Open for Innovation KNIME®

Components/Metanodes

KNIME AG, Zurich, Switzerland
Version 4.0 (last updated on 2019-06-05)

Table of Contents

Introduction . . . 1
- Metanodes vs Components . . . 1

Creating Components/Metanodes . . . 2
- Collapsing Nodes into Metanodes . . . 2
- Encapsulating Nodes into Components . . . 3

Configuring Components/Metanodes . . . 4

Component Dialogs . . . 7
- Quickforms . . . 7

Custom Views on Components . . . 10
- Layouting for Components . . . 10

Job Manager . . . 17
- Standard Execution . . . 17
- Streaming Execution . . . 17

Introduction

Components/Metanodes are nodes that contain sub-workflows, that is, a number of further nodes. In this guide, we explain how to use Components/Metanodes for organizing and tidying up workflows, how to create custom views and how Components/Metanodes can be used for reusability.

Metanodes vs Components

Components are similar to Metanodes, but they encapsulate complete, isolated functionality.

The main differences are:

- Variable Scope: The variable scope of a Component is local which makes them self-containing and less polluting to the parent workflow. A flow variable defined inside a Component is by default not available outside it, and a flow variable defined outside the Component is by default not available inside it.
- Custom views: Components can also have custom views, which are acquired from the interactive views of Quickform nodes and JavaScript nodes inside the Component. The view corresponds to a WebPortal page on the KNIME Server.
- Custom configuration dialogs: Components can have custom configuration dialogs.

The differences between Metanodes and Components are summarized in Table 1.

Table 1. Comparison of Metanodes and Components valid for KNIME Analytics Platform 3.1 and above

	Metanodes	**Components**
Quickforms	Legacy	Standard
Variable scope	Global	Local
Custom views	Not supported	Supported

© 2018 KNIME AG. All rights reserved.

Creating Components/Metanodes

Collapsing Nodes into Metanodes

To collapse a set of nodes into a Metanode, first select the nodes you want to collapse into a Metanode.

- Drag a rectangle with a mouse over the nodes in the workflow editor
- Or, press and hold the "Ctrl" button and select the nodes with a mouse click
- Now right click the selection, and select *Create Metanode* in the menu as shown in Figure 1

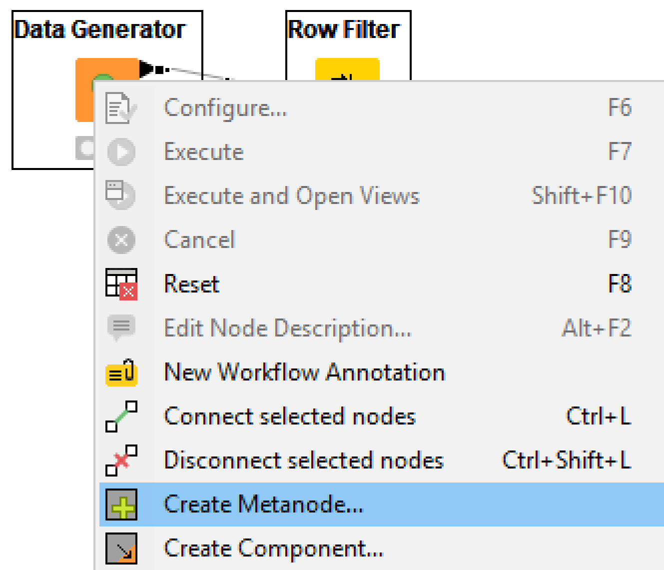


Figure 1. Creating a Metanode

- A dialog shown in Figure 2 appears asking you to give the Metanode a name

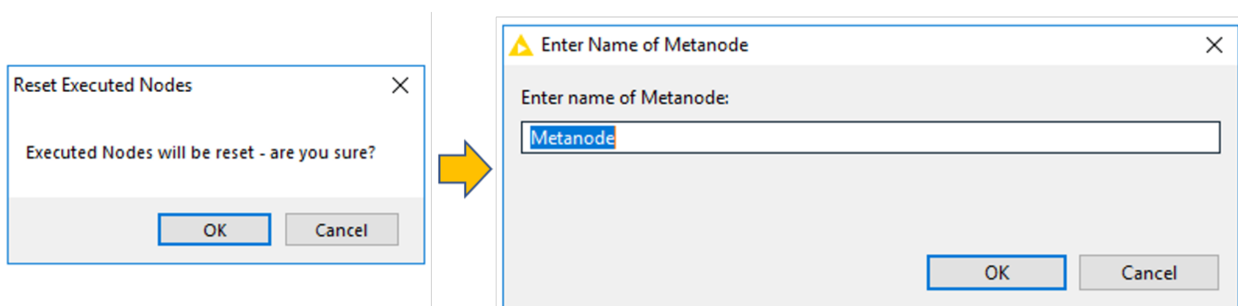


Figure 2. Giving a Metanode a Name

- Click OK, and you will see a gray box in the workflow editor in place of the single nodes. Appropriate input and output ports will appear for the Metanode based on the connections coming into and out of it. Notice that collapsing nodes into a Metanode

© 2018 KNIME AG. All rights reserved.

resets the nodes.

If you want to open the sub-workflow inside the Metanode, and possibly change the settings for single nodes, double click the Metanode. Now the sub-workflow will open in a separate workflow editor.

Encapsulating Nodes into Components

Create a Component by:

- Selecting the nodes you want to encapsulate in the workflow editor by drawing a rectangle with a mouse over them
- Or, pressing "Ctrl" and selecting single nodes with a mouse click
- Then, right clicking the selection and selecting *Create Component* in the menu shown in Figure 3

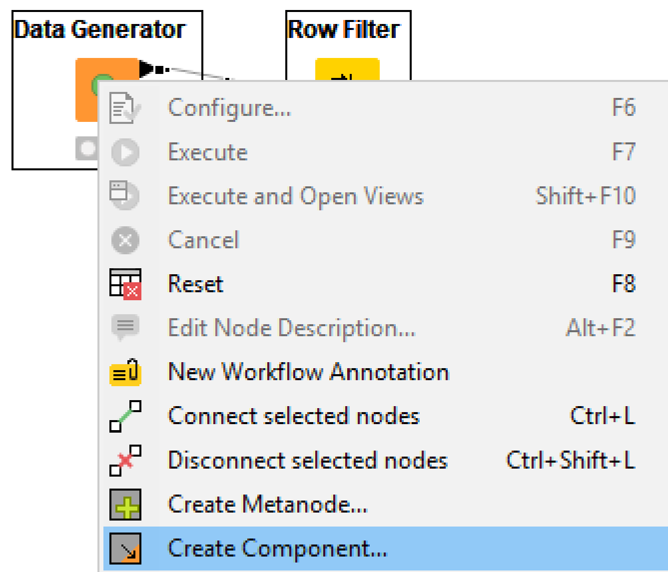


Figure 3. Creating a Component

© 2018 KNIME AG. All rights reserved.

Configuring Components/Metanodes

The settings for Components/Metanodes are the Metanode name, and the input and output ports and their types.

To change the settings for a Metanode:

- Right click the Metanode and select *Reconfigure* as shown in Figure 4. In the dialog (shown in Figure 5) that opens, you can change the Metanode name, and add and remove input and output ports of any type.

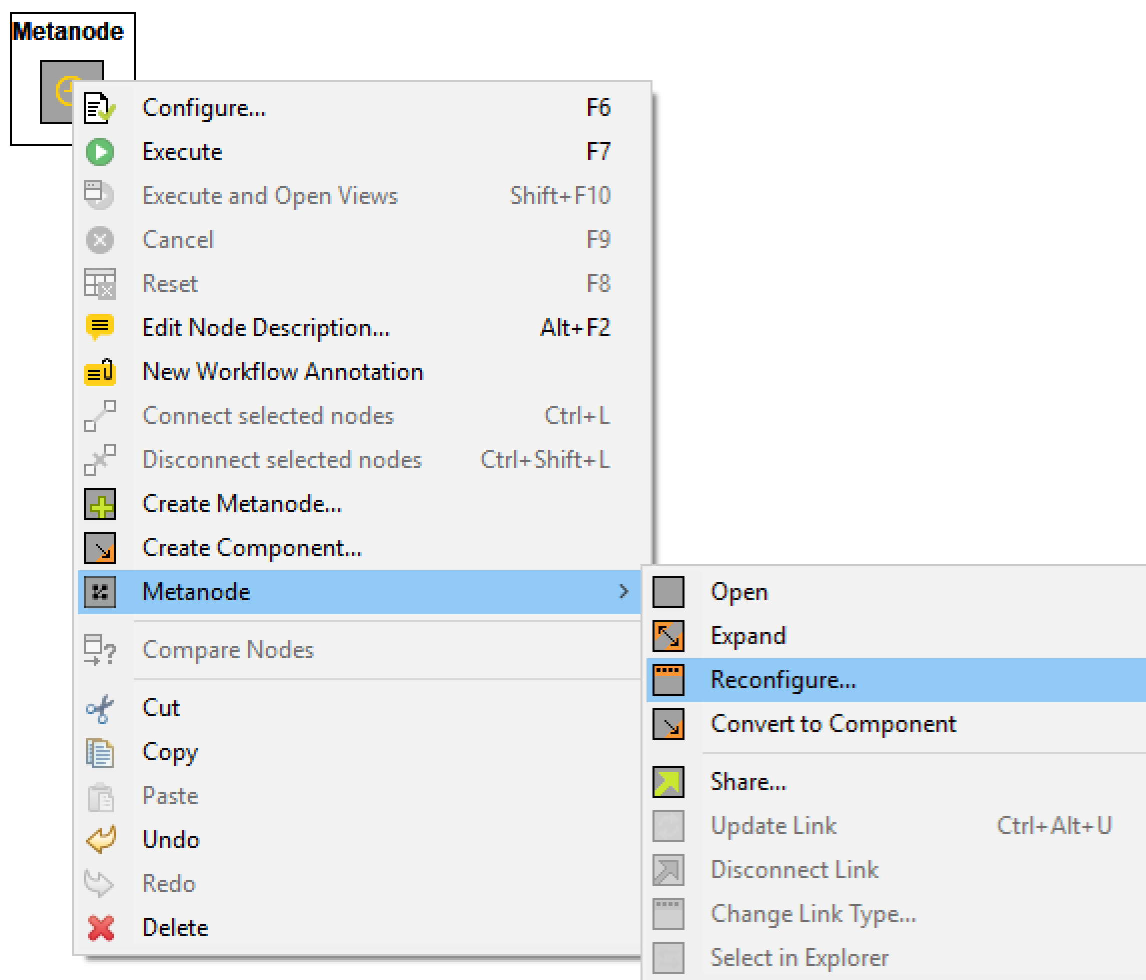


Figure 4. Metanode context menu

© 2018 KNIME AG. All rights reserved.

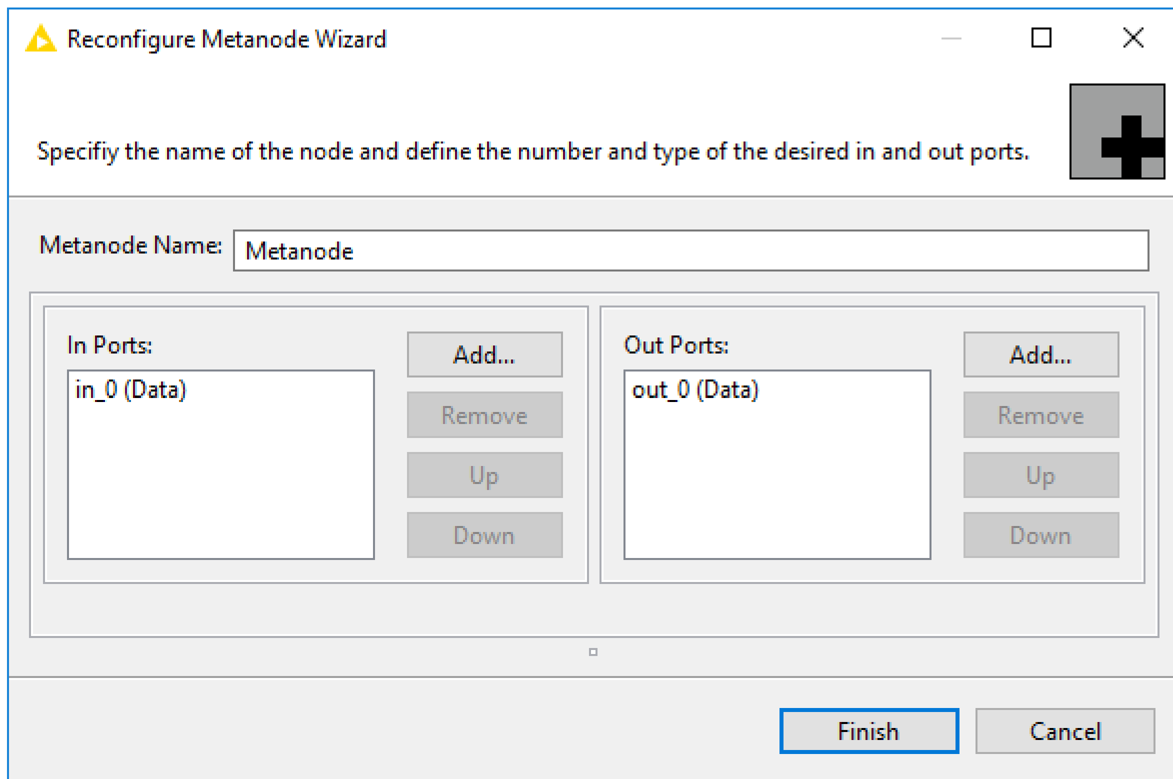


Figure 5. Reconfiguring a Metanode

You can change these settings for a Component in much the same way: expand the *Component* item in its context menu, and select *Setup…*. Now, a dialog opens similar to the dialog shown in Figure 5.

In the wizard shown in Figure 5, you can apply the following settings:

- Add input and output ports by clicking the *Add…* button and selecting the port type in the dialog (shown in Figure 6) that opens.

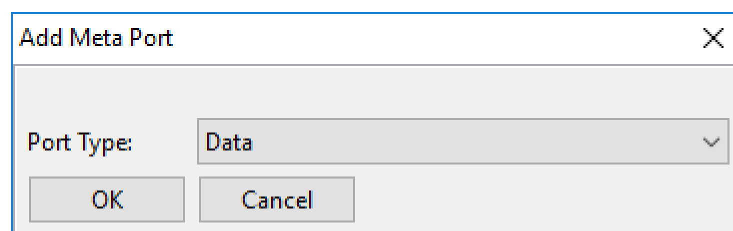


Figure 6. Adding a New Port to the Component/Metanode

- Remove existing input and output ports by clicking the *Remove* button. Notice that you have to remove all connections coming to and from the port before you can remove it.
- Change the order of the input and output ports using the *Up* and *Down* buttons. Changing the order of the ports doesn't affect any functionality of the workflow but is often practical to avoid crossing connections and thus improve the visual appearance of the workflow.

To return the nodes within a Component/Metanode into their original, uncollapsed state, right

© 2018 KNIME AG. All rights reserved.

click the Metanode and select *Expand* in the menu shown in Figure 4. To change a Component back to a simple Metanode, select *Convert to Metanode* instead of *Expand*.

© 2018 KNIME AG. All rights reserved.

Component Dialogs

In this section, we will explain how to create a custom configuration dialog for a Component using Quickform nodes.

Quickforms

A Quickform node can provide input parameters for other nodes in the workflow. If you use one or more Quickform nodes inside a Component, the configuration dialog of the Component will show all these configuration options you created inside it in it’s custom dialog.

Quickform nodes enable different types of user inputs such as string input, integer input, selecting one value in a list and more. You can find all available Quickform nodes in the node repository in the "Workflow Control" category under the "Quickforms" category.

The output of a Quickform node often is a flow variable. In the configuration dialog of the quickform node, you can define the name and default value of the flow variable, along with some settings to control the appearance of the custom dialog, if the Quickform node is used inside a Component.

Open the custom configuration dialog of the Component by right clicking and selecting *Configure…* in the menu.

Figure 7 shows the configuration dialog of the Value Selection node, where you can define the input label, description, default selection option, and some visual properties.

© 2018 KNIME AG. All rights reserved.

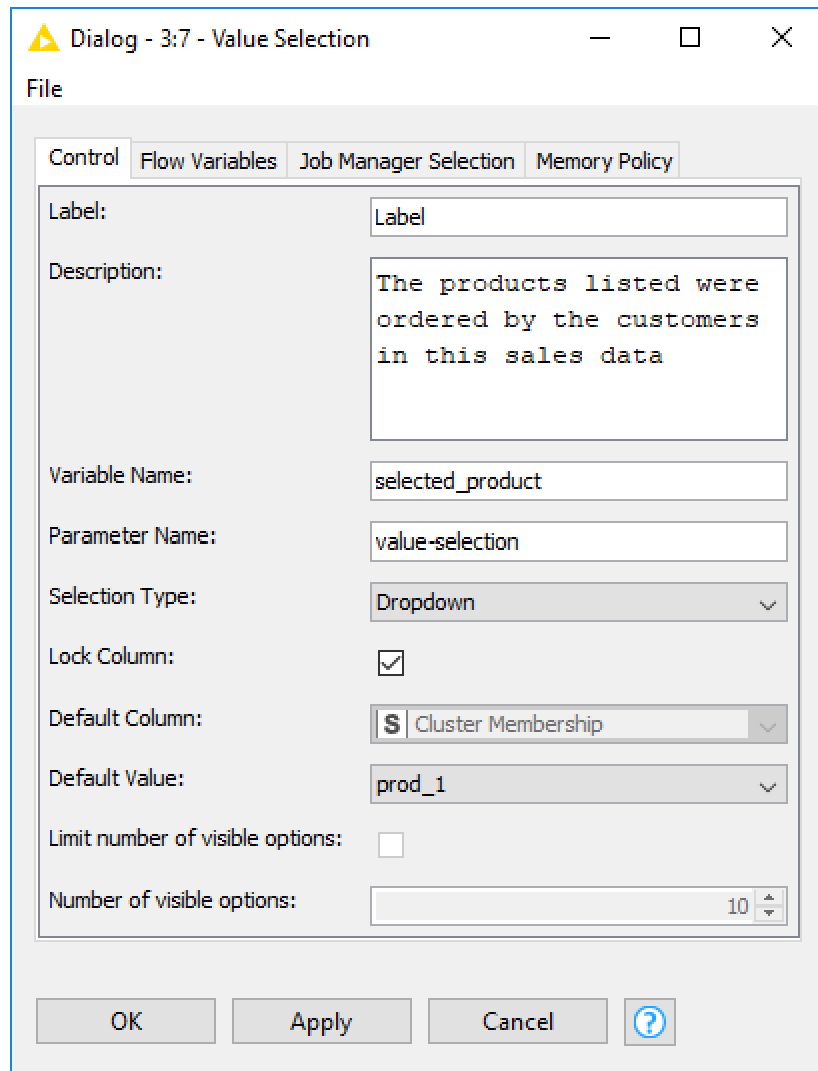


Figure 7. Configuration Dialog of a Quickform node

Another node can access the flow variable output of a Quickform node, if the flow variable output of the Quickform node is connected to it, as shown in Figure 8. The value of the output of the Quickform node is either defined by its default value (as defined in the configuration dialog of the Quickform node) or corresponds to the value provided by the user in the custom configuration dialog of the Component.

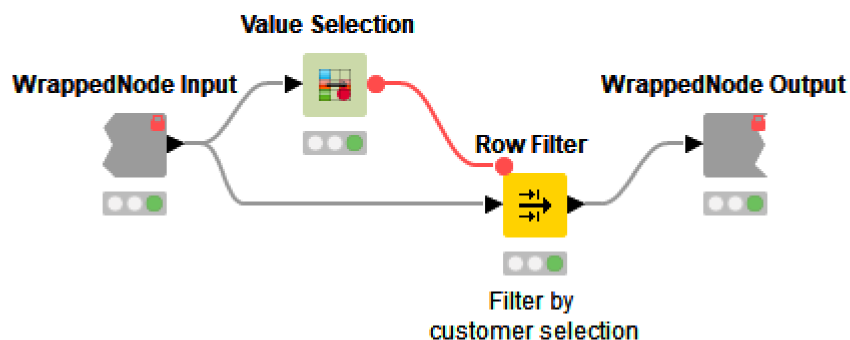


Figure 8. Configuring a node with a value defined by a user

© 2018 KNIME AG. All rights reserved.

If you deploy the workflow to a KNIME Server, the same configuration option is shown on a WebPortal page, as shown in Figure 9.

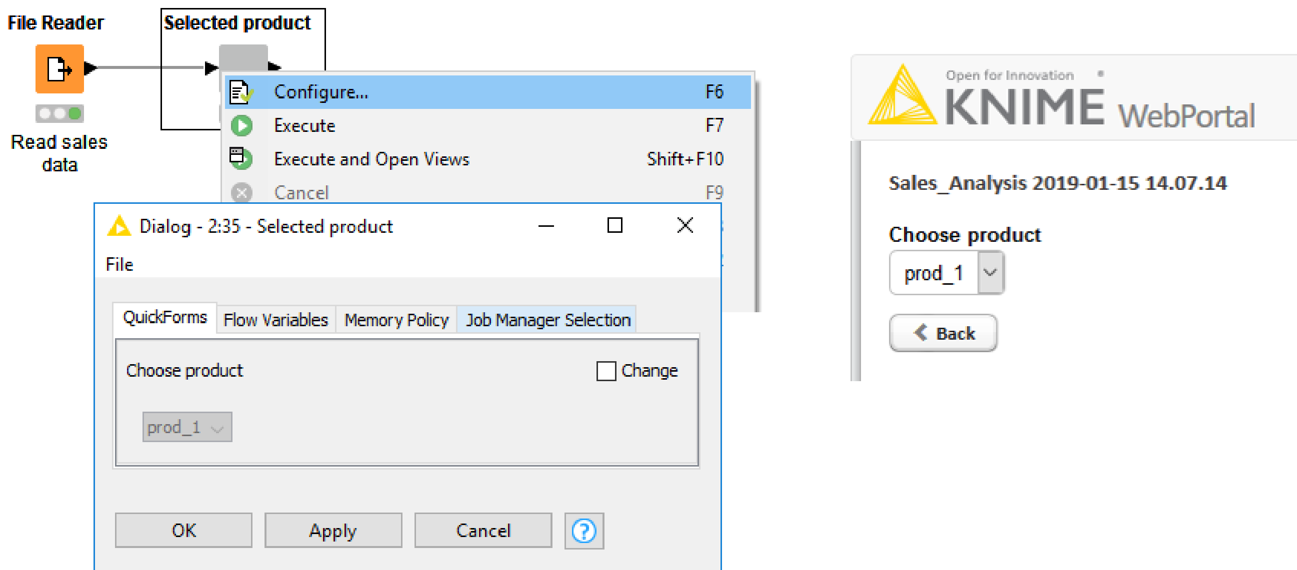


Figure 9. Value Selection Option in the Configuration Dialog of a Component and on a WebPortal Page

© 2018 KNIME AG. All rights reserved.

Custom Views on Components

Besides custom configuration dialogs, Components can have custom views. The custom views are composite views containing the interactive views of Quickform nodes and JavaScript nodes inside the Component. The composite view will be available as a web page on the KNIME WebPortal.

Layouting for Components

Any Component that contains at least one Quickform or JavaScript-enabled view can have a layout defined, e.g. to lay out two views beside each other. To access the layout editor, open the Component and click the layout editor button in the top toolbar:

Visual layout editor

The visual layout editor allows you to create and adjust layouts using a drag & drop grid. To get started it's important to understand three basic concepts:

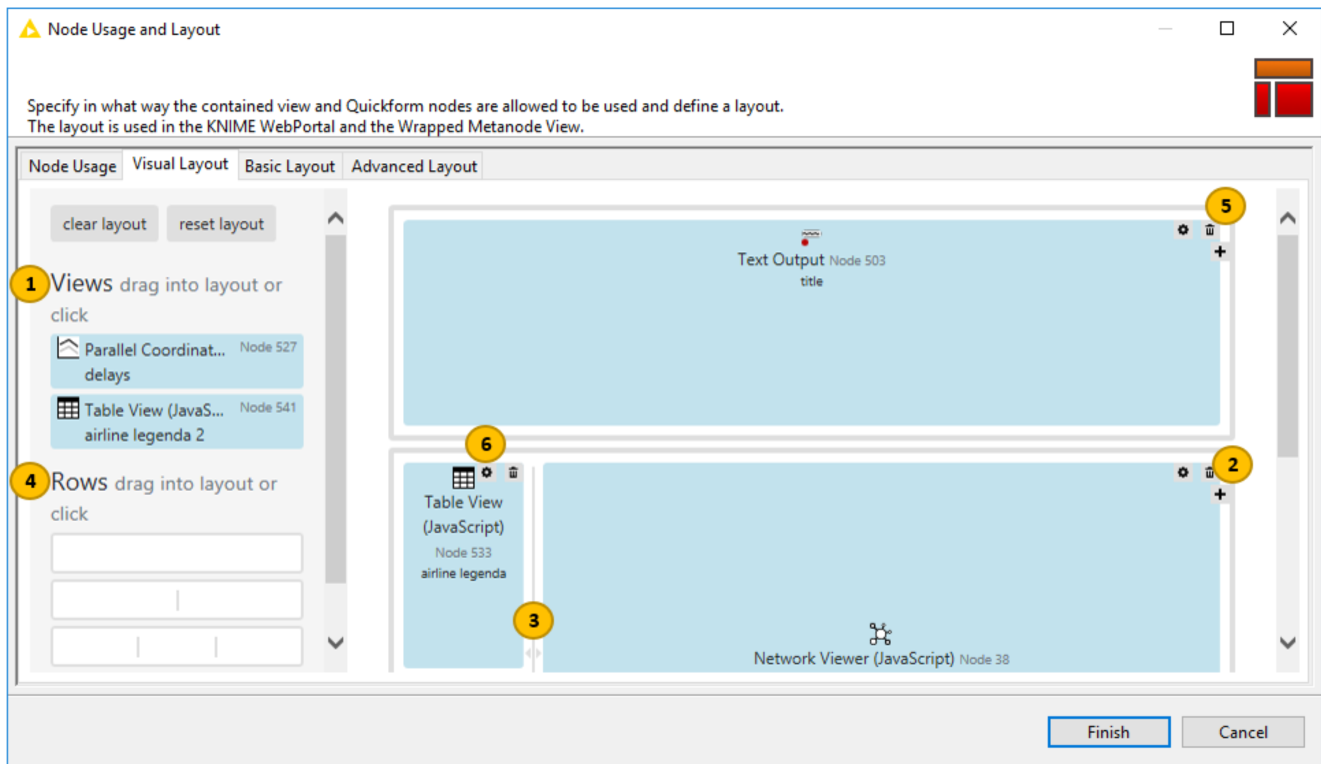
- a layout consists of one or more **rows**. Each row can have up to 12 columns.
- a **column** determines the width of its components, therefore it can be resized when there is more than one column in a row.
- one or more **views** can be added to a column.

For example, to layout two views side by side you could create two columns with either equal or unequal widths and add one view to each.

All available views are initially inserted below each other into the layout.

The visual layout editor consists of a left panel which shows a list of all JavaScript views in your Component which have not yet been added to the layout and an interactive preview of the layout on the right.

© 2018 KNIME AG. All rights reserved.



To **add a view**, drag it from the left panel (1) to the desired position in the layout preview.

To **add a column**, click the '+' buttons in the layout preview (2).

To **resize a column**, click and move the resize handle between columns (3).

To **add a row**, drag a row template from the left panel (4) to the desired position in the layout preview. You can choose between different templates, e.g. 1-column, 2-column, 3-column; but you can also add and remove columns later on.

To **delete a view, column or row** use the trash bin button (5). This is only available for columns and rows when they are empty, i.e. do not contain views, rows, or columns.

To **move a view** into another column drag it to the layout preview. Complete rows can also be moved by dragging.

Note that **nesting is possible**: columns can contain rows as well as views, those nested rows can contain columns, and rows, and views, and so on…

Adjusting the height of views

Each view has default sizing settings which can be changed via the cog icons in the layout preview (6). Basically you can choose between automatic height based on the content of the view or aspect ratio sizing (16:9, 4:3 or 1:1). When using automatic height it’s possible to define minimal and maximal sizes.

© 2018 KNIME AG. All rights reserved.

Advanced layouting

The layout structure is saved in a JSON format which advanced users can edit directly in the 'Advanced layout' tab.

The layout is based on the framework Bootstrap and uses its 12 column-based, responsive grid system. For more information and documentation about Bootstrap please visit the Bootstrap website.

The JSON format generated by the visual layout editor is shown in Figure 10. It's a good starting point for understanding the structure and further customization and fine tuning.

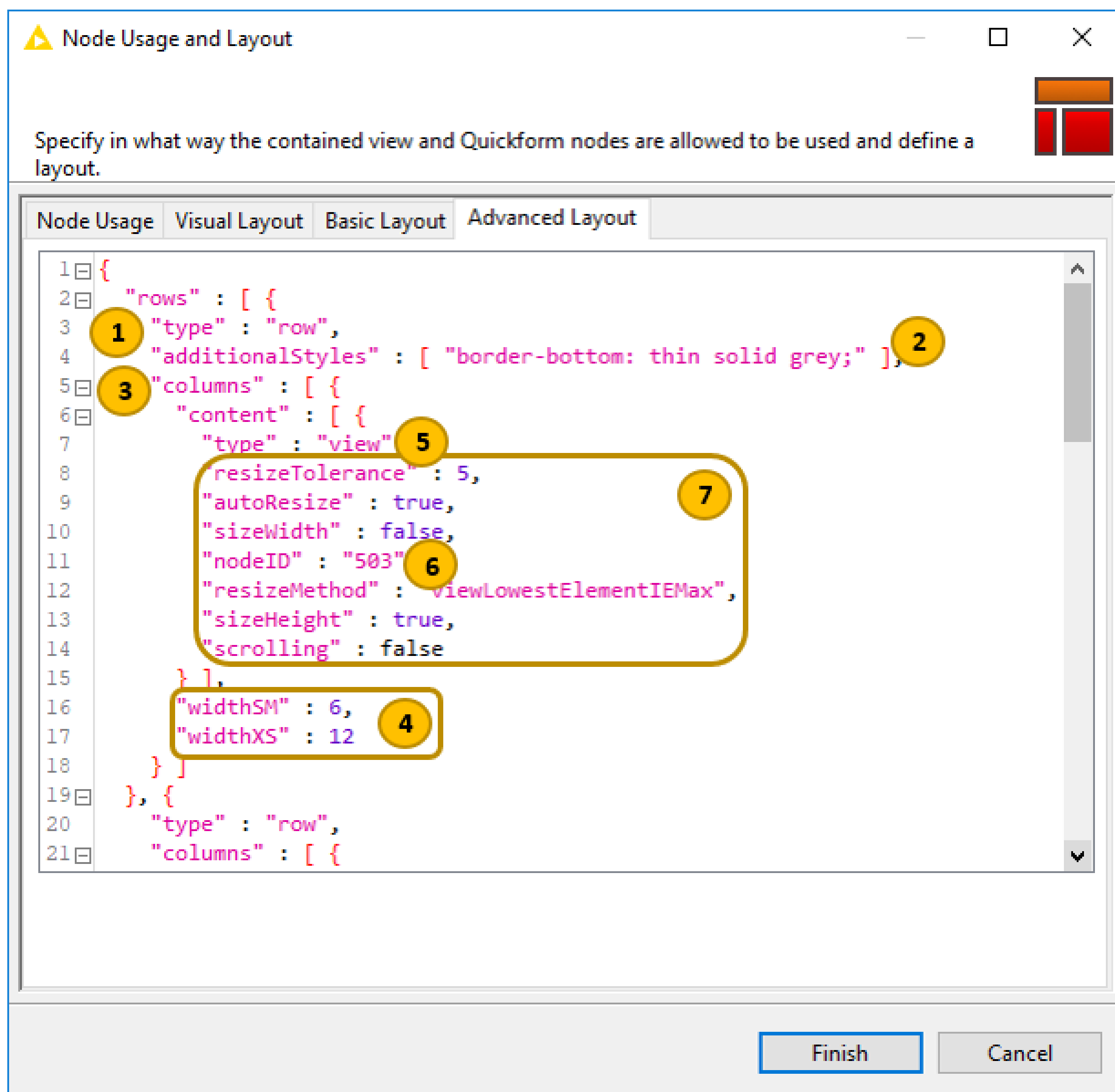


Figure 10. Component Layouting in JSON Format

The summary below describes the JSON structure in a non-normative descriptive way. The numbers in parenthesis indicate where you can find the elements in the JSON format in Figure 10.

© 2018 KNIME AG. All rights reserved.

Row (1)

A row is the outer most element that can be defined and is the first element inside the layout container. The JSON structure's outer layer is an array of rows. A row contains a number of layout-columns.

To further customize a row, optional fields can be used. `additionalClasses` can be used to provide an array of class names to append to the created HTML row element, `additionalStyles` (2) is an option to directly insert CSS style commands on the element. For example, to create a visual separator between one row and the next, a bottom border can be used like this:

```
"additionalStyles" : [ "border-bottom: thin solid grey;" ]
```

The grey line that appears in the custom view output of the Component is shown in Figure 11.

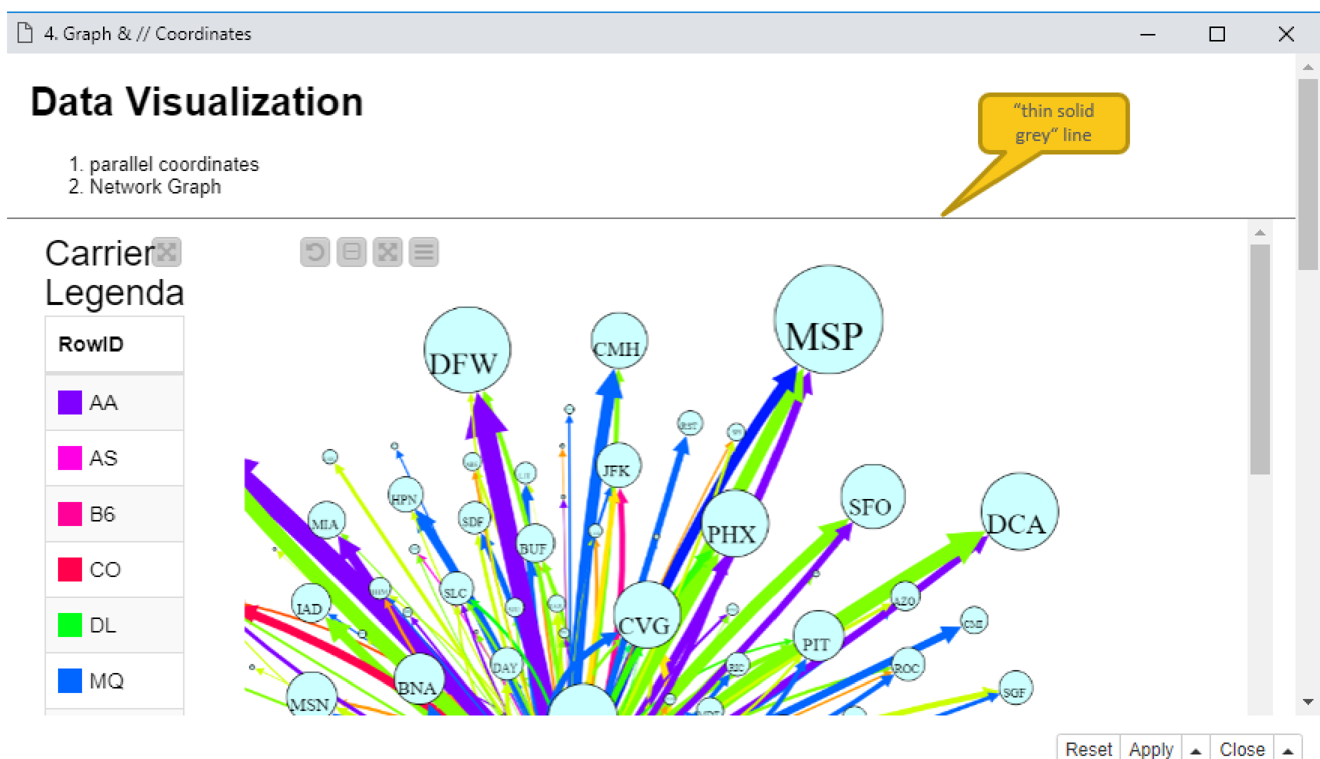


Figure 11. Custom View Output of a Component with Additional Styling

Column (3)

A column is a layout element inside a row which determines the width of its components. To define a width, a number between 1 and 12 (corresponding to the Bootstrap column widths) has to be used. 12 means taking up 100% of the width, whereas 6 would be 50% of the width. This way it is possible to define a layout with components side by side by providing their

© 2018 KNIME AG. All rights reserved.

relative widths. E.g. if three components are to be laid out horizontally with equal column widths use a row with three columns of width 4. If the sum of widths for a particular row is larger than 12, the "extra" columns are wrapped onto a new line.

Responsive layouts (4)

It is also possible to define multiple widths of the columns so that they can adapt to the screen size. With this option responsive layouts can be achieved.

To define the responsive width of a column, use at least `widthXS` and one or more of the following fields: `widthSM`, `widthMD`, `widthLG` which corresponds to the Bootstrap size definitions of XS, SM, MD and LG.

The content of a column can be an array of one of three things: another set of rows (providing the possibility to create nested layouts), regular HTML content (to insert plain HTML elements into the layout), or a node reference (to embed the contents of a JavaScript-enabled KNIME node). As for rows, it is also possible to further customize the column using the optional fields `additionalClasses` and `additionalStyles`.

HTML content

It is possible to include plain HTML into the layout by placing a content element of type `html` inside a column. To insert the content a single field `value` is used. Please note that Bootstrap is present at runtime, so regular Bootstrap elements can be used. This way glyphicons, labels, badges, jumbotrons, … are available to define the layout of the page. For example:

```
[...]
"content":[{
    "type":"html",
    "value":"<h2 >Title defined in layout</h2><p><span class='glyphicon glyphicon-th' \
             aria-hidden='true' style='margin-right: 5px;'></span>Glyphicon</p>"
  }]
[...]
```

View content (5)

To embed the contents of a KNIME node inside the layout, a content element with type `view` has to be used. The element has quite a few ways to customize the sizing and behavior of the content, which are explained in the table further down.

Referencing the node is done by the field `nodeID` (6), which takes the ID-suffix of the node as a string argument. If nodes exist inside the Component which are not referenced by the

© 2018 KNIME AG. All rights reserved.

layout, a warning message appears underneath the editor. Errors will also be issued for referencing nodes twice or referencing non-existing nodes.

To understand the sizing concepts better it is important to know that each node's content is wrapped in its own `iframe` element (this allows us to encapsulate the implementation and avoids reference and cross-scripting issues). As `iframe` elements do not adapt to their content's size automatically, they need to be resized to achieve the desired behavior. In principal there are three options available:

Size-based methods

This method uses an iframe-resizer library to resize the `iframe` according to the size of its contents. This means that the content has to have a concrete size explicitly or implicitly set. determining the size is possible using different approaches, which are explained on the iframe-resizer GitHub page Size-based resize methods all start with the prefix `view…` in the JSON structure.

Aspect-ratio based methods

If a node's view is set to adapt to its parent's size, rather then implicitly providing a size, the size-based methods will either not work or cause strange behavior. To allow these views to take up an appropriate amount of space in the layout an aspect ratio setting can be used. Here the width is taken as 100% of the horizontal space available at that position in the layout and the height is calculated according to the given ratio. To achieve this resizing behavior, Bootstrap's responsive embed concept is used. Aspect-ratio based resize methods start with the prefix `aspectRatio` in the JSON structure.

Manual method

It is also possible to completely manually trigger resize events at appropriate times. This requires the node's implementation to make the appropriate resize calls itself.

List of available fields (7)

Field name	Explanation / Possible Values
`nodeID`	ID-suffix of referenced node
`minWidth`	Constrain the size of the `iframe` by setting a minimum width in pixels.

© 2018 KNIME AG. All rights reserved.

Field name	Explanation / Possible Values
`minHeight`	Constrain the size of the `iframe` by setting a minimum height in pixels.
`maxWidth`	Constrain the size of the `iframe` by setting a maximum width in pixels.
`maxHeight`	Constrain the size of the `iframe` by setting a maximum height in pixels.
`resizeMethod`	The resize method used to correctly determine the size of the `iframe` at runtime. Can be any of the following values: `viewBodyOffset`, `viewBodyScroll`, `viewDocumentElementOffset`, `viewDocumentelementScroll`, `viewMax`, `viewMin`, `viewGrow`, `viewLowestElement`, `viewTaggedElement`, `viewLowestElementIEMax`, `aspectRatio4by3`, `aspectRatio16by9`, `aspectRatio1by1`, `manual`
`autoResize`	Boolean only working with size based resize methods. Use this to enable or disable automatic resizing upon window size or DOM changes. Note that the initial resize is always done.
`resizeInterval`	Number only working with size based resize methods. Sets the interval to check if resizing needs to occur. The default is `32` (ms).
`scrolling`	Boolean only working with size based resize methods. Enables or disables scroll bars inside `iframe`. The default is `false`.
`sizeHeight`	Boolean only working with size based resize methods. Enables or disables size adaption according to content height. The default is `true`.
`sizeWidth`	Boolean only working with size based resize methods. Enables or disables size adaption according to content width. The default is `false`.
`resizeTolerance`	Number only working with size based resize methods. Sets the number of pixels that the content size needs to change, before a resize of the `iframe` is triggered. The default is `0`.
`additionalClasses`	Array of additional classes added to the HTML container element.
`additionalStyles`	Array of additional CSS style declaration added to the HTML container element.

Debugging resize behavior

If the WebPortal is used in debug mode, log messages are printed to the browser’s console for all size-based resize method components. Also all JavaScript libraries will be included in a non-minified version to allow for JavaScript debugging in the browser.

To inspect the custom output view in KNIME Analytics Platform, right click the Component and select *Interactive View: …*.

© 2018 KNIME AG. All rights reserved.

Job Manager

A job manager defines how nodes or workflows are executed, e.g. in which order, and how the data are passed from one node to another. In each configuration dialog, you can find a *Job Manager Selection* tab. It is set to "standard" by default, but you can change the job manager for single nodes and Components.

The different settings for the Job Manager are as follows:

Standard Execution

In the standard execution model, the operations in a workflow are executed node by node. Data are passed from one node to another after the entire input data of a node has been processed. The dataset passed to the next node is the intermediate output table, which you can inspect by opening the output table of a node in the middle of a workflow.
If you open the Job Manager Selection tab in any configuration dialog, and see the job manager selection `<<default>>`, then the node operation is executed according to the standard execution model.

Streaming Execution

In the streaming execution model, data is passed from one node to another as soon as it is available. That is, all data do not have to be processed. Therefore, the streaming execution model leads to a faster execution speed because only the rows "in transit" are concerned. Streaming execution is also memory efficient because intermediate tables are not reserved.

Streaming execution can be applied to single nodes or entire workflows inside a Component. The Streaming Executor is an extension available in KNIME Labs, which you can install by navigating to *File - > Install KNIME Extensions…*.

How to see which nodes support streaming execution:

- Click the arrow in the top right corner of the node repository.
- In the menu that opens, select *Show Additional Info*. Now, the information on ability for streaming execution appears next to the node names.
- If you select *Show Streamable Nodes Only* in the menu, only streamable nodes will appear in the interface. Notice that the non-streamable nodes can still be a part of a workflow inside a Component, which is executed in the streaming model. They will simply be executed in the standard execution model.

© 2018 KNIME AG. All rights reserved.

To switch from standard to streaming execution:

- Select *Simple Streaming* in the *Job Manager Selection* tab in the configuration dialog of a Component or a single node. If the streaming execution option is not available for the node, or you have not yet installed the extension, you will only see the `<<default>>` option in the menu.

The streaming mode to execute a Component:
If you use the streaming model to execute a Component, the sub-workflow inside it is always executed entirely. The intermediate output tables of the nodes inside the Component are not available, because they are not reserved. As shown in Figure 12, the nodes in the workflow don't have traffic lights below them, but bars showing the execution progress. The nodes are connected by dotted connections, and the numbers above them show the number of records that have passed that particular connection.

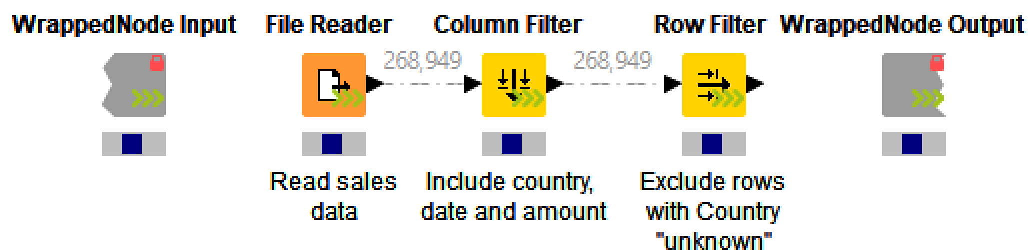


Figure 12. Streaming Execution inside a Component

© 2018 KNIME AG. All rights reserved.

Open for Innovation®
KNIME

KNIME AG
Technoparkstrasse 1
8005 Zurich, Switzerland
www.knime.com
info@knime.com

The KNIME® trademark and logo and OPEN FOR INNOVATION® trademark are used by KNIME AG under license from KNIME GmbH, and are registered in the United States. KNIME® is also registered in Germany.