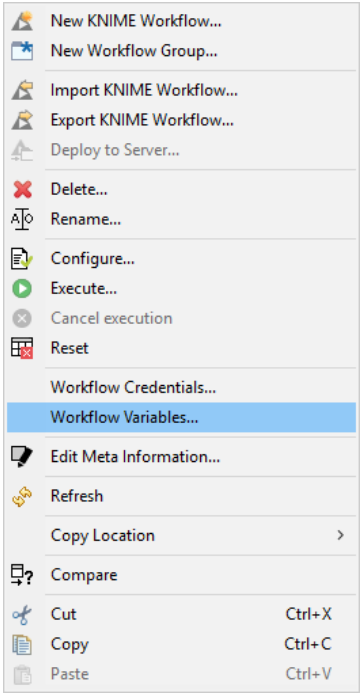
In the “Workflow Variable Administration” window (Fig. 5.3), there is:

- The list of global flow variables for this workflow
- Three buttons:
    - The “Add” button introduces a new global flow variable
    - The “Edit” button allows the three parameters for the selected flow variable to be modified
    - The “Remove” button deletes the selected flow variable
- Click the “Add” button to create a new global flow variable.
- The “Add/Edit Workflow Variable” window opens (Fig. 5.4).

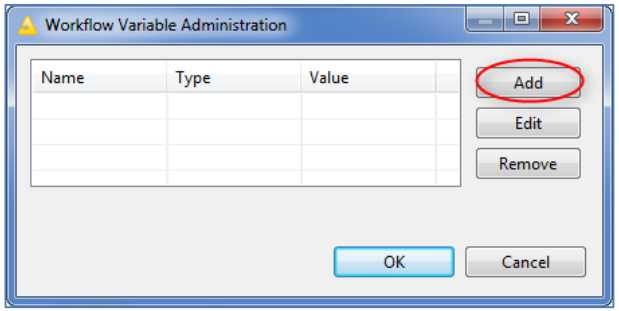
In the “Add/Edit Workflow Variable” window:

- Define name, type, and default value for the new global flow variable
- Click “OK”.

5.2. Option “Workflow Variables” in workflow’s context menu




5.3. “Workflow Variable Administration” window



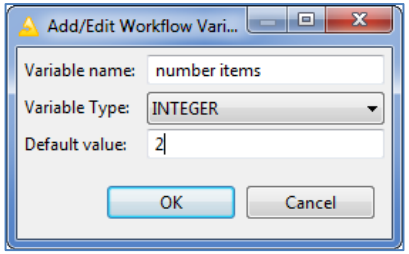



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

Back in the “Workflow Variable Administration” window:

- The newly defined global flow variable has been added to the list
- Click “OK”, to complete the flow variable creation process.
- Since the value of this flow variable potentially affects all nodes in the workflow, you are now prompted to reset all nodes of the workflow.

5.4. “Add/Edit Workflow Variable” window




Following these steps, we created a new global flow variable named “number items”, of type Integer, and with default value 2.

**Note.** This global flow variable is visible by all nodes in the “Flow_Vars” workflow. However, it will not be visible by nodes in other workflows.

## 5.3. Flow Variable Values as Node Settings

Once a flow variable has been created, it can be used to overwrite the settings in a node configuration window. In our example, in the workflow named “Flow_Vars”, the flow variable “number items” must overwrite the value of the matching pattern in the filtering criterion of the “Row Filter” node.

In the configuration window of some nodes, a “Flow Variable” button (Fig. 5.5) is displayed on the side of some of the settings. For example, you can find it in the “Row Filter” node, in the “matching criteria” panel for the “use pattern matching” option, to the right of the “pattern” textbox (Fig. 5.1).



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

## The “Flow Variable” Button

5.5. The “Flow Variable” button

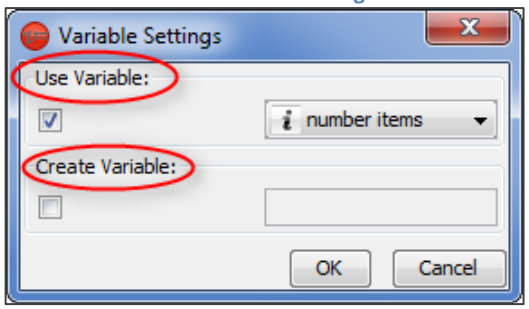
The “Flow Variable” button allows you to use the value of a flow variable to overwrite the value of the corresponding node setting.

By clicking the “Flow Variable” button, the “Variable Settings” window opens and asks whether:

- The value from an existing flow variable should be used for this setting. If yes, then:
  - Enable the “Use Variable” flag
  - Select one of the existing flow variables from the menu
  - The value of the selected flow variable now defines the value of this setting at execution

  OR
- A new flow variable should be created with the current value of this node setting. In this case:
  - Enable the “Create Variable” flag
  - Provide a name for the new flow variable
  - A new flow variable with the selected name and with that setting value is created and made available to all subsequent nodes in the workflow

5.6. The “Variable Settings” window




In the “Flow_Vars” workflow, we selected the existing flow variable “number items” - created in section 5.2 - to overwrite the value of pattern matching setting in the “Row Filter” node named “*quantity = "number items" via pattern matching criterion*”.

You can check which flow variables are available at a given point in a workflow by:

- Opening the resulting data table of the node (last option in the node context menu)
- Selecting the “Flow Variables” tab

The “Flow Variables” tab contains the full list of flow variables, with their current values, available for that node.

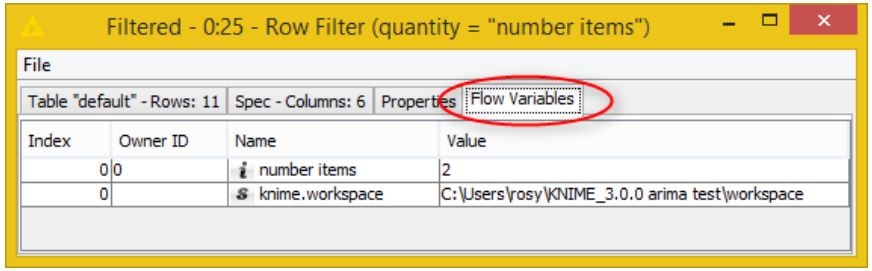


This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

Not all configuration settings display a “Flow Variable” button. The “Flow Variable” button has been implemented for only some settings in some nodes.

A more general way to overwrite the value of a configuration setting through the value of a flow variable involves the “Flow Variables” tab in the node’s configuration window.

5.7. The “Flow Variables” tab in the data table of the “Row Filter” node shows one flow variable only available at this point of the workflow, named “number items”, of Integer type, and with value 2, besides the default knime.workspace flow variable




## The “Flow Variables” Tab in the Configuration Window

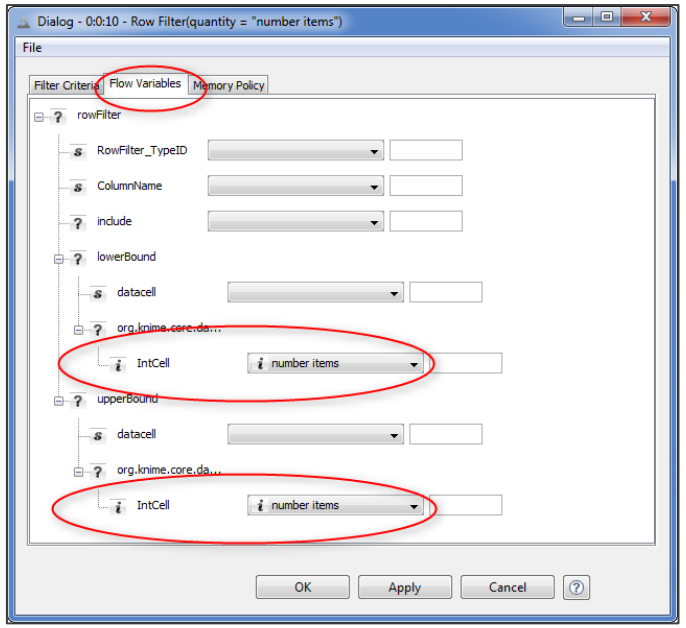
- Open the configuration window of a node
- Select the “Flow Variables” tab

The “Flow Variables” tab allows to overwrite each of the node configuration settings with the value of a flow variable.

- Find the desired configuration setting and open the menu on the right containing the list of available flow variables
- Select the flow variable to overwrite this setting
- Click “OK”

At execution time, the node will assign the value of the selected flow variable to that setting.

5.8. The “Flow Variables” tab in the configuration window of a “Row Filter” node with “use range checking” filter criterion enabled






This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

In “Flow_Vars” workflow, we performed the same row filtering as before (i.e. column “quantity” = value flow variable “number items”) , but this time using the “use range checking” filter criterion with “lower bound” = “upper bound” = value of flow variable “number items”. This should implement exactly the same type of row filtering as by using the pattern matching filtering criterion and the “number items” flow variable as pattern to match.

Since the configuration settings named “lower bound” and “upper bound” in the “Row Filter” node do not have a “Flow Variable” button, we need to overwrite their values via the “Flow Variables” tab.

We created a new “Row Filter” node, opened its configuration window, and enabled the “use range checking” criterion. In the “Flow Variables” tab of the configuration window, we expanded the top category called “rowFilter”, then the “lowerBound” category, and finally the “org.knime.core.da…” category. There we set the value of the “IntCell” parameter to take the value of the flow variable named “number items” through the combobox menu on its right. We repeated the same sequence of operations to set the value of parameter “IntCell” in the “upperBound” category (Fig. 5.8). This “Row Filter” node has been named “*quantity = "number items" via range checking criterion*”.

After defining the value of a configuration setting by means of a flow variable, a warning message, characterized by a yellow triangle, appears in the lower part of the configuration window. The simple goal of this message is to warn the user that this particular parameter is controlled by a flow variable and therefore changing its value manually within the configuration window will be ineffective. In fact, if a configuration setting value has been set through a flow variable, the flow variable value will always overwrite the current setting at execution time.

In order to make effective the manual change of the parameter value, the user needs to disable the flow variable for this setting. This can be obtained by either disabling both options in the “Variable Settings” window (Fig. 5.6), accessible through the “Flow Variables” button, or by selecting the first empty option in the combobox menu in the “Flow Variables” tab of the configuration window (Fig. 5.8).

**Note.** In rare cases, it is still possible that some settings are not accessible even through the “Flow Variables” tab.

## 5.4. Creating a Flow Variable from within a Workflow

Let’s change the problem now and suppose that we do not know ahead the matching pattern for the filtering criterion. In this case, the matching pattern becomes known only during the workflow execution. Here we need to update the flow variable “on the fly” or, even better, we need to create the flow variable “on the fly” inside the workflow. In our case, the flow variable will be created and its value assigned as soon as it becomes known. There are three ways to proceed with this:

- transform a data value into a flow variable;



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

- transform a configuration setting into a flow variable;
- create a new flow variable and feed it on the fly.

All nodes dealing with flow variables can be found in categories “Workflow Control” -> “Variables” and “Workflow Control” -> “QuickForms”.

## Transform a Data Value into a Flow Variable

Let’s imagine that we want to analyze only the sale records of the country with the highest total number of sold items. To find out the total number of sold items for each country, in the “Flow_Vars” workflow, we connected a “GroupBy” node to the output port of the “Date&Time-based Row Filter” node. The “GroupBy” node was set to group the data rows by country and to calculate the sum of values in the “quantity” column. A “Sorter” node sorted the resulting aggregations by “sum(quantity)” in descending order. Thus the country reported in the first row of the final data table is the country with the highest number of sold items. On another branch in the workflow, a “Row Filter” node should retain all data rows where “country” is that country with the highest number of sold items, as reported in the first row of the output data table of the “Sorter” node.

Depending on the selected time window, we do not know ahead which country has the highest number of sold items. Therefore, we cannot assign the country name to a flow variable before running the workflow. We should first find that country, then transfer it into a flow variable, and finally use it to execute the “Row Filter” node.  The node, that transfers data table cells into an equal number of flow variables, is the “TableRow To Variable" node. The ”TableRow To Variable” node is located in the category “Workflow Control” -> “Variables”.

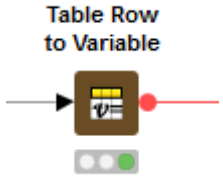
The “Workflow Control” category contains nodes that help to handle the data flow in the workflow. In particular, the sub-category “Variables” contains a number of utility nodes to deal with flow variables from inside the workflow, such as to transform data cells into variables and vice versa.

### TableRow To Variable

The ”TableRow To Variable” node takes a data table at the input port (black triangle), transforms the data cells into flow variables, and places the resulting set of flow variables at its output port (red circle).

No configuration settings are required for this node.

5.9. The “TableRow To Variable” node






This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

**Note.** The output port of the "TableRow To Variable" node is a red circle. Red circle ports carry flow variables either as input or as output.

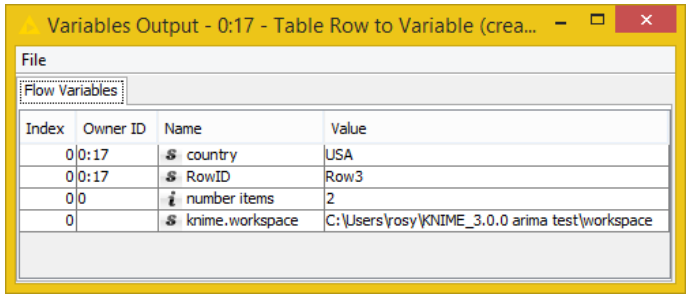
Going back to the "Flow_Vars" workflow, we isolated the "country" data cell of the first row in the output data table of the "Sorter" node, by using an additional "Row Filter" node to keep the first row only and a "Column Filter" node to keep the "country" column only. The resulting data table has only one cell containing the name of the country with the highest number of sold items. This one-cell data table was then fed into a "TableRow To Variable" node and therefore transformed into a flow variable named "country".

**Note**. The "TableRow To Variable" node creates as many flow variables as many input columns, each flow variable carrying the name of the original input column. The "TableRow To Variable" node also works only on the first row of the input data table. If more than one row is presented to the node, the newly created flow variables will take the values available in the first row of the corresponding input columns.

The list of flow variables available after the "TableRow To Variable" node can be seen by selecting the last item of the node context menu. This view contains the flow variables available for subsequent nodes. This includes all previously defined flow variables plus all the flow variables created by the "TableRow To Variable" node itself.

The view shows the new "country" variable with value "USA" for this execution run, the "RowID" variable also created by the TableRow To Variable" node, and the previously created global flow variable named "number items".

5.10. Flow Variables available in the workflow "Flow_Vars" after the "TableRow To Variable" node




## Transform a Configuration Setting into a Flow Variable

Let's suppose now to have a data table containing some values about specific countries. We simulated that table with a "Table Creator" node with just one column of values for the desired country (Fig. 5.11). The column name is the country name. We would like to extract all data rows for that country from the original input data table from file sales.csv. We should then use the column name as a matching pattern in the "Row Filter" node. It is actually possible to transfer the value of any configuration setting into the value of a new flow variable.



This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

The “Flow Variables” tab in each configuration window shows two boxes close to each setting name. One is the menu combobox that we have already seen and that is used to overwrite the setting value with an existing flow variable (Fig. 5.8). The other box, the last one on the right, is a textbox and implements the opposite pass; that is, it creates a new flow variable named according to the string in the textbox and containing the value of the corresponding configuration setting.

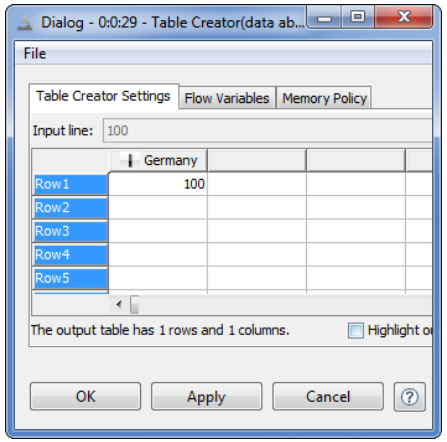
In the Table Creator node, the column name is actually a configuration setting. So, in the “Flow Variables” tab of its configuration window, we found the setting “columnProperties” -> “0” -> “ColumnName” which corresponds to the column name of the column number 0 in the configuration window of the Table Creator. We then decided to transfer this setting value into a new flow variable named “var_country” (Fig. 5.13) by filling the second textbox with the desired flow variable name.

If we now check the output data table of the “Table Creator” node and we select the “Flow Variables” tab, we see a new flow variable named “var_country” with value “Germany”, exactly the name of the column # 0 in the “Table Creator” node.

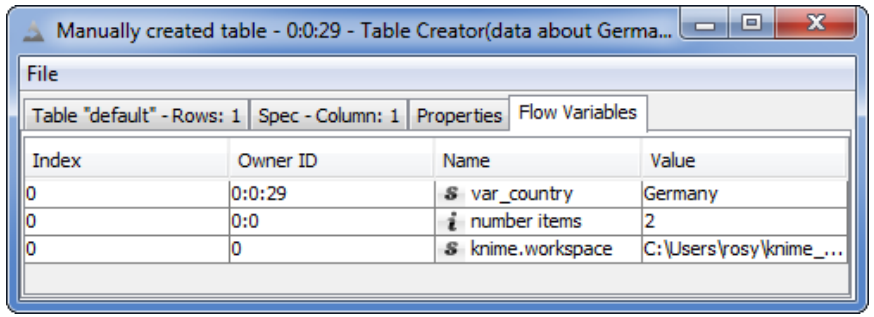


This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

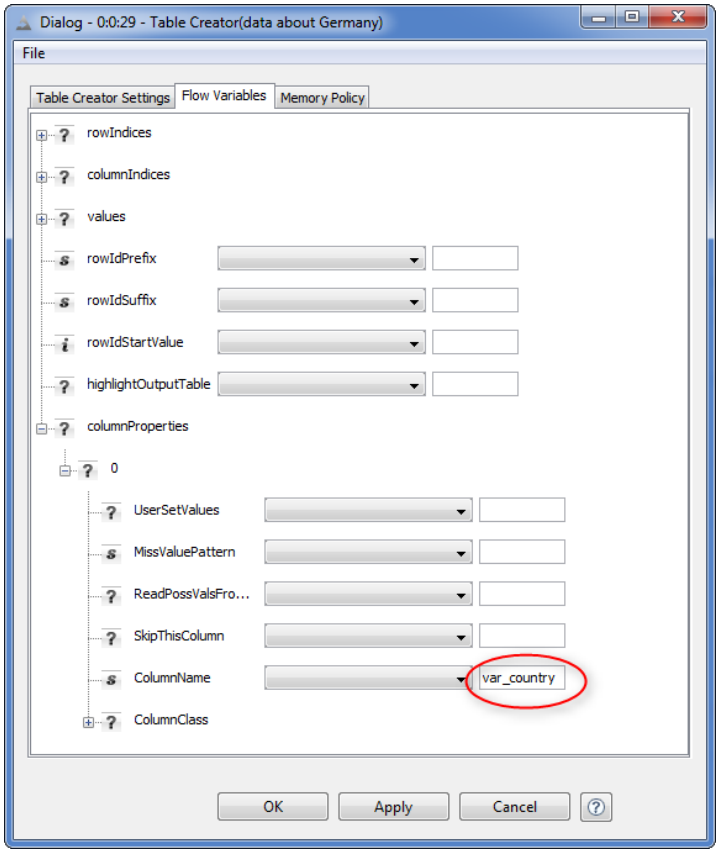
### 5.11. The "Table Creator" node: “Settings” Tab




### 5.12. Output Flow Variables of the "Table Creator" node in the "Flow_Vars" workflow




### 5.13. The "Table Creator" node: “Flow Variables” Tab



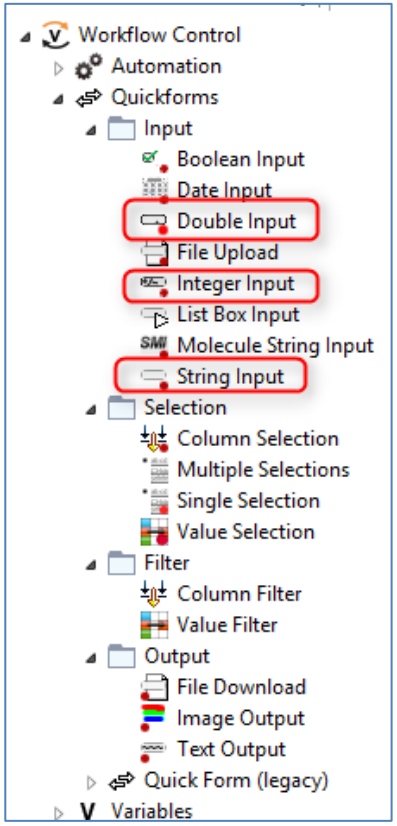



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

## Quickforms to Create Flow Variables

Another way to create a flow variable in the middle of a workflow is to use a Quickform node. Quickform nodes are located in the category “Workflow Control”/”Quickforms”, create flow variables, and provide interactive forms for a variety of tasks (see section 5.7 below). The simplest Quickform nodes to generate flow variables are the “Boolean Input”, the “Double Input”, the “Integer Input”, and the “String Input” node.

### 5.14. The simplest "Quick Form" nodes to generate flow variables




All Quickform nodes create a flow variable of a specific type, with a name and a value. Since they do no processing, they need no input. Thus, these nodes have no input and just one output port of flow variable type (red circle). They also share the same kind of configuration window, requiring: the name and the default value of the flow variable to be created, optionally some description of the flow variable purpose, and an explanation label to help the user for the assignment of new values.

Let’s have a look again at the example workflow “Flow_Vars” and particularly at the “Row Filter” node. The “Row Filter” node selects all those data rows with a pre-defined number of products (“quantity” column) by means of the flow variable “number items” and the range checking criterion. A local flow variable like the global one, named “number items” and containing the number of sold items for the filtering criterion, can also be created from inside the workflow with a Quickform node. This flow variable must be of type Integer to match the values in data column “quantity”.

We can use the “Integer Input” node from the “Workflow Control”/“Quickform” category. The “Integer Input” node was configured to produce a flow variable of type Integer, named “qkform_num_items”, with default value 2.

The “Boolean Input” node, the “Double Input” node, and the “String Input” node are structured the same way as the “Integer Input” node, with the only difference that they produce flow variables of type Boolean, Double, and String respectively. For example, if we now want to write the results of the filtering operation to a CSV file and if we want to parameterize the output file path by using a flow variable of type String, we could use a “String Input” node.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

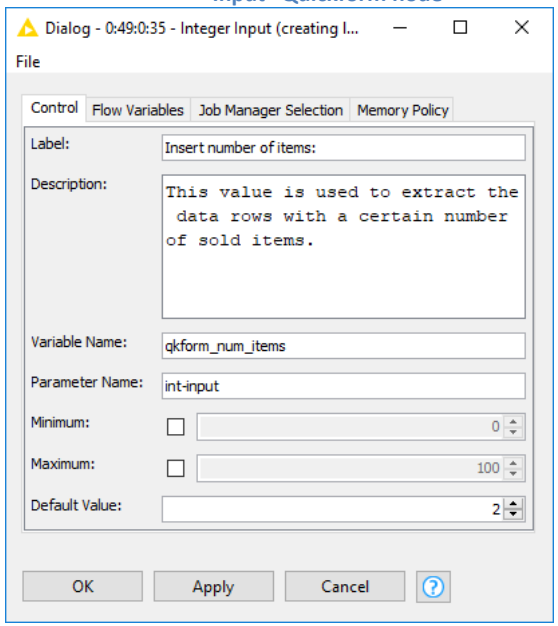
## Integer Input

The “Integer Input” Quickform node creates a new flow variable of type Integer, assigns a default value to it, adds it to the list of already existing flow variables, and presents it at the output port.

Its configuration window requires:

- The flow variable name
- The flow variable default value
- An optional description of the purpose of this flow variable
- A label that will be used as a help for the user when updating the flow variable with new values
- A parameter name for external identification, for example when running the workflow in batch mode
- An integer range (Minimum and Maximum) to verify and accept valid values

5.15. Configuration window of the “Integer Input” Quickform node






This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

## 5.5. Inject a Flow Variable through the Flow Variable Ports

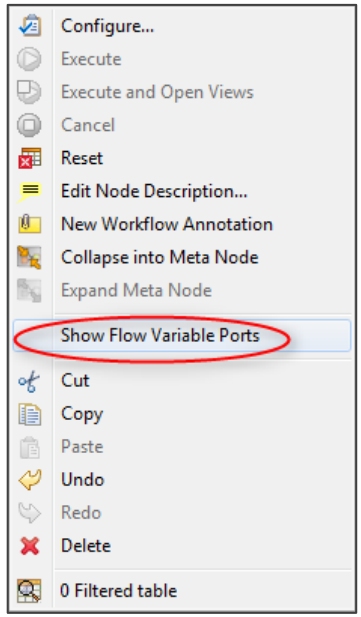
At this point, in our example workflow, we have four flow variables to feed a possible "Row Filter":

- "number items" which is a global flow variable;
- "country" which contains the name of the country with the highest number of sold items;
- "var_country" which contains the name of the country in the reference table from the "Table Creator" node;
- "qkform_num_items" generated by the "Input Integer" Quickform node.

Let's concentrate for now on the flow variable "country". This flow variable is at the output of the "TableRow To Variable" node. How do we make a "Row Filter" node aware of the existence of this new flow variable? How do we connect the "TableRow To Variable" node to the "Row Filter" node? We need to insert, or inject, the new flow variables back into the workflow, to make them available for all subsequent nodes.

All KNIME nodes have visible data ports and hidden flow variable ports. In the context menu of each node, the option "Show Flow Variable Ports" makes the hidden flow variable ports visible. We use these flow variable ports to inject flow variables from one node to the next or from one workflow branch to another. Notice that there are two flow variable ports: one port on the left to import flow variables from other nodes and one port on the right to export flow variables to other nodes.

5.16. Option "Show Flow Variable Ports" in the context menu of a node





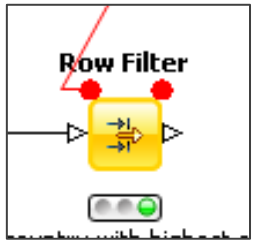

This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

## Flow Variable Injection into the Workflow

When the context menu option “Show Flow Variable Ports” is selected, the node shows two red circle ports on the top. The port on the left accepts the incoming flow variables and the port on the right exposes the updated flow variables available at this node.

In order to force, i.e. inject, flow variables into a node:

- Show its flow variable ports via the context menu,
- Connect the output variable port of the preceding node to the input variable port (i.e. the left one of the two flow variable ports) of the current node.




5.17. The flow variable ports of a node

The list of flow variables made available to subsequent nodes can be inspected by opening the output data table (last item in node context menu) and selecting the tab named “Flow Variables”. After the injection of new flow variables, the flow variable list should include the newly created flow variables as well as the flow variables that existed beforehand.

The flow variables injected into a node are only available for this and the connected successor nodes in the workflow. Nodes cannot access workflow variables injected later on in the workflow.

**Note.** Flow variable injection becomes necessary only when we need to transfer a flow variable from a branch of the workflow to another or when we need to connect a flow variable output port to the next node. In all other cases, new flow variables are automatically transferred from one node to the next through the data flow (black triangle ports).

In “Workflow_Vars” workflow, we first introduced a “Row Filter” node after the “TableRow To Variable” node; next, we displayed its flow variable ports through its context menu option; finally we connected the flow variable output port of the “TableRow To Variable” node to the input variable port (i.e. the flow variable port on the left) of the “Row Filter” node. As a result, the flow variable named “country” became part of the group of flow variables available to the “Row Filter” node and to the following nodes in the workflow. Finally, we configured the “Row Filter” node to use the pattern matching filter criterion and the value of the flow variable “country” as the matching pattern.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

**Note.** Flow Variable ports can also be used as a barrier point to control the execution order of nodes, i.e. nodes connected through a flow variable line will not start executing until all upstream nodes have been executed.

Sometimes we might want both flow variables, “country” and “var_country”, to be available to a “Row Filter” node. In this case, the only input flow variable port is not enough to inject two flow variables at the same time. KNIME has a node, named “Merge Variables”, to merge together many flow variables from different branches of a workflow.

### Merge Variables

The “Merge Variables” node merges flow variables into one stream. If flow variables with the same name are to be merged, the standard conflict handling is applied: most top inputs have higher priority and define the value of the post-merging variable.

This node needs no configuration settings.

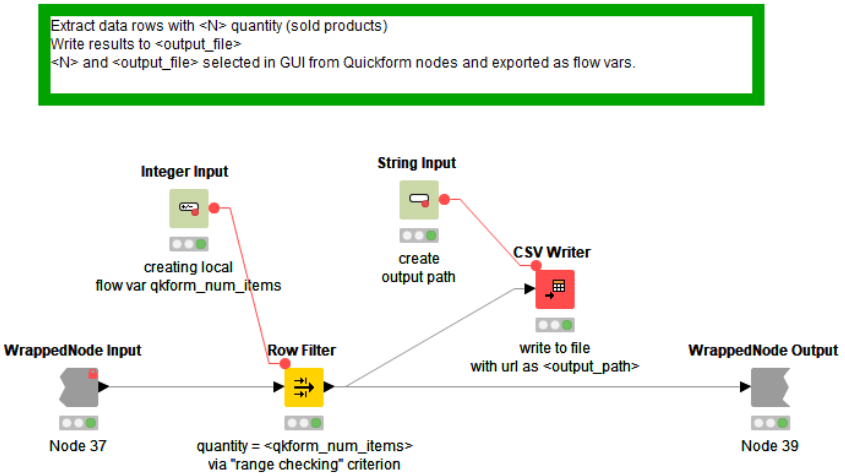
## 5.6. Quickforms, Wrapped Meta-nodes, and KNIME WebPortal

Let’s go back to that part of the “Workflow_Vars” workflow that used a Quickform node (an “integer Input” node) to overwrite the value of the upper and lower bound of the range checking criterion in the “Row Filter” node. Let’s add here a second Quickform node, a “String Input” node, to define the name of the output CSV file where to write the row filtering results.

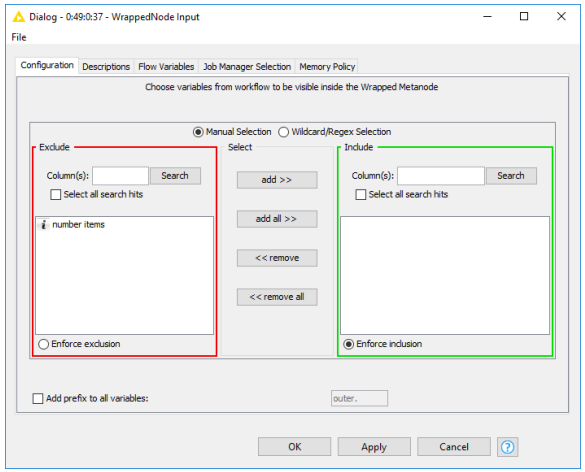


This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

5.18. "Integer Input" and "String Input" Quickform nodes inside a wrapped metanode to define the row filtering criterion and the output file path




5.19. Configuration window of the “WrappedNode Input” node in the wrapped meta-node shown in figure 5.18.




Often workflows become quickly too crowded with too many nodes. In order to clean up the workflow and collect together nodes belonging to the same logic unit, metanodes can be introduced. Metanodes are nodes containing other nodes. To create a new metanode from a set of existing nodes, select the nodes of interest, then right-click and select option “Collapse into Metanode”. The metanode is automatically created including all selected nodes and with the right number of input and output ports. Double-clicking a metanode, open its content.

An evolution of a simple metanode is a wrapped metanode. A wrapped metanode can be created by selecting the nodes of interest, right-clicking and selecting the option “Encapsulate into Wrapped Metanode”. Ctrl-double-click on a wrapped metanode opens its content. A wrapped metanode limits the input and output of flow variables in and out of the metanode. This vacuum environment is useful against the quick proliferation of flow variables. Content isolation ensures safer coding inside the wrapped node with lower risk of mixing up flow variables and their values. It is still possible to import or export a flow variable into or out of a wrapped metanode, if we set this explicitly.

The WrappedNode Input and the WrappedNode Output nodes in the wrapped metanode (Fig. 5.18) have a configuration window. Double-clicking them opens their configuration window, which offers and include/Exclude framework to allow flow variables to enter or exit the wrapped metanode.

A general wrapped meta-node that does not contain Quickform or JavaScript view nodes has no configuration window. The “Configure” option in its context menu is inactive. However, if a wrapped meta-node contains one or more Quickform or JavaScript View nodes, it acquires a configuration



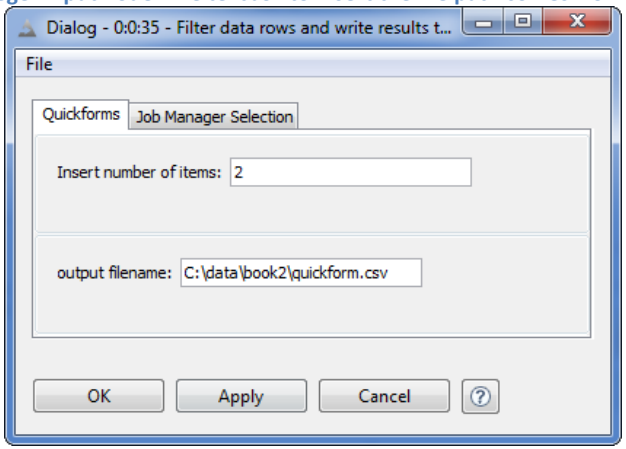
This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

window; that is the option “Configure” in the meta-node context menu becomes active. The configuration window is then filled with the textboxes and menus from the Quickform nodes.

The wrapped meta-node, shown in figure 5.18 and containing an Integer Input Quickform node and a String Input Quickform node, produces the configuration window in figure 5.20, requiring the output file path and the integer value for the flow variable “qkform_num_items” controlling the “Row Filter” node.

When running on a KNIME WebPortal, each wrapped metanode produces a web page displaying the same fields as in the acquired configuration window of the wrapped metanode.

5.20. Newly acquired configuration window of the meta-node containing the "Integer Input" node and the "String Input" node. The box to enter integer values comes from the Integer Input node. The textbox to insert the file path comes from the String Input node.




It is possible to transform a simple metanode into a wrapped metanode, by selecting “Metanode”-> “Wrap” from its context menu.

**Note**. Unlike meta-nodes, double-clicking a wrapped metanode leads to its configuration window and not the sub-workflow.

It is good practice to wrap the Quickform nodes inside wrapped metanodes rather than inside simpler meta-nodes. Indeed, since Quickform nodes generate flow variables and flow variables might end up having common names with other flow variables external to the wrapped nodes, it is good practice to keep the two sets of flow variables separated using the wrapped nodes as barrier.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

## 5.7. Transform a Flow Variable into a Data Value

Sometimes it is necessary to use the flow variable values as data values. To transform flow variables into data cells, we can use the "Variable To TableRow" node. The "Variable To TableRow" node mirrors the "TableRow To Variable" node. That is, it transforms a number of selected flow variables into cells of a data table.

### Variable To TableRow

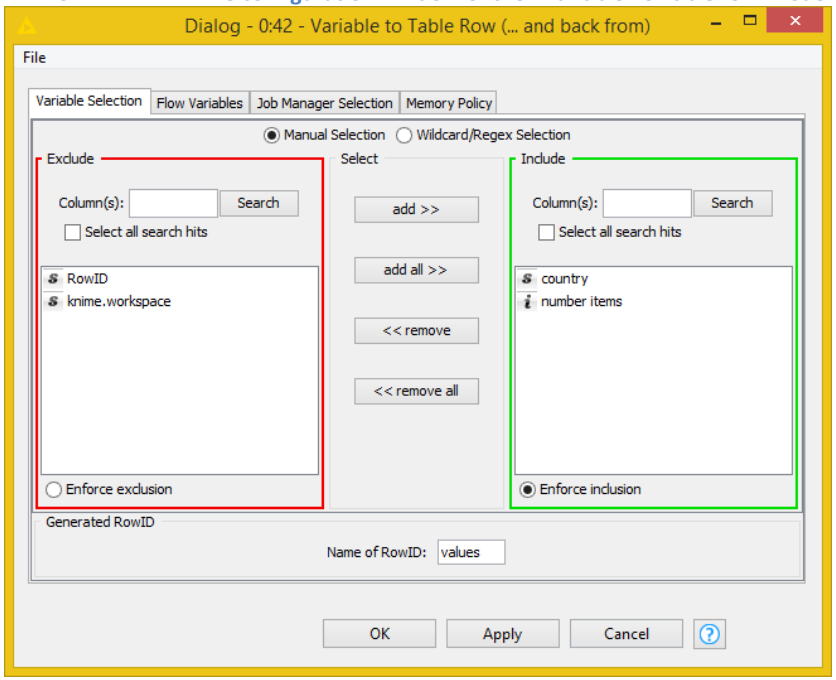
The "Variable To TableRow" node transforms selected flow variables at the input port (red circle) into cells of a data table at the output port (black triangle).

The configuration settings require the selection of the flow variables to be included in the output data table through an include/exclude panel.

The output data table contains two columns:

- One column with the name of the flow variable;
- The second column with the flow variable values at execution time.

5.21. The configuration window of the "Variable To TableRow" node




In our example workflow, "Flow_Vars", we connected a "Variable To TableRow" node to the "TableRow To Variable" node, in order to perform the opposite operation and transform all current flow variables into data cells.



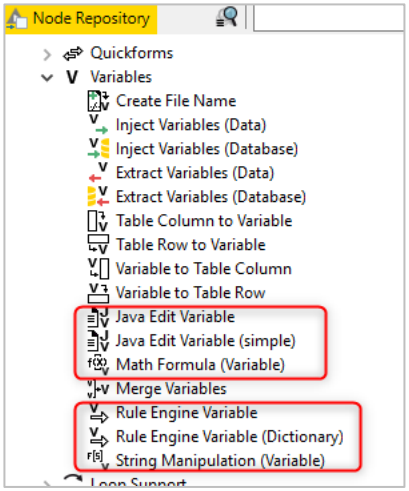
This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

## 5.8. Modifying Flow Variable Values

For the country with the highest number of sales, we now want to restrict the analysis scope to only those records with a high value in the "amount" column. This requires an additional "Row Filter" node implementing a filtering criterion on "amount" > x, where x is the threshold above which an amount value is declared high.

Let's suppose that different thresholds are used for different countries. For example, an amount is considered high for a sale in USA if it lies above 500; it is considered high in other countries if it lies above 100. In one case, we should set x = 500 and in the other cases x = 100. Therefore, the threshold x cannot be set as the static value of a global flow variable. Indeed, the flow variable value needs to be adjusted as the workflow runs.

5.22. Nodes modifying values in flow variables. These nodes mimic the corresponding nodes working on data. That is, same configuration window, but variable input and output ports rather than data ports.




There are a number of nodes dedicated to modifying the values in flow variables. They all mirror existing nodes operating on data tables, like Java Snippet, Math Formula, Rule Engine, and String Manipulation. The only difference of the variable edition with respect to the data edition of these nodes consists in the input/output ports. The nodes operating on data get data as input and produce data as output. The corresponding nodes operating on variables get variables as input and produce variable as output. Besides that, the node configuration windows mimic the configuration windows of the corresponding nodes working on data.

To complete our example workflow "Flow_Vars", we have added a few nodes for variable value manipulation.

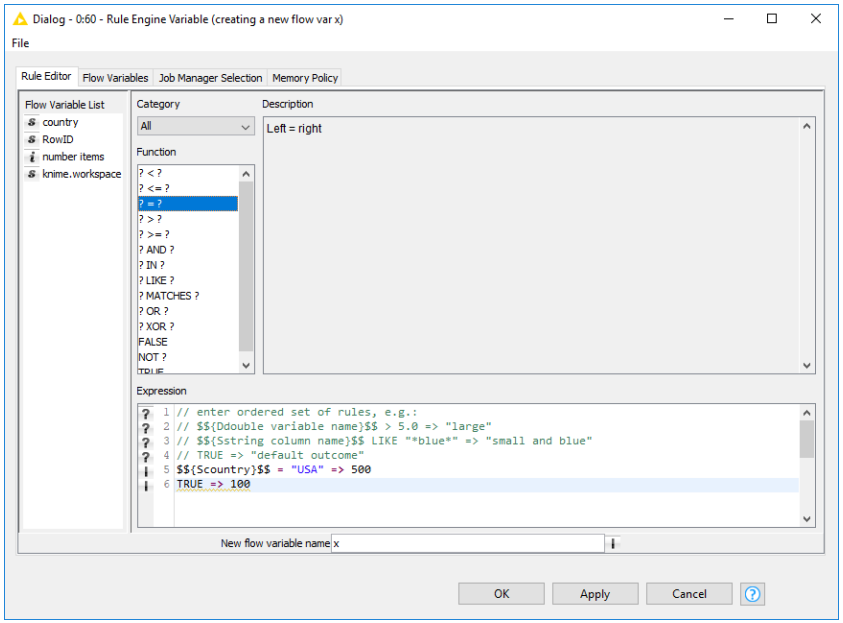
A "Rule Engine Variable" and a "Java Edit Variable" both implement the task of creating the varying threshold x, that is equal to 500 if variable "country" value is "USA" and to 100 otherwise.



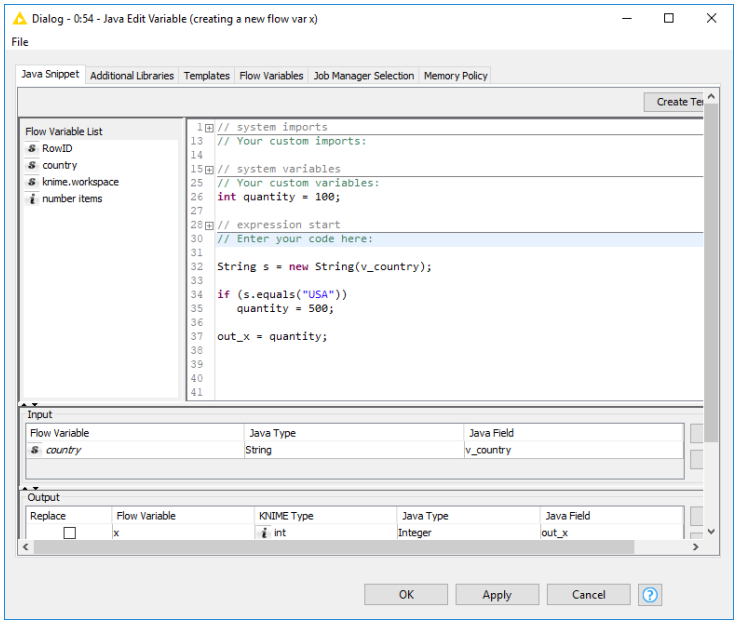
This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

A “String Manipulation Variable” node transforms the content of flow variable “country” from just the country name to “Country: <country name>”. At the same time a “Math Formula Variable” node doubles the value of flow variable “number items”.

5.23. Configuration window of the "Rule Engine Variable" node to create a flow variable named "x" with value 500 if flow variable "country" is set to "USA" and 100 otherwise.




5.24. Configuration window of the "Java Edit Variable" node to create a flow variable named "x" with value 500 if flow variable "country" is set to "USA" and 100 otherwise.



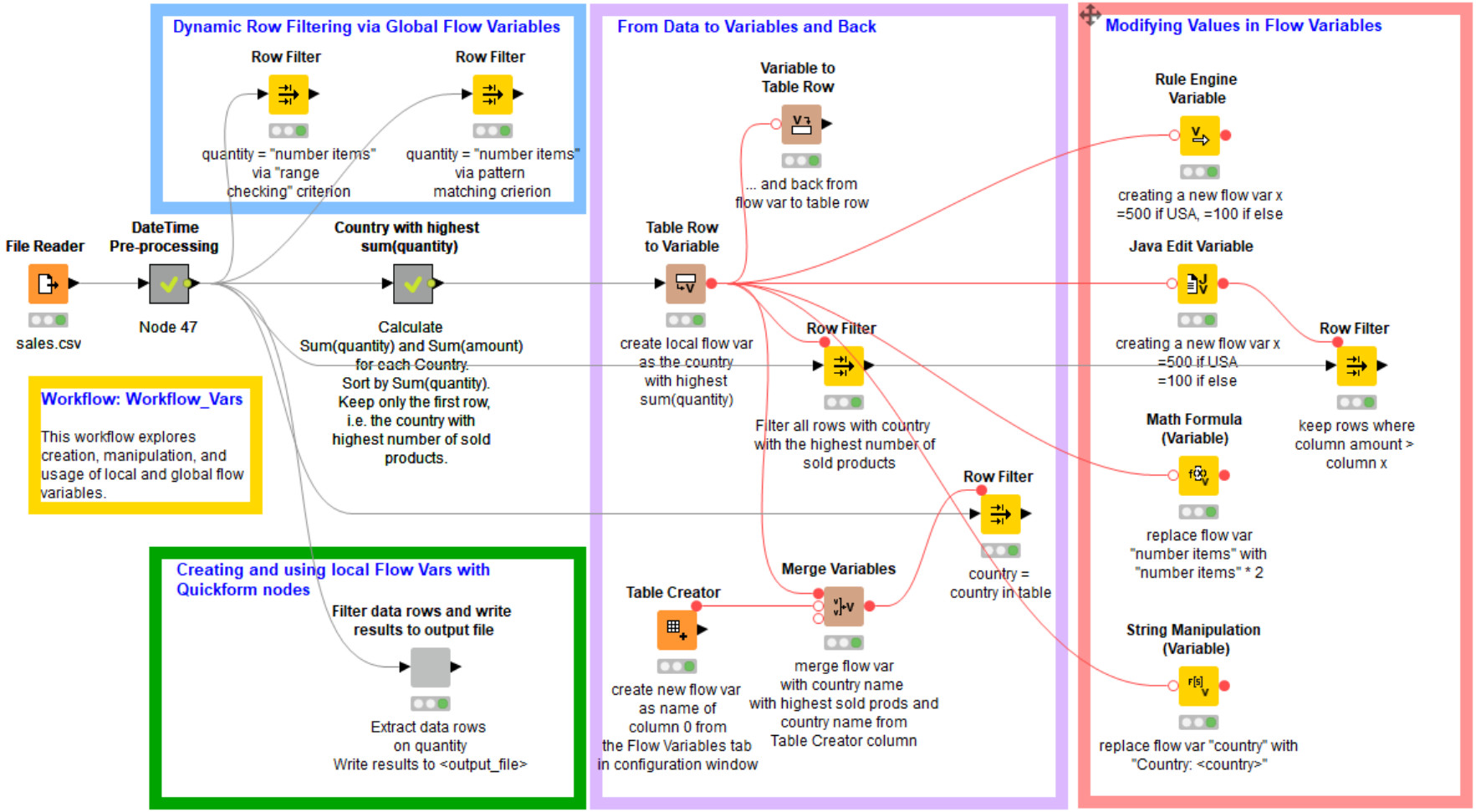

**Note**. The type of the returned flow variable has to be consistent with the return type of the Java/Rule Engine/Math Formula/String Manipulation code.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

**Note**. Even though all these "… Variable" nodes have an input port for flow variables, this port is often optional. That is, the nodes can also work without input, which makes them ideal nodes to start a workflow without reading data.

5.25. The final workflow in "Chapter5/Flow_Vars"

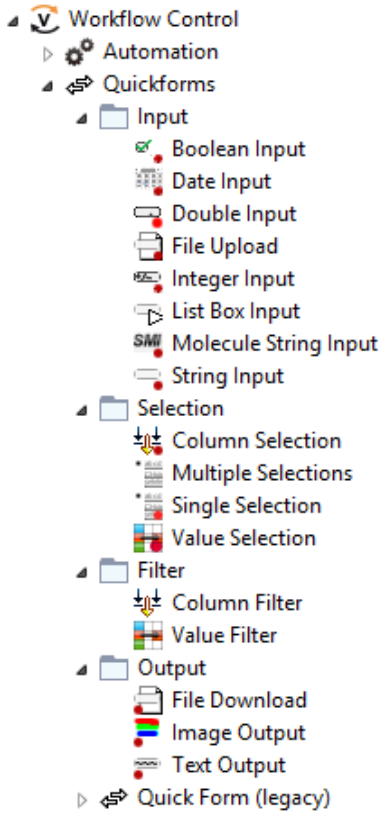





This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

## 5.9. Other Quickform Nodes

5.26. The "Quickforms" category




In the remaining part of this chapter, we would like to explore some of the many options offered by the nodes in the "Quickforms" category and the possibility to build composite, complex, and organized views in wrapped metanodes. In this section we concentrate on the nodes in the "Quickforms" category.

In the previous sections of this chapter, we have encountered already two Quickforms nodes: the "Integer Input" node and the "String Input" node. Like all Quickforms nodes, these two nodes provide an input form to enter the flow variable value. A wrapped metanode containing such nodes would then include their input forms into its configuration window. The same wrapped metanode executing on a KNIME WebPortal generates a web page displaying the same input forms.

The Integer Input and String Input node are probably the Quickforms nodes with the simplest input forms. The "Quickforms" category offers a wide choice of similar nodes, with more complex and flexible input forms.

In this section, we will explore two interesting and widely used Quickforms nodes: the "File Upload" node and the "Value Selection" node.

To demonstrate the usage of these two nodes, we have created a new workflow in folder "Chapter5", named "Composite_Views". Instead of reading the dataset directly from the file "sales.csv" in a File Reader node, we upload it from wherever it is located to read it into the workflow. For this we use a "File Upload" node. Then, we select the data for just one country using a "Value Selection" node. The "File Upload" node and the "Value Selection" node are contained in the wrapped metanode named "Select Country from list". The extracted data for the selected country are exported from the wrapped metanode.

The "File Upload" node is used to trigger the File Browser UI, to find, select, and import the file URL as a flow variable. The file URL variable is then passed to a File Reader node that reads and imports the data set from the file. Once the content of the file has been imported, we feed its content into the "Value Selection" node to select one of the possible countries available in column "country" (Fig. 5.29).

The configuration window of the wrapped metanode therefore displays a "Browse" button to search for file and a drop-down menu to select one of the countries available in the input table (Fig. 5.30).



This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

## Value Selection

The “Value Selection” Quickform node extracts a list of unique values from a column in the input data table and loads them in a menu or a group of radio buttons.

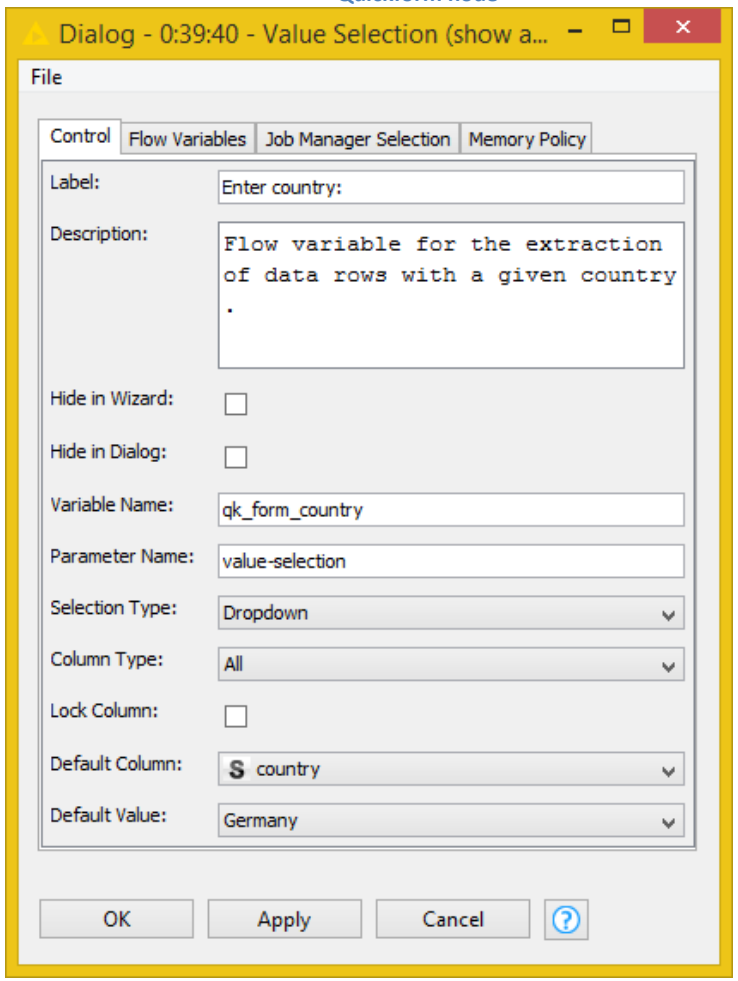
The menu or the radio buttons will be displayed in the configuration window of the node itself, in the meta-node where the node has been possibly included, and in the step-wise execution on the KNIME WebPortal. A value needs to be selected from the input form. Then a flow variable is created with the selected value and will pass it over to the subsequent part of the workflow. To configure the node we need at least:

- The type of input form, whether menu or radio buttons, to display in the node configuration window, metanode configuration window, and WebPortal.
- The name of the flow variable to create
- The name of the input column to be used to extract the list of unique values for the menu / radio buttons
- The selected column can be changed via the input form; to avoid that the “Lock Column” flag must be checked
- The default value to assign to the flow variable from the menu list; this value can be changed via the selected input form

Optionally, it would help to have a description of what this flow variable has been created for, and a label that explains what is supposed to be selected.

The usual flags to hide this form during WebPortal execution or from a possible wrapped node configuration window apply.

5.27. The configuration window of the “Value Selection” Quickform node




A parameter name is also required to assign a value to the flow variable from external applications.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

## File Upload

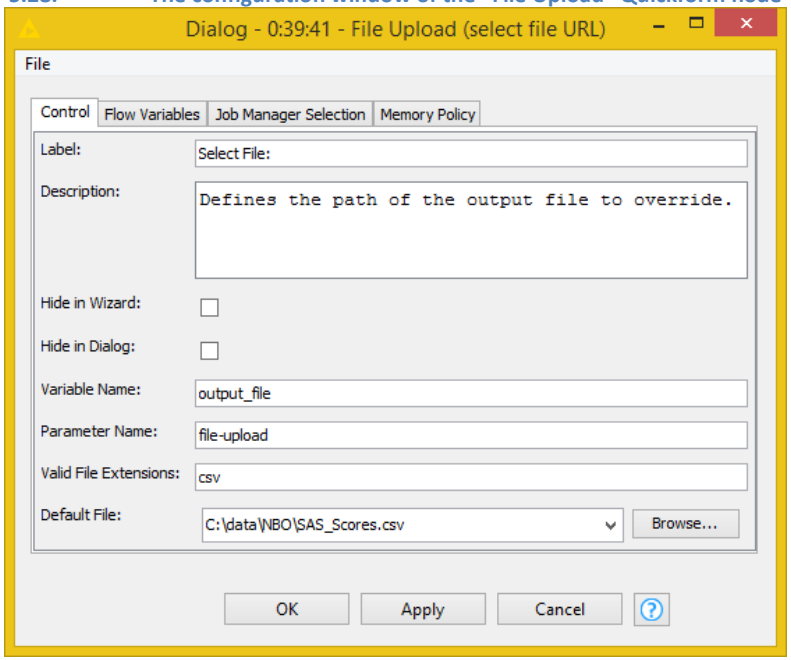
The “File Upload” node explores the folder of the default file path and loads all found files into a menu. The file path can be set using the GUI triggered by the “Browse” button.

The input form for this node then is the file search GUI triggered by the Browse button from the configuration window of the node itself, the configuration window of the meta-node where the node has been possibly included, and from the step-wise execution of the KNIME WebPortal.

When the file is selected from the browse GUI, a flow variable is created containing the file URL and passed over to the subsequent part of the workflow.

Usually this node is connected to a reader node, like a “File Reader” node or to writer node, like a “CSV Writer” node, for importing/exporting a data table from/to a file. To configure the node we need:

5.28. The configuration window of the “File Upload” Quickform node




- The name of the flow variable to create
- The default file path (it must be a valid path)

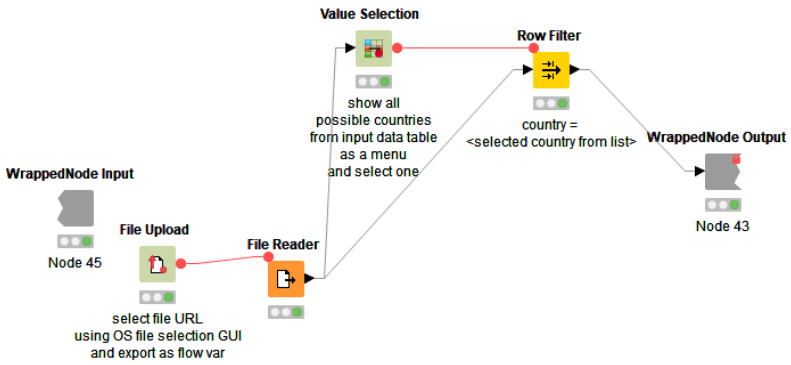
Optionally, the following information can help to display a more informative GUI form:

- The file extension to limit the number of files uploaded in the menu by their extension
- A description of what this flow variable has been created for
- A label that instructs on what to select
- The usual flags to hide this form during WebPortal execution or from a possible wrapped node configuration window
- A parameter name to assign a value to the flow variable from external applications

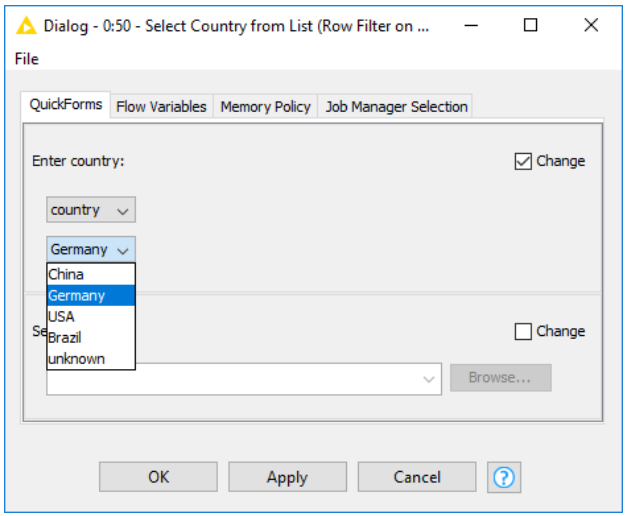


This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

5.29. Sub-workflow in wrapped meta-node named “Select Country from list” with the two Quickform nodes: a “File Upload” and a “Value Selection” Quickform node




5.30. Configuration window of meta-node “Select Country from list”. Notice the input forms (a file browser and a drop-down menu) coming form the internal Quickform nodes.




Above we have described two important Quickform nodes (“Value Selection” and “File Upload”). However, the “Quickform” category contains many other useful nodes. For example, the “Multiple Selection” and “Single Selection” Quickform node execution presents a list of values in a menu and allows for the selection of one or more of them. The “Value Filter” node works similarly to the “Value Selection” node, but returns a table with the one or more selected values instead of a flow variable. The “Column Selection” Quickform node lets the user select a data column and transmits the name of the selected column to the following nodes by means of a flow variable.

## 5.10. Composite View in Wrapped Metanodes

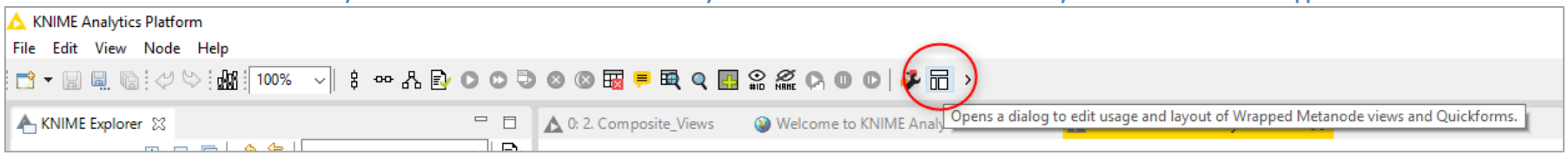
As we have stated already a few times in this chapter, wrapped metanodes include input forms from internal Quickform nodes in their configuration window. They also include views from internal Javascript nodes in their view window. Both input forms and views get displayed on the KNIME WebPortal during execution. So far, we have also seen that elements in the wrapped metanode’s view and configuration window are arranged ina vertical sequence. It does not have to be just vertical.



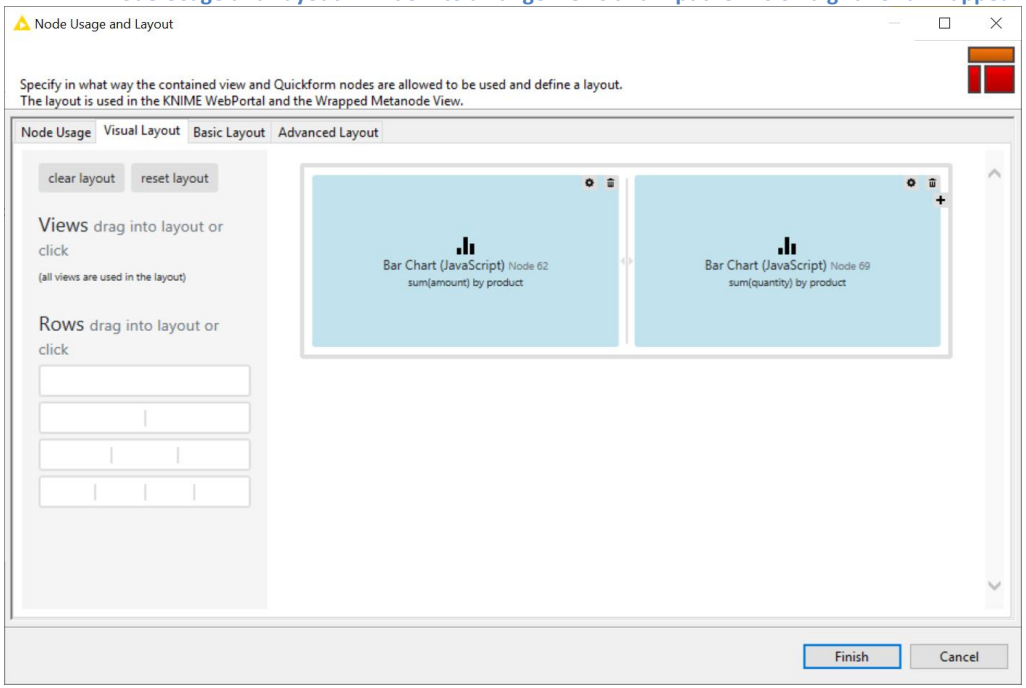
This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

If you are more artistically inclined and would like to place the metanode's items on a grid, you can do so using the layout button in the tool bar of the KNIME Analytics Platform workbench. Just open the wrapped metanode, by ctrl-doubleclick or by right-click -> "Wrapped Metanode" -> "Open" and click the layout button (Fig. 5.31). The "Node Usage and Layout" window opens. In this window, items are organized on a grid. Here you can assign the desired (x,y) coordinates to each one of the item to place it in the desired position of the view and configuration window.

5.31. The Layout button in the tool bar of KNIME Analytics Platform. This button is enabled only when we are inside a wrapped metanode.




5.32. "Node Usage and Layout" window to arrange views and input forms on a grid for a wrapped metanode




There are many ways to place items in the layout grid of a wrapped metanode via the four tabs of the "Node Usage and Layout" window.

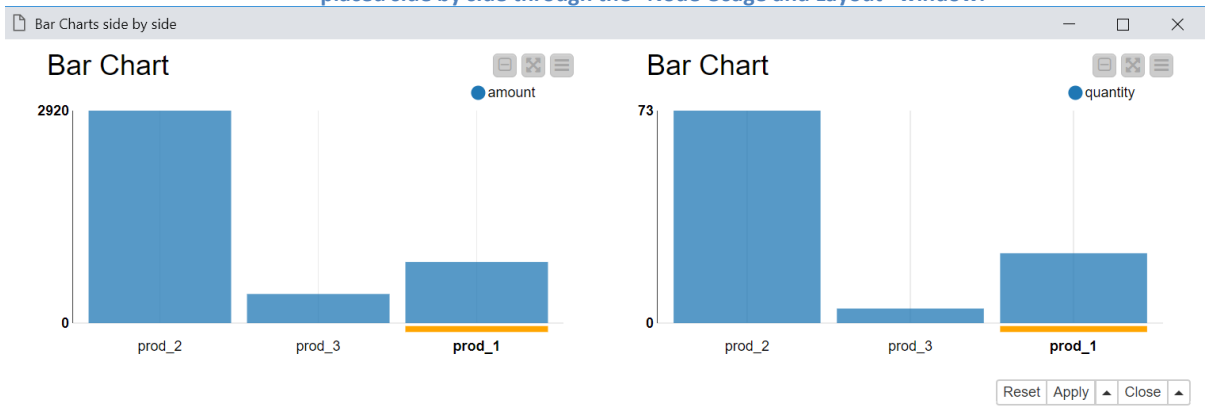


This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

- "Node Usage" sets whether the view or input form will be available in the wrapped metanode configuration window (dialog) and/or in the metanode and WebPortal View.
- "Visual Layout" provides a drag and drop layout solution (Fig. 5.32). On the left, layout subgrids on a row are available: 1x1, 2x1, 3x1, or 4x1. Clicking on one of these items, automatically adds them to the layout. By drag&drop, you can then place your views or Quickforms in any of the cells in the layout grid. Notice the trash bin button in the top right corner of each cell to empty it of its content.
- "Basic Layout" lets you place the items on the view grid or hide them. For example, if you have two items and you want to place them side by side, then you should use row=1, column=1 for the first item and row=1, column=2 for the second item. The width of each item is expressed in inches and refers only to the default size of the view window. The item size changes anyway, when resizing the view.
- "Advanced Layout" lets you place the items on the same view grid, but via a JSON structure.

In the workflow "Composite_Views", the wrapped metanode named "Bar Charts side by side" contains two bar charts built on the data of the file sales.csv. The two bar charts are built with two "Bar Chart (Javascript)" nodes summing for each product once the number of sold items ("quantity") and once the money amount ("amount"). We wanted to have the two bar charts displayed side by side in the metanode view. Thus, in the "Node Usage and Layout" window we placed the first chart at (1,1) and second chart at (1,2), giving to both charts an equal size 6.

5.33. View of the metanode named "Bar Chart side by side" containing two bar charts generated by two "Bar Chart (Javascript)" nodes. The two bar charts have been placed side by side through the "Node Usage and Layout" window.




Notice that all views produced by JavaScript based nodes are connected. Selecting groups/points/data rows in one view, automatically selects the same groups/points/data rows in the other view, if so set in the configuration window and/or in the interactive menu of the view. In the figure above, selecting the group of data for one product in one chart automatically selects the same group of data in the other chart.



This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

The last mention should go to a group of special nodes that offers control functions for charts and plots in a composite view. We will show here just one for all the nodes of this type: the “Range Slider Filter Definition” node.

The “Range Slider Filter Definition” node operates on numerical columns. It defines the range allowed for the numbers in the column represented by the upcoming Javascript nodes. It also produces a visual Javascript item in the form of a slider. In the view, changing the numerical range set in the slider, affects the number of records displayed in the connected charts and plots.

## Range Slider Filter Definition

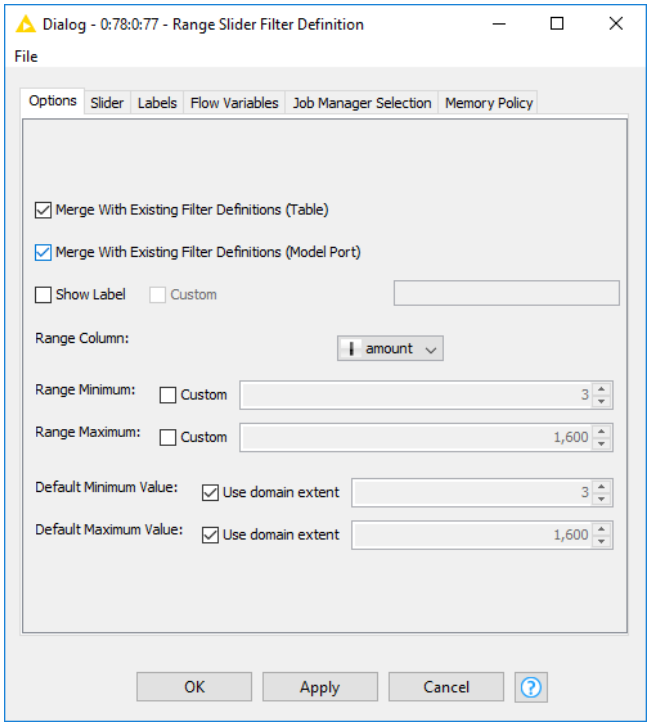
The “Range Slider Filter Definition” node defines the range of some numerical values to be plotted by a subsequent Javascript node.

In the view it produces a slider item. Changing the range in this slider, changes the values displayed in the connected chart or plot.

The configuration window requires:

- The numerical column to apply the range to.
- The minimum and maximum value for the full range. Default values are inherited from the column domain.
- The minimum and maximum value for the range defined by the slider handle. Again, default values are inherited from the column domain.
- If you are using a cascade of slider nodes, you can preserve the previous settings and build on top of them, by enabling the two merge checkboxes at the top. You can keep previous filters either in the result table or in the model or both.
- Optionally, an explanatory label can be added to clarify the range slider operation.

5.34. The configuration window of the “Range Slider Filter Definition” node



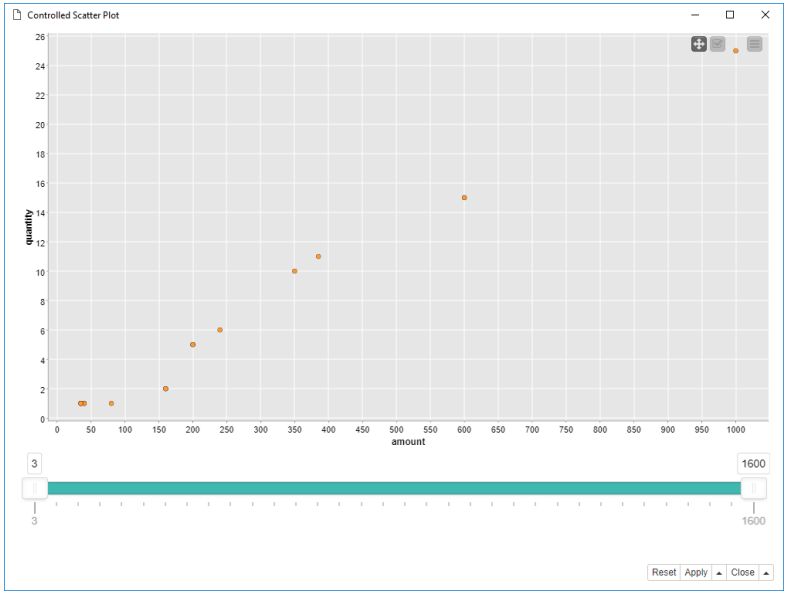

In our example workflow, named “Composite_Views”, we created a wrapped metanode, named “Controlled Scatter Plot”, to contain a “Scatter Plot Javascript” node and a “Range Slider Filter Definition” node. The scatter plot was set to display the sales input data in the (“amount”, “quantity”) space.



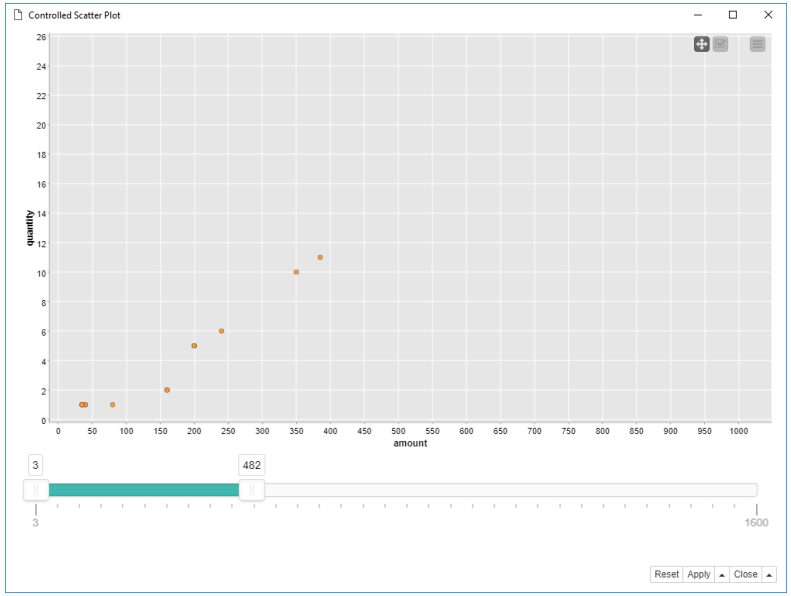
This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

The slider was hooked to column “amount” to filter the displayed data points in the plot. Figure 5.35 shows the view of the composite node, with all data points plotted in the scatter plot. Figure 5.36 shows the same metanode view, with only sales with “amount” lower than 482 plotted in the scatter plot. This last range [3-482] has been manually defined by moving the slider handle in the metanode composite view.

5.35. The configuration window of the “Range Slider Filter Definition” node




5.36. The configuration window of the “Range Slider Filter Definition” node



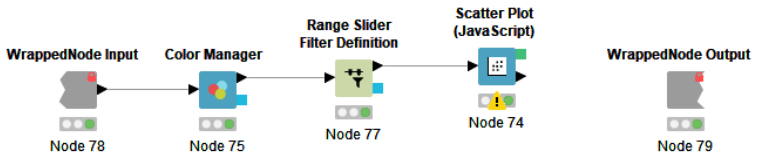

Sub-workflow contained in wrapped metanode “Controlled Scatter Plot” is shown in figure 5.37.

We conclude here this chapter with a picture of the final workflow “Composite_Views”, showing the wrapped metanode “Select Country from List” including “File Upload” and “Value Selection” Quickform nodes; the wrapped metanode “Bar Charts side by side” with the two bar charts placed close to each other; the metanode “Controlled Scatter Plot” displaying a scatter plot controlled by a slider item.

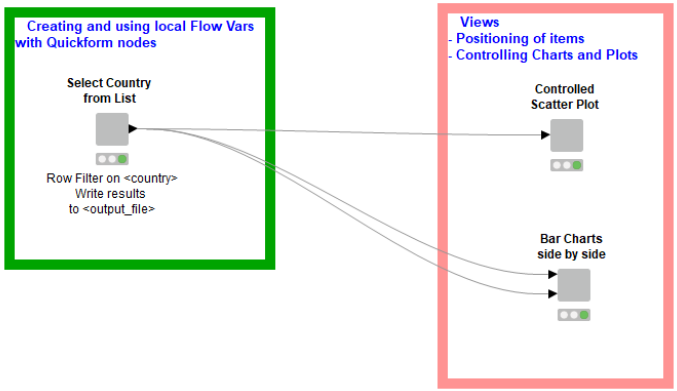


This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

5.37. Sub-workflow in wrapped metanode "Controlled Scatter Plot". Notice the "Range Slider Definition" node feeding (and controlling) the data in the "Scatter Plot (JavaScript)" node.




5.38. Final workflow "Composite_Views"




## 5.11. Exercises

### Exercise 1

Read the file sales.csv from the KCBData folder.

Select sale records for a specific month and year, like for example March 2009, March 2010, January 2009, January 2010 using flow variables for the month and the year values.

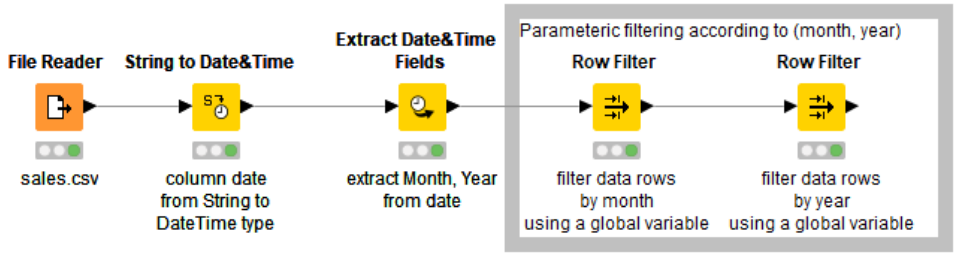
**Solution to Exercise 1**

To implement an arbitrary selection of data records based on month and year, we applied a "Date Field Extractor" node and defined two flow variables: one for the year and for the month. Two "Row Filter" nodes were then extracting the data rows with that particular month and that particular year as indicated by the two workflow variables' values.

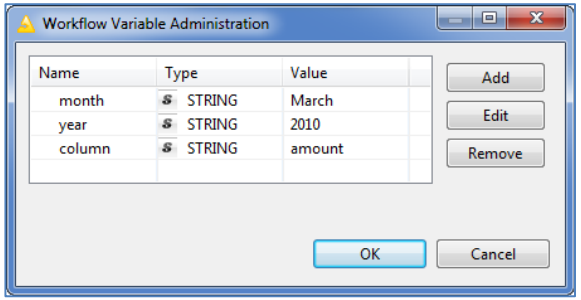


This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

5.39. Exercise 1: Workflow




5.40. Exercise 1: Global Workflow Variables




## Exercise 2

Using file “cars-85.csv”, build a data set with:

- The cars from the car manufacturer that is **most** present in the original data
- The cars from the car manufacturer that is **least** present in the original data

Put your name and today’s date on every row of the new data set, to identify author and time of creation.

**Solution to Exercise 2**

To identify the most/least represented car manufacturer in the data, we counted the data rows for each value in the “make” column with a “GroupBy” node; we sorted the aggregated rows by the count value in descending/ascending order; we kept only the first row and only the “make” column; we transformed this single value in the “make” column into a flow variable named “make”; and finally we used a “Row Filter” to keep only the rows with that value in the “make” column.

We then defined a flow variable, named “user”, for the whole workflow to hold the username.

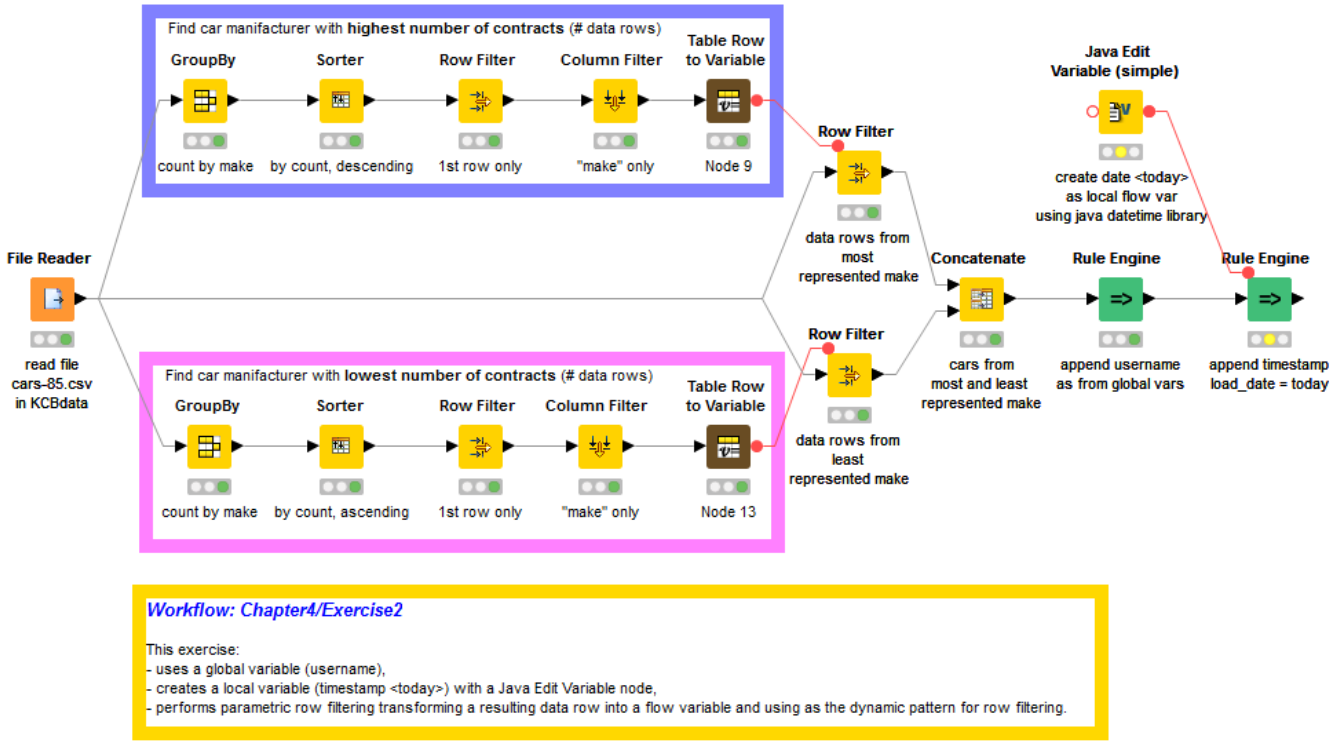
After concatenation of the two data sets, a “Rule Engine” node writes the username from the flow variable “user” in a column “user”.

We used a “Java Edit Variable” node to create today’s date and to append it to the data set in a column called “load_date” by means of a second “Rule Engine” node.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

## 5.41. Exercise 2: Workflow

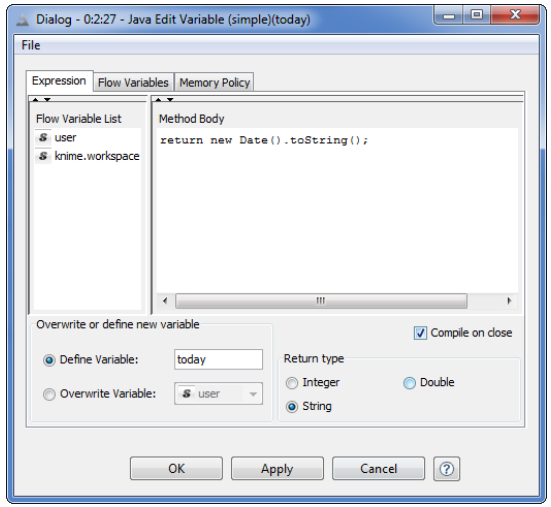



**Note**. A "Java Snippet" node could also write today's date in the "load_date" column. However, if you have more than one data set to timestamp, it might be more convenient to use only one "Java Editor Variable" node rather than many "Java Snippet" nodes.

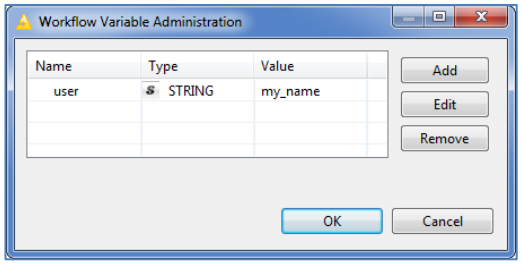


This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

5.42. Exercise 2: Configuration window of the “Java Edit Variable (simple)” node




5.43. Exercise 2: Global flow Variables “user”




## Exercise 3

Define a maximum engine size, for example 300, and put it into a flow variable named “max”.

Read file “cars-85.csv” and build two data sets respectively with:

- All cars with “engine_size” between “max” and “max”/2
- All cars with “engine_size” between “max”/2 and “max”/4

The workflow must run with different values of the maximum engine size.

**Solution to Exercise 3**

Define “max” as an integer global flow variable, with value 300.

Create flow variables “max2” with value “max”/2  and “max4” with value “max”/4 with two “Math Formula (Variable)” nodes.

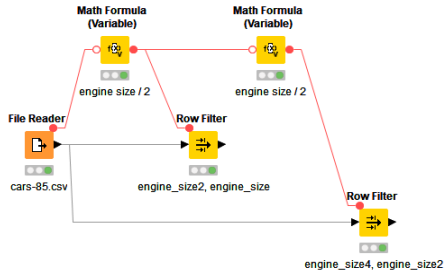


This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

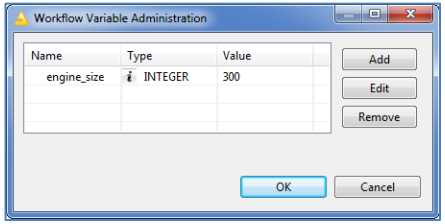
Configure two “Row Filter” nodes with “Use Range Checking” as filter criterion and with the appropriate flow variables in lower bound and upper bound to build the required data tables.

**Note**. Remember to check the “Convert to Int” flag in the “Math Formula (Variable)” nodes. Column “engine_size” is of type Integrer and then the “Row Filter” node sees only flow variables of type Integer.

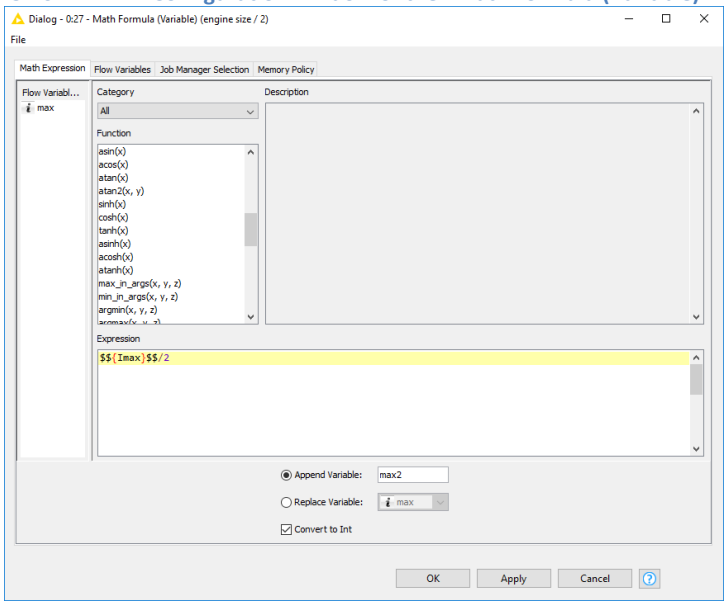
### 5.44. Exercise 3: Workflow




### 5.45. Exercise 3: Global Workflow Variables




### 5.46. Configuration window of the “Math Formula (Variable)” node






This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

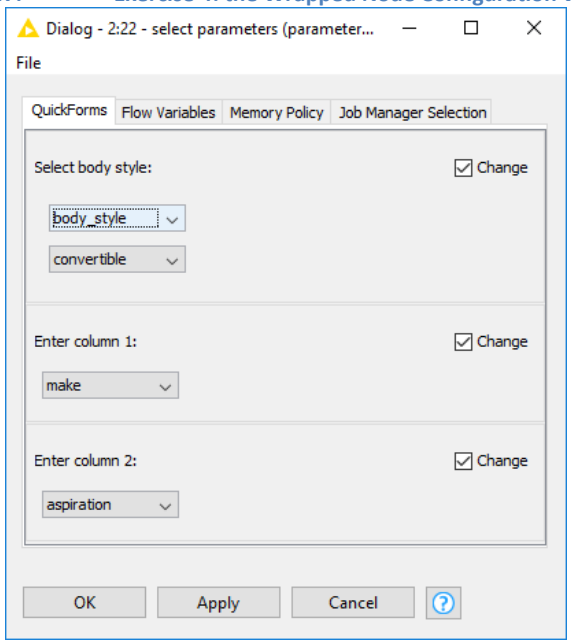
## Exercise 4

Using the “cars-85.csv” file, build a wrapped metanode with the following configuration window to select data rows with a specific body style and to replace missing values in column 1 with the values in the same data row in column 2.

Limit the choice of columns to only String columns.

For the merging operation use the “Column Merger” node.

5.47. Exercise 4: the Wrapped Node Configuration Window




**Solution to Exercise 4**

First, we read the data and we extract all String columns with a “Column Filter” node. Then we define all the parameters for the configuration window with Quickform nodes:

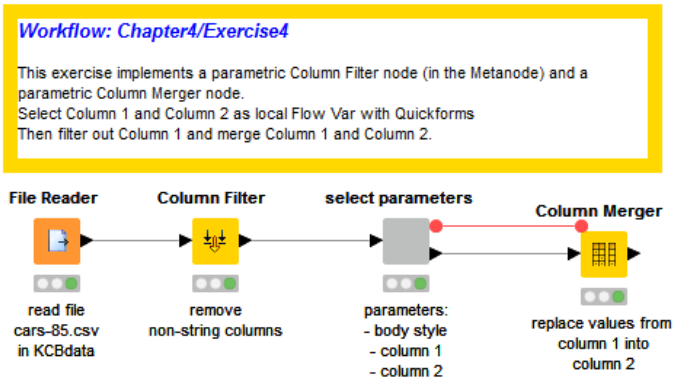
- With a “Value Selection” Quickform node applied to the “body style” column, we select the records of cars with a particular body style;
- Using a “Column Selection” Quickform node we select the first column to feed the “Column Merger” node;



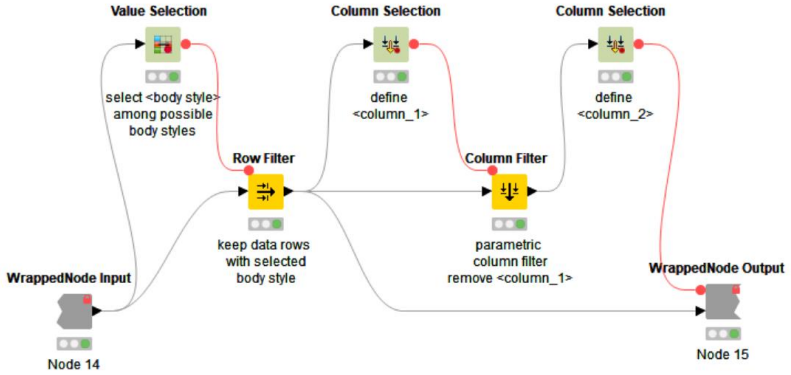
This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

- Using a second “Column Selection” Quickform node, we select the second column to feed the “Column Merger” node;
- We configure a “Column Merger” node using the variables defined for the first and second column.

5.48. Exercise 4: the Workflow




5.49. Exercise 4: The Sub-Workflow in the wrapped metanode






This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

# Chapter 6. Loops

## 6.1. What is a Loop

In the “Node Repository” there is a full category called “Workflow Control” for logical and control operations on the data flow. The “Workflow Control” category contains a number of sub-categories, like:

- “Automation” contains a number of utility nodes to improve the controlled execution and orchestration of workflows;
- “Quickforms” has nodes to enter variables and variable values through a GUI in wrapped nodes and on the WebPortal;
- “Variables” contains all those nodes that create, delete, and update flow variables (Chapter 4).
- “Loop Support” contains those nodes that allow the implementation of recursive operations; that is of loops;
- “Switches” contains the nodes that can switch the data processing flow one way or another;
- “Error Handling” nodes switch to an alternative workflow branch in case of error
- “Meta Nodes” contains a few pre-packaged commonly used loops, for example to inject variables or to read all files in a folder.

In this section we will work with the nodes in the “Loop Support” sub-category.

A loop is the recursive execution of one or more operations. That is, a loop keeps re-executing the same group of operations (the body of the loop) till some condition is met that ends the loop. A loop starts with some input data; processes the input data a number of times; each time adds the results to the output data table; and finally stops when the maximum number of times is reached or some other condition is met. A loop then has a start point, an end point to verify that the stopping condition is met, and a body of operations that is repeated. Each execution of the loop body is called a loop iteration. Loops do not work only on processing data, but also on flow variables. Similar loops can be constructed to export data or flow variables. The output data table at each iteration can appended as a column set or concatenated as a row set to the previous results.

A very commonly used loop iterates across a list of files - for example all files found in a folder with the “List Files” node - reads the content of each file, and piles up the results into the loop output data table. This of course works only if all files have a compatible data structure.

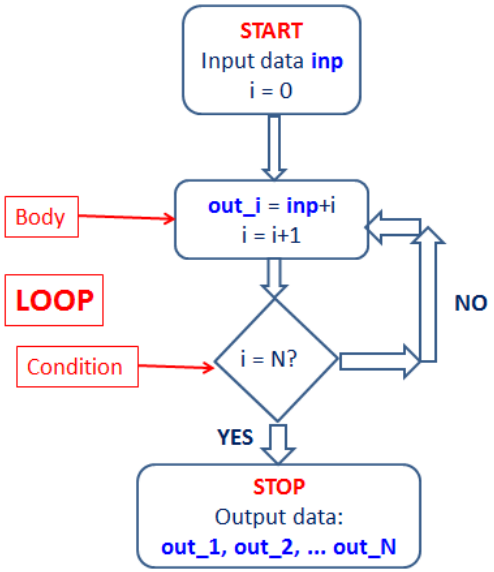


This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

Another example could be to grow an initial numerical value by the current iteration value for N iterations. If the initial numerical value is named “inp” and set to 5, at each loop iteration we add the value of the current iteration “i” - where “i”=1,2,3,4,5,…,N - till “i” = “N”. If “N” is set to 10, the output data table will contain values 5, 6, 7, 8, 9, 10, 11, 12, 13, and 14. In this example, the initial value “inp”=5 can be contained in a flow variable and the output values can be stacked or appended in a data table.

Finally, the classic loop example adds one unit to an integer variable N times. Starting from the variable initial value, we add 1 at each iteration to the result of the previous iteration. The loop stops when we have reached the maximum number of iterations N. The loop output in this case is just one value, i.e. the final sum and not a data table. This one output value could then be exported as a flow variable rather than as a data table. Notice that in this loop each iteration works with the result of the previous iteration. While this is common in programming languages it is less common in data analytics. In data analytics the goal is usually to transform the entire data set and less often to micro-transform the variables beneath. The KNIME Analytics Platform has only one loop for this kind of task: the recursive loop.

6.1. Loop Example




In KNIME Analytics Platform there are a number of nodes that can be used to implement loops: either loops on a pre-defined number of iterations or loops that stop when some condition is met. Loops in KNIME are always built with at least two nodes:

- The node to start the loop
- The node to end the loop
- A few additional but not necessary nodes to build the body of the loop

**Note**. Not all loops are in the “Workflow Control/Loop Support” category. You will find a few specialized loops here and there in other sub-categories, such as “Ensemble Learning”, “IO”, “Optimization”, and more. Just type “loop” in the search box on top of the “Node Repository” panel and see how many nodes show up that are implementing some kind of loop.



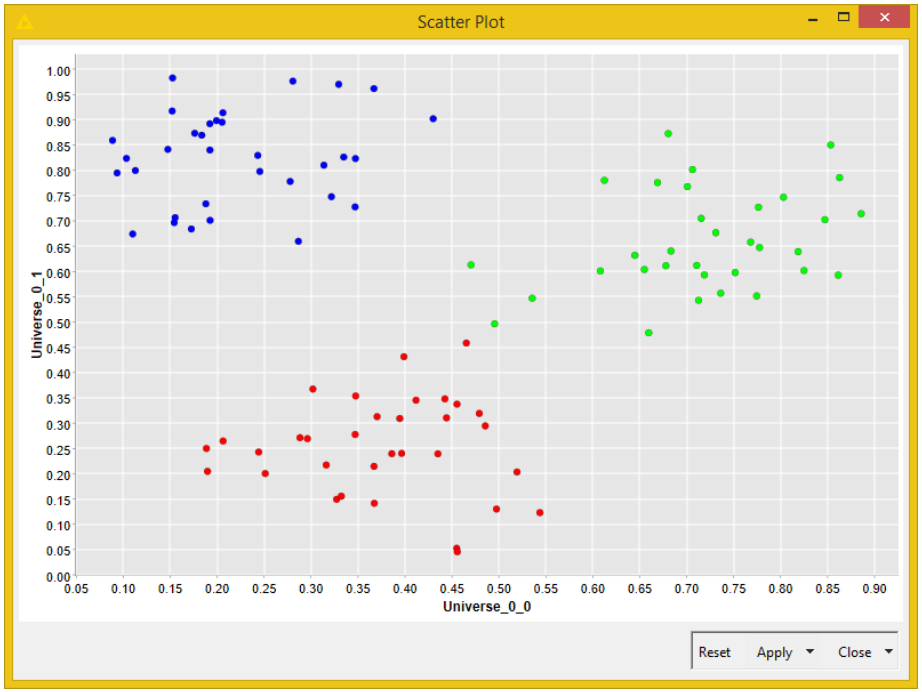
This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

## 6.2. Loop with a pre-defined number of iterations

The easiest loop to implement is the loop with a pre-defined number of iterations. That is, the loop where a group of operations is repeated a fixed number of times.

In this section we build an example workflow on a data set with 3 clusters distributed across two variables (Fig. 6.2). The goal is to produce a copy of this dataset shifted one unit to the right on the x-axis. Let's repeat this operation 4 times to obtain 4 copies of the original three clusters, each copy shifted one more unit to the right than the previous copy. To build the required 4 copies of the original data set, we introduce a loop with 4 iterations. Each iteration moves the original data set of "i" units to the right, where "i" is the number of the current iteration, and concatenates the resulting data with the data processed so far.

6.2. Data set with 3 clusters across two variables




In a new workflow group named "Chapter 6", we created a new empty workflow called "Counting Loop 1". First of all, the workflow needed to read in the original data set. Since we did not have such a data set available, we artificially created it by using the "Data Generator" node.



This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

## Data Generator

The “Data Generator” node generates artificial random data normalized in [0,1], grouped in equal size clusters, and with a fixed number of attributes. Data points for each cluster are generated following a Gaussian distribution with a given standard deviation and adding a certain fraction of random noise to the data.

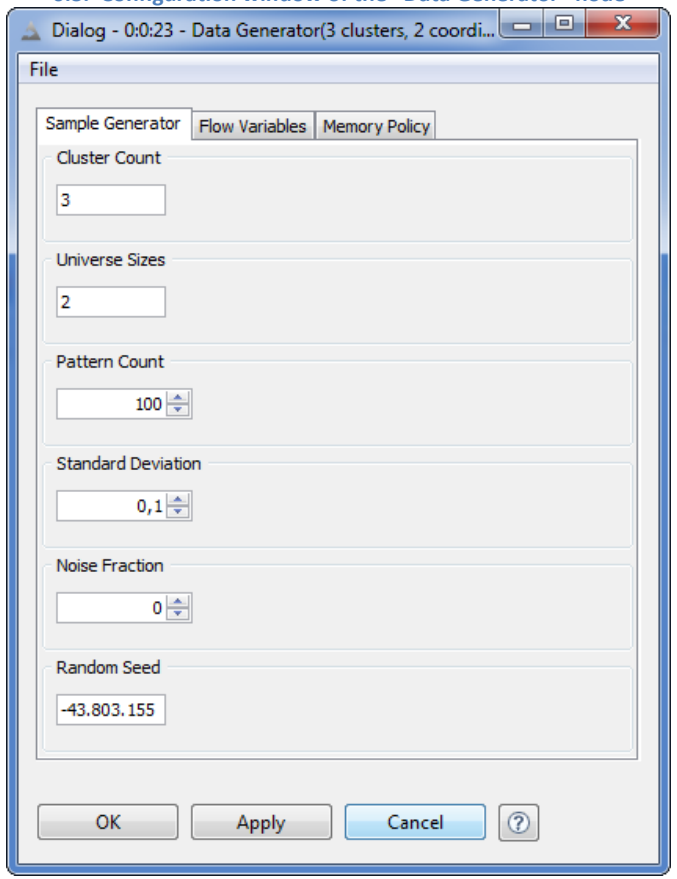
The “Data Generator” node is located in the “IO” -> “Other” category. In order to perform its task, the “Data Generator” node needs the following settings:

- The number of clusters to be created
- The number of space coordinates (i.e. “Universe Sizes”)
- The total number of data rows (i.e. “Pattern Count”). An equal number of patterns (= number of data rows/number of clusters) is subsequently generated for each cluster.
- The standard deviation used to generate data in each cluster. Notice that it is the same standard deviation value for all clusters.
- The noise fraction to apply to the generated data.
- A random seed to make this random pattern generation reproducible

The “Data Generator” node offers two output ports:

- One port with the generated data rows, each with its own cluster ID
- One port with the cluster centers in the space coordinates

6.3. Configuration window of the “Data Generator” node




**Note**. Data can be generated in more than one universe, each universe with a different number of clusters and a different number of coordinates.

In case of more than one universe to be created, the cluster counts and the universe sizes are set as a sequence of comma separated numbers. For example, 2 universes with 3 and 4 clusters and 2 and 3 coordinates each are described by “Cluster Count” = “3,4” and “Universe Sizes”=“2,3”.

We then started the new workflow “Counting Loop 1” with a “Data Generator” node to generate one single universe with 100 data rows equally distributed across 3 clusters, on a 2 attribute space, with a standard deviation 0.1 for each cluster, and no noise added. The node was named “3 clusters,



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

2 coordinates". The generated random data and their cluster centers are reported in the figures below. In order to visualize the data and their clusters, we added a "Color Manager" node and a "Scatter Plot (Javascript)" node into the workflow. The data visualization performed with the "Scatter Plot (Javascript)" node on a two-coordinate space - "Universe_0_0" and Universe_0_1" - is shown in figure 6.2.

6.4. Generated random data for workflow „Counting Loop 1"

Random data with cluster ID - 2:23 - Data Generator(3 clusters...

File

Table "default" - Rows: 99 | Spec - Columns: 3 | Properties | Flow Variables

| Row ID | Universe_0_0 | Universe_0_1 | Cluster Membership |
|---|---|---|---|
| Row0 | 0.296 | 0.269 | Cluster_0 |
| Row1 | 0.206 | 0.265 | Cluster_0 |
| Row2 | 0.367 | 0.142 | Cluster_0 |
| Row3 | 0.19 | 0.205 | Cluster_0 |
| Row4 | 0.327 | 0.149 | Cluster_0 |
| Row5 | 0.367 | 0.215 | Cluster_0 |
| Row6 | 0.288 | 0.271 | Cluster_0 |
| Row7 | 0.316 | 0.217 | Cluster_0 |
| Row8 | 0.332 | 0.156 | Cluster_0 |
| Row9 | 0.442 | 0.348 | Cluster_0 |
| Row10 | 0.412 | 0.345 | Cluster_0 |
| Row11 | 0.386 | 0.24 | Cluster_0 |
| Row12 | 0.251 | 0.2 | Cluster_0 |

6.5. Cluster centers of the generated random data in workflow "Counting Loop 1"

Cluster centers - 2:23 - Data Generator(3 cl...

File

Spec - Columns: 2 | Properties | Flow Variables

Table "default" - Rows: 3

| Row ID | Universe_0_0 | Universe_0_1 |
|---|---|---|
| Cluster_0 | 0.376 | 0.249 |
| Cluster_1 | 0.734 | 0.682 |
| Cluster_2 | 0.227 | 0.834 |

The goal was to generate 4 copies of the original data set and shift each copy one unit to the right. That is, the first copied data set should be identical to the original data set inside [0,1] x [0,1]; the second copied data set should be like to the original data set but inside [1,2] x [0,1], and so on. We then needed a loop cycle to shift the data 4 times, each time to a different region. In particular, since we already knew how many times the copy had to be made, we used a loop with a pre-defined number of iterations, i.e. 4. To move the original data set one unit to the right 4 times, the loop was to add the iteration number to the data set x-values at each iteration. Thus:

- **iteration 0:** x = x+**0** -> the data set is the exact copy of the original data set
- **iteration 1:** x = x+**1** -> the data set is moved one unit to the right
- **iteration 2:** x = x+**2** -> the data set is moved two units to the right
- **iteration 3:** x = x+**3** -> the data set is moved three units to the right



This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

A loop with a pre-defined number of iterations starts with a “Counting Loop Start” node. The most generic loop ending node is the one named “Loop End”.

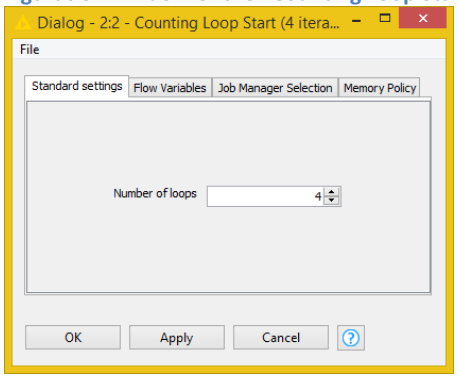
## Counting Loop Start

The “Counting Loop Start” node starts a loop with a pre-defined number of iterations.

That is, it starts a cycle where the operations, between the “Counting Loop Start” node and the end loop node, are repeated a pre-defined number of times.

The only setting required for such a loop start node is the number of pre-defined iterations (“Number of loops”).

6.6. Configuration window of the “Counting Loop Start” node

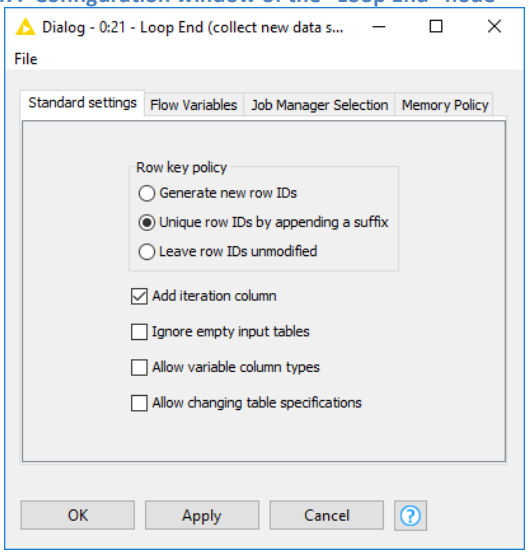



## Loop End

The “Loop End” node closes a loop and **concatenates** the resulting data tables from all loop iterations. It requires the following configuration settings:

- The row key policy; that is: keeping the old RowIDs, make the old RowIDs unique through suffix, or just generate new unique RowIDs.
- The flag specifying whether to append a column to the output data set with the iteration number. Iteration numbers start with 0.
- The flags needed for possible incongruent data across iterations: empty input tables, different output column types, and different table specs in general.

6.7. Configuration window of the “Loop End” node






This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

**Note**. The "Loop End" node **concatenates** the resulting data tables produced at each iteration. However, other loop end nodes can append the resulting data tables as additional columns, one iteration after the other.

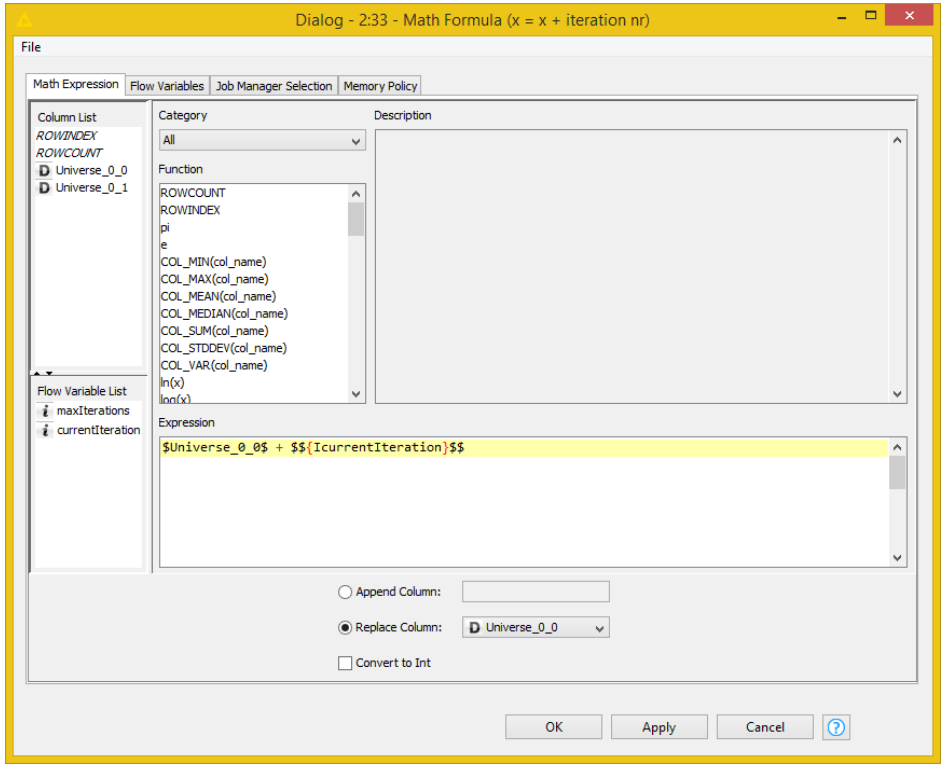
The "Counting Loop Start" node, while starting a loop, also creates two new flow variables: one, named "currentIteration", contains the current iteration number and the other, named "maxIterations", contains the total number of iterations.

So far, we have started and closed a loop with a pre-defined number of iterations, without any operations in between. We still need to introduce a loop body that moves each copy of the original data table one step further to the right on the x-axis. To do that, we can exploit the values in the "currentIteration" flow variable created by the "Counting Loop Start" node. Indeed, we used a "Math Formula" node where we added the $${IcurrentIteration}$$ flow variable to the $Universe_0_0$ attribute (the x-axis).

In the "Flow Variable List" panel on the bottom left, you can see the two new flow variables introduced by the "Counting Loop Start" node: "currentIteration" and "maxIterations". Double-clicking one of them automatically introduces it in the formula editor with the right syntax. The same for any column listed in the panel above, named "Column List".

The math formula $Universe_0_0$ + $${IcurrentIteration}$$ overwrites the Universe_0_0 coordinate, effectively moving each copy of the data to the right on the x-axis as many units as the iteration number.

6.8. Configuration window of the "Math Formula" node used to implement the loop body

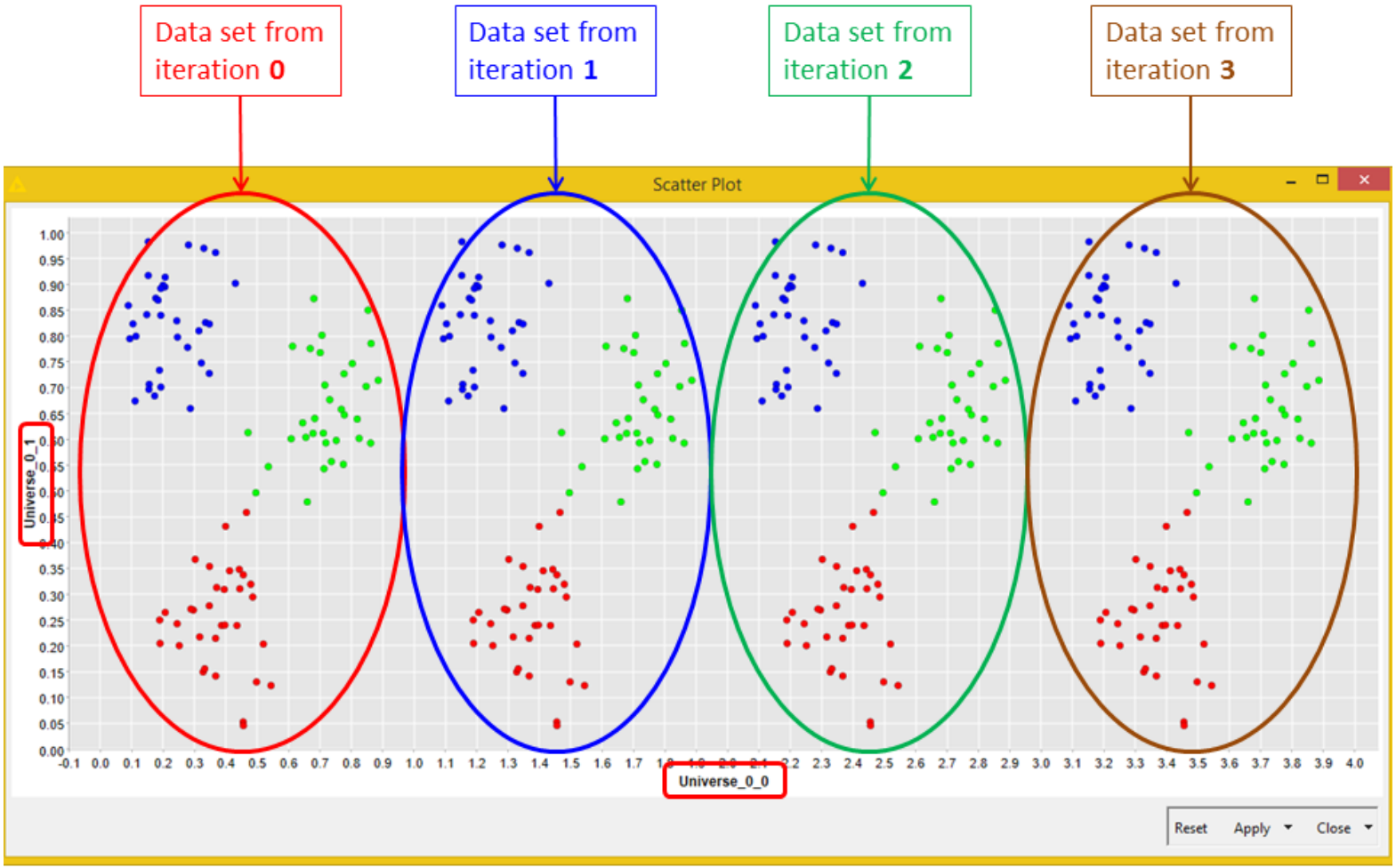



The final data set then consist of 4 identical vertical stripes of data equally spaced in [0, 4] on the x-axis. We used a "Scatter Plot (Javascript)" node again to verify the new data placement in the final data table.



This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

6.9. Visualization of the data set resulting from the loop implemented by in the “Counting Loop 1” workflow




Finally, figure 6.9 shows the “Counting Loop 1” workflow.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

## 6.3. Dedicated Commands for Loop Execution

The result in figure 6.9 can only be seen after the whole workflow has been executed. However, loops have a number of dedicated execution options. The table below shows most commands available from the context menu (right-click menu) of loop end and loop start nodes and their functionalities.

| **Loop End nodes** | **Loop Start nodes** |
|---|---|
| - "**Execute**" runs the whole loop with all required iterations. To execute the whole loop you need to select "Execute" in the loop end node<br>- "**Cancel**" stops the loop execution and resets the loop<br>- "**Reset**" resets all nodes in the loop<br>- "**Select Loop**" selects the whole loop: loop start, loop end, and loop body<br>- "**Pause Execution**" pauses the loop execution at the end of the current iteration and allows investigation of the intermediate results in the loop body.<br>- "**Step Loop Execution**" executes the next loop iteration and pauses at the end, allowing the visualization of the intermediate results in the loop body.<br>- "**Resume Loop Execution**" restarts the loop execution previously paused by the "Pause Execution" or the "Step Loop Execution" command. | - "**Execute**" only executes the loop start node, i.e. it only transfers the input data into the loop<br>- "**Cancel**" stops the loop execution and resets the loop<br>- "**Reset**" resets all nodes in the loop<br>- "**Select Loop**" selects the whole loop: loop start, loop end, and loop body |

To reset the loop, it is enough to select the "Reset" option in the context menu of any node in the loop, including the loop start, the loop end node, and any node in the loop body.

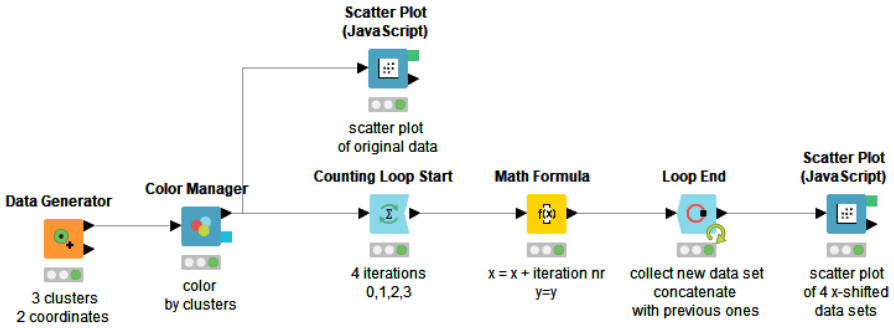


This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

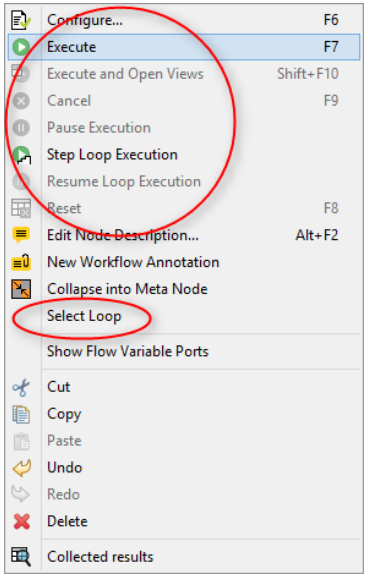
The last three options in the loop end nodes - Pause, Resume, and Step Loop Execution - make loop execution very flexible and easy to debug.

**Note**. It might be necessary, in some cases, to execute the loop nodes manually for the first time to be able to configure them properly.

6.10. The "Counting Loop 1" workflow




6.11. Context menu of the "Loop End" node.

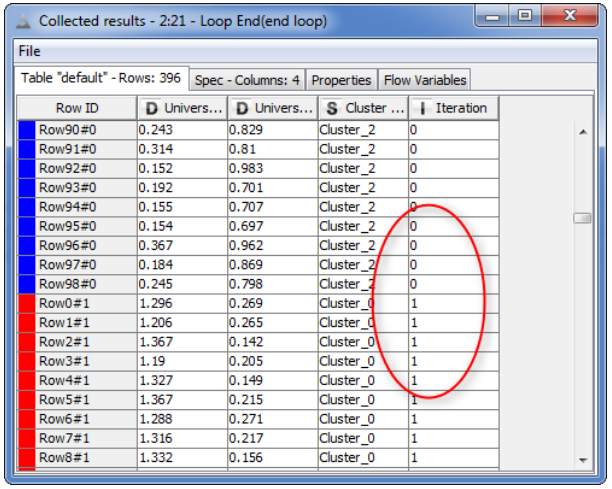





This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

## 6.4. Appending Columns to the Output Data Table

6.12. Output Table of the "Loop End" node in the "Counting Loop 1" workflow



Collected results - 2:21 - Loop End(end loop)

File

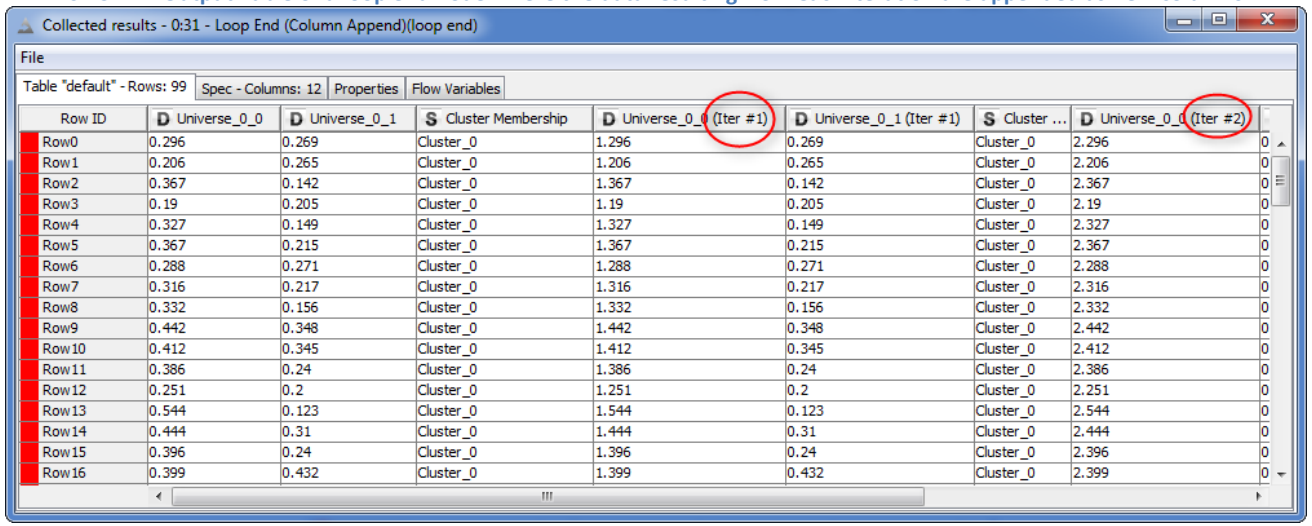
Table "default" - Rows: 396 | Spec - Columns: 4 | Properties | Flow Variables

| Row ID | D Univers... | D Univers... | S Cluster ... | i Iteration |
|---|---|---|---|---|
| Row90#0 | 0.243 | 0.829 | Cluster_2 | 0 |
| Row91#0 | 0.314 | 0.81 | Cluster_2 | 0 |
| Row92#0 | 0.152 | 0.983 | Cluster_2 | 0 |
| Row93#0 | 0.192 | 0.701 | Cluster_2 | 0 |
| Row94#0 | 0.155 | 0.707 | Cluster_2 | 0 |
| Row95#0 | 0.154 | 0.697 | Cluster_2 | 0 |
| Row96#0 | 0.367 | 0.962 | Cluster_2 | 0 |
| Row97#0 | 0.184 | 0.869 | Cluster_2 | 0 |
| Row98#0 | 0.245 | 0.798 | Cluster_2 | 0 |
| Row0#1 | 1.296 | 0.269 | Cluster_0 | 1 |
| Row1#1 | 1.206 | 0.265 | Cluster_0 | 1 |
| Row2#1 | 1.367 | 0.142 | Cluster_0 | 1 |
| Row3#1 | 1.19 | 0.205 | Cluster_0 | 1 |
| Row4#1 | 1.327 | 0.149 | Cluster_0 | 1 |
| Row5#1 | 1.367 | 0.215 | Cluster_0 | 1 |
| Row6#1 | 1.288 | 0.271 | Cluster_0 | 1 |
| Row7#1 | 1.316 | 0.217 | Cluster_0 | 1 |
| Row8#1 | 1.332 | 0.156 | Cluster_0 | 1 |

In section 6.2 we have seen that the “Loop End” node concatenates together the data tables produced at each loop iteration. A part of the output table of the “Loop End” node in the “Counting Loop 1” workflow is shown in figure 6.12. The last column of the data table, named “Iteration”, is created by the “Loop End” node (if so specified in the configuration window), and shows the iteration number during which each data row was created.

However, sometimes we might like to append the data table resulting from each iteration as new columns to the previous results. That is, at each iteration new data columns are generated and appended to the current output data table. Figure 6.13 shows an output data table where column “Cluster Membership (Iter #1)” contains the data generated from column “Cluster Membership” at the end of loop iteration number 1, and column “Universe_0_0 (Iter #2)” contains the data generated from column “Universe_0_0” at the end of loop iteration number 2, and so on.

6.13. Output Table of a loop end node where the data resulting from each iteration are appended as new columns



Collected results - 0:31 - Loop End (Column Append)(loop end)

File

Table "default" - Rows: 99 | Spec - Columns: 12 | Properties | Flow Variables

| Row ID | D Universe_0_0 | D Universe_0_1 | S Cluster Membership | D Universe_0_0 (Iter #1) | D Universe_0_1 (Iter #1) | S Cluster ... | D Universe_0_0 (Iter #2) | |
|---|---|---|---|---|---|---|---|---|
| Row0 | 0.296 | 0.269 | Cluster_0 | 1.296 | 0.269 | Cluster_0 | 2.296 | 0 |
| Row1 | 0.206 | 0.265 | Cluster_0 | 1.206 | 0.265 | Cluster_0 | 2.206 | 0 |
| Row2 | 0.367 | 0.142 | Cluster_0 | 1.367 | 0.142 | Cluster_0 | 2.367 | 0 |
| Row3 | 0.19 | 0.205 | Cluster_0 | 1.19 | 0.205 | Cluster_0 | 2.19 | 0 |
| Row4 | 0.327 | 0.149 | Cluster_0 | 1.327 | 0.149 | Cluster_0 | 2.327 | 0 |
| Row5 | 0.367 | 0.215 | Cluster_0 | 1.367 | 0.215 | Cluster_0 | 2.367 | 0 |
| Row6 | 0.288 | 0.271 | Cluster_0 | 1.288 | 0.271 | Cluster_0 | 2.288 | 0 |
| Row7 | 0.316 | 0.217 | Cluster_0 | 1.316 | 0.217 | Cluster_0 | 2.316 | 0 |
| Row8 | 0.332 | 0.156 | Cluster_0 | 1.332 | 0.156 | Cluster_0 | 2.332 | 0 |
| Row9 | 0.442 | 0.348 | Cluster_0 | 1.442 | 0.348 | Cluster_0 | 2.442 | 0 |
| Row10 | 0.412 | 0.345 | Cluster_0 | 1.412 | 0.345 | Cluster_0 | 2.412 | 0 |
| Row11 | 0.386 | 0.24 | Cluster_0 | 1.386 | 0.24 | Cluster_0 | 2.386 | 0 |
| Row12 | 0.251 | 0.2 | Cluster_0 | 1.251 | 0.2 | Cluster_0 | 2.251 | 0 |
| Row13 | 0.544 | 0.123 | Cluster_0 | 1.544 | 0.123 | Cluster_0 | 2.544 | 0 |
| Row14 | 0.444 | 0.31 | Cluster_0 | 1.444 | 0.31 | Cluster_0 | 2.444 | 0 |
| Row15 | 0.396 | 0.24 | Cluster_0 | 1.396 | 0.24 | Cluster_0 | 2.396 | 0 |
| Row16 | 0.399 | 0.432 | Cluster_0 | 1.399 | 0.432 | Cluster_0 | 2.399 | 0 |



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

In order to demonstrate how such a loop output data table has been produced, we slightly modified the workflow used in the previous section and renamed it “Counting Loop 2”. The “Counting Loop 2” workflow is in general identical to the “Counting Loop 1” workflow except for the choice of the loop end node. Here, instead of using the generic “Loop End” node, we used the “Loop End (Column Append)” node.

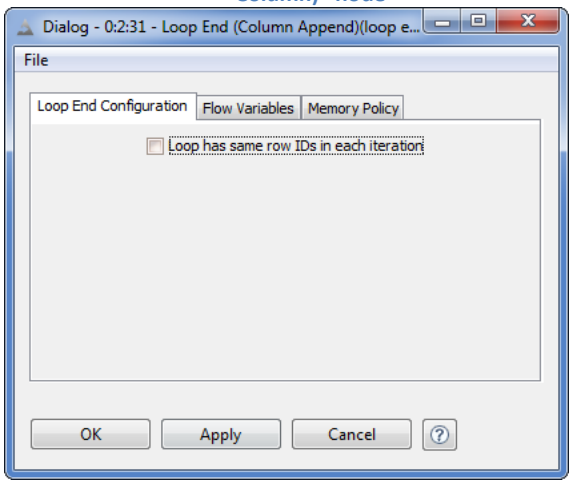
## Loop End (Column Append)

The “Loop End (Column Append)” node marks the end of a loop and collects the resulting data tables from all loop iterations.

At each iteration, the output data columns are appended as a set of new columns to the output data table collected so far.

This node requires minimal configuration, just a flag on whether keep the same RowIDs at each iteration. This flag, when enabled, speeds up the joining process of the new data columns to the table of the so far collected results.

6.14. Configuration window of the “Loop End (Append Column)” node




The “Loop End (Column Append)” node is the appropriate choice to close a loop, if the columns produced by the loop body are supposed to be appended to the output table.

By using the “Loop End (Column Append)” node to close the loop in the “Counting Loop 2” workflow, the data table in figure 6.13 is generated after the loop execution. Here an x-shifted copy of the original data set is generated at each iteration and then appended as a new set of columns to the current output data table.

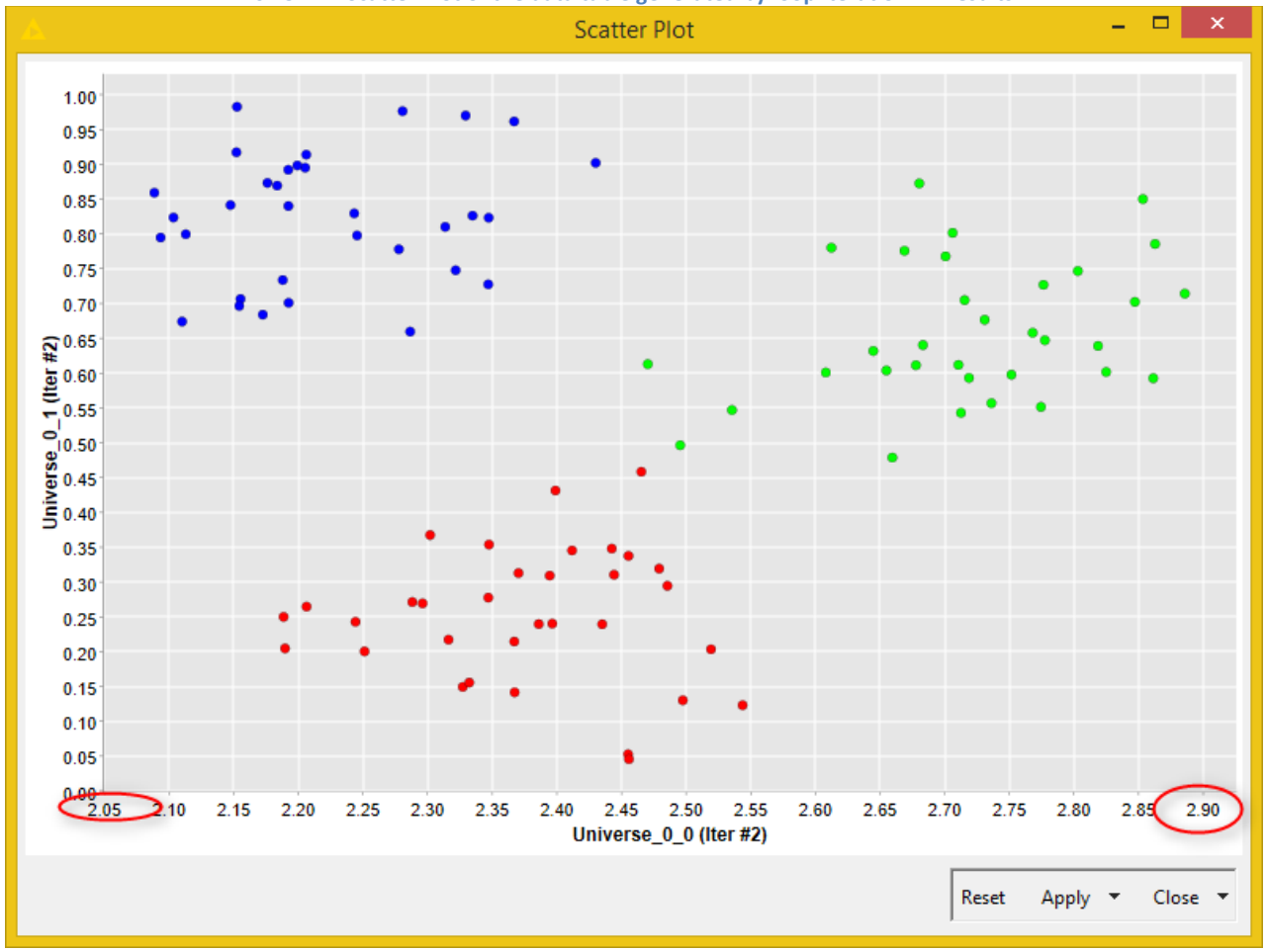
Notice that the “Scatter Plot (Javascript)” node at the end of the “Counting Loop 2” workflow can only visualize two coordinates at a time and therefore we cannot have the full visualization of the results of all loop iterations as in figure 6.9. Thus, in figure 6.15 we visualized “Universe_0_0 (iter #2)” vs. “Universe_0_1 (iter #2)” with a scatter plot. There you can see that the x-range falls in [2,3] and not in [0,1] like for the original data set.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

**Note**. The “Loop End (Column Append)” node appends all columns produced in the output table, not only the processed ones. Therefore the input columns, if not removed with a “Column Filter” node, will appear multiple times (as many times as many iterations) unaltered in the final output table.

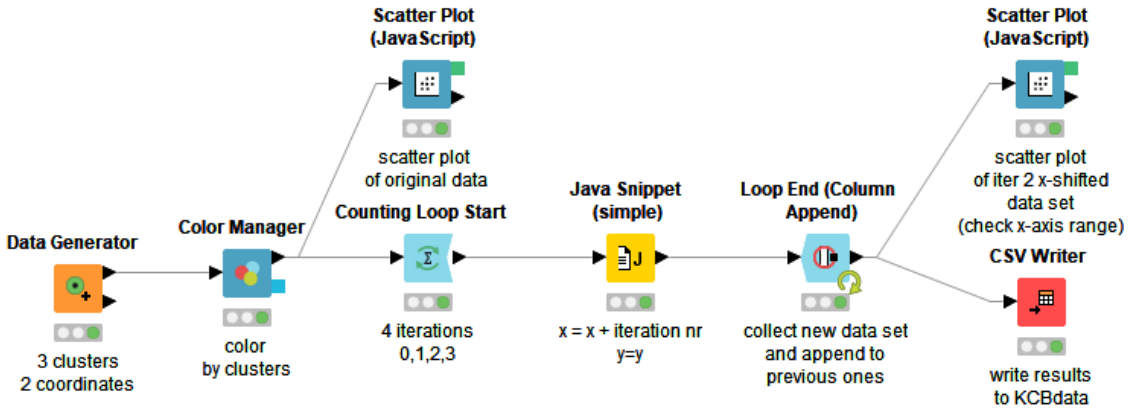
6.15. Scatter Plot of the data table generated by loop Iteration # 2 results






This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

6.16. Workflow "Counting Loop 2"




## 6.5. Loop on a List of Columns

Let's suppose now that we want to create two more shifted data sets, but shifted differently on the x-axis and on the y-axis, and that we also want to append the new columns to the resulting data set. We could, of course, modify the "Counting Loop 2" workflow to iterate only twice and, depending on the iteration number, apply custom x- and y-shifts to the data set. We could also use a loop that iterates on a list of selected columns.

A loop that iterates on a list of columns starts with a "Column List Loop Start" node. This kind of loop iterates across all columns, one by one, and runs the group of operations specified in the body loop for each column. The loop then can end with a "Loop End (Column Append)" node or just with a "Loop End" node, depending on how we would like to collect the results in our final data table.

In order to demonstrate how to implement a loop that iterates over a list of columns, we created a new workflow named "Loop on List of Columns".

This workflow reads the data set produced by the "Counting Loop 2" workflow, including the original data set and 3 copies of the same data set differently shifted across the x-axis. The goal here is to multiply the x-axis of each iteration set by a different numerical factor. How do we identify each iteration set? If you remember, the "Loop End (Column Append)" node was renaming columns with duplicate names by appending a "(Iter #n)" suffix to the column name. Thus data set columns from iteration n are identified by the suffix "(Iter #n)" in the column names.

The goal is to multiply the x ("Universe_0_0") and y ("Universe_0_1") value of each iteration set by the iteration number in the column name plus one. So, "Universe_0_0 (Iter #2)" and "Universe_0_1 (Iter #2)" should both be multiplied by 3, while "Universe_0_0" and "Universe_0_1" by 1 and so on. To



This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

define the correct multiplying factor, we need to work on each column name String and extract the number in the “Iter #” suffix. Once we have it, we need to multiply all values in that column by the associated numerical factor (= iteration number in column name +1).

## Column List Loop Start

The “Column List Loop Start” node iterates over a list of columns in the input table, one by one.

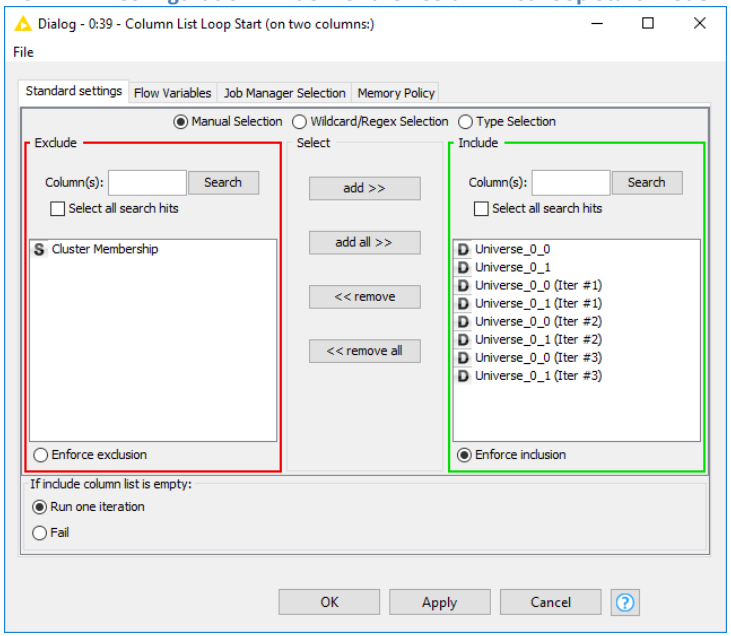
The columns on which to iterate are selected in the configuration window by means of an “Exclude/Include” framework. Columns can be included manually, through wildcard and regex selection, or based on type.

The “Column List Loop Start” node separates the columns in the input data table into iteration columns (“Include” panel) and non-iteration columns (“Exclude” panel).

At each iteration, the non-iteration columns are processed and appended to the resulting data table together with the current iteration column, while all other iteration columns are excluded from the results.

Additionally a strategy can be defined in case the iteration column is empty.

6.17. Configuration window of the “Column List Loop Start” node




We used a “Column List Loop Start” node to iterate on the different columns of the input data set. The configuration window includes all “Universe*” columns in the iteration group and excludes only the column named “Cluster_Membership”. Such inclusion could be performed manually (those are not that many columns) or using the wild card based selection. In case of the wildcard based selection, “Universe*” should be used as pattern with wildcard to match the column names. After execution, the “Column List Loop Start” node produces two flow variables: “currentIteration” with the current iteration number and “currentColumnName” with the name of the current column in the loop.

Our goal included the structure of the final table to be identical to the structure of the input table, only with different values in each columns. In order to append the data columns into the same position as in the original table, the loop was closed with a “Loop End (Column Append)” node.

The loop was built with two parallel branches.

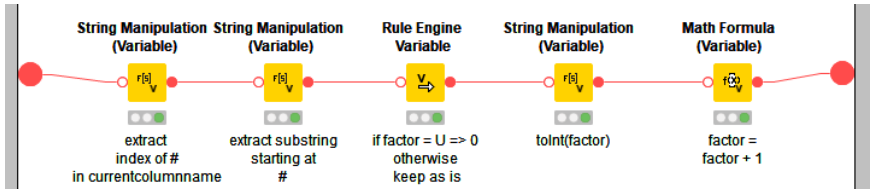


This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

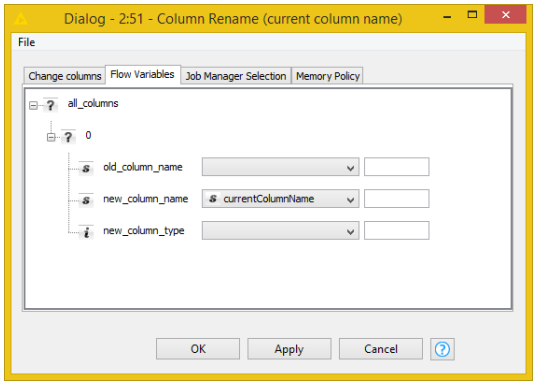
21. One branch defines the multiplying factor. That is, it checks the flow variable “currentColumnName” for the presence of “#”; it then extracts the character following the position of “#”; if “#” was not found, default multiplying factor is 0, otherwise is the number following “#”; the last step is to add +1 to the multiplying factor. All of this is obtained with three “String Manipulation” nodes, one “Math Formula (Variable)” node, and one “Rule Engine Variable” node. This whole part has been grouped under a metanode named “define factor”.

22. One branch eliminates “Cluster_Membership” column from the iteration set, renames the current column to an anonymous “temp_column”, applies the multiplying factor from the parallel metanode, and changes the column name back to its original one.
Renaming the current iteration column to “temp_column” is necessary to allow the “Math Formula” node to work on any input column independently of its name. The renaming process is carried out through a “Column Rename” node using the value in the flow variable “currentColumnName” to overwrite the “old_column_name” setting. The inverse renaming process is carried out again through a “Column Rename” node using the value in the flow variable “currentColumnName” to overwrite this time “new_column_name” in the configuration settings.
After the column is renamed, it is presented to the “Loop End (Column Append)” closing node, to be appended to the other already processed data columns.

**6.18. Content of metanode “define factor”. “String Manipulation (Variable)” node detects “#” in column name via the “indexOf()” function, it then extracts the number following “#” if any; if no number, “factor” is 0 otherwise is the detected number; finally the “Math Formula (Variable)” node adds +1 to flow variable “factor”.**




**6.19. Value of flow variable “currentColumnName” overwrites configuration setting “new_column_name” in the last “Column Rename” node**



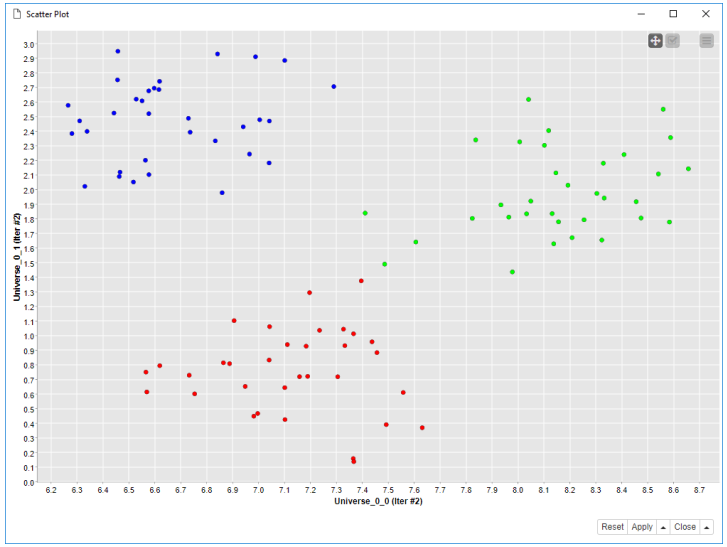

Finally, a “Joiner” node rescues the “Cluster_Membership” data column from the original data table. This will be used for coloring in the “Scatter Plot (Javascript)” node.

Figure 6.20 and figure 6.21 show the scatter plot using the newly generated columns “* (Iter #2)” and “*(Iter #3)” multiplied by a different factors. The shape look similar but the x- and y-range are different.

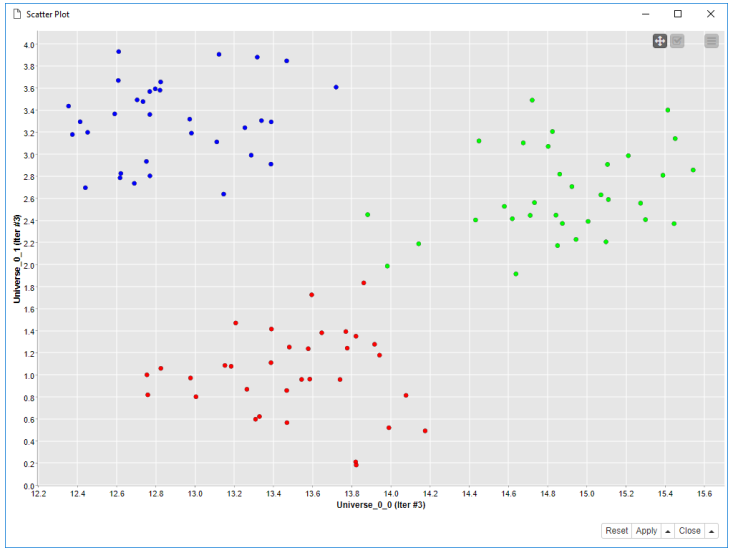


This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

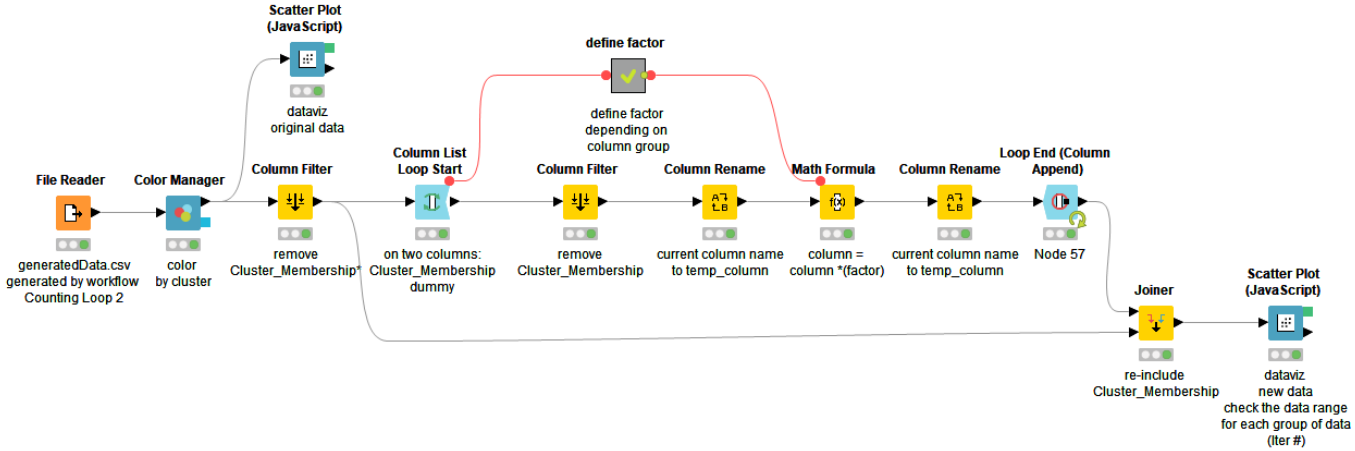
6.20. Data points from altered “Iter #2” columns. Notice the range of the axis and compare with plot on the right.




6.21. Data points from altered “Iter #3” columns. Notice the range of the axis and compare with plot on the left.




6.22. Workflow "Loop on List of Columns"






This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

## 6.6. Loop on a List of Values

Let's now use the "sales.csv" file from the KCBdata folder that we have used in some previous sections. The goal of this section is to calculate some statistics measures about the sales in each country.

- The statistical measures for both numerical and nominal columns are calculated through a "Statistics" node.
- Data about each country are isolated through a "Row Filter" node.
- The list of unique countries included in the data column "country" is extracted with a "GroupBy" node using the "country" column as the group column and no aggregation columns. A "GroupBy" node with only one group column and no aggregation columns produces the list of unique values available in the group column.

What we are still missing is a loop that allows us to iterate over all countries in the list and to calculate the sales statistics for each country at each iteration.

In the `"Workflow Control" -> "Loop Support"` category, the node "TableRow To Variable Loop Start" starts exactly the kind of loop that iterates across a list of values.

### TableRow To Variable Loop Start

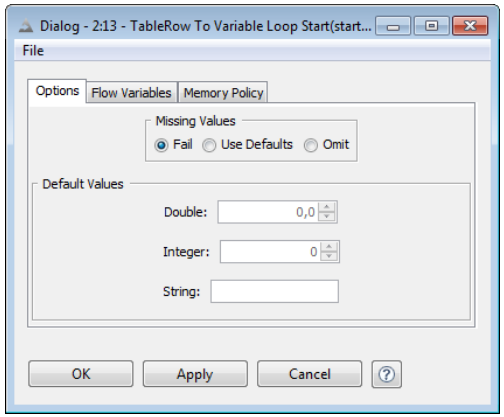
The "TableRow To Variable Loop Start" starts a loop that iterates on a list of values.

This node takes a data table at the input port and, row by row, transforms the data cells into flow variables with the same name as their column name. At the next iteration, the flow variables update their value with the next value from the corresponding column.

The "TableRow To Variable Loop Start" node does not require any configuration settings, besides the selection of the strategy for missing value handling.

The loop performs as many iterations as the number of rows in the input table at the loop start node.

6.23. Configuration window of the "TableRow To Variable Loop Start" node






This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

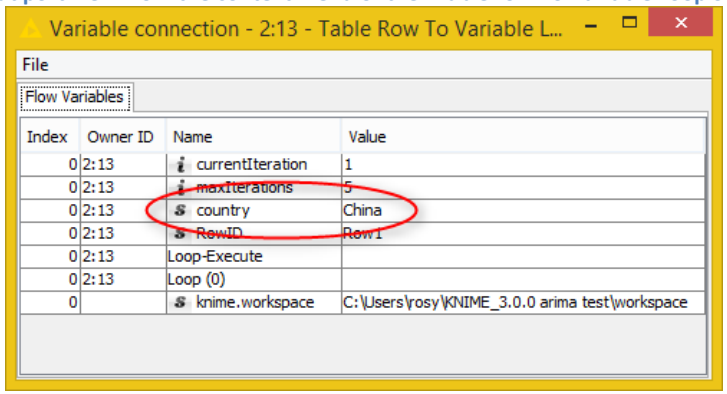
In a new workflow named “Loop on List of Values”, we started by reading the data from the “sales.csv” file with a “File Reader” node. After that, we isolated the data column “country” with a “Column Filter” node and we built the list of countries with a “GroupBy” node using “country” as group column and no aggregation columns. The output data table of the “GroupBy” node contains just one column named “country” with a list of unique country values. This is the input data table of the “TableRow To Variable Loop Start” node.

If we execute the “TableRow To Variable Loop Start” node and look at the output port view (right-click the node to open its context menu and select the option “Variable Connection”), we can see that there is a new flow variable named “country” with value “Brazil” and that the current iteration is numbered 0. “Brazil” was indeed the first value in the “country” column in the input data table.

We closed the loop with a generic “Loop End (2 ports)” node (we will explain this later). If we now run “Step Loop Execution” from the context menu of the “Loop End (2 ports)” node, at the second iteration we see that the flow variable named “country” takes the next value in the input list: “China”, and so on.

We still need to build the body of the loop.  The goal was to isolate the data rows for each country, to calculate the statistical measures through a “Statistics” node, to add the country to the measures, and to export the final results. After the “TableRow To Variable Loop Start” node and before the “Loop End (2 ports)” node, we introduced a “Row Filter” node to select only the data rows for the current value of the flow variable “country”; a “Statistics” node to calculate mean and variance and other statistical measures on the selected data group; and a “Variable to Table Column” node to add the current value of flow variable “country”.

6.24. Output port view from the context menu of the "TableRow To Variable Loop Start" node




The execution of a loop on a list of values can become quite slow, if the data set to loop on is very large and the list of values is quite long. One way to make the workflow execution faster is to cache the data group. There is a node that is used to cache data: the “Cache” node.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

### Cache

The “Cache” node caches all input data onto disk.

This is particularly useful when the preceding node performs a column transformation. In fact, many column transformation nodes only hide the unwanted part of the input data, showing what the result should be but really keeping in memory the whole input data table. For example, a “Filter Column” node just hides the unwanted columns from the output.

A “Cache” node instead caches only the visible data from the input table. Therefore, a “Cache” node after a “Column Filter” node loads only the visible input data and this might make the workflow execution faster, especially with loops when a data table is iterated many times.

The “Cache” node does not require any configuration settings.

A “Cache” node was inserted inside the loop after the “Row Filter” node to speed the loop execution at each iteration.

The “Statistics” node has three output ports:

- the top port for the statistics on numerical columns,
- the lowest port for the statistics on nominal columns,
- the port in the middle for the histogram tables.

Our goal is to collect the statistical measures for both nominal and numerical columns. We need then a loop end node capable to collect two data flows. The “Loop End (2 ports)” node can do exactly that. It has two input ports and two output ports. It collects the loop results from two different branches at its input ports and produces the concatenated data tables at the output ports.

The final version of the “Loop on List of Values” workflow, named “filter by country”, is shown in the figure below.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

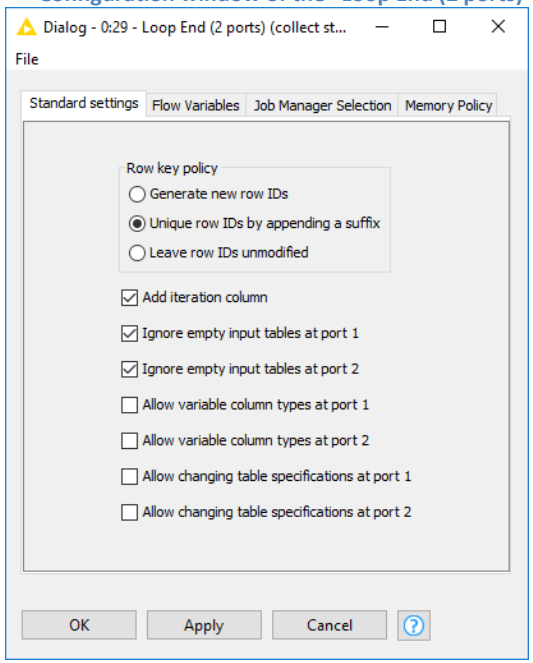
## Loop End (2 ports)

The “Loop End (2 ports)” node closes a loop, at each iteration takes the loop results from two loop branches at the two input ports, and concatenates them with the previously built data tables at the two output ports. This means that with this node we can export two separate data tables built in the loop body.

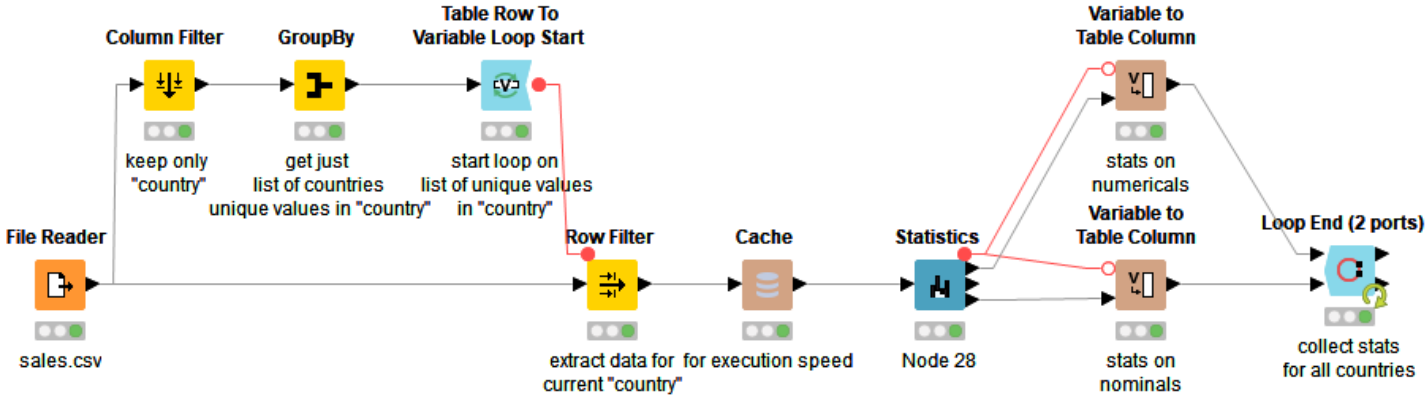
The “Loop End (2 ports)” node requires the following flags:

- The row key policy; that is: keeping the old RowIDs, make the old RowIDs unique through suffix, or just generate new unique RowIDs.
- The flag specifying whether to append a column to the output data set with the iteration number. Iteration numbers start with 0.
- The flags needed for possible incongruent data across iterations: empty input tables, different output column types, and different table specs in general. Such flags must refer to the first and the second output port.

6.25. Configuration window of the “Loop End (2 ports)” node




6.26. Workflow "Loop on List of Values"






This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

## 6.7. Loop on Data Groups and Data Chunks

In the previous section we have isolated the list of countries and calculated a few sales statistics for each one of them. Now, would not that be much easier to loop on the groups directly? The “Group Loop Start” node identifies unique values in a selected data column, extracts the associated groups of data rows in the input data table, and loops over them. This makes the task described in the previous section even easier!

Instead of having a “Table Row to Variable Loop Start” node followed by a “Row Filter” node and a “Cache” node like in the previous section, we just introduced a “Group Loop Start” node. The “Group Loop Start” node was set to loop on unique values in the “country” column, i.e. on the groups of data defined by each country value.

The loop body was then again made by the “Statistics” node and the two “Variable to Table Column” nodes to append the current country name to the resulting data tables. The loop was then closed with a “Loop End (2 ports)” node.

Since the “Group Loop Start” node is equivalent to the “Table Row to Variable Loop Start” followed by a “Row Filter” node, the output data tables produced by the “Loop End (2 ports)” node should be identical in this new workflow - named “Loop on Groups of Data” - as in the workflow generated in the previous section - named “Loop on List of Values”.

The “Workflow Control” -> “Loop Support” category includes a type of loop similar to the loop on groups: the loop on chunks. This loop divides the input data table into smaller pieces (chunks) and iterates from the first piece to the last one till the end of the data table has been reached. So, for example, a data set with 99 data rows can have 3 chunks of 33 data rows each.

The biggest advantage in using a chunk loop is not in the number of iterations on the input data set, since it only covers the input data set once, but the speed. Indeed, processing smaller chunks of data at a time speeds up execution. A second advantage in using a chunk loop is that different processing can be applied to different chunks of data.

A chunk loop starts with a “Chunk Loop Start” node and ends with any of the loop end nodes.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

## Group Loop Start

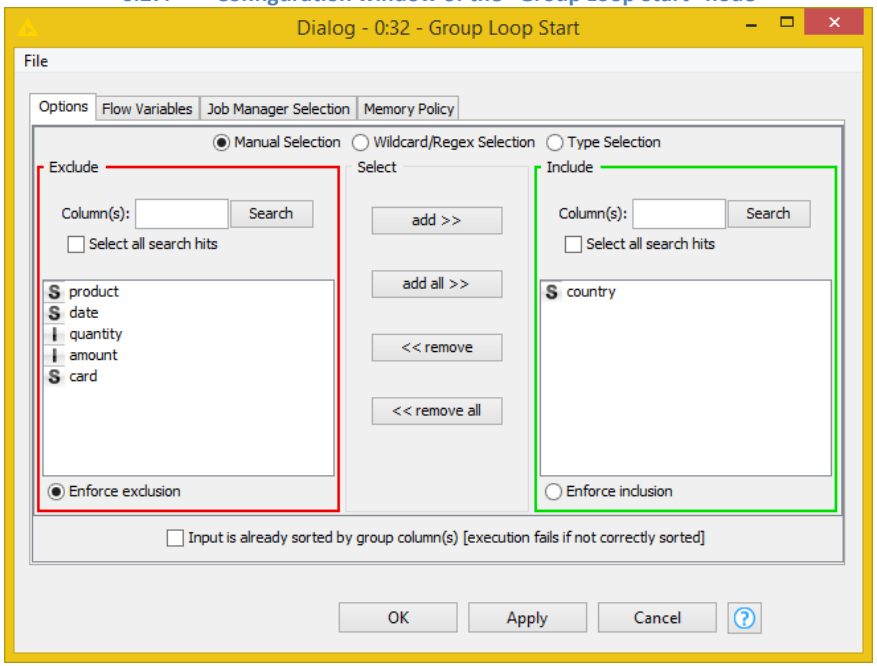
The “Group Loop Start” node starts a group loop. It identifies a list of unique values in one or more of the input data columns, detects the groups in the input data table associated with each one of these values, and iterates over those groups.

The configuration window of the “Group Loop Start” node requires the data column(s) from whose values to build the data groups.

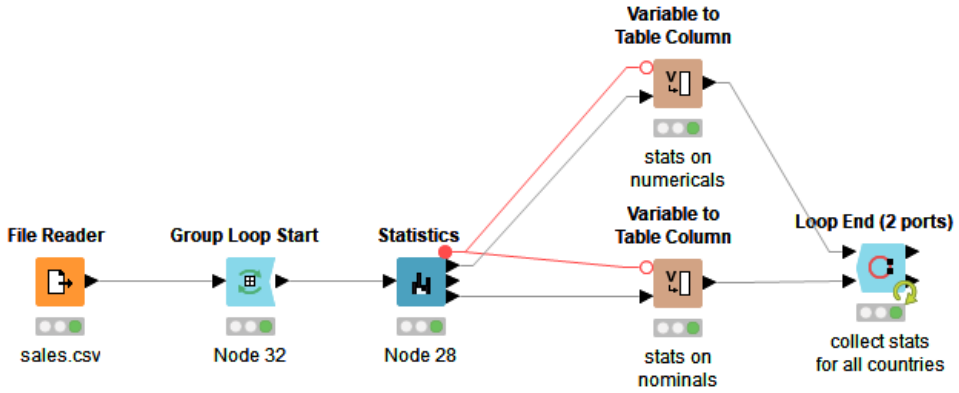
The column selection is obtained by means of an Exclude/Include framework. Columns can be included manually, based on a wildcard or regex expression, or based on type. Data columns in the “Include” frame will be used to group the data rows; those in the “Exclude” frame will not.

In case of manual selection, “Enforce Inclusion” and “Enforce Exclusion” add possible new data columns to the “Exclude” frame or to the “Include” frame respectively.

6.27. Configuration window of the “Group Loop Start” node




6.28. Workflow "Loop on Groups of Data”. This workflow performs exactly the same task as the workflow named “Loop on List of Values” reported in figure 6.26.






This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

## Chunk Loop Start

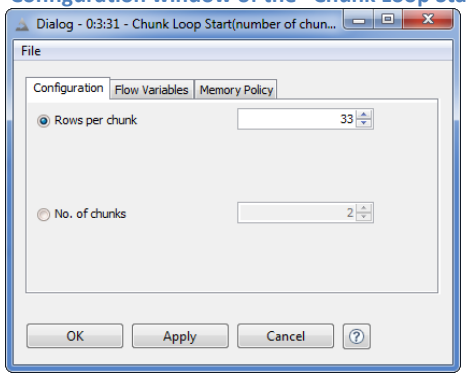
The “Chunk Loop Start” node starts a chunk loop.

It divides the input data table into a number of pieces (chunks) and iterates over those chunks.

The configuration window of the “Chunk Loop Start” node requires:

- Either the (maximum) number of data rows in each chunk
- Or the number of chunks

6.29. Configuration window of the “Chunk Loop Start” node




We created a new workflow named “Chunk Loop”. This workflow uses the same data generated by the “Data Generator” node named “3 clusters, 2 coordinates" described in the section 6.2. We set the “Data Generator” node to produce 99 data rows distributed across 3 clusters along 2 coordinates. The original data set then contains 33 data rows for cluster 1, 33 data rows for cluster 2, and 33 data rows for cluster 3 and defining chunks of 33 rows fits the clusters size perfectly.

We implemented a chunk loop on chunks of 33 data rows. For each chunk, we shift the data along the x-axis of as many units as the iteration number. This translated into:

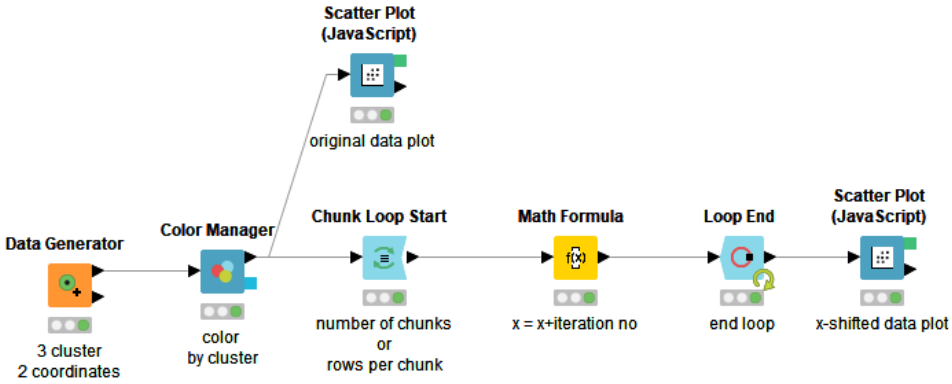
- A “Chunk Loop Start” node to start the chunk loop, with “Rows per chunk” set to 33;
- A “Math Formula” node with “$Universe_0_0$ + $${IcurrentIteration}$$” where $Universe_0_0$ is a data column and {IcurrentIteration}$$ the loop flow variable
- A generic “Loop End” node to close the loop and concatenate the results of each iteration.
- We then inserted a “Scatter Plot” node to visualize the final data table.

The “Chunk Loop” workflow is shown in figure 6.30, while the data set before and after the progressive shifting is shown in figure 6.31 and 6.32 respectively.

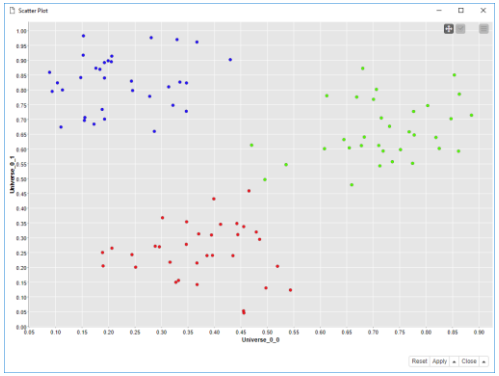


This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

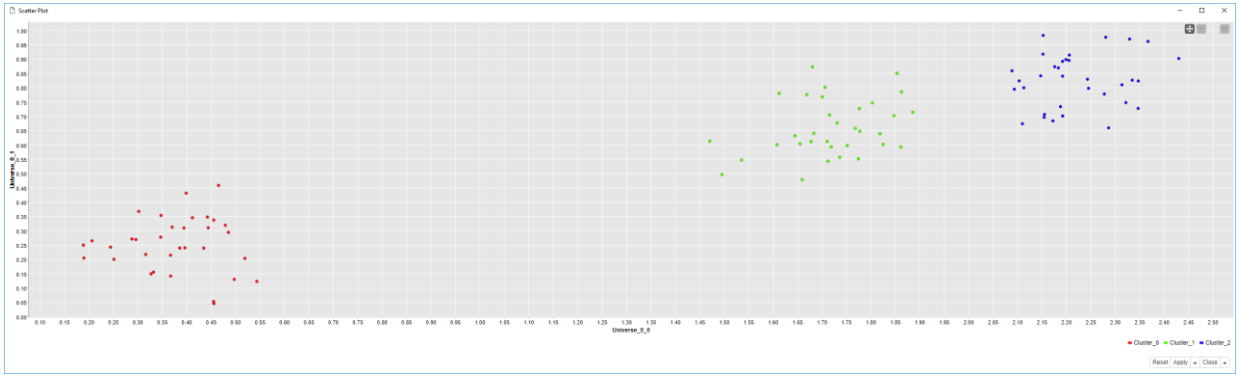
6.30. Workflow "Chunk Loop"




6.31. Original Data Set



6.32. Progressively x-Shifted Data Set after executing workflow "Chunk Loop"



Another node in the "Loops" category that might be worth mentioning is the "Breakpoint" node. The "Breakpoint" node can be inserted in a loop body and, while it does not process data, it might give some control on the loop execution.



This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

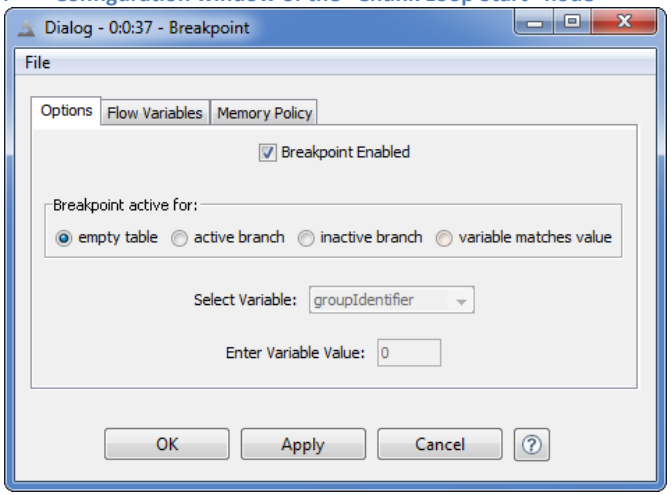
### Breakpoint

The “Breakpoint” node prevents the loop execution when the input table fulfills a user-specified condition, like for example the input data table is empty or a variable matches some set value.

The configuration window then needs:

44. The flag enabling the breakpoint
45. The condition for which the breakpoint has to become active and disable the loop execution (like for example an empty input data table)
46. The name of the flow variable and its breakpoint value if the “variable matches value” condition has been selected.

6.33. Configuration window of the “Chunk Loop Start” node




## 6.8. Keep Looping till a Condition is verified

In the first sections of this chapter we have seen how to repeatedly execute a group of operations on the input data for a pre-defined number of times. However, there are other cases where we do not know upfront how many iterations are needed. We only know that the loop has to stop when a certain condition is met.

In this section, we are going to show how to implement the second solution: iterating till a condition is met. In order to implement a “do-while” cycle, i.e. a loop that iterates till a condition is met, we use the “Generic Loop Start” node to start the loop and the “Variable Condition Loop End” to end the loop and collect the results.

### Generic Loop Start

The “Generic Loop Start” node, located in the `“Workflow Control” -> “Loop Support”` category, starts a generic loop on all data rows without any previous assumptions and, because of that, it needs no configuration settings.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

The “Generic Loop Start” node starts a generic loop on all data rows without any assumptions. The “Variable Condition Loop End” implements the stop condition and checks it at the end of each iteration. If the loop condition is verified the “Variable Condition Loop End” ends the loop and makes the results, collected till now, available at the output port. Otherwise it passes the control back to the loop start node and proceeds with the next iteration.

## Variable Condition Loop End

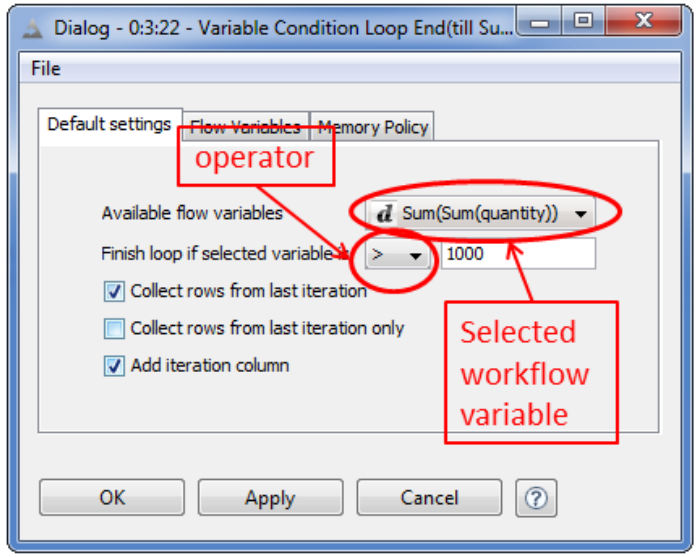
The “Variable Condition Loop End” implements a condition to terminate a loop. At the end of each iteration the condition is evaluated and, if it is true, the loop is terminated and the collected results are made available at the output port.

The terminating condition can only be implemented on a flow variable. The configuration window then requires:

- the flow variable on which the condition is evaluated
- the condition (i.e. comparison operator + value)
- a few flags to exclude or only include the last iteration results and/or to append the iteration column

A second output port shows the progressive values of the flow variable used to implement the loop condition throughout the loop iterations.

6.34. Configuration window of the “Variable Condition Loop End” node




**Note**. The terminating condition can only be implemented on one of the flow variables available to the “Variable Condition Loop End” node. This means that the flow variable for the loop condition has to be defined before the loop end node is reached. It can of course be updated at each iteration.

In order to show how such a loop can be implemented, we created a new workflow under the “Chapter6” workflow group and we named it “Loop on Condition”. This workflow implements a game. I think of a number and write it into the “Table Creator” node. Then I prompt the user to guess it. The user can try a new guess three times, after which the game just ends either way.

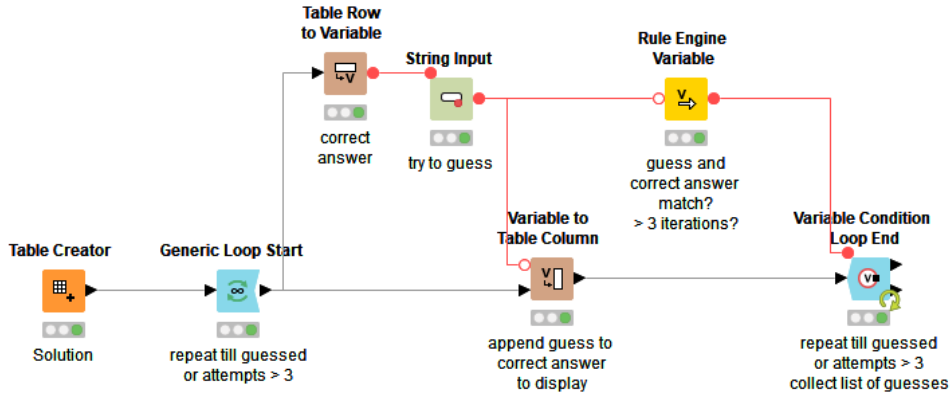
This workflow starts with a “Table Creator” node containing my mystery number. Then a “Generic Loop Start” node starts the guessing loop. In the loop the user is prompted to guess the number through a “String Input” Quickform node. Guess and number are compared in a “Rule Engine Variable” node



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

that produces 1 if the guess was correct OR the number of iterations was > 2, 0 otherwise. The result is stored in a flow variable named “success”. The loop should terminate when success = 1. This condition is checked in the “Variable Condition loop End” that closes the loop and the workflow.

6.35. Workflow "Loop on Condition"




**Note**. Like other loop end nodes, the “Variable Condition Loop End” node introduces a few new flow variables related to the loop execution, like “currentIteration” which contains the current iteration number.

## 6.9. Recursive Loop

All loops described in the previous sections do not change the content of the loop input data table across iterations. At each new iteration the input data table is unchanged. Each iteration might extract a different part of the input data table, but the data table itself is always the same. For example, the counting loop loops a number N of times on the same input data table. That is, all so far seen loops do not have memory.

In the workflow named “Counting Loop 1” and implemented in section 6.2 the iteration number is added to the initial x values. In this section, we want to re-elaborate that workflow as to increment the x values in the input data one more unit at each new iteration. Therefore, we need a loop that can pass the incremented x-values back to the loop start: that is we need a loop with memory. There is only one such a loop in the KNIME Analytics Platform: the recursive loop.

A recursive loop starts with a “Recursive Loop Start” node and ends with a “Recursive Loop End” node. The particularity of this loop lies in the loop end node. The “Recursive Loop End” node has two input ports: one to collect the iteration output data table (like all other loop end nodes) and one to pass



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

the processed data back to the loop start node as the new input data for the next iteration. The recursive loop node pair enables the passing of a data table from the “Recursive Loop End” node back to the “Recursive Loop Start” node. The output data table collected at the current iteration may or may not be the input data table for the next iteration. Hence the two distinct input ports of the “Recursive Loop End” node.

We copied the old “Counting Loop 1” workflow and renamed it “Recursive Loop”. As in the original workflow, the “Data Generator” node generates 100 data rows distributed across 3 clusters along two coordinates (“Universe_0_0” and “Universe_0_1”). However, instead of the counting loop we introduced a recursive loop, with a “Recursive Loop Start” node and a “Recursive Loop End” node with maximum number of iterations set to 4.

The goal of this loop is to increment the initial x-values (“Universe_0_0”) 1 unit at each iteration. So the loop body was implemented through a “Math Formula” node with mathematical expression “$Universe_0_0$ + 1”, that is adding one unit to all x-values at each iteration. As usual two “Scatter Plot” nodes show the data cluster distribution before and after the loop.

### Recursive Loop Start

The “Recursive Loop Start” node starts a recursive loop and must be used together with a “Recursive Loop End” node.

The “Recursive Loop Start” node and the “Recursive Loop End” node communicate with each other at the end of each iteration. At iteration 0 the “Recursive Loop Start” node uses the input data able to feed the loop body. After iteration 0, the “Recursive Loop Start” node feeds the loop body with the data table received from the second input port of the “Recursive End Loop” node.

No configuration settings needed.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

## Recursive Loop End

The “Recursive Loop End” node closes a recursive loop, started by a “Recursive Loop Start” node. The “Recursive Loop End” node has two input ports.

- The first input port collects the results of the current iteration and concatenates them with the results from the previous iterations. At the end of the loop the final data table will be passed to the only output port.
- The second input port receives the data table to pass back to the “Recursive Loop Start” node to feed the next loop iteration.

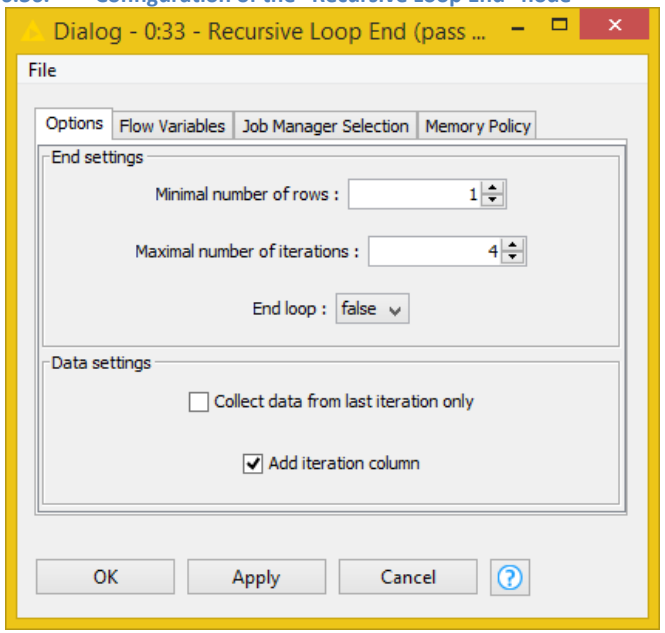
The first three settings in the configuration window are emergency stop settings, to avoid infinite loops. The loop will stop if:

61. A maximum number of iterations has been reached
62. The data table pass back to the loop start node contains less than the required minimum number of rows
63. A variable overriding the “end loop” option takes value “true”

The last two settings regulate the data structure.

64. Add a data column containing the iteration number to the final output data table
72. Collect the output data only from the last iteration or concatenating the results from all iterations.

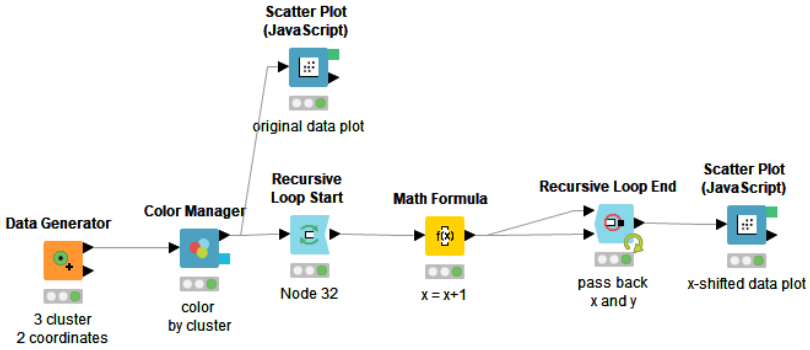
6.36. Configuration of the “Recursive Loop End” node






This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

6.37. Workflow "Recursive Loop"




## 6.10. Exercises

### Exercise 1

Generate 5400 data rows in a two-dimensional space grouped in 6 clusters each with a standard deviation value of 0.1 and no noise. Assign different colors to the data of each cluster and plot the data by means of a scatter plot.

Then process the data of each cluster in a different way according to the formulas below and observe how the clusters have changed by means of a scatter plot again. To make the workflow run faster and be more flexible, use a loop to process the data.

x = Universe_0_0, y = Universe_0_1

**Cluster 0:** x = x
**Cluster 1:** x = sqrt(x)
**Cluster 2:** x = x + iteration number
**Cluster 3:** x = x * iteration number
**Cluster 4:** x = y
**Cluster 5:** x = x*x



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

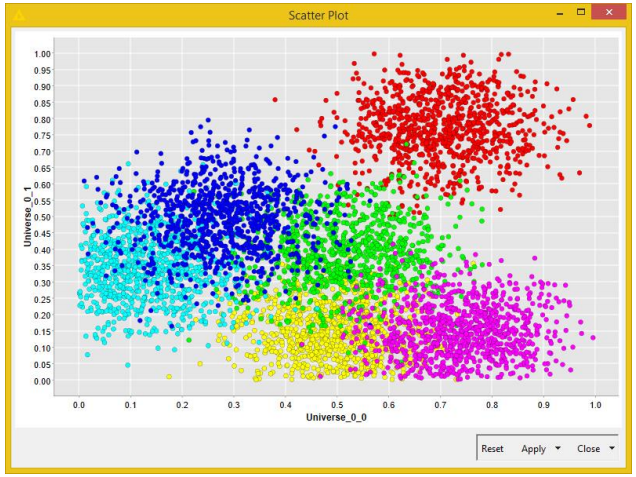
**Solution to Exercise 1**

To process the data in a loop, we used a “Chunk Loop Start” node. We defined the chunk size to be 900 data rows in order to have exactly one cluster in each chunk. We then closed the loop with a generic “End Loop” node which concatenates the output tables into the final result. The different processing for each cluster is implemented with a “Java Snippet (simple)” node with the following code:

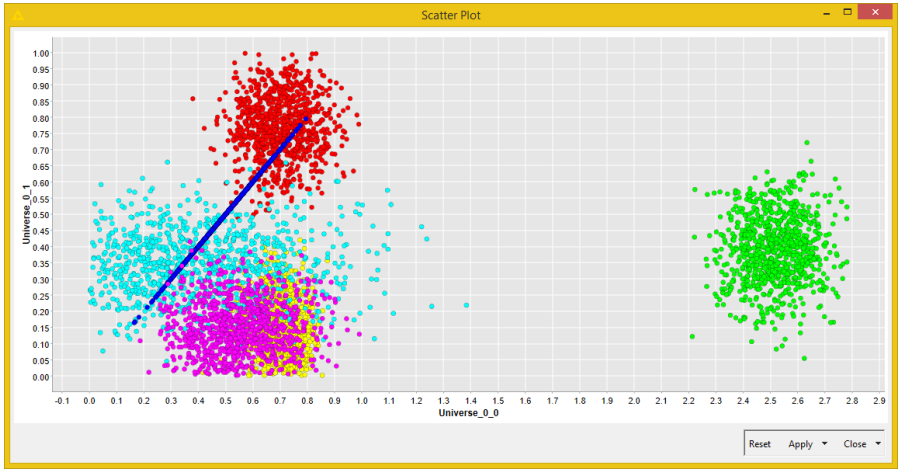
```
Double x = 0.0;
if      ($${IcurrentIteration}$$ == 0)     x = $Universe_0_0$;
else if($${IcurrentIteration}$$ == 1)     x = Math.sqrt($Universe_0_0$);
else if($${IcurrentIteration}$$ == 2)     x = $Universe_0_0$+$${IcurrentIteration}$$;
else if($${IcurrentIteration}$$ == 3)     x = $Universe_0_0$*$${IcurrentIteration}$$;
else if($${IcurrentIteration}$$ == 4)     x = $Universe_0_1$;
else                                      x = $Universe_0_0$*$Universe_0_0$;
return x;
```

The scatter plot of the original data and the scatter plot of the processed data are shown in the figures below.

6.38. Scatter Plot of the original data



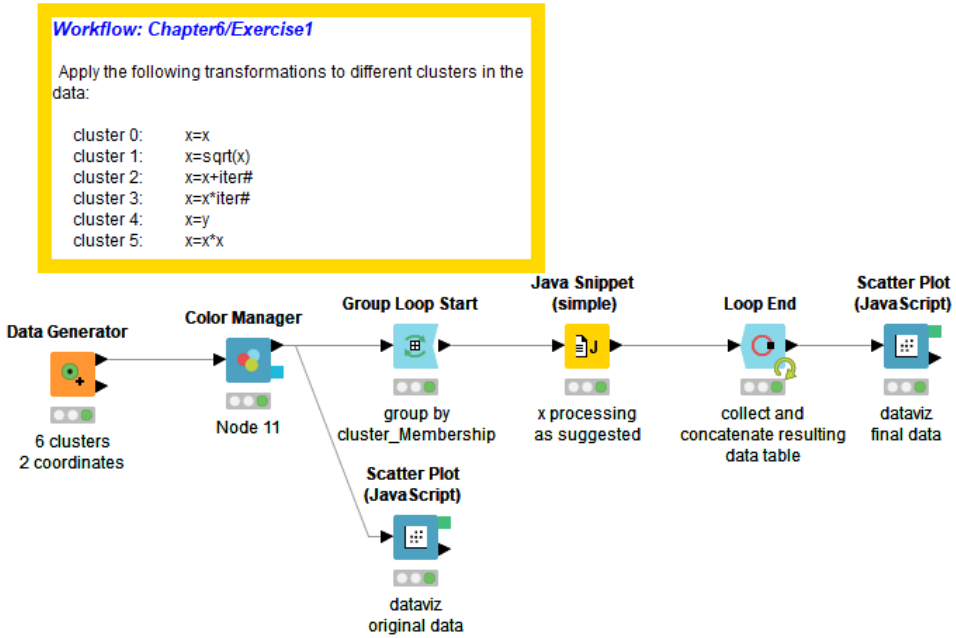

6.39. Scatter Plot of the resulting data






This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

6.40. Exercise 1: The workflow




## Exercise 2

On the 15th of each month a course takes place, starting from 15.01.2011. Teacher "Maria" teaches till the end of March, teacher "Jay" till the end of September, and teacher "Michael" till the end of the year. The course is held in San Francisco from May to September and in New York the other months. Generate the full table of the courses for the year 2011 with course date, teacher name, and town.

**Solution to Exercise 2**

To solve this problem we started from a data set with the first course date only: 15.01.2011.

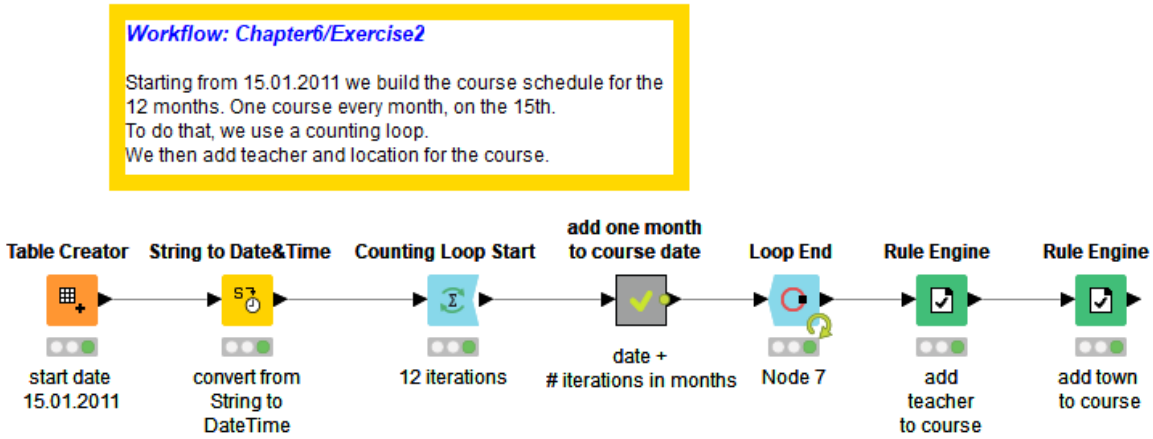
Then we used a "Counting Loop Start" node that iterates 12 times on the initial data set and generates a new date at each iteration.

The "teacher" and "town" columns are both obtained with a "Rule Engine" node.

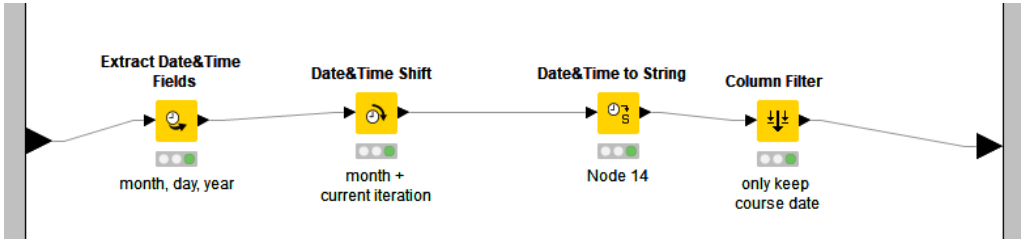


This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

6.41. Exercise 2: The workflow




6.42. Exercise 2: The "add one month to course date" meta-node




## Exercise 3

This exercise gets rid of duplicated unnecessary data rows.

In the folder KCBdata there is a file called “wrong_sales_file.txt” which contains sales records, each one with a contract number, as well as duplicate records. In fact, for each sale you can find an older record with a few missing values and a more recent record with all field values correctly filled in. The column “load_date” contains the date of record creation. Of course, we want to get rid of the old records with missing values and keep only the recent records with all fields filled in. In this way, we get a sales table with a unique record ID, i.e. the contract number, and only the most up to date values.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

**Solution to Exercise 3**

6.43. Exercise 3: The workflow


In the solution workflow, we read the “wrong_sales_file.txt” with a “File Reader” node. If we execute a “RowID” node on the “contract nr” column and with the flag ”ensure uniqueness” not enabled, then the “RowID” node’s execution fails. This means that the “contract nr” column contains non-unique values. Indeed, for each “contract nr” we have two records in the data table: an old one with many missing values and a recent one with all fields filled in.

In order to filter out the older records with missing values, we loop on the “contract nr” list with a “TableRow To Variable Loop Start” node. At each iteration, we keep only the records with the “contract nr” of the current iteration (“Row Filter” node), we sort the selected records by “load_date”, in



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

descending order, and we keep only the first row (the second “Row Filter” node), which is the most recent record. The loop is then closed by a generic “Loop End” node.

If we now run a “RowID” node on the loop results, which is similarly configured to the first “RowID” node of this exercise, it should not fail anymore.

The same result could have been obtained with just a GroupBy node, grouping by contract number and taking just the first (or last depending on sorting) of all other values.

**Note**. Since KNIME has been thought in terms of data tables, loops are rarely needed. Before using a loop make sure that a dedicated node for what you have in mind does not exist!

## Exercise 4

Let’s suppose that the correct file called “sales.txt” had been saved in many pieces. In particular, let’s suppose that each column of the file had been saved together with the “contract nr” in a single file under “KCBdata/sales”.

This exercise tries to find all the pieces and to collect them together to form the original file “sales.txt”. In “KCBdata/sales” we find files like “sales_<column name>.txt” containing the “contract nr” and the “<column name>” columns of the sale records. There are 6 files for 6 columns: “card”, “amount”, “quantity”, “card”, “date”, “product”.

**Solution to Exercise 4**

Our solution to this exercise, builds a data table with a “Table Creator” node which includes only the column names in a column named “type”.

Then a “TableRow To Variable Loop Start” starts a loop that:

- loops over the list of column names in “type”,
- builds the file path with a “String Manipulation (Variable)” node with: `join($${Sfile-path}$$, "sales_", $${Stype}$$, ".csv")`
- passes the file path as a workflow variable to a “File Reader” node,
- reads the file via the “File Reader” node,
- collects the final results with a “Loop End (Column Append)” node.

If the “contract nr” values are read as RowIDs for each file, then the “Loop End (Column Append)” node joins all these columns providing the flag for the ”Loop has same rowIDs in each iteration” is enabled in its configuration window.



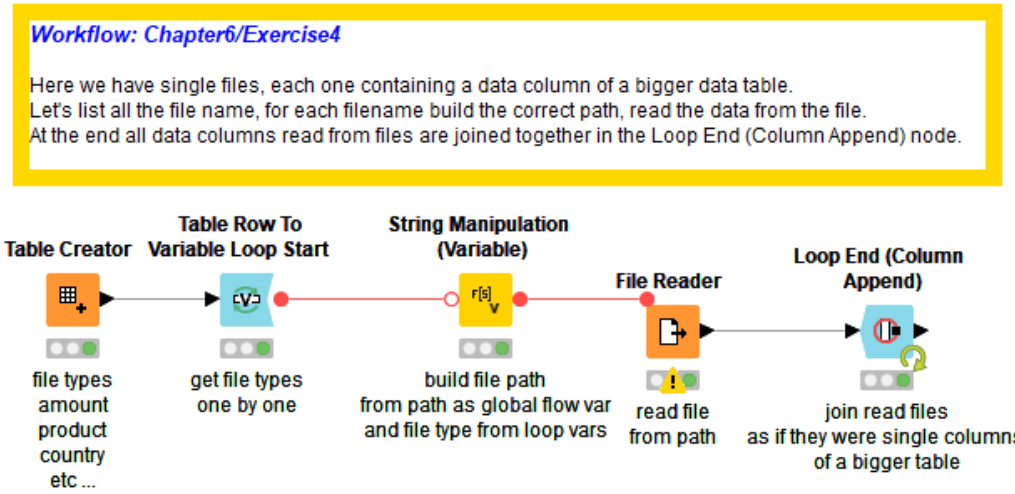
This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

The ”File Reader” node needs to use the flow variable called “filename” to read the file and the flow variable named “type”, introduced by the “TableRow To Variable Loop Start”, in order to name the column read at each iteration.

To set the column name with a flow variable in a “File Reader” node, you need to:

- go to the “File Reader” node’s configuration window,
- select the “Workflow Variables” tab,
- expand the “Column Properties” node and then “0”
- assign the value to the “Column Name” field via the workflow variable.

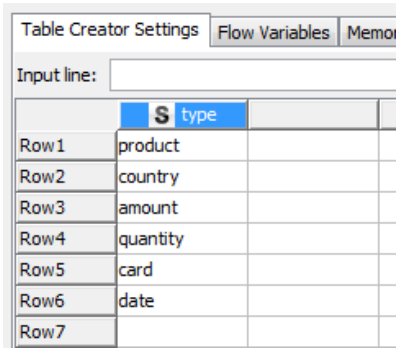
### 6.44. Exercise 4: The workflow



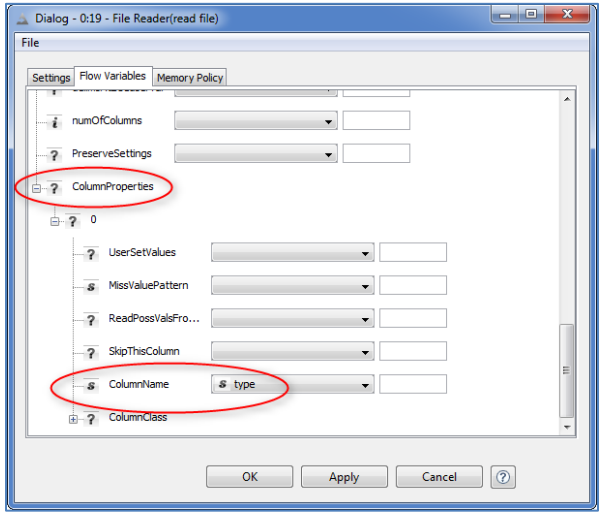



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

6.45. The Configuration Window of the "Table Creator" node




6.46. The “ColumnName” setting in the "Workflow Variables" Tab of the configuration window of the "File Reader" node to name the data column read at each iteration






This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

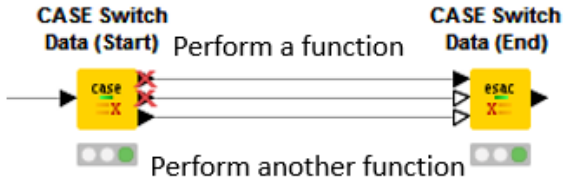
# Chapter 7. Switches

## 7.1. Introduction to Switches

KNIME workflows consist of a sequence of nodes connected together. The sequence can be linear, where one node is connected just to the next, or it can branch off to multiple parallel routes. Sometimes in some situations it might be preferable to execute only some of the parallel branches of the workflow and not the others. This is where the KNIME node group, “Switches”, becomes useful.

A “Switch (Start)” node determines the flow of data via one or more workflow branches. Data flows into a “Switch (Start)” node and is then directed down one or more routes where a number of specific operations are performed, while all other routes remain inactive. All routes originating from a “Switch (Start)” node are finally collected by a “Switch (End)” node. The KNIME switch concept is illustrated in figure 7.1, where a switch with 2 alternative parallel routes has been implemented.

7.1. Generic illustration of data flow control via Switches




Depending on the type of “Switch (Start)” node, the data can flow out through one or more ports. In figure 7.1, for example, the “CASE Switch Data (Start)” node has three output ports, of which only one is active at a time. In this particular configuration, the data has been directed to flow out from the third output port of the “CASE Switch Data (Start)” node only. The top output ports, in fact, are blocked, as depicted by a red cross. The data then flows through a number of nodes implementing “Another Function” till the “CASE Switch Data (End)” node, thus completing the switch process. Which output port of the “Switch (Start)” node is active, and therefore which route the data takes, can be controlled in the configuration window.

All “Switch (Start)” and “Switch (End)” nodes are located in the “Workfow Control” -> “Switches” category in the “Node Repository” panel. There are two main types of “Switch” nodes: the “IF Switch” nodes and the “CASE Switch” nodes. The “IF Switch” starts a two branch switch closed by the “End IF” node. The “CASE Switch … (Start)” node starts three parallel branches and results from the three branches are collected by a CASE Switch … (End)”



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

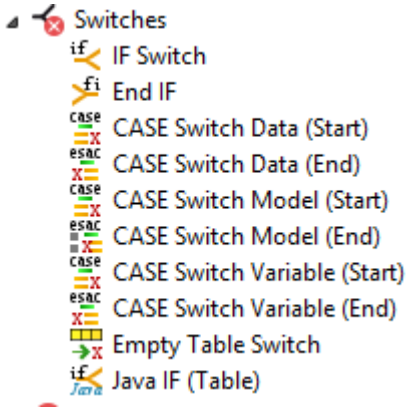
node. “End IF” and “CASE Switch … (End)” nodes write the switch type backwards on the node icon. Thus “if” and “fi” form one pair and “case” and “esac” the other.

While there is only one type of IF Switch pair, there are many types of CASE switch nodes. Some of the CASE Switch nodes deal with flow variables, others with data, and others with models. You can also mix and match, i.e. you can start with a “CASE Switch Data (Start)” node, passing data to its output branches, and close with a “CASE Switch Variable (End)” node, collecting results of type flow variable to pass on to the next nodes, or vice versa.

In the “Workflow Control” -> “Switches” category we can also find a “Java IF (Table)” node. The “Java IF (Table)” node works similarly to an “IF Switch” node, but allows for the flow of data to be controlled by a Java method.

Finally, the last node in this category is the “Empty Table Switch” node. The output of an “Empty Table Switch” node becomes inactive (i.e. connected nodes are on an inactive branch) if the input table is an empty data table. This avoids the execution of subsequent nodes on tables with no actual content and possibly the workflow failure.

7.2. The nodes contained in the “Switches” category




## 7.2. The “IF Switch”- “END IF” switch block

There are two “IF Switch” nodes: the “IF Switch” node and the “Java IF (Table)” node. The “IF Switch” node is a simple switch to change the data flow in one direction or another. The direction can be controlled manually or by means of a workflow variable. The “Java IF (Table)” node also controls the data flow but by means of a Java method. Both these nodes use the same “End IF” node to terminate the switch data flow control block.

Let’s start with the simplest of the two switch nodes: the “IF Switch” node. In the workflows that you have imported for this book (the link was provided in the purchase email), two workflows are available in folder “Chapter7” that demonstrate how to set the “IF Switch” node manually or automatically: the “Manual IF Switch” workflow and the “Automatic IF Switch” workflow.

The “Manual IF Switch” workflow reads “cars-85.csv” file from the folder KCBdata, bins the data set using the fuel consumption in mpg for either city driving or highway usage, and calculates the statistics of “curb weight”, “engine size”, and “horse power” over the defined bins. To alternatively bin the data set on different data column values, we need two branches inside the workflow: one branch to bin the data set on “city mpg” data column and the other branch to bin the data set on the “highway mpg” data column.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

In order to produce two parallel alternative branches in the workflow, the data flow is controlled by an “IF Switch” node. The “IF Switch” node implements two branches: the top branch is supposed to bin the “city mpg” data column, while the bottom branch is supposed to bin the “highway mpg” data column. The “IF Switch” node is configured to enable the top branch by default. If the data is supposed to flow through the bottom branch, a manual change has to be made to the setting in its configuration window.

## IF Switch

The “IF Switch” node allows data to be branched off to either one or both of the two available output ports. The direction of the data flow can be controlled by manually changing the node’s configuration settings or automatically by means of a flow variable.

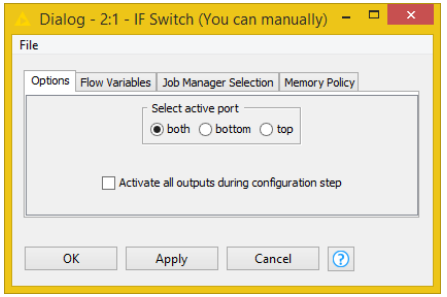
In the configuration window:

- The “Options” tab contains the options for the manual setting of the active output port; option “Activate all outputs during configuration step” enables all outputs during configuration of other subsequent nodes.
- The “Flow Variables” tab allows to overwrite the active output port in the “Options” tab with a workflow variable value.

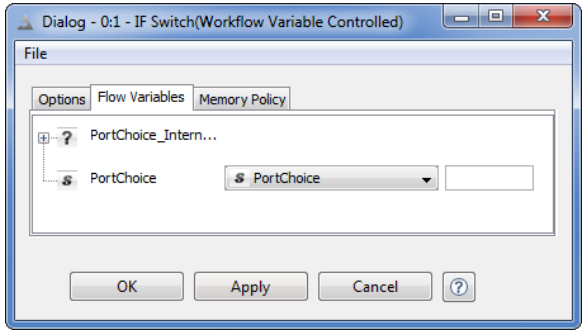
The “Options” tab in the configuration window contains three radio buttons to manually select the path for the data flow. By selecting “top” or “bottom” or “both” the corresponding output port(s) is/are activated.

Alternatively, the “Flow Variables” tab allows to overwrite the configuration parameter “PortChoice”, set in the “Options” tab, with a String-type workflow variable. The allowed workflow variable values are: “top”, “bottom”, and “both”.

7.3. Configuration window of the “IF Switch” node: “Options” tab




7.4. Configuration window of the “IF Switch” node: "Flow Variables" tab




The goal of this workflow is to calculate some statistics for columns “curb weight”, “engine size” and “horse power”, sometimes on the quantiles of column “city_mpg” and sometimes on the quantiles of column “highway_mpg”. So, in both branches following the “IF Switch” node, an “Auto-Binner” node was used to bin the data set based on sample quartiles of “city_mpg” on one branch and of “highway_mpg” on the other branch of the IF block. The “Auto-Binner” node returns the input data set with an additional column containing the numbered bins. This column has the header name of either



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

"city mpg [Binned]", if the top branch is taken, or "highway mpg [Binned]", if the bottom branch is selected. The switch block is then closed by an "END IF" node.

## End IF

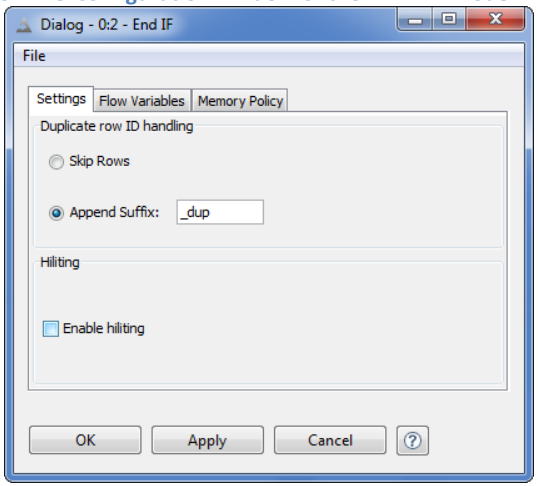
The "End IF" node closes a switch block that was started by either an "IF Switch" node or a "Java IF (Table)" node.

The "END IF" node has two input ports and accepts data from either the top, or bottom, or both input ports. If both input ports receive data from active branches, then the result is the concatenation of the two data tables.

The configuration window of the "End IF" node requires only a duplicate row handling strategy among two possible alternatives: either skip duplicate rows or append a suffix to make the RowIDs unique.

Check the "Enable hiliting" box if you wish to preserve any hiliting after this node.

7.5. The configuration window of the "END IF" node




A "GroupBy" node was appended at the end of the switch block to calculate the statistics of "curb weight", "engine size", and "horse power" over the defined bins produced by the active workflow branch. Since it would be cumbersome to manually change the configuration settings of the "GroupBy" node every time the active output port of the "IF Switch" node is changed, we renamed the binned column in both branches to carry the same column header, i.e. "Binned.MPG". The "Manual IF Switch" workflow is shown in figure 7.6.

The "Auto-Binner" node does not belong to the "Switches" category. However, since it has been used to implement the "Manual IF Switch" workflow, we spend a few words here to describe how it works and what it can be used for. The "Auto-Binner" node groups numeric data in intervals - called bins. Unlike the "Numeric Binner" node, the "Auto-Binner" node first divides the binning data columns into a number of equally spaced bins; then labels the data as belonging to one of those bins. The column to be binned is selected via an "Exclude/Include" framework, manually or via Regular / wildcard expression.

The manual selection of the column to be binned is based on an "Exclude/Include" framework: the columns to be used for binning are listed in the "Include" frame on the right; the columns to be excluded from the binning process are listed in the "Exclude" frame on the left.

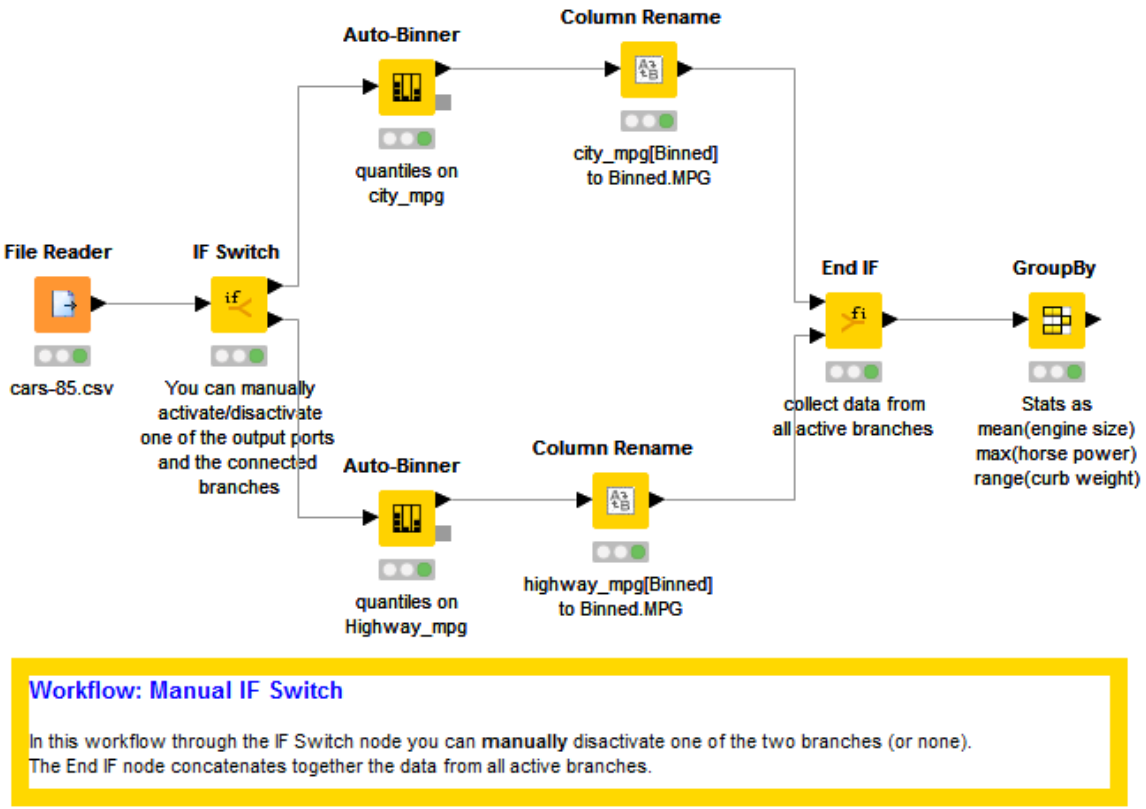


This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

To move single columns from the “Include” frame to the “Exclude” frame and vice versa, use the “add” and “remove” buttons. To move all columns to one frame or the other use the “add all” and “remove all” buttons. A “Search” box in each frame allows searching for specific columns, in case an excessive number of columns impedes an easy overview of the data.

The selection based on a regular expression or a wildcard requires the regex or wildcard matching pattern as configuration setting.

7.6. The “Manual IF Switch” workflow






This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

## Auto-Binner

The configuration window offers the choice between two methods to build the bins:

87. As a fixed number of equally spaced bins;

88. As pre-defined sample quantiles.

The configuration window also offers the choice between two naming options for the bins:

89. “Bin1”, “Bin2”, … , “BinN”

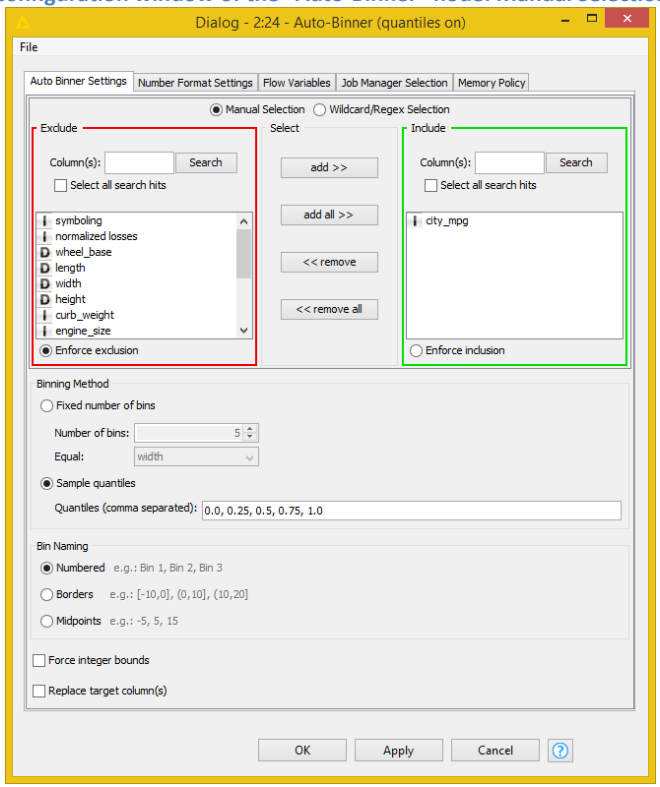
90. By using the bin borders, e.g. [0.5674, 0.7899]

There is then a flag to avoid this non-round boundaries of the bin intervals.

You can also choose to overwrite the original column by means of another flag.

The “Number Format Settings” tab allows for custom formatting of double and string bin labels.

7.7. Configuration window of the “Auto-Binner” node: Manual Selection




Pre-defined sample quantiles produce bins corresponding to the given list of frequencies. The smallest element corresponds to a frequency of 0 and the largest to a frequency of 1. The applied estimation method is Type 7 which is the default method in R, S and Excel.

**Note**. The “Auto-Binner” node does not allow customized bins. Please use the "Numeric Binner" node if you want to define custom bins.

Alternatively to the manual setting of the active port(s) in the “IF Switch” node configuration window, we could define a workflow variable of type String, named “PortChoice” for example, and we could use that variable to overwrite the manual settings in the “IF Switch” node’s configuration window (Fig. 7.4). If we activate the output port of the “IF Switch” node via a flow variable, we do not need to manually change the node configuration settings



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

to enable the other branch of the switch block. In this case, it is enough to change the value of the flow variable “PortChoice”, for example from “top” to “bottom”.

The new workflow with the automatic activation of the output port(s) of the “IF Switch” node is named “Automatic IF Switch” and is available again in folder “Chapter7”. This workflow is identical to the “IF Manual Switch” workflow, except for the flow variable “PortChoice” and the configuration settings of the “IF Switch” node. In fact, the “IF Switch” node is configured so as to overwrite the active port choice with the value of the flow variable “PortChoice”.

## 7.3. The “Java IF (Table)” node

A similar node to the “IF Switch” node is the “Java IF (Table)” node. In the “Java IF (Table)” node the active port, and therefore the data flow direction, is controlled through some Java code. The return value of the Java code determines which port is activated and which branch of the workflow is active.

A switch block, then, can start with a “Java IF (Table)” node, branch off into two different data flow paths, and collect the results of the two branches with an “End IF” node. The “JAVA IF (Table)” node can be found like all other switch nodes under the “Workflow Control” -> “Switches” category.

To demonstrate the use of the “Java IF (Table)” node, we created a new workflow, named “Java IF & Tables”, in the workflow group called “Chapter7”. The goal of the workflow was to produce a box plot on car price, engine size, and horse power for either two- or four-door cars for data from the “cars-85.csv” file.

The selection of either two-door cars or four-door cars was implemented by means of a “Java IF (Table)” switch node governed by a flow variable called “Doors”. This flow variable can take two string values, either “two” or “four”, which determines the behaviour of the “Java IF (Table)” switch node. Finally, an “Box Plot (Javascript)” node produces the box plot on the data coming out of the switch block. Depending on the value of the “Doors” flow variable and therefore on the active branch in the switch block, a box plot for engine size, horse power, and scaled price of two- or four-door cars is produced.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

## Java IF (Table)

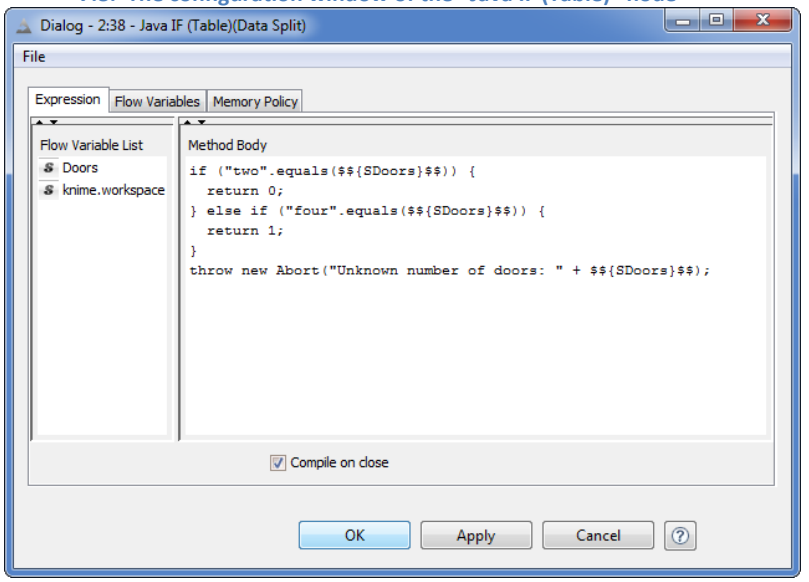
The “Java IF (Table)” node acts like an “IF Switch” node. It takes one input data table and redirects it to one of two output ports.

However it differs from the “IF Switch” node, because it can activate only one output port at a time.

The main difference between the “IF Switch” node and the “Java IF (Table)” node resides in the management of the output port activation. The “Java IF (Table)” node executes a piece of Java code with return value 0 or 1. Return value 0 activates the top output port, while return value 1 activates the lower output port.

The configuration window of the “Java IF (Table)” node resembles the configuration window of the “Java Edit Variable” node. It contains:

7.8. The configuration window of the “Java IF (Table)” node




- A “Method Body” panel for the piece of Java code to be executed at execution time
- A “Flow Variable List” panel where all flow variables available to this node are listed

Like the “Java Edit Variable” node, the Java code in the “Java IF (Table)” node operates only on flow variables. A flow variable can be inserted into the “Method Body” panel by double clicking any of the entries in the “Flow Variable List” panel.

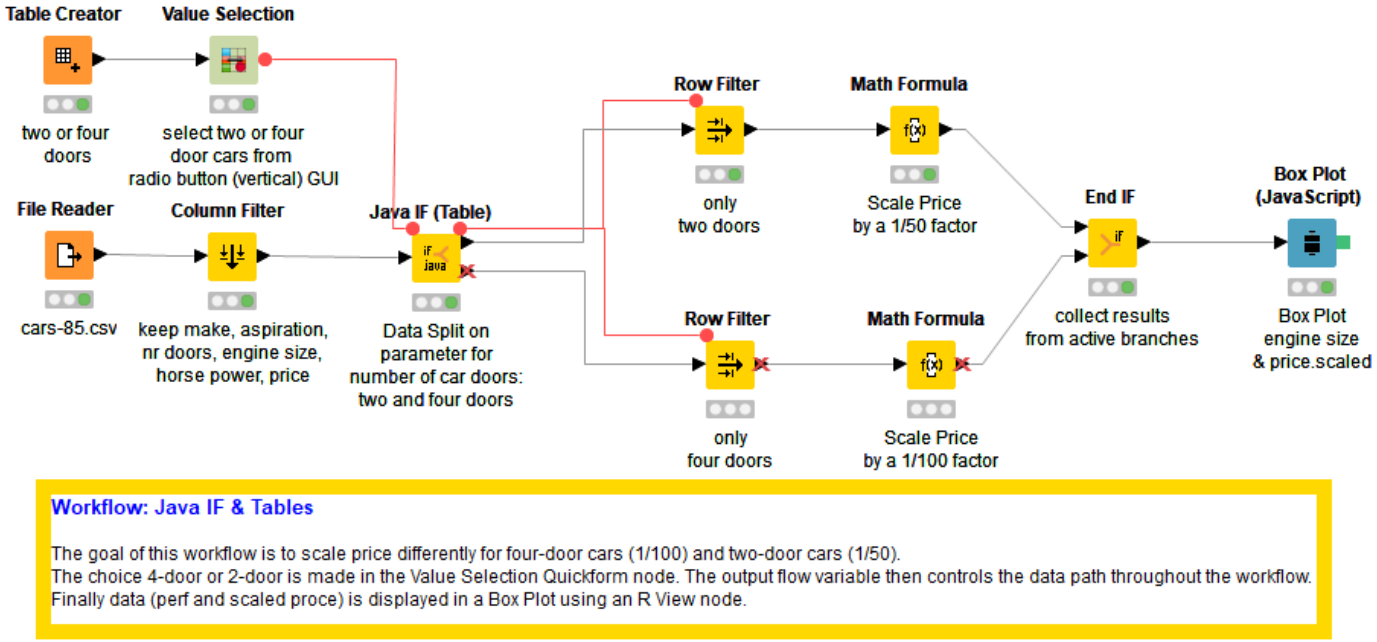
Unlike the “Java Edit Variable” node, the configuration window of the “Java IF (Table)” node cannot output any value for the return variable, just 0 or 1. Any other return value will result in an error at execution time.

**Note**. Inside the “Java IF (Table)” node configuration window, an exception handling statement “`throw new Abort();`” is required at the end of the Java code, to handle results that do not produce a 0/1 return value.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

7.9. The “Java IF & Tables” workflow




## 7.4. The CASE Switch Block

We have finally reached the most commonly used switch block in the KNIME workflows: the CASE switch block.

The “IF Switch” node and the “Java IF (Table)” node offer a choice between two and only two data flow paths. In some situations, we might wish to have more options than just two data flow paths. The “CASE Switch … (Start)” node opens a switch block which is then completed by inserting a “CASE Switch … (End)” node.

The “CASE Switch … (Start)” node offers the choice of three possible mutually exclusive data flow paths. The “CASE Switch … (End)” node collects the results from the active path(s) of the switch block. There are many types of CASE Switch nodes: while they all implement the same switch functionality, they operate on different data, such as data tables, flow variables, or models. The node name indicates which kind of data the node is working on.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

## CASE Switch Data (Start)

Any “Case Switch … (Start)” node starts a switch block with three data flow branches. It has three output ports, of which only one can be active at a time. The three output ports are indexed by an integer number, such as 0, 1, and 2.

The configuration window requires only the index of the active output port.

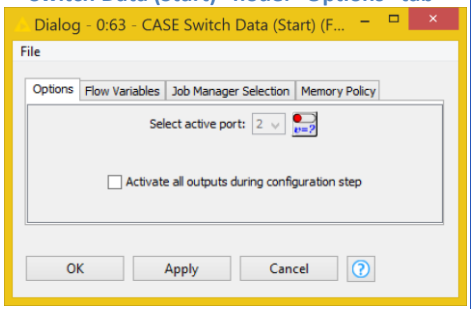
Similarly to the “IF Switch” node, the active port index can be provided manually via the “Options” tab or automatically via the “Flow Variables” tab in the configuration window.

The “Options” tab offers the list of the output port indices to choose from.

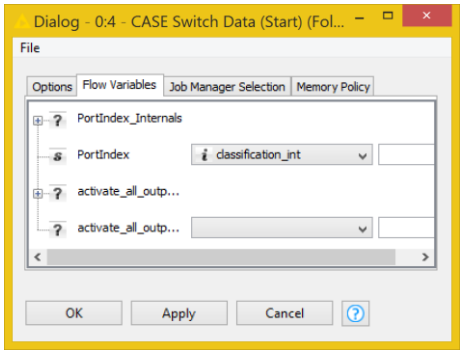
The “Flow Variables” tab overwrites the active port index parameter, named “PortIndex”, with the value of the selected flow variable. The value of a flow variable can be assigned to the output port index through the flow variable button located on the right of the “Choose Active Port” list in the “Options” tab.

The configuration window of the “CASE Switch (Model (Start)” and “CASE Switch Variable (Start)” nodes is identical to the configuration window of the “CASE Switch Data (Start)” node, depicted here.

7.10. The configuration window of the “CASE Switch Data (Start)” node: “Options” tab




7.11. The configuration window of the “CASE Switch Data (Start)” node: “Flow Variables” tab




In “Chapter7”/“CASE Switch” workflow, there is an example of a CASE switch block. This workflow was implemented to work on the “cars-85.csv” data. In particular, it was implemented to build alternatively a few data classification models on the input data set: either a Decision Tree, a Probabilistic Neural Network (PNN), or a rule system.

The decision of which model to implement is supposed to be arbitrarily made by the workflow end user. The workflow then required three separate branches: one to implement the decision tree, one to implement the PNN, and one to implement a rule system. An “IF Switch” node here was not enough: “CASE Switch” block was needed.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

To toggle between the models, a new flow variable was created and named “ClassificationModel”. This flow variable is used to indicate the type of analysis to do and can take three values only: “rule system”, “decision tree”, and “pnn”.

## CASE Switch Data (End)

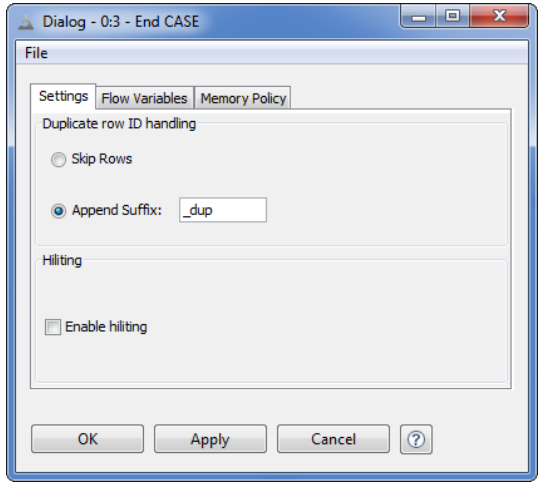
The “CASE Switch Data (End)” node closes a CASE switch block with multiple branches and collects the resulting data table(s). It has three input ports. If more than one branch is active at the same time, the “CASE Switch Data (End)” node concatenates the resulting data tables together. This is a very straightforward node, the only problem being the possible existence of rows with duplicate RowIDs.

The configuration window only requires a strategy to handle rows with the same RowIDs. Two options are offered:

1. Skip rows with duplicate IDs
2. Make those RowIDs unique by appending a suffix to the duplicate values.

Other “CASE Switch … (End)” nodes operating on flow variables or models offer a similar configuration window, with a similar selection scheme for the strategy to cope with multiple active input branches.

7.12. The configuration window of the “CASE Switch Data (End)” node



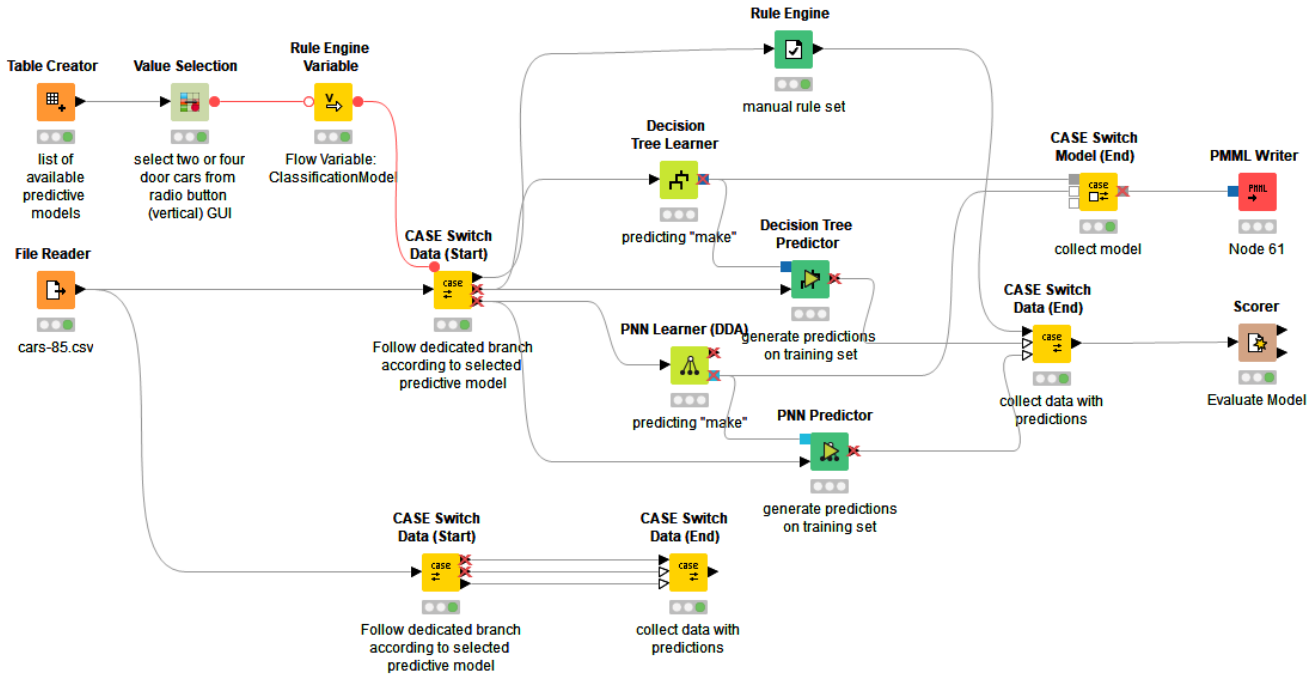

However, a “CASE Switch … (Start)” node cannot understand string values, like “decision tree” or “pnn”. It only takes integer values in order to define which output port is active. So, the string value of the “Classification Model” workflow variable had to be converted to an integer value. We used a “Rule Engine (Variable)” node, to transform “rule system” into 0, “decision tree” into 1, and “pnn” into 2 and we assigned the final value to a new flow variable “classification_int”. This last flow variable was used to control the active port in the “CASE Switch Data (Start)” node.

Next, we implemented a decision tree, a pnn, and a rule system respectively on each one of the three branches coming out of the “CASE Switch data (Start)” node. Each model produces two kinds of results: the model itself and the data tables with the predictions. In order to collect both results, we used two end nodes to close the switch block: a “CASE Switch Model (End)” node to collect the models, and a “CASE Switch Data (End)” node to collect the predicted data. The models are then saved to a file and the predictions are evaluated by a “Scorer” node.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

7.13. The “CASE Switch” workflow




## 7.5. Transforming an Empty Data Table Result into an Inactive Branch

In “Chapter7” folder, there is another workflow that implements a CASE switch block: the “Empty Table Replacer” workflow. This workflow reads the “cars-85.csv” data, keeps only the cars of a particular “make”, as defined in a workflow variable named “CarMake”. Then it follows three different branches depending on the type of wheel drive of the car: “4wd”, “fwd”, and “rwd”. Each branch keeps only the cars with this particular wheel drive type and sends the resulting data table to a “CASE Switch Data (End)” node. Finally a “GroupBy” node counts the number of cars by fuel-type for that “make” with that type of wheel drive.

In the “Empty Table Replacer” workflow, the “CASE Switch Data (Start)” node is controlled by a flow variable, named “wheeldrive”. The flow variable “wheeldrive” can only take “rwd”, “fwd”, and “4wd” string values, which are then transformed into output port indices by a “Rule Engine (Variable)” node to control the active port in the “CASE Switch Data (Start)” node.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

The data selection in the switch branches is implemented by “Row Filter” nodes on the wheel drive type. The “Row Filter” nodes are controlled by the same flow variable “wheeldrive” that controls the active branch in the switch block.

What happens, though, if one of the “Row Filter” nodes returns an empty data table? For example, if you assign the value “rwd” to the flow variable “wheeldrive”, the “Row Filter” node in the corresponding active branch produces an empty data table. Indeed, sometimes the execution of a workflow branch may result in an empty data table being created. Empty tables can still be processed by nodes further down in the workflow, like the “GroupBy” node, but they could also result in a lot of warning messages, wasted time, and maybe even data inconsistencies. To prevent this from happening, the “Empty Table Switch” node enables two different output ports: one is active when an empty data table is created and one is active when a value-filled data table is created.

### Empty Table Switch

The “Empty Table Switch” node has two output ports. It activates the lower output port for an empty input data table; otherwise it activates the top output port. This allows for the creation of an alternate branch in case of an empty input data table and avoids the execution of subsequent nodes on tables with no actual content.

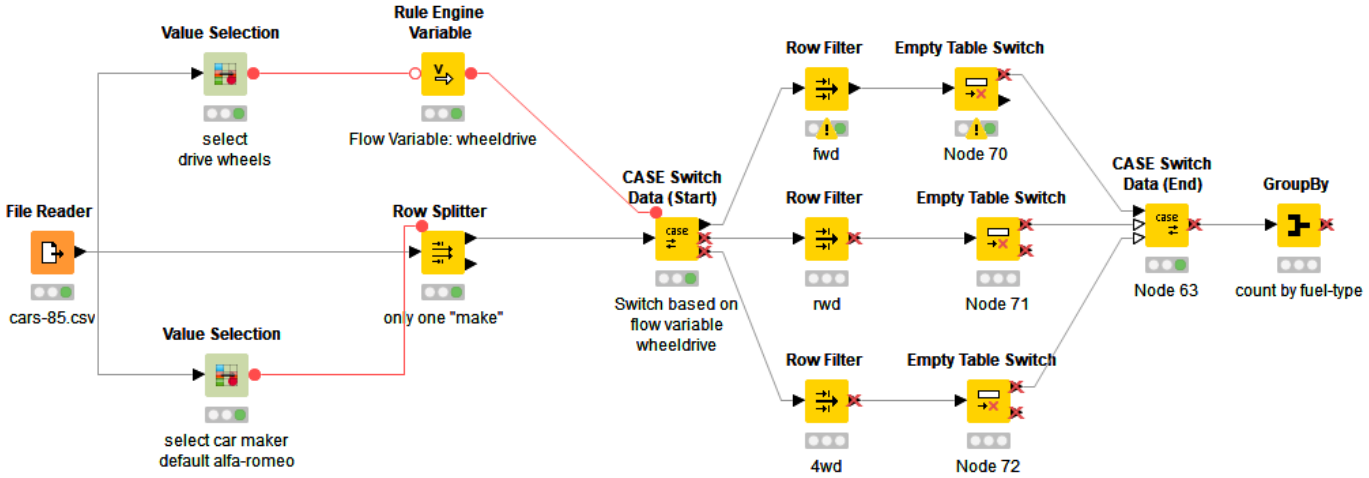
The configuration window of the “Empty Table Switch” node does not requires any setting.

In each branch of the CASE switch block, an “Empty Table Switch” node was introduced, to block further processing of the workflow in case the result of the branch is an empty data table. If we now run the workflow with the flow variable “wheeldrive” = “fwd”, the “Row Filter” node of the second branch produces an empty data table, the output of the “CASE Switch Data (End)" node collects no results, the “GroupBy” node becomes inactive and is not executed.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

7.14. The “Empty Table Replacer” workflow




## 7.6. Exercises

### Exercise 1

A situation where the “IF Switch” node is useful is when you need to perform one operation on some row entries and an alternate operation on the other rows. Let us examine this using a trivial scenario.

Create a table containing the numbers 1-10. Then, by using an “IF Switch” node, create a workflow to generate an additional integer column produced by multiplying the even numbers by a factor of 3 and the odd numbers by a factor of 10.

**Solution to Exercise 1**

The first step is to create a table containing the numbers 1-10, which is achieved using the “Table Creator” node and by labeling the integer column as “Number”. The aim here is to perform an operation on alternate lines. The rest from a division by 2 is calculated for each number by means of a “Math Formula” node. The value of this rest defines the multiplying factor, 3 or 10. An easy solution to solve this problem could have been to apply a “Rule Engine” node and transform the rest value into the corresponding multiplying factor. A “Math Formula” node then would have finished the trick, multiplying each number for the newly created multiplying factor.

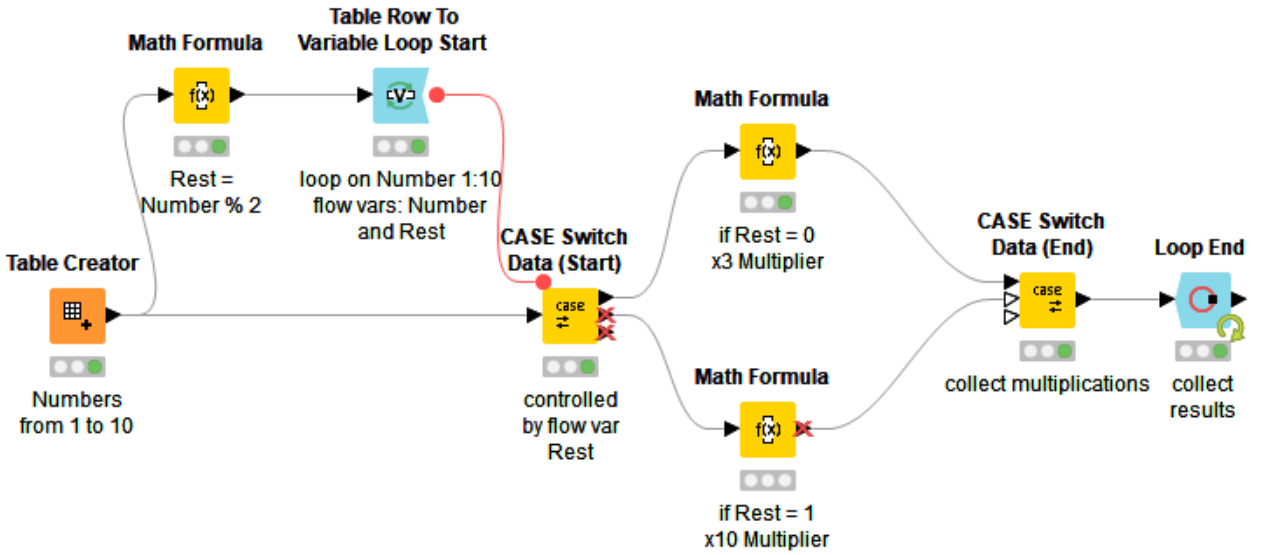


This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

However, we want to use a switch block here. As this can be viewed as a row operation, we looped over the rows, performed the right multiplication, and then concatenated the final results. We then started a loop with the "TableRow to Variable Loop Start" node, turning each number and each rest value into flow variables.

A "CASE Switch Data (Start)" node implemented the switch for different multipliers. Numbers are multiplied by 3 on one of the two branches and by 10 on the other. The active port of the "CASE Switch Data (Start)" node is controlled by the rest value: 0 enables the upper port; 1 enables the second output port. We selected a "CASE Switch Data (Start)" node, because we could control its output port with integer numbers. An "IF Switch" node would need Strings as "top", "bottom", and "both" to control its output ports. In order to transform the rest values from Double to Integer, we enabled the "Convert to Int" flag at the bottom of the "Math Formula" node configuration window.

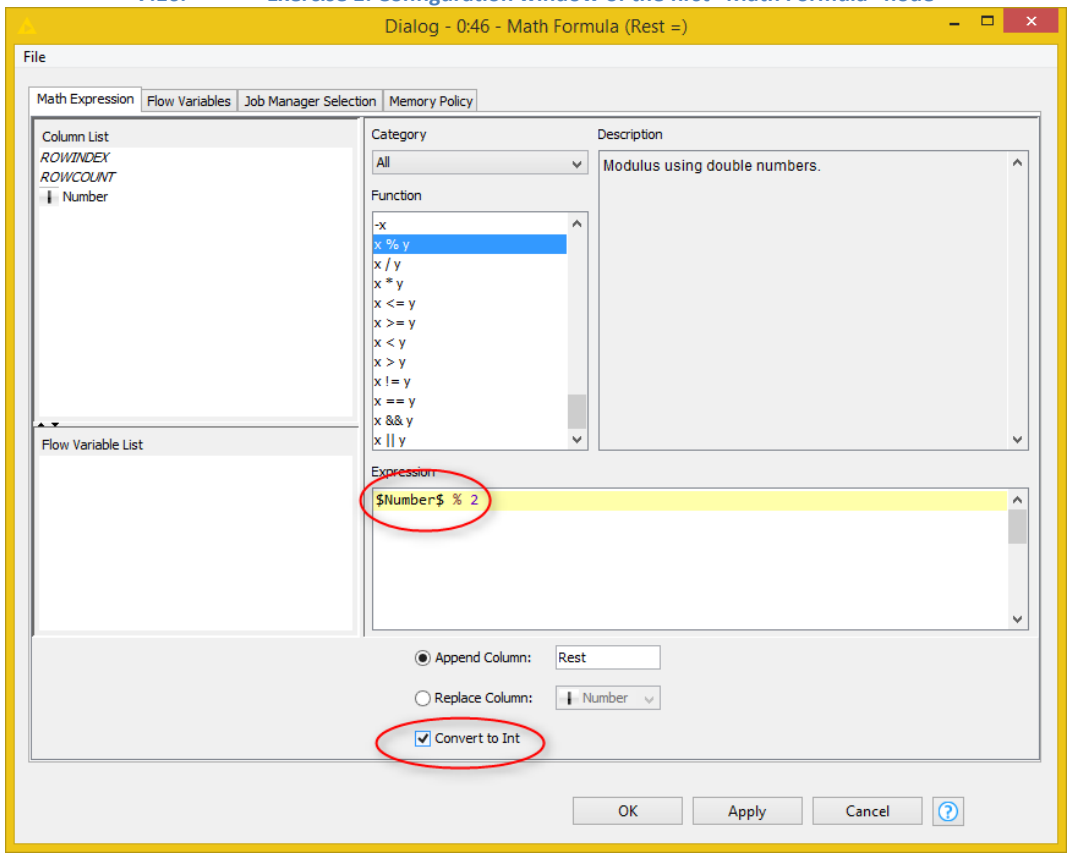
### 7.15. Exercise 1: The workflow






This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

7.16. Exercise 1: Configuration window of the first “Math Formula” node




## Exercise 2

In this exercise we show how to implement a control of the workflow processes based on time and date.

Using the “cars-85.csv” file, create a workflow that manipulates data differently depending upon the day of the week:

- On Wednesdays, write out an R box plot for engine size, horse power, and scaled price and save it as a PNG image;



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

- Do not perform any operations on the weekend
- For the remaining days keep only the “make”, “aspiration”, “stroke”, and “compression ratio” data columns.

Use a “CASE Switch” node to implement the branches for different daily data manipulation.

**Solution to Exercise 2**

First of all, we need to extract the current date and time information. We did that with a “Create Date&Time Range” node, followed by an “Extract Date&Time Fields” node and a “TableRow to Variable” node. We are interested in the DAY_OF_WEEK information, expressed as an integer 1-7, 1 being Sunday.

A subsequent “Rule Engine (Variable)” node implements the following rule:

```
$${IDay of week (number)}$$ = 1 OR $${IDay of week (number)}$$ = 7 => 0
$${IDay of week (number)}$$ = 4 => 1
TRUE =>  2;
```

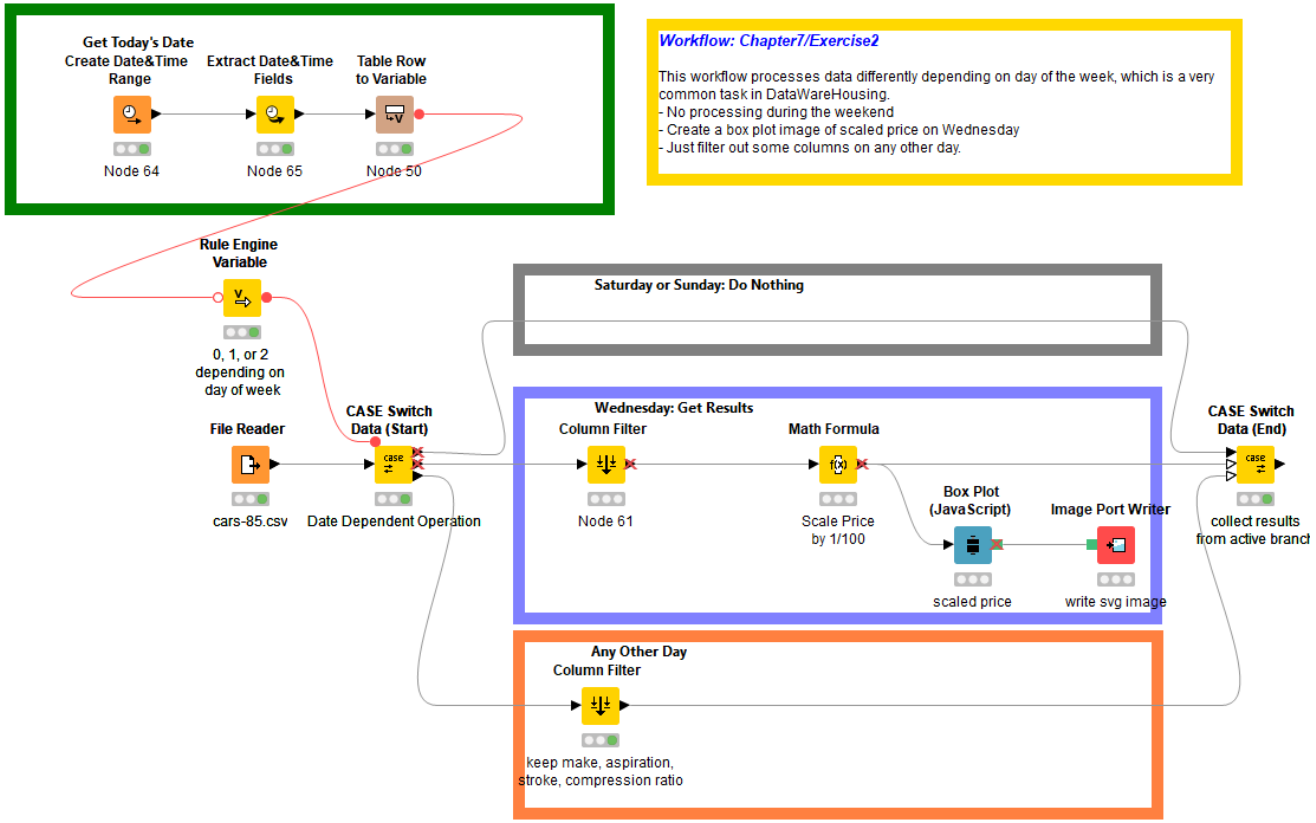
This returns 0 if it is a weekend, 1 if it is a Wednesday, and 2 otherwise. These numbers are then used to control the active port of a subsequent “CASE Switch” node.

Three branches are sprouting from the “CASE Switch” node and the final results are collected by an “End CASE” node.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

## 7.17. Exercise 2: The workflow






This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

# Chapter 8. Advanced Reporting

## 8.1. Introduction

So far, we have seen many advanced KNIME features: from the flow variables to the usage of date/time nodes; from the implementation of loops to the calling of web services, and so on. At last, the exploration of the advanced features of the KNIME Reporting tool is covered in this chapter.

The KNIME Reporting tool was described in detail in chapter 6 of the first KNIME user guide ("The KNIME Beginner´s Luck" [1]). However, a few aspects of the KNIME Reporting tool were not included, since previous knowledge of a number of advanced features was required. In this chapter, we want to describe the advanced features of the KNIME Reporting tool that were ignored in the previous book. This chapter covers:

- how report parameters can be created by means of flow variables;
- how to display parameter values in a report;
- the introduction of dynamic content into a report, by means of dedicated BIRT and Javascript functions;
- how to import images from a workflow into a report.

In order to give continuity to the example material for the first and this second KNIME user guide, the workflow example for this chapter is "New Projects", extending the workflow example named "Projects" and developed for chapter 6 in [1].

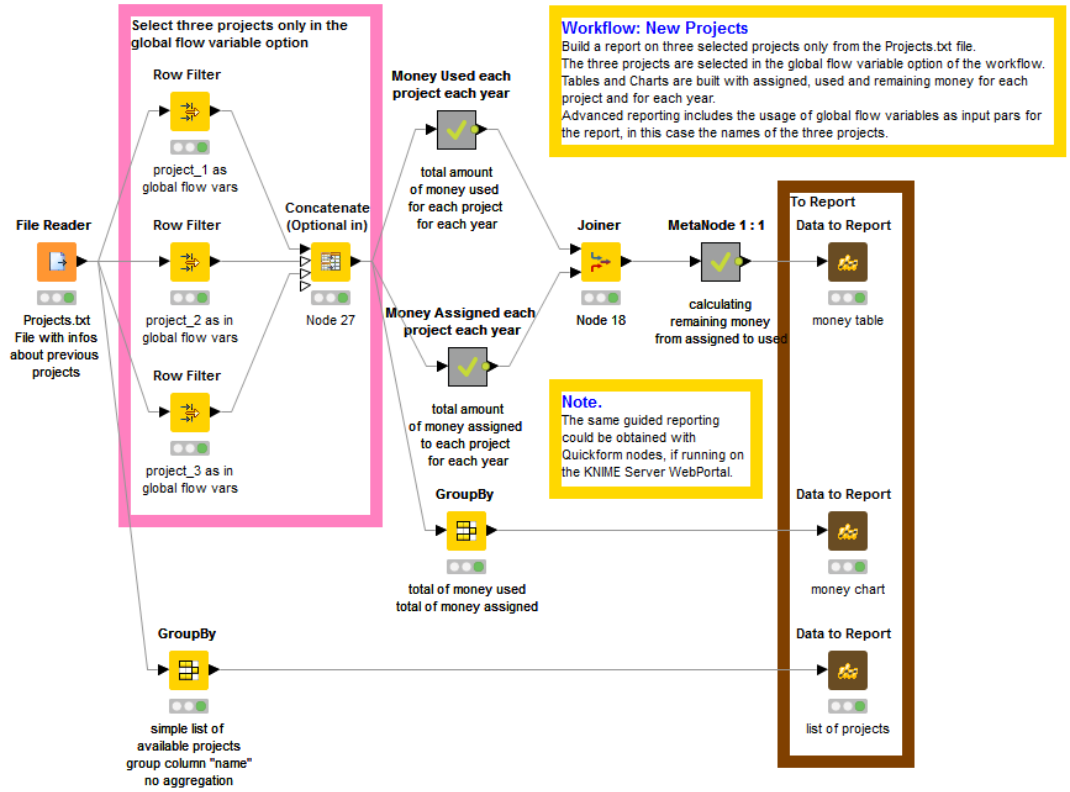
The "New Projects" workflow works on the "Projects.txt" file from folder KCBdata. The "Projects.txt" file contains a list of project names with the corresponding amount of money assigned and used for each quarter of each year between 2007 and 2009. The "New Projects" workflow builds a pivot table with the project names and the sum of the money assigned, the sum of the money used, and the sum of the money remaining (= assigned - used) for each project and for each year between 2007 and 2009. The resulting data fill a few tables and two bar charts in the associated report.

The final version of the "New Projects" workflow is displayed in figure 8.1 and the resulting pivot data table, from the "money table" node, in figure 8.2.



This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

### 8.1. The "New Projects" workflow




### 8.2. The data table, from the “money table” node, of the "New Projects" workflow

Table View - 0:15 - Data to Report(money table)

File Hilite Navigation View Output

| Row ID | S name | D used 2007 | D used 2008 | D used 2009 | D assigned 2007 | D assigned 2008 | D assigned 2009 | D remain 2009 | D remain 2008 | D remain 2007 |
|---|---|---|---|---|---|---|---|---|---|---|
| Row0 | Blue | 1,360 | 1,277 | 1,565 | 1,360 | 1,277 | 1,565 | 0 | 0 | 0 |
| Row1 | Gobi | 1,203 | 1,424 | 1,740 | 1,203 | 1,424 | 1,740 | 0 | 0 | 0 |
| Row10 | White | 860 | 1,087 | 1,420 | 860 | 1,087 | 1,420 | 0 | 0 | 0 |
| Row2 | Kalahari | 630 | 800 | 1,192 | 630 | 800 | 1,192 | 0 | 0 | 0 |
| Row3 | Kara Kum | 800 | 888 | 1,516 | 800 | 888 | 1,516 | 0 | 0 | 0 |
| Row4 | La Guajira | 1,020 | 1,404 | 1,496 | 1,020 | 1,404 | 1,496 | 0 | 0 | 0 |
| Row5 | Mojave | 1,800 | 1,819 | 1,860 | 1,800 | 1,819 | 1,860 | 0 | 0 | 0 |
| Row6 | Patagonia | 864 | 2,098 | 1,359 | 864 | 2,098 | 1,359 | 0 | 0 | 0 |
| Row7 | Sahara | 806 | 1,457 | 1,495 | 806 | 1,457 | 1,495 | 0 | 0 | 0 |
| Row8 | Sechura | 3,200 | 2,966 | 3,940 | 3,200 | 2,966 | 3,940 | 0 | 0 | 0 |
| Row9 | Tanami | 453 | 0 | 453 | 453 | 0 | 453 | 0 | 0 | 0 |



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

## 8.2. Report Parameters from global Flow Variables

The report of the “New Projects” workflow contains three tables, each one with the data of all 11 projects, and two bar charts. Due to the high number of projects, the tables and the bar charts in the report can be quite hard to read. In this chapter we reduce the number of projects to a maximum of three. With only the data of three projects, the tables and the bar charts become much easier to read. In order to make the reports parametric, we created three report parameters containing the names of the three projects to be displayed in each report.

We created three flow variables, named respectively “project_1”, “project_2”, and “project_3”, to hold the names of the projects to be displayed in the report. Initially we worked with global flow variables, but later we moved to local flow variables created with Quickform nodes. How to create and use flow variables in a workflow is described earlier on in this book. We set “Sahara”, “Blue”, and “White” as the three default values for the three flow variables.

We then changed the “New Projects” workflow accordingly. After the “File Reader” node, we placed three “Row Filter” nodes. The first “Row Filter” node selects the data for the project contained in “project_1” flow variable; the second “Row Filter” node selects the data for the project contained in “project_2” flow variable: and the third “Row Filter” node selects the data for the project contained in “project_3” flow variable. Each “Row Filter” node was configured to use the pattern matching feature and to match the values in the data column “names” with the value in the selected flow variable. The data tables at the output ports of the three “Row Filter” nodes were then concatenated together by a “Concatenate (Optional in)” node.

The data table at the output port of the “Concatenate (Optional in)” node was subsequently fed into the two blocks that calculate the money used and assigned by and to each project each year. For space reasons, we collapsed these two blocks in two meta-nodes. In order to do that, we selected the three nodes of the group, right-click any of them, and selected the “Collapse into a meta-node” option. The new meta-node was automatically created and filled in with the selected nodes. As a new name we assigned the corresponding annotation content to each one of the meta-nodes.

As in the previous version of the “New Projects” workflow, the data tables resulting from the two meta-nodes were joined together, the amount of the remaining money was calculated, and the final data table was exported for reporting with a “Data to Report” node named “money table”. During execution, this “Data to Report” node exports its input data table into the report data set “money table”, to be subsequently used to build the tables in the report.

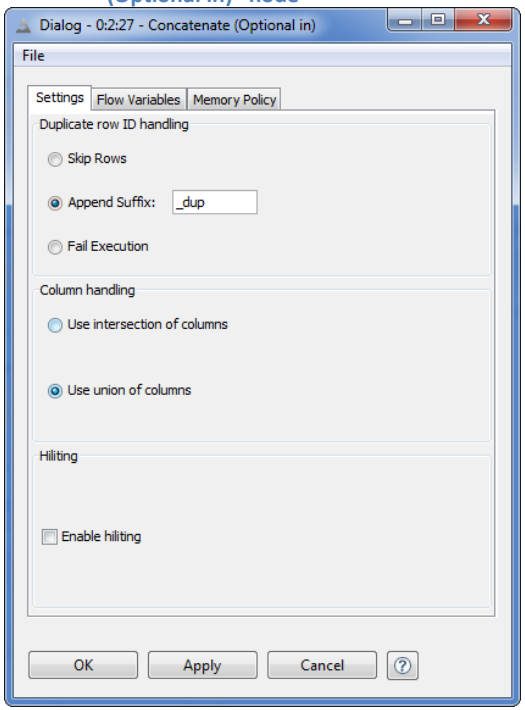
A “GroupBy” node was placed after the “Concatenate (Optional in)” node to calculate the total amount of money assigned (used by) each project each year. Its output data table was marked for reporting with a “Data to Report” node named “money chart”. This “Data to Report” node generates the data set called “money chart” to be used for the bar charts in the report layout.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

## Concatenate (Optional in)

8.3. Configuration window of the "Concatenate (Optional in)" node




The "Concatenate (Optional in)" node is a variation of the "Concatenate" node.

Like the "Concatenate" node, it concatenates up to four data tables.

The node has "optional" input ports (displayed as gray outlined triangles). Optional ports don't have to be connected for the node to execute.

The configuration window requires:

- A method to handle rows with duplicate IDs
- A choice between outputting the intersection or the union of the input data columns
- A flag to enable hiliting

The "New Projects" workflow programmatically selects some data from the original data set, on the basis of the values in the flow variables, "project_1", "project_2", and "project_3". We would like to transfer this parameterization to the report, so that we could also run the report only for a maximum of three selected projects at a time. Let's start by using only global flow variables.

We created the three required global flow variables. Now let's execute the "New Projects" workflow and switch to the reporting environment. We do not change the report layout; we just select "Run"-> "View Report" -> "As HTML" from the top menu. A new tab appears on your default web browser, named "PARAMETER SELECTION PAGE" and asking for the report parameter values. Indeed the report now is controlled by three parameters: one named "project_1", one named "project_2", and one named "project_3".

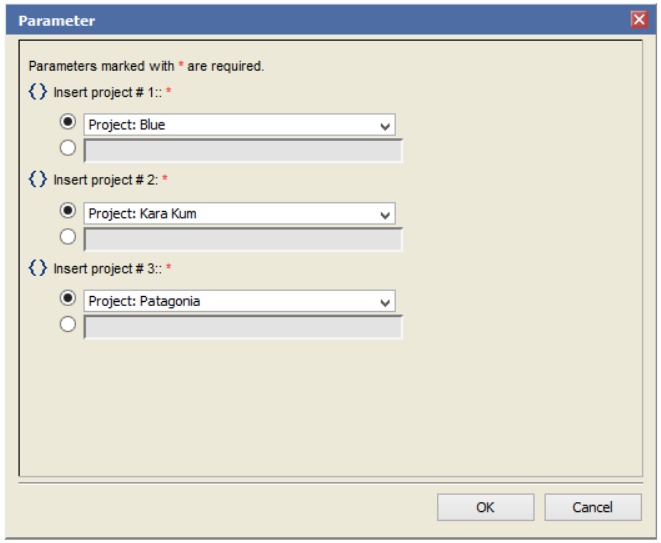


This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

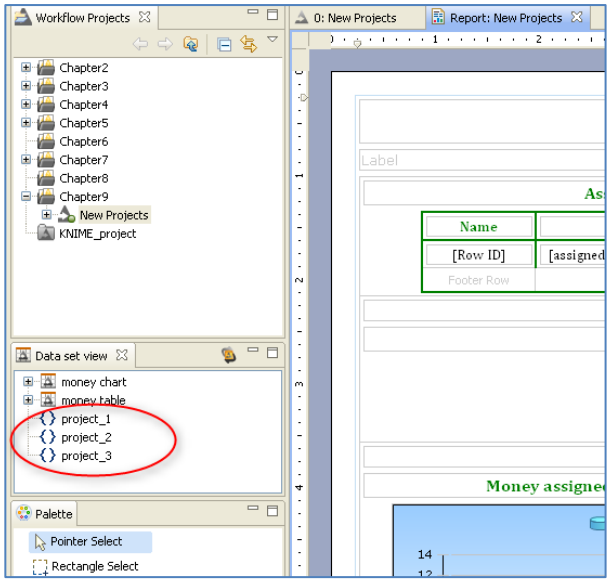
**Note**. When switching from the KNIME Analytics workbench to the reporting environment, the software looks for all “Data to Report” (or “Image to Report”) nodes, transfers their input data tables/images into report data sets, names them from the original comment under the “Data to Report” nodes, looks for all global flow variables, and transforms the global flow variables into report parameters with the same name and the same default values.

The global flow variables found in the underlying workflow are displayed as report parameters in the panel named “Data Set View” in the middle on the left (Fig. 8.5).

8.4. The “PARAMETER SELECTION PAGE” tab in the web browser to insert the Report Parameter values




8.5. The "Data Set View" panel in the reporting perspective




In the “PARAMETER SELECTION PAGE” window, we insert three values for the three report parameters and press “OK”. The result is a web page with tables and bar charts built only on the data of the three selected projects, which is exactly the kind of report we wanted to build.

If we want the report to display the data for only two projects, we give the third workflow variable/report parameter the name of a non-existing project. The third “Row Filter” node in the workflow then produces an empty data table and the report only shows the data for 2 existing projects.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

## 8.3. Customize the "PARAMETER SELECTION PAGE" web page

The default "PARAMETER SELECTION PAGE" window is very anonymous and not very informative. It is possible to customize it:

- To show a nicer layout,
- To be more informative,
- To offer only a number of pre-defined answers, so that user's errors, like typos, are less likely to happen.

In order to change the request display for a specific parameter, we double-click the name of this report parameter in the "Data Set View" panel located on the left, for example "**{}** project_1".

A new window, named "Edit Parameter" opens. This window allows customizing the request display for the selected report parameter. In fact, we can customize:

- The "Prompt Text" from the original "Please enter a value …";
- The "Display Type" from the default text box;

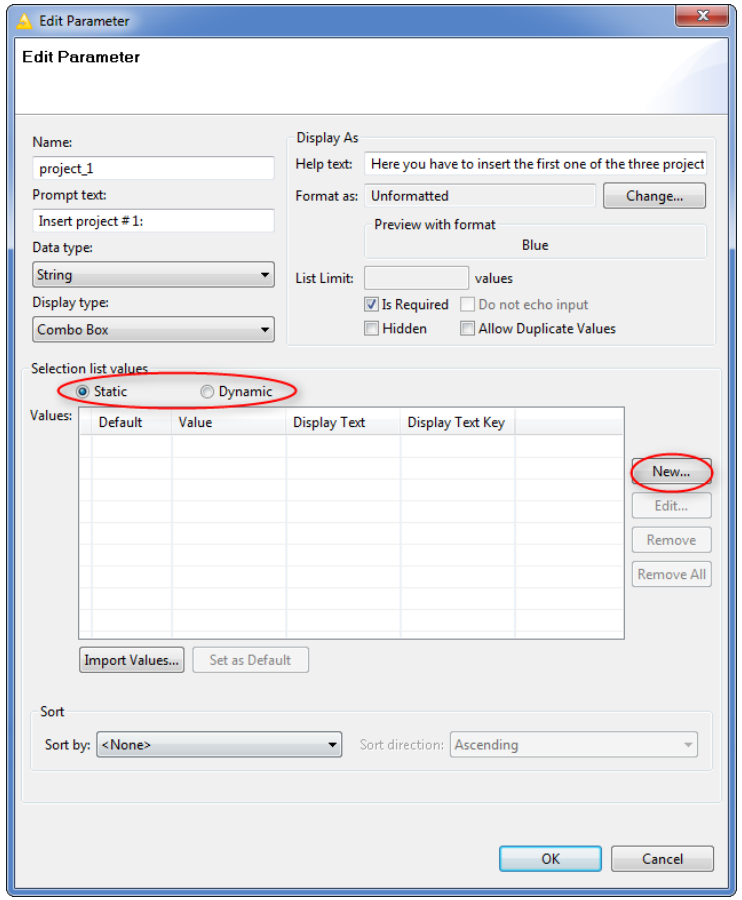
The "Display Type" in particular offers a few options on how to enter the report parameter: via a "Text Box", a "List Box", a "Combo Box", and a "Radio Button".

The "Text Box" is a box in which to type free text and therefore it is the most error prone option.

The "List Box" and the "Combo Box" both offer a list of pre-defined values to the user. List Boxes and Combo Boxes are safer against typos than a simple Text Box. The difference between the "List Box" and the "Combo Box" lies in the fact that the first one allows multiple values, while the second one does not.

Finally, the "Radio Button" only allows a mutually exclusive choice between two possible values. Very minimal error is possible here on the user's side.

8.6. The "Edit Parameter" window






This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

The parameters “Name” and “Data Type” of the parameter can also be changed, generating in this way a new report parameter.

For our three global flow variables / report parameters, we provided:

- “Insert the first project:” as prompt text,
- A “Combo Box” as the modality to enter the parameter value,

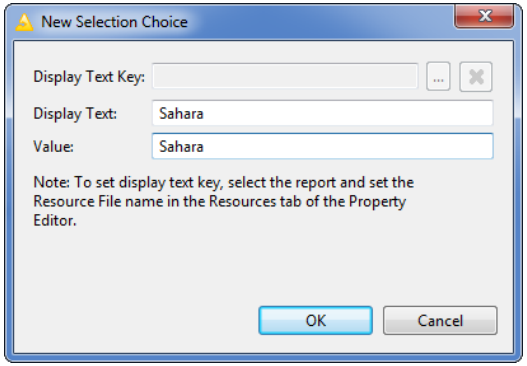
At this point, we needed to fill in the “Combo Box” with a list of pre-typed values. There are two options to pass a list of values into a “Combo Box”: a static list or a dynamic list of values. A static list of values is a list of values that was created once and can only be edited manually. A dynamic list of values is created on the fly from some data sets when running this report.

**Static List of Values**

To fill in the list with static values, in the “Edit Parameter” window:

- Select the “Static” radio button in the “Selection List Values” frame;
- Click the “New” button on the right;
- The “New Selection Choice” window opens;
- In the “New Selection Choice” window:
    - Enter the list value and its display text;
    - Click “OK”;
- The pair (value, display text) appears in the list of values in the “Enter Parameter” window;
- Perform the same operations for all remaining values to be in the list

8.7. The "New Selection Choice" window




**Note**. The list value and the display text might not be the same. For example, the report parameter might refer to month abbreviations. Since month abbreviations are locale dependent, you can use the month’s short form as the “Value” list item in your preferred locale and the full month text as the “Display Text” list item, e.g. “Value” = “Mar” or “Mrz” depending on your locale and “Display Text” = “March”. “Value” is not visible in the “PARAMETER SELECTION PAGE” page.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

**Dynamic List of Values**

To enter all values in the list manually could be time consuming and even impossible if the list of values is really long. An alternative is to fill the list dynamically from the data of one of the report data sets. In order to fill the list of values into the "ComboBox" dynamically, that is from one of the report data sets, go to the "Edit Parameter" window:

- Select the "Dynamic" radio button in the "Selection List Values" frame;
- Select the report data set with the desired value list;
- Select the column of the data set that contains the values for the list;
- Select the column of the data set that contains the display texts for the list;
- Click "OK".

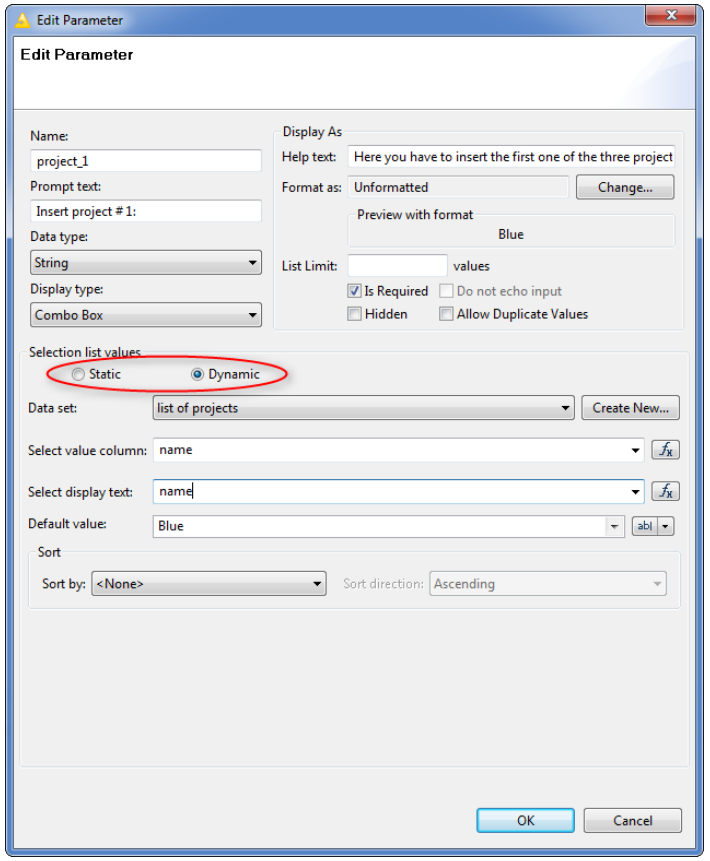
Every parameter can be so customized in the parameter window in the "PARAMETER SELECTION PAGE".

For the "New Projects" report, we decided to use a "ComboBox" to select the name of the project for each one of the three report parameters. We decided to fill in the "ComboBox" dynamically. In fact, since we were not sure that the list of projects will not change in the future, we preferred the dynamic solution rather than the static one.

We introduced a new "GroupBy" node, named "list of projects", in the "New Projects" workflow. The "GroupBy" node was supposed to collect all project names and make them available in the report for the "Combo Box" list of values. The "GroupBy" node was then connected to a "Data to Report" node named "list of projects" as well.

Going back to the reporting environment, a new "list of projects" data set was shown in the "Data Views" panel. Then for each one of the three report parameters, we selected:

8.8. The "Edit Parameter" window when the option dynamic is selected to fill the list of values for the "ComboBox"






This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

- The “ComboBox” as “Display type”;
- The “Dynamic” option to fill in the “ComboBox” with a list of values;
- “list of projects” as “data set”;
- “name” as the column containing the list of values;
- “name” again as the column containing the “Display text” strings;

The window in the “PARAMETER SELECTION PAGE” shown in figure 8.4 is generated. For each parameter there is a “ComboBox” with a list of pre-defined values and its “Prompt Text" says “Insert project # <n>”.

**Note**. Below the “ComboBox” there is still a “TextBox” where free text can be entered. This can be used to type in a text that is not offered by the “Combo Box” list. For example, we could use it to type the name of a non-existing project to limit the report to only two or even one project.

## 8.4. The Expression Builder

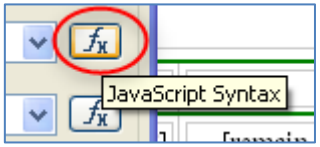
In the “Edit Parameters” window, at the right of the “select value column” and of the “Select display text” boxes, there is a button with an “fx” symbol, as depicted in figure 8.9. This button can be often found at the side of text/combo boxes in the KNIME Reporting tool to access the “Expression Builder” window.

The “Expression Builder” window helps to build a customized text or mathematical expression around some column values or just by itself.

For example, we might want to display the following text for each workflow variable value: “Project: <project-name>”. This expression can be built by means of the “Expression Builder” window. In the “Expression Builder” window we find:

- In the top panel, the “Expression Builder” editor, where expressions can be assembled;
- In the middle panel, a list of operators that can be used for mathematical expressions;
- In the bottom panel, the lists of functions, operators, and column values that can be inserted into the Expression Builder Editor.

8.9. Button to the "Expression Builder" window



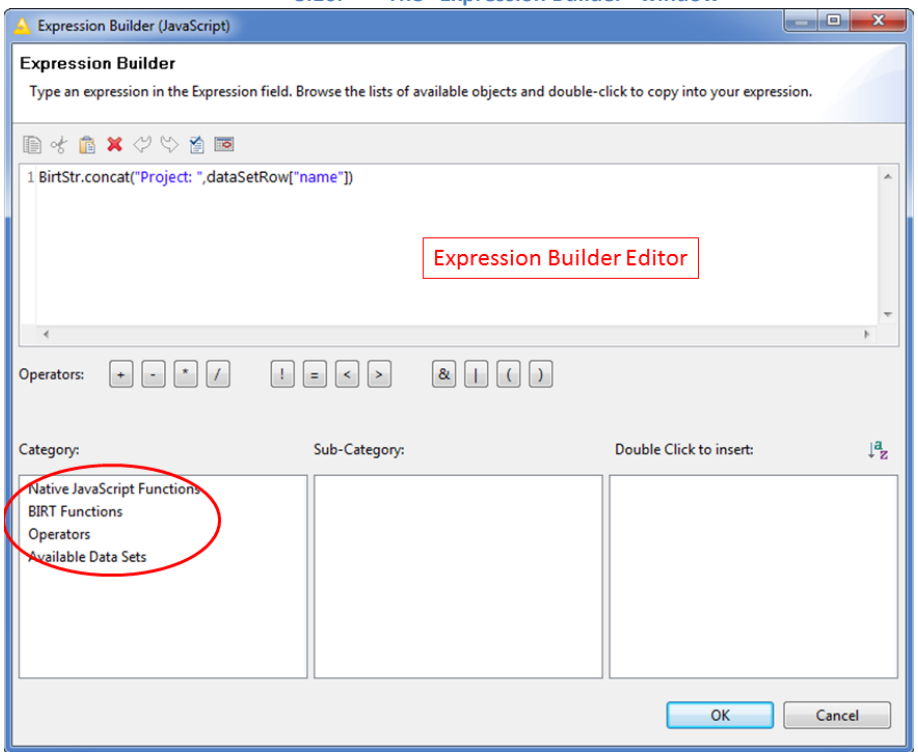



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

Let's have a closer look at the bottom panel. On the left we find the function categories to choose items from. The last item is the "**Available Data Sets**" category and offers a link all available data sets and their columns for this particular "Expression Builder" instance. The data sets are listed in the center sub-panel and the corresponding data columns in the right sub-panel.

**Note**. This Expression Builder originates from the "Edit Parameter" window, where the "list of projects" data set was selected. Therefore the only available data set here is "list of projects". In other instances of the Expression Builder originating from different items in the report, more data sets might be available.

8.10. The "Expression Builder" window




The "**Operators**" category offers a number of mathematical/logical operators. An extract with the most frequently used operators is shown in the middle panel.



This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

The "**BIRT Functions**" category includes a number of BIRT specifically designed functions in the fields of finance, date/time manipulation, duration, mathematics, string manipulation, and string or number comparison.

The "**Native JavaScript Functions**" category includes a number of JavaScript functions. These turn out to be particularly useful when the report is created using the HTML format.

Double-clicking an item in a sub-panel, such as a column name, a BIRT function, an operator, or a Java Script function, automatically inserts the item into the Expression Builder editor above.

For example, in order to get the display text as: "Project: <project-name>":

- double-click one of the report parameters in the "Data Set" panel;
- fill the "Edit Parameter" window as in figure 8.8;
- click the "fx" button at the right of the "Select display text" box;
- In the "Expression Builder" window:
    - open the "BIRT Functions" category in the left panel and the "BirtStr" sub-category in the center panel;
    - double-click the `concat()` function in the right panel;
    - the text `BirtStr.concat()` appears in the Expression Builder Editor;
    - position the cursor in between the parenthesis;
    - type the text `<"Project: ",>` (the quotation marks are needed for Java-style strings);
    - open the "Available Data Sets" category in the left panel and then the "list of projects" sub-category in the center panel;
    - double-click the "name" column;
    - the text `BirtStr.concat("Project: ",dataSetRow["name"])` appear in the expression builder editor;
    - click "OK"
- click "OK" back in the "Edit Parameter" window

This operation was repeated for all three report parameters, "project_1", "project_2", and "project_3.

**Note**. In the Expression Builder editor text strings must be typed in quotation marks (Java-style). Text items in a `concat()` function have to be separated by a comma.

In this section we have seen how the "Enter Parameters" window can be customized, so that less errors and typos occur on the user's side. More complex customizations are possible by exploiting the BIRT dedicated functions and the native JavaScript functions more in depth.



This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

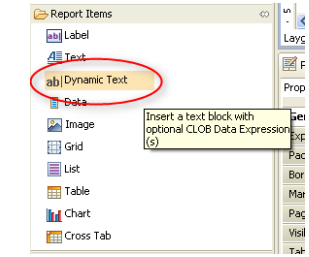
## 8.5. Dynamic Text

It is good practice to display the report parameter values at the top of the report. In this way, the reader knows immediately under which conditions the displayed data was collected and processed. The report parameter values, however, are different at each run. Therefore we need a dynamic item that updates the top description at each run.

A dynamic report item is the “Dynamic Text”, which can be found in the “Report Items” list in the bottom left panel. The “Dynamic Text” item displays a small text, built with the “Expression Builder”. The “Expression Builder” window generated by a “Dynamic Text” offers the list of the available report parameters in addition to the lists of operators and BIRT and Java Script functions, as seen in the previous section.

Let’s drag and drop a “Dynamic Text” item from the “Report Items” list to the top of the report layout, to directly under the title. The “Expression Builder” opens. The “Expression Builder” editor is still empty. In the bottom panel, on the left, we can now see 4 categories: “Report Parameters”, “Native JavaScript Functions”, “BIRT Functions”, and “Operators”.

8.11. The "Dynamic Text" item in the "Report Items" panel

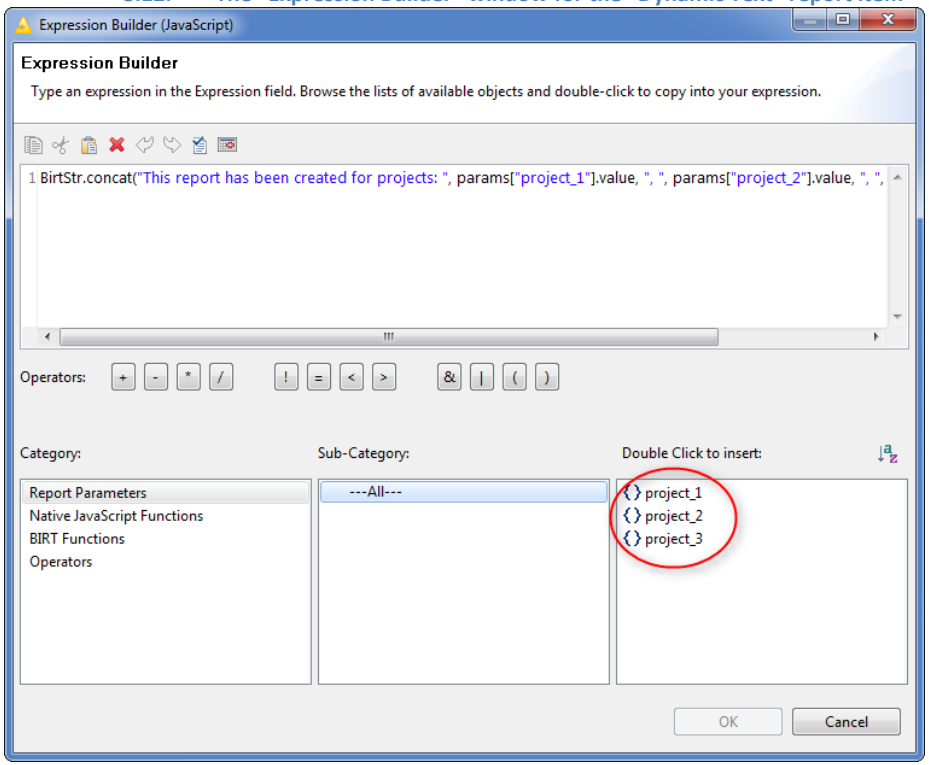



The “Report Parameters” category is new, while the “Data Sets” category has disappeared in comparison with the previous instance of the “Expression Builder”. The “Report Parameters” category contains the list of all available report parameters.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

8.12. The "Expression Builder" window for the "Dynamic Text" report item




We wanted to introduce a text at the top of the report, something like: “This report has been created for projects: <project-name1>, <project-name2>, <project-name3>”.

To do that, we used the BIRT function `concat()` once more. In the “BIRT Functions” category on the left, we selected the “`BirtStr`” sub-category in the center panel and then the `concat()` function in the right panel. In the “Expression Builder” editor in the top panel the text `BirtStr.concat()` appeared. Inside the parentheses, we typed the text <`"This report has been created for projects: "`, > with the text in quotation marks and followed by a comma.

Then, we selected the “Report Parameters” category, the “All” sub-category, and double-clicked “project_1”. `params["project_1"].value` appeared in the “Expression Builder” editor. We typed a comma after that and the text <`", "`, > in quotation marks and again followed by a comma. The same sequence of operations was repeated for the remaining two workflow variables “project_2” and “project_3”, till the following text was present in the “Expression Builder” editor (Fig. 8.12):

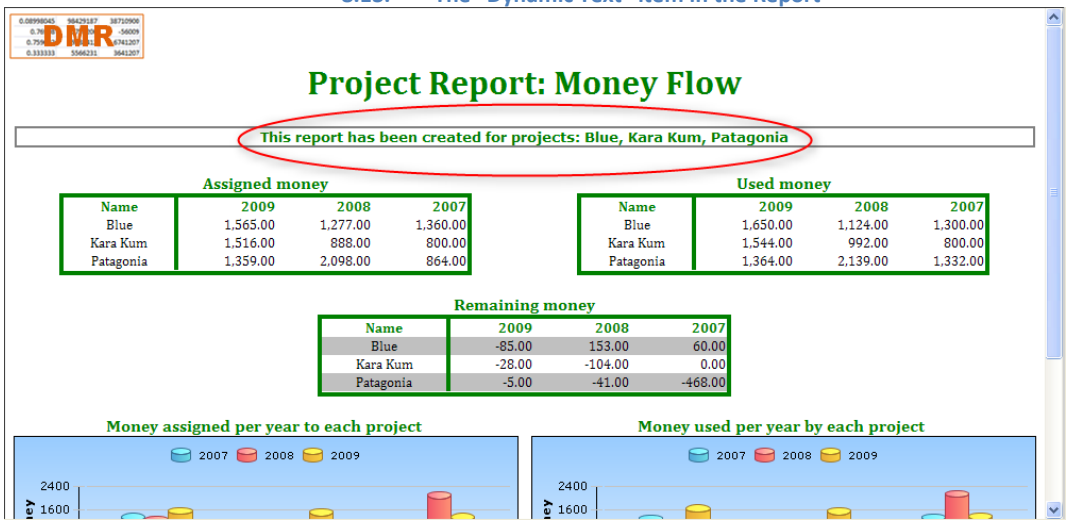


This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

```
BirtStr.concat(   "This report has been created for projects: ",
                   params["project_1"].value, ", ", params["project_2"].value, ", ", params["project_3"].value)
```

We clicked “OK”, the “Expression Builder” window closed, and the “Dynamic Text” item was set in the report with this text. The “Dynamic Text” item was then formatted as centered, with green font color and bold style.

8.13. The "Dynamic Text" item in the Report




**Note**. Only global flow variables get imported as parameters into the report environment. Local flow variables are created and stay inside the workflow.

So, creating flow variables through Quickform nodes, produces a step-wise Wizard on a web browser when the workflow is run on a WebPortal, coltrolling the report indirectly. However the flow variable values are not accessible from inside the report. If you want to write a dynamic text including the local flow variables generated inside the workflow, you need to export them as data set with a “Data To Report” node.

## 8.6. BIRT and JavaScript Functions

In the “BIRT Functions” and “JavaScript Functions” categories for the “Dynamic Text” item, there are other useful functions, for example the date and time functions and the mathematical functions. Indeed, it is good practice to include the creation date in the report, just to be sure not to read an excessively old report. We could insert the current date as a running title in the report, to the right of the logo.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

In the reporting environment, in the “Master Page” tab, in the top header frame, we inserted a grid with two columns and one row. The left cell already included the logo [1]. In the right cell we wanted to insert a “Dynamic Text” item displaying the current date at report creation.

Let’s drag and drop a “Dynamic Text” item from the “Report Items” list panel on the bottom left to the right grid cell in the top header frame of the “Master Page” tab. The “Expression Builder” window opens. In order to display the current date we have many options.

We can choose for example a straightforward “BIRT Functions” -> “BirtDateTime” -> “Today()” function. “Today()” returns a timestamp date which is midnight of the current date. The function in the “Expression Builder” window then runs as `BirtDateTime.today()` and the current date in the running title of the report looks like in figure 8.14.

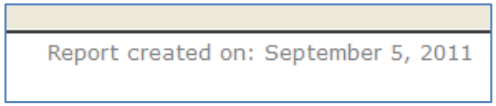
8.14. Current Date from "Today()" BIRT Function


The “Today()” function offers no formatting options. If we want to have the current date in a customized format we must build that ourselves. “BIRT Functions” -> “BirtDateTime” offers a number of functions to extract components from a DateTime object, like `day(DateTime)`, `month(DateTime)`, `year(DateTime)` and so on. We could extract the date components and combine them with a `BirtStr.concat()` function to get the desired date format. After extracting date parts from the result of the `Today()` function and combining them with a `concat()` function, we get, for example, the following formula in the “Expression Builder” window:

```
BirtStr.concat(   "Report created on: ",
                  BirtDateTime.month(BirtDateTime.today(), 2), " ",
                  BirtDateTime.day(BirtDateTime.today()), ", ",
                  BirtDateTime.year(BirtDateTime.today()))
```

8.15. Customized Current Date from "Today()" BIRT Function

and the following current date format in the report running title:




The “BIRT Functions” -> “BirtDateTime” sub-category also offers a number of functions to add and subtract time from a DateTime object. For example, the running title could use the following formula with the `addQuarter()` function:

```
BirtStr.concat("Report valid from: ",
BirtDateTime.today(),
```



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

```
" to: ",
BirtDateTime.addQuarter(BirtDateTime.today(),1)
)
```

8.16. Use of the "addQuarter()" function in the Running Title

which produces a date on the running title as follows.


**Note**. Notice the change in date locale between figure 8.14 and figure 8.16. The change is due to the introduction of the `concat()` function, which automatically sets the locale for the DateTime values as well.

In this section we have only shown the BIRT functions related to the DateTime object, because they are the most commonly used. However, the "BIRT Functions" category offers many built-in functions for mathematical expressions, finance quantities, string manipulation, etc …

If the report output document is HTML, we could also take advantage of the built-in JavaScript functions, which are more articulated and varied than the built-in BIRT functions.

## 8.7. Import Images from the underlying Workflow

The last advanced reporting feature that we would like to discuss in this chapter is the dynamic image transfer between a workflow and its report. In this chapter, we take the data generated by the "New Projects" workflow and we attach a red or green traffic light to each project, depending on project performance in 2009. That is, if the "remain 2009" is below 0 we depict a red traffic light close to the project name in the report table; if "remain 2009" is above or equal to 0 we append a green traffic light.

To illustrate this concept, we have created a new workflow in the workflow group "Chapter9" and we named it "traffic lights". We have also created two images: one with a red traffic light (file "red.png") and one with a green traffic light (file "green.png"). These images are available in the Download Zone in the "images" folder. We also generated a new file with the data table built in the workflow developed in the previous sections. The file was named "Totals Projects.csv" and can be found in folder KCBdata.

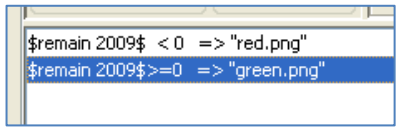


This copy of the book "KNIME Advanced Luck" is licensed to: Laszlo Sallo

The “traffic lights” workflow starts with a “File Reader” node to read the “Totals Projects.csv” file, with the project name as RowIDs. A “Column Filter” node then removes all columns besides the RowIDs with the project names and the “remain 2009” column. The “remain 2009” column contains the difference between the money assigned to each project and the money used by each project in year 2009.

A “Rule Engine” node appends the name of the image file of the red\green traffic light in a new column named “image”, depending on the value of the “remain 2009” column. That is, column “image” contains the name of the image file of the red traffic light (“red.png”), if “remain 2009” is below zero, and the name of the image file of the green traffic light (“green.png”), if “remain 2009” is above or equal to 0.

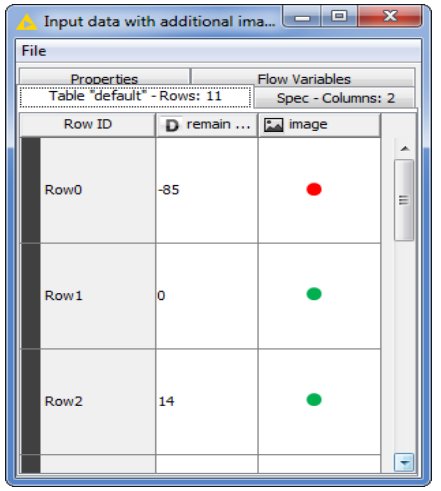
8.17. Rules implemented by the "Rule Engine" node




The path to the folder containing the image files is implemented as a local flow variable through a “String Input” Quickform node and set to:

```
knime://knime.workflow/../../KCBdata/images
```

8.18. Output Data Table from a "Read Images" Node




The image file name in the “image” column is then transformed into the full path of the image file by a “String Manipulation” node. The “String Manipulation” node combines the kflow variable “image_path” with the values in the “image” column. The new path values replace the old filename values in the “image” column.

A “Read Images” node reads the images pointed by the file paths in the “image” column. For each row, a red or a green traffic light image is then read.

Finally, the data table is fed into a “Data to Report” node to be exported into a report data set. The “Data to Report” node was named again “money table”.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

## Read Images

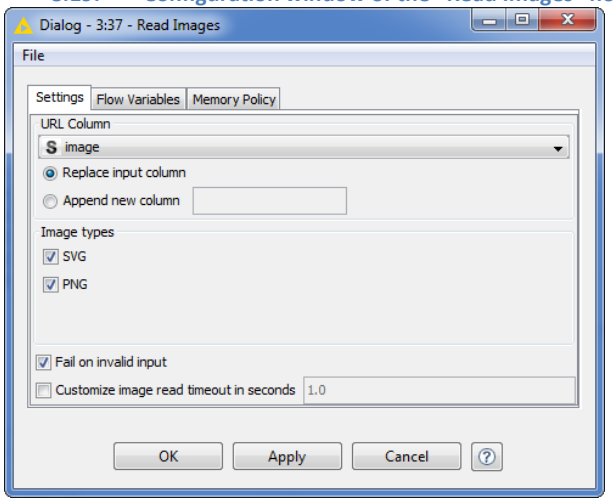
The “Read Images” node reads image files from the URLs stored in a selected column of the input data table.

The “Read Images” node is located in `“IO” -> “Read” -> “Other”`.

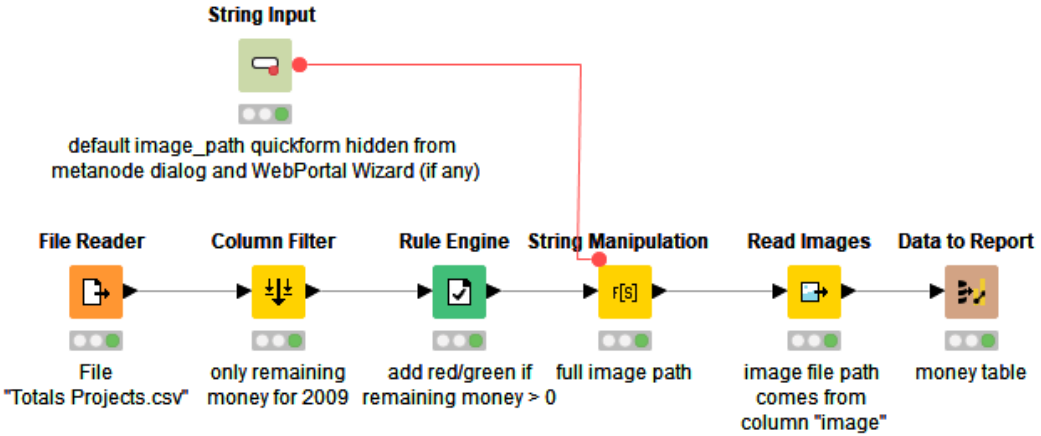
The configuration window requires:

- The column containing the URLs of the PNG image files
- A flag to stop or continue in case of a reading error
- A radio button to create a new column or replace an existing one
- The image format(s)
- A read time out

8.19. Configuration window of the “Read Images” node




8.20. The "traffic lights" workflow




After the “traffic lights” workflow was finished, we switched to the reporting perspective and we started to build a very simple report to show the traffic light images.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

In the “Master Page” tab, we set “Orientation” to “Landscape”. In the “Layout” page, we inserted a title (font color=blue, font size=16, font style=cold, alignment=center) with the text “Use of Conditional Images in Reports”. Under the title we placed a small table with three columns only: the image of the traffic light (“image”), the project name (“RowID”), and the remaining money for 2009 (“remain 2009”).

If we just dragged and dropped the “money table” data set from the “Data Set Views” panel into the report editor, the “image” data would have been rendered as a sequence of characters and not as an image. To make the report render the data in the “image” column as an image, we needed to specify that this is an image. In order to import an image value in a table cell of the report:

- Right-click the empty data cell of the table column where the images should go;
- Select “Insert” and then “Image”;
- In the “Edit Image Item” window :
    - Select the radio button, “Dynamic Image”, at the top;
    - Click the “Select Image Data …” button;
    - In the “Select Data Binding” window:
        - Select the column of the data set containing the images (“image” in our example);
        - Click “OK”
- Back in the “Edit Image Item”, click “Insert”.

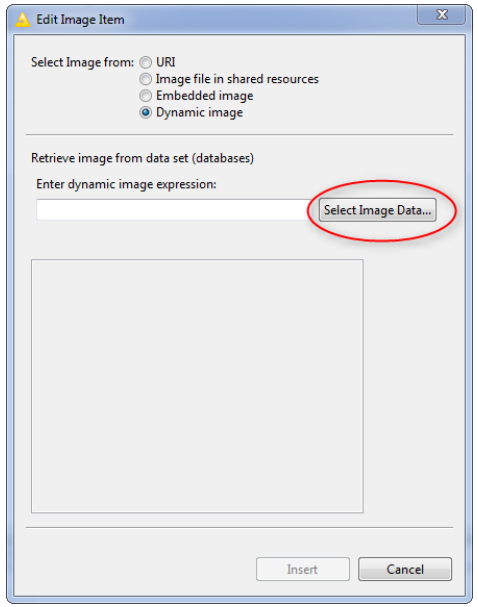
After inserting an image in your table, a square with a red cross appears in the table cell. This is because the reporting editor cannot visualize images. However, the “Preview” should show them clearly in your report layout. In our example, we have imported red and green traffic lights depending on the “remain 2009” column contents. The table in the report should then show rows with a red traffic light and rows with a green traffic light.

**Note**. It is necessary to import the “image” column as an image and, in particular, as a dynamic image for the content of the “image” column to be appropriately interpreted.

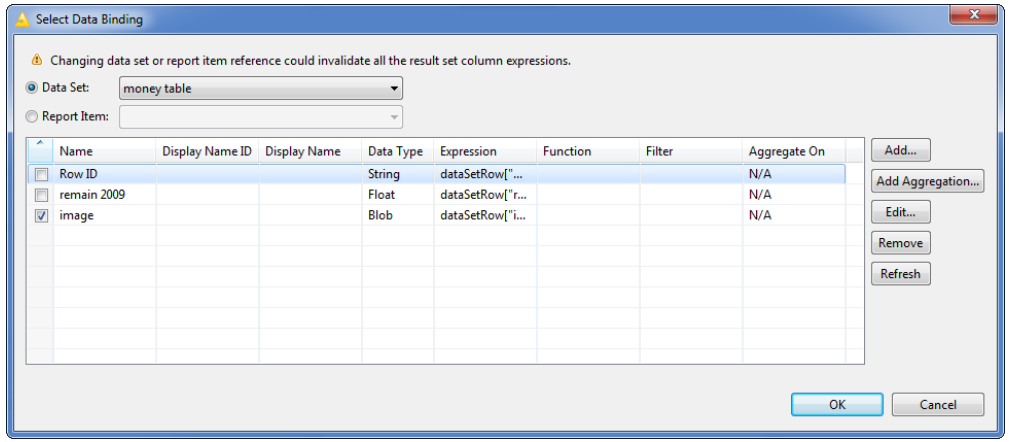


This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

### 8.21. The "Edit Image Item" window



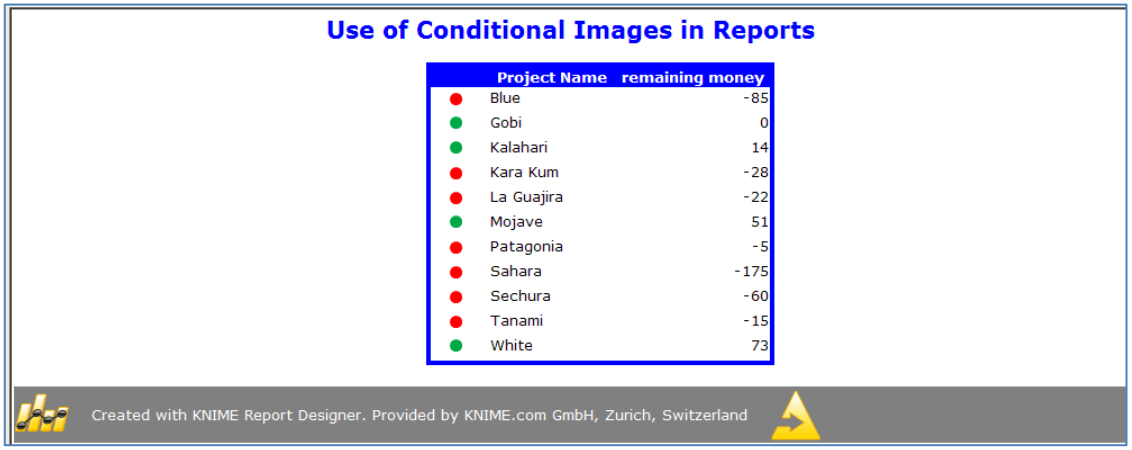

### 8.22. The "Select Data Binding" window






This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

8.23. The final report of the “traffic lights” workflow




## 8.8. Exercises

### Exercise 1

Using data of the file named “Projects.txt” from the KCBdata folder build a report as follows:

- With a title;
- With a table with “project name”, “used money”, and “assigned money”;
- Limit the data in the table to a given quarter and reference year;
- Display the report parameters used to generate the report;
- Display the current date;
- Customize the “Edit Parameters” window for all report parameters: use a static list of values for some and a dynamic list of values for other parameters.



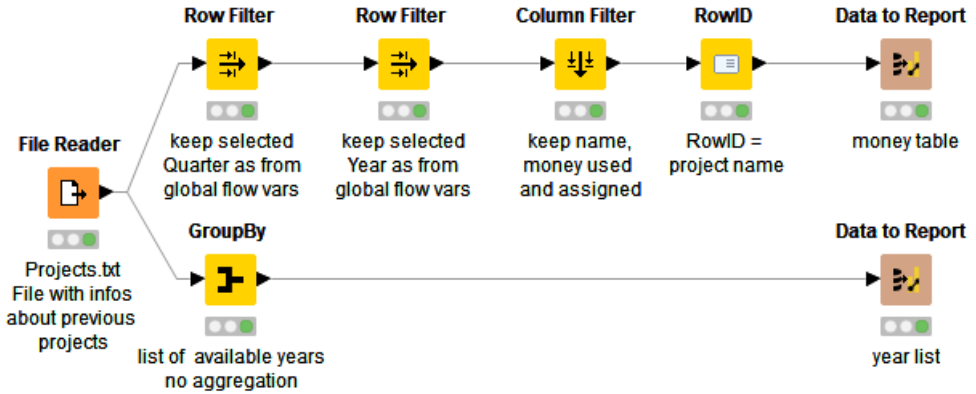
This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

**Solution to Exercise 1**

The workflow named “Exercise 1” is shown in figure 8.24. It includes a “File Reader” node to read the file “Projects.txt”, two “Row Filter” nodes to select a given reference year and a given quarter, and a “Column Filter” node to keep only the project name and the two money related columns. The final data table gets exported into a report data set named “money table”.

To make the “Row Filter” nodes work with parametric values for “reference year” and “quarter”, we introduced two flow variables: “Year” (Integer, default value=2009) and “Quarter” (String, default value = Q1). These two workflow variables generated the two corresponding report parameters. We decided to use a static list of values for the report parameter named “Quarter” and a dynamic list of values for the report parameter named “Year”. The “GroupBy” node collects the list of available years for the dynamic list of values.

8.24. Exercise 1: Workflow




We then built a minimal report with a title and a table generated directly by drag and drop of the “money table” data set. We formatted the title and the table as shown in figure 8.26. The report inherits two report parameters from the underlying workflow: “Quarter” and “Year”.

Under the title we placed a grid with two “Dynamic Text” items: one to display the report parameters and one to display the current date.

The “Dynamic Text” displaying the report parameters uses the following expression in the “Expression Builder” window:

```
BirtStr.concat("Money Summary for Year: ", params["Year"].value, " and Quarter: ", params["Quarter"].value)
```

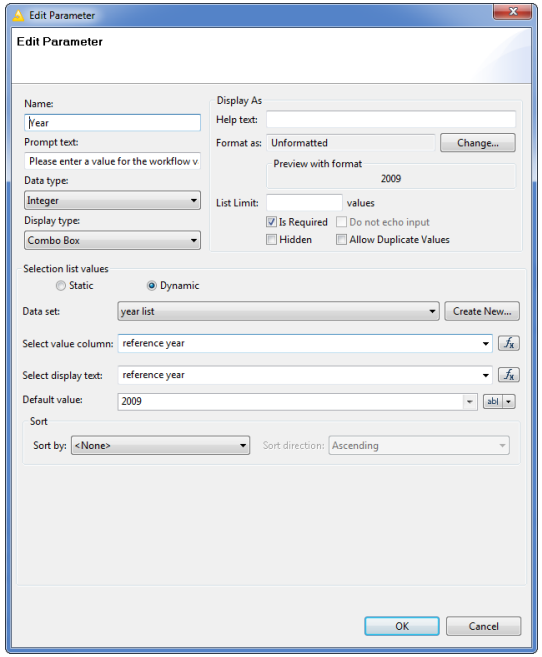
The “Dynamic Text” displaying the current date uses the following expression in the “Expression Builder” window: `BirtDateTime.today()`.



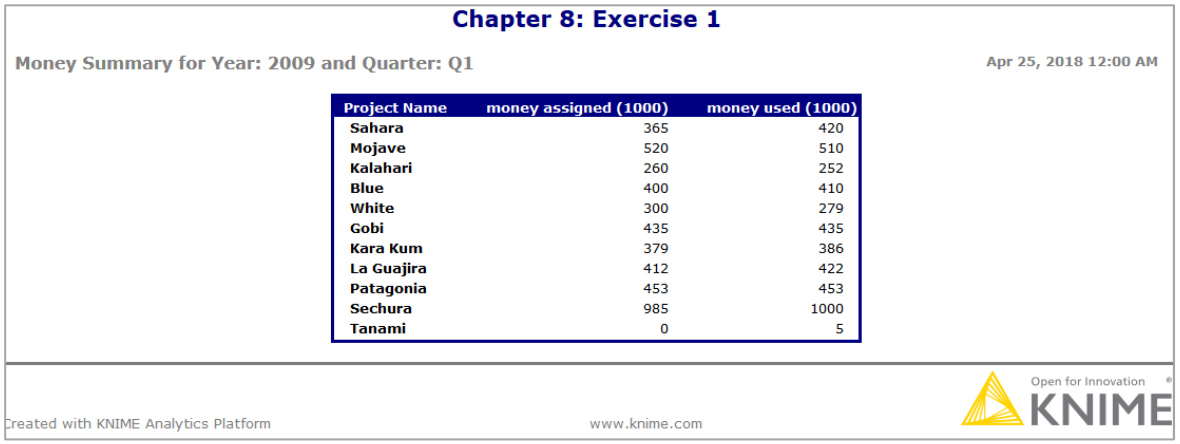
This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

Finally, we customized the “PARAMETER SELECTION PAGE” window. We used a static list of values for “Quarter” report parameter. There we manually inserted the following values: “Q1”, “Q2”, “Q3”, and “Q4”. We used a dynamic list of values for the “Year” report parameter. The list of values was taken from the report data set called “year list”.

8.25. Exercise 1: The "Edit Parameter" window for report parameter "Year"




8.26. Exercise 1: The Report




## Exercise 2

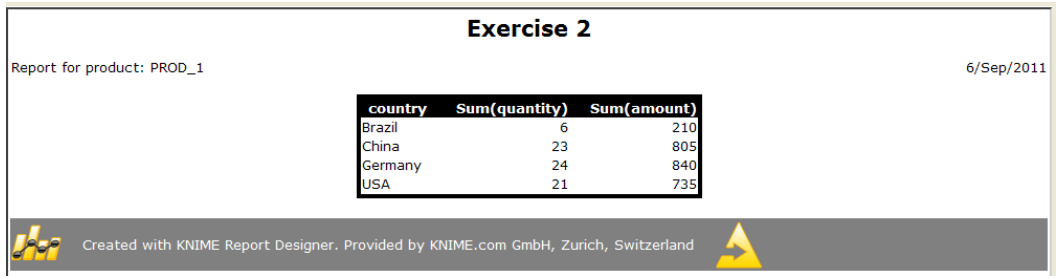
With the data from file “sales.csv” from the KCBdata folder, set a report to look like figure 8.27. In particular:

- The report has to be created for either product “prod_1” or product “prod_2”;
- Write which product has been used, in the top part of the report under the title;
- Produce a current date in the format “dd/MMM/yyyy”.



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

8.27. Exercise 2: The target report



**Exercise 2**

Report for product: PROD_1 6/Sep/2011

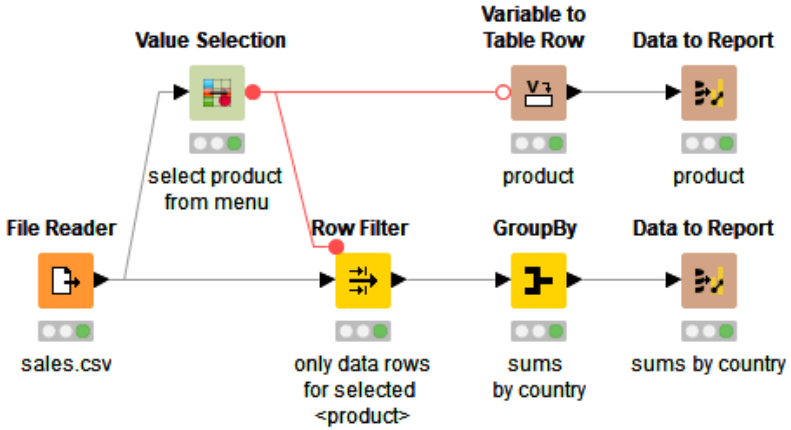
| country | Sum(quantity) | Sum(amount) |
|---|---|---|
| Brazil | 6 | 210 |
| China | 23 | 805 |
| Germany | 24 | 840 |
| USA | 21 | 735 |

Created with KNIME Report Designer. Provided by KNIME.com GmbH, Zurich, Switzerland

**Solution to Exercise 2**

The workflow to calculate the data for the report table is shown in the figure below.

8.28. Exercise 2: The workflow




In addition, one global flow variable has been created and named “product”. This flow variable becomes the report parameter when switching to the reporting environment. The “product” workflow variable should accept two values only: “prod_1” for “product 1” and “prod_2” for “product 2”. The dynamic text items below the title use the following expression respectively:

```
BirtStr.concat("Report for product: ",
               BirtStr.toUpper(params["product"].value))
```

The “toUpper” function changes the argument string into a full upper case string.



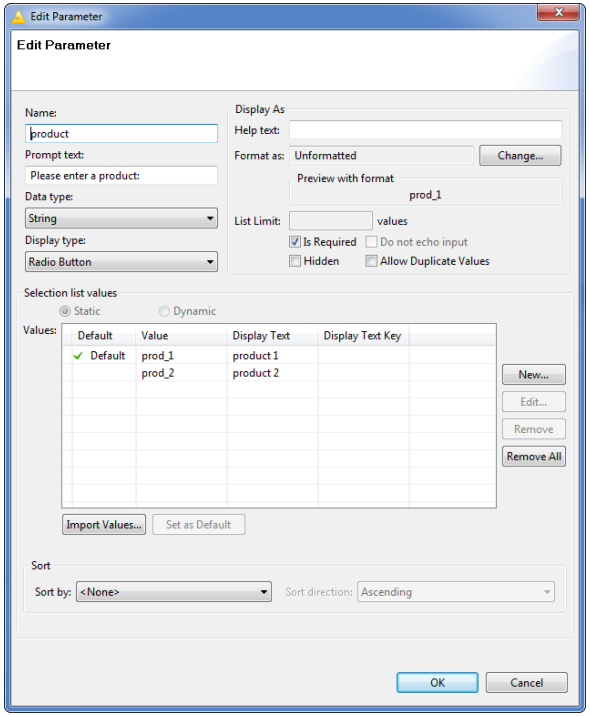
This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

```
BirtStr.concat( BirtDateTime.day(BirtDateTime.today()), "/",
                BirtDateTime.month(BirtDateTime.today(),3), "/",
                BirtDateTime.year(BirtDateTime.today()))
```

This sequence of “concat()”, “day()”, “month()”, and “year()” produces the desired date format.

In the “Edit Parameters” window for the “product” report parameter we have chosen a radio button with the choice between “prod_1” and “prod_2”. The list of values for a radio button is a static one and has to be filled in manually.

8.29. Exercise 2: The "Edit Parameter" window




## Exercise 3

Using data in the “sales.csv” file from the KVBdata folder, build a report with:



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

- A table with the product name, the total sum of amount, and the total sum of quantity for each product;
- The image of each product at the end of the row.

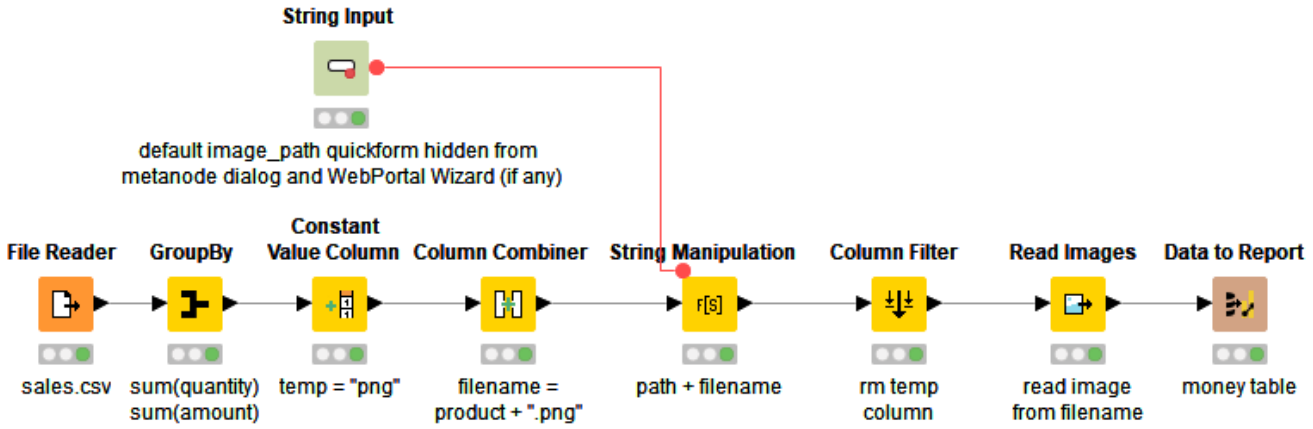
**Solution to Exercise 3**

The product PNG image files are located in “KCBdata\images” and are named after the product they represent.

We created a workflow with the image file path as workflow variable, named “path”.

After reading the “sales” data set, we built the required table with product name, sum(amount), and sum(quantity) using a “GroupBy” node named “sums”. In a temporary column named “temp” we wrote the “png” string in each row by using a “Rule Engine” node. We combined the product name with the “png” extension with a “Column Combiner” node and subsequently with the image file path with a “Java Snippet (simple)” node and we placed the result in a column named “filename”. After removing the temporary column “temp”, we read the PNG images from the file path into the “filename” column with a “Read Images” node. Finally we exported the data table to a report by means of a “Data to Report” node named “money table”.

8.30. Exercise 3: The workflow




**Note**. The file path to be read by the “Read Images” node has to start with “knime://” to allow for relative paths.

In the report, we created a title and a table from the “money table” data set. However, just creating the report table from the data set by drag and drop does not render the images in the table properly. We needed to:

- remove the data cell “filename”;



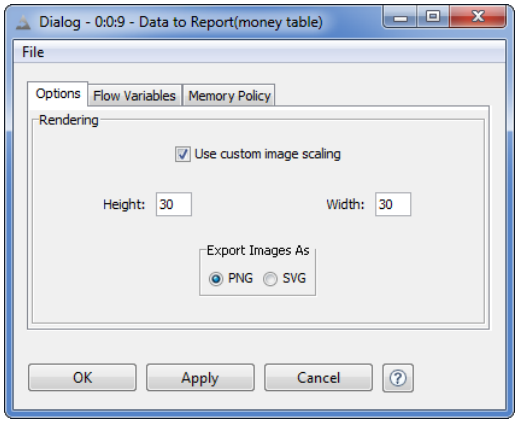
This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

- right-click the empty data cell and select “Insert” and then “Image”;
- in the “Edit Image Item” window:
    - select “Dynamic image”
    - click the “Select Image Data …” button
    - In the “Select Data Binding” window:
        - Select the column name that contains the images
        - Click “OK”
    - Click “Insert”

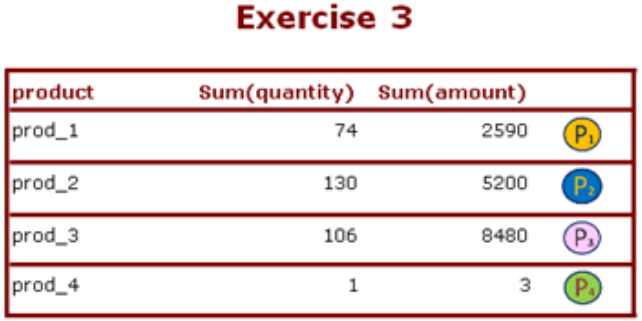
The images now display properly in the table, even though they might be a bit too large. To change the image sizes, you can:

- Either resize the image icon in the report layout manually
- Or change the “Height” and “Width” preferences in the configuration settings window of the “Data to Report” node back in the workflow.

8.31. Exercise 3: The configuration window of the "Data to Report" node




8.32. Exercise 3: The report






This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

# References

[1] Silipo R., “KNIME Beginner’s Luck”, KNIME Press (2010) (https://www.knime.com/knimepress/beginners-luck)
[2] Frank A. and Asuncion A., “UCI Machine Learning Repository [http://archive.ucs.uci.edu/ml], Irvine, CA: University of California, School of Information and Computer Science (2010)
[3] Roy T. Fielding, Richard N. Taylor, “Principled design of the modern Web architecture”, Journal ACM Transactions on Internet Technology (TOIT) Vol. 2 Issue 2, 115-150 (2002)
[4] NIST/SEMATECH, “e-Handbook of Statistical Methods”, (http://www.itl.nist.gov/div898/handbook/pmc/section4/pmc4.htm)



This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

# Node and Topic Index





This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

## G



## H



## I



## J



## K



## L



## M





This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

## P



## Q



## R



## S



## T



## U



## V



## W



## X





This copy of the book “KNIME Advanced Luck” is licensed to: Laszlo Sallo

Open for Innovation ®
KNIME

# *KNIME Advanced Luck: A Guide for Advanced Users*

*This book is the sequel to the introductory text "KNIME Beginner's Luck". Building upon the reader's first experience with KNIME, this book presents some more advanced features, like looping, selecting workflow paths, workflow variables, reading and writing data from and to a database, dealing with date and time objects, and more.*

*All new concepts, nodes, and features are demonstrated through worked examples and the learned knowledge is reinforced with exercises. All example workflows and exercise solutions can be downloaded at purchase.*

*The goal of this book is to elevate your data analysis from a basic exploratory level to a more professionally organized and complex structure.*

***About the Authors***

***Dr Rosaria Silipo** has gained her PhD in biomedical engineering, at the University of Florence, Italy, in 1996. She has been analyzing data ever since, during her postdoctoral program at the University of Berkeley and during most of her following job positions (Nuance Comm., Viseca). She has many years of experience in data analysis in many different application fields. In particular, she is interested in customer care, finance, and text mining. In the last few years she has been using KNIME for her consulting work.*

***Dr Jeanette Prinz** earned her PhD in Bioinformatics at the Technical University Munich (Germany), in collaboration with the Helmholtz Centre Munich in 2015. During her work there she became an enthusiastic KNIME user. She has a strong background in clinical data analysis, developing machine learning models to infer molecular mechanisms related to disorders and drug action. She applied this, for example, as Research Fellow at the Mayo Clinic (AZ, USA) in 2016-2017.*

ISBN: 978-3-9523926-0-7