Technical Handbook i-views 6.0

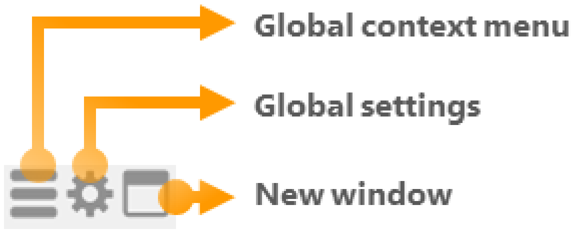
Table of contents

1. Knowledge-Builder

This technical handbook comprises all advanced configuration of the i-views Knowledge-Builder, Admin-Tool, View Configuration Mapper and services as well. Basic fundamentals about how to use the Knowledge-Builder are described in the User's Manual.

1.1. Global actions and settings

All actions and settings, which are independent from the Knowledge Graph context, are so-called "global actions" or "global settings". They are available in the upper right corner of the Knowledge-Builder as long as the start screen is visible or when an element in the organizer is chosen on the left side of the Knowledge-Builder:



- Global context menu: Provides actions for administrative purposes
- Global settings: Contains user-changeable settings or overall settings that can be changed by the administrator only
- New window: Useful for opening a selected item (e. g. import mapping, view configuration…) in a new dialog window.

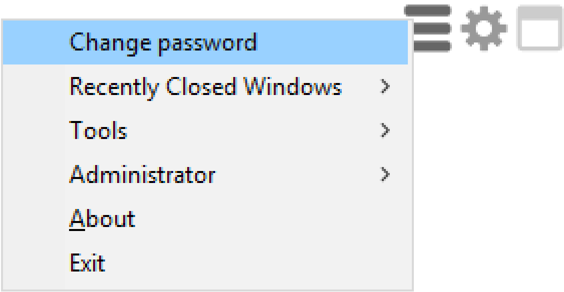
  Advantages:

  - View doesn't get lost when another item is chosen in the main window of the Knowledge Builder
  - View is opened without organizer, thus offering more display space

1.1.1. Global context menu

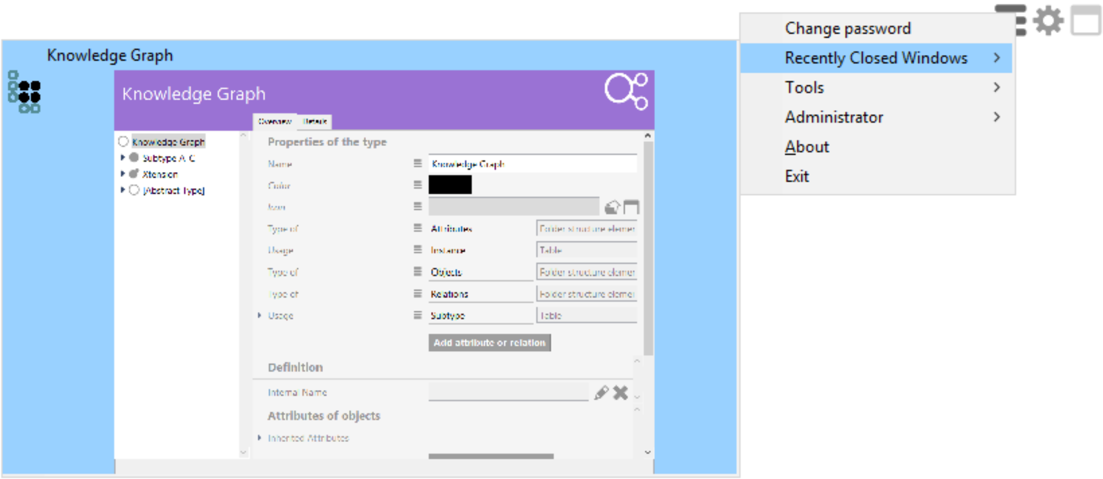
Change password

For the logged in account (non-administrative and administrative), this option provides changing the backend password for accessing the Knowledge Graph by means of the Knowledge Builder.



Recently closed windows

Since i-views 5.4, this feature is included as standard in this menu. Recently closed windows can be reopened again without the need to search for the respective element view.



Tools

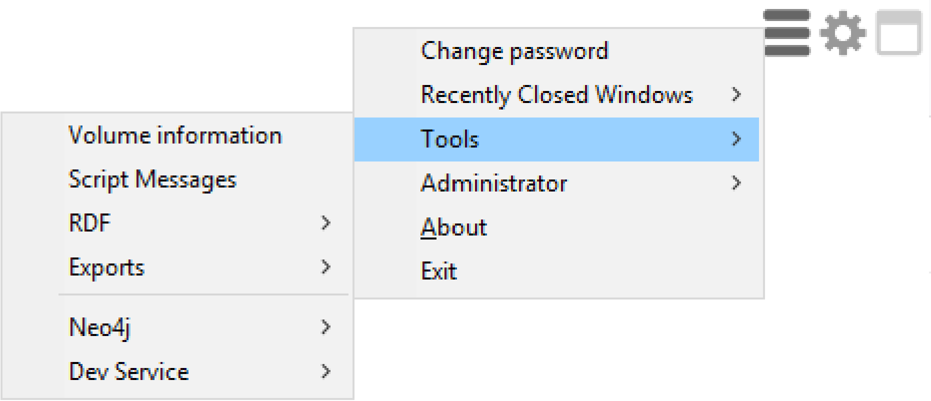
The tools actions provide several functionalities:

- **Volume information:** Shows a dialog window with detailed information about the amount of types and instances of the Knowledge Graph and the size of the volume in which the Knowledge Graph is stored.
- **Script messages:** When using or debugging JavaScript, the script messages dialog can be used for displaying feedback returned by the $k.log() method in the script itself.

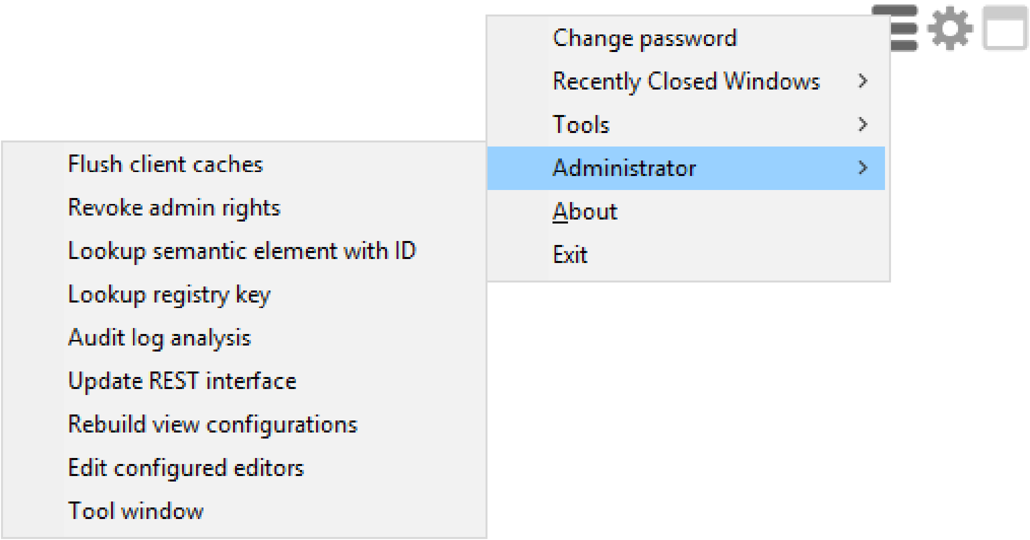
  NOTE The visibility of script messages depends on the configuration of the bridge.

- **RDF:** Provides the options "RDF import" and "RDF export". For more information, see Chapter 1.5.4 "RDF-import and -export".
- **Exports:** Provides export options concerning JavaScript-API, viewconfig JSON-schema, REST-API as OpenAPI 2.0 and KScript XML Schema.

- **Dev Service:** By means of the DEV Service, the i-views browser extension tool can be used to open the respective element/view/panel of the viewconfiguration in the Knowledge Builder by right-clicking onto the relevant part in the browser. The i-views browser extension is an extra which is available upon request. Pay attention that several Knowledge-Builders cannot use the DEV Service at the same time if they use the same DEV Service port as specified in the global settings.



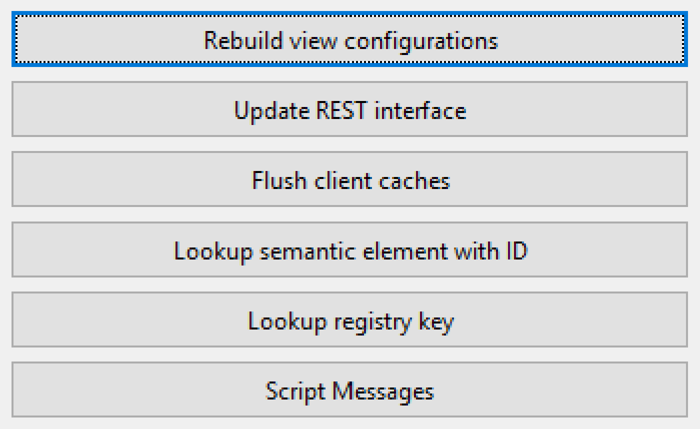
Administrator



- **Flush client caches:** This action triggers in all tools connected to this graph the invalidation of caches. In normal operation caches should be invalidated automatically. In rare cases the invalidation fails, this action can trigger it manually.
- **Revoke admin rights:** This option allows an administrative user to revoke the own administrative access in order to test rights management configuration in the Knowledge Builder. The administrative access can be restored again by deactivating the option.
- **Lookup semantic element with ID:** For analyzing messages returning a semantic element ID (=

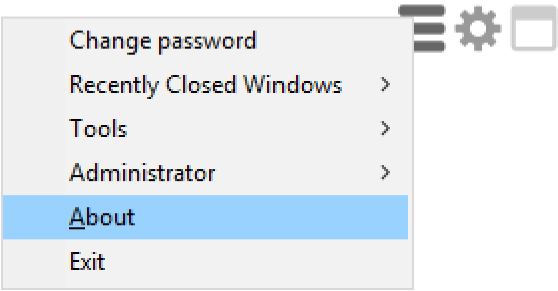
"frame ID"), the ID can be input here to determine the corresponding semantic element.

- **Lookup registry key:** Offers search for registered objects within the Knowledge-Builder (e. g. registered queries, scripts or types).
- **Audit log analysis**
- **Update REST interface:** Global available action for updating the REST interface, serves as substitute when local REST update button ⇩ is not present (visible) at the moment.
- **Rebuild view configurations:** Global available action for updating the viewconfiguration, serves as substitute when the local viewconfiguration update button ⟳ is not present at the moment.
- **Edit configured editors:** If detail editors are configured as a view for elements of the Knowledge Graph, they can be administered here in one space.
- **Tool window:** Provides an overall avalilable tools menu for often needed actions. This comes in handy when many windows are opened at the same time:



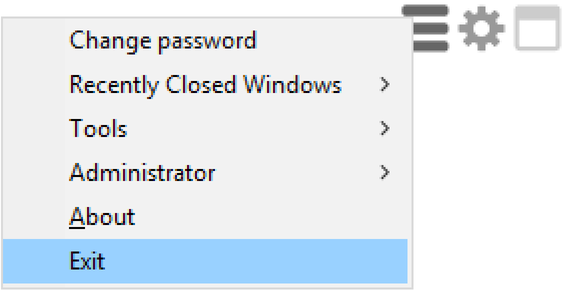
About

Recalls information about configuration, licensing and components of the Knowledge Graph as available in the login window.



Exit

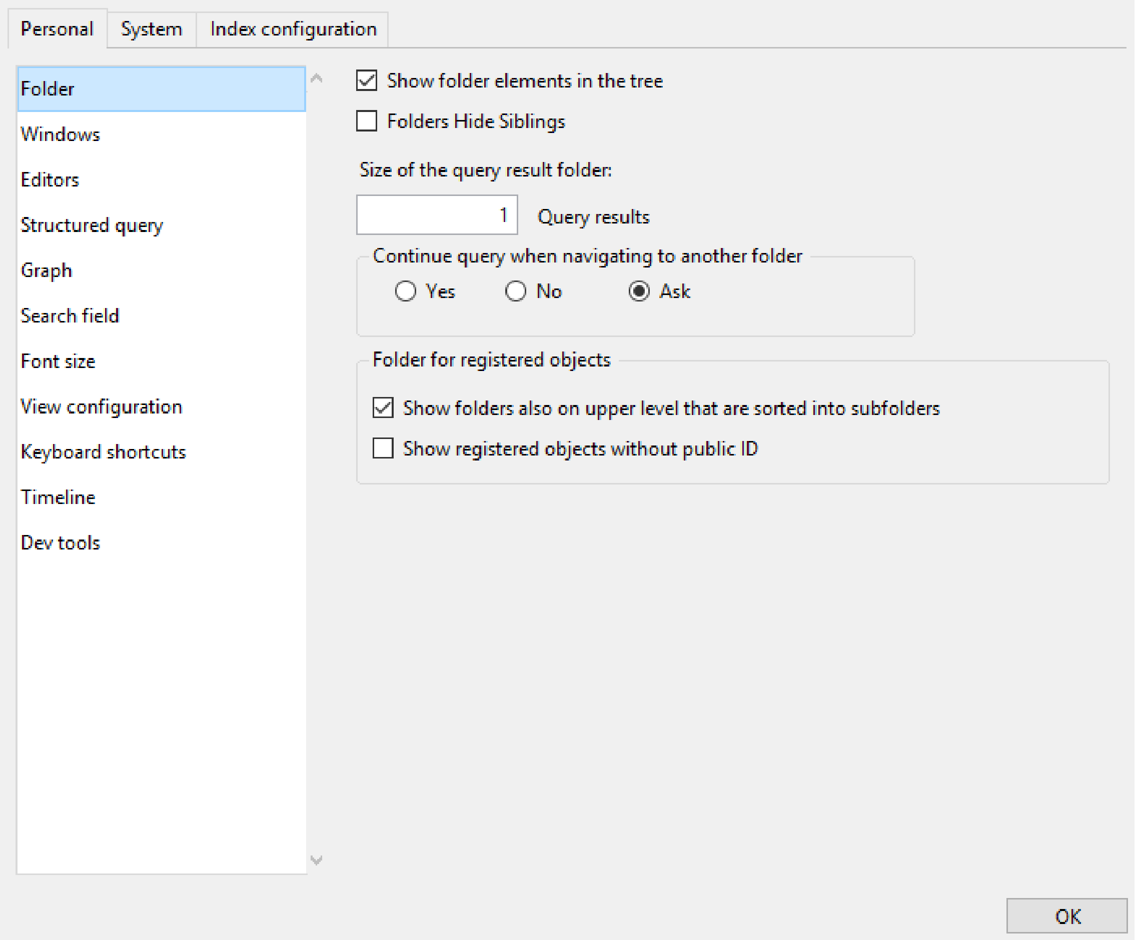
Exits the Knowledge Builder.



1.1.2. Personal settings

Personal settings are available and adjustable for the logged in Knowledge Builder user exclusively. The options are described in the following subchapters in detail.

1.1.2.1. Folder



☑ Show folder elements in the tree

□ Folders Hide Siblings

Size of the query result folder:

1 Query results

Continue query when navigating to another folder

○ Yes ○ No ● Ask

Folder for registered objects

☑ Show folders also on upper level that are sorted into subfolders

□ Show registered objects without public ID

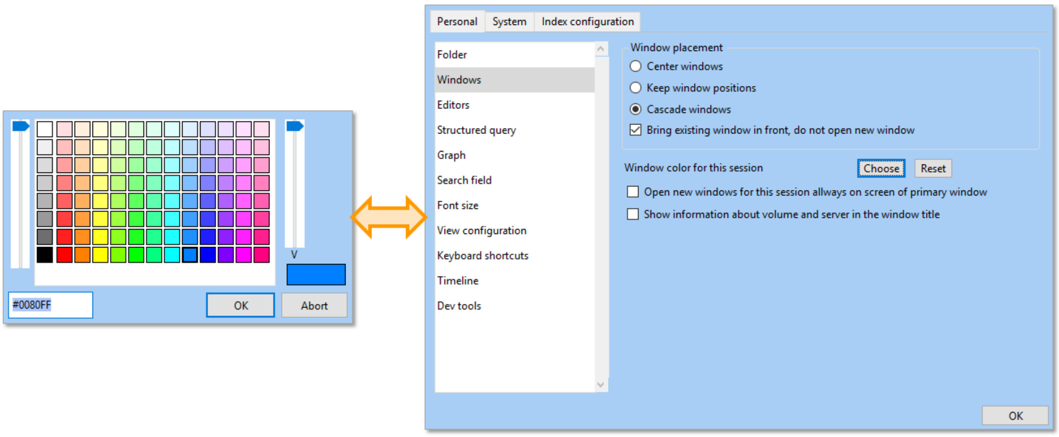
- **Show folder elements in the tree:** Determines whether the content of the folder is displayed as subnode in the folder tree. This option can be useful to improve clarity of the tree in case of many folder sub elements.

- **Folders hide siblings** Should all siblings folders become invisible while a folder is opened.
- **Size of the query result folder:** Number of search result sets of the structured queries that have been recently executed within the KB. The search result set will then be listed within FOLDER > Query results. A search result set entry consists of the timestamped search result list, containing the found semantic elements which can be display in the graph, including their causes. Reducing the size will take effect when executing the next query.
- **Continue query when navigating to another folder**
- **Show folders also on upper level that are sorted into subfolders**
- **Show registered objects without public ID**

1.1.2.2. Windows

The windows settings determines the behaviour of the Knowledge Builder itself and its dialog windows.

- **Center windows:** New windows will always be opened in a centered position.
- **Keep window positions:** Reopens the same view in a window with the same window position.
- **Cascade windows:** Stacks all windows of the same type in a cascading manner so that all their titles can be seen at once.
- **Bring existing window in front, do not open new window:** Reuses windows if they are already open, preventing the increase of opened windows for clarity reasons.
- **Window color for this session:** If several Knowledge Builder are opened at once, this option helps to distinguish between the different Knowledge Graphs by setting a temporary color of the window frames per Knowledge Builder per session:



#0080FF

Personal | System | Index configuration

Folder
Windows
Editors
Structured query
Graph
Search field
Font size
View configuration
Keyboard shortcuts
Timeline
Dev tools

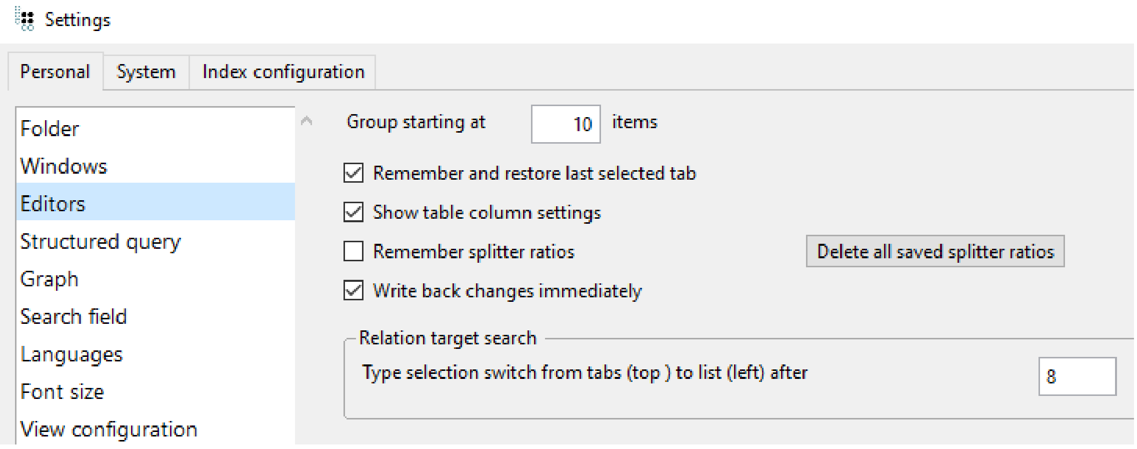
Window placement
○ Center windows
○ Keep window positions
● Cascade windows
☑ Bring existing window in front, do not open new window

Window color for this session

□ Open new windows for this session allways on screen of primary window
□ Show information about volume and server in the window title

- **Open new windows for this session always on screen of primary window:** If several screens are used, a new window always is opened on the main screen.
- **Show information about volume and server in the window title:** For distinguishing between the windows of different Knowledge Graphs from different Knowledge Builder, the Knowledge Graph name and the server will be shown in the title of all opened windows. Serves for the same purpose of clarity like setting the window color.

1.1.2.3. Editors



Group starting at 10 items

☑ Remember and restore last selected tab

☑ Show table column settings

☐ Remember splitter ratios

☑ Write back changes immediately

Relation target search

Type selection switch from tabs (top) to list (left) after 8

Group starting at [...] items

This option leads to properties of the same type being bundled into a dropdown accordion if the given number is reached.

Remember and restore last selected tab

Allows the detail editors being displayed with the same tab selected as in a previous access during the session.

Show table column settings

If enabled, all tables in the KB will have a menu to the right of their filters which replaces the button to clear all filters. The menu contains the option to clear all filters and to customize which columns this specific table should show. These changes can be set to last until the view is left, the KB is closed or until they are changed again.

Remember splitter ratios

If enabled, most views that can be dragged, will save their ratios and restore them the next time they are displayed.

Delete all saved splitter ratios

Resets all splitter ratios saved by the previous option.

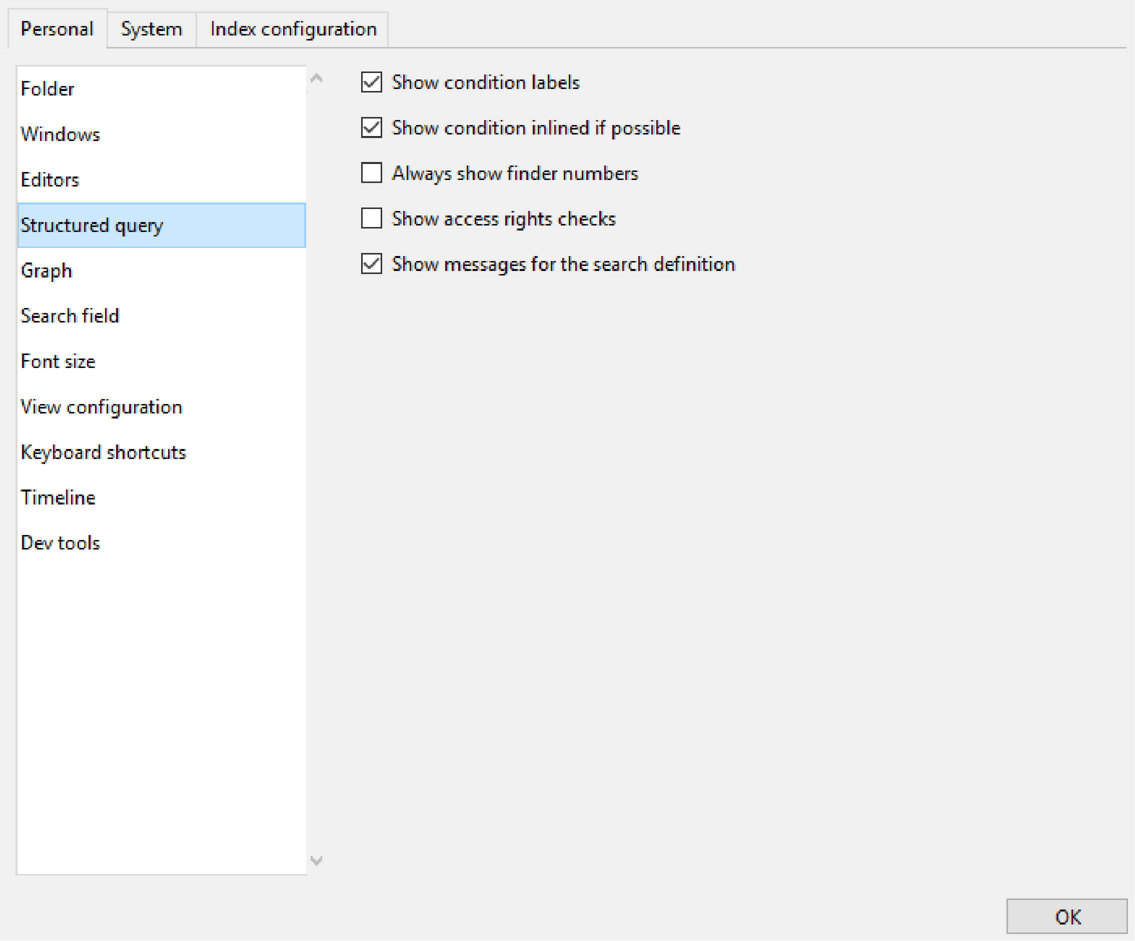
Write back changes immediately

This option takes effect on the backend (Knowledge Builder) only. When activated, element properties are written to the Knowlede Graph immediately in order to validate them against schema rules first before applying the changes. If disabled, properties can be edited and the changes are written to the graph using the additional "Apply" button, which is displayed at the bottom of the editor view. For the web frontend, (buttons with) actions of the action type "validate" and "save" serve this purpose.

Type selection switch from tabs (top) to list (left) after [...]

When the relation target selection dialog is openend due to editing a relation, the relation targets are not shown separated by tabs on the top edge but in forms of categories listed on the left side of the dialog.

1.1.2.4. Structured query



Personal | System | Index configuration

- Folder
- Windows
- Editors
- Structured query
- Graph
- Search field
- Font size
- View configuration
- Keyboard shortcuts
- Timeline
- Dev tools

☑ Show condition labels

☑ Show condition inlined if possible

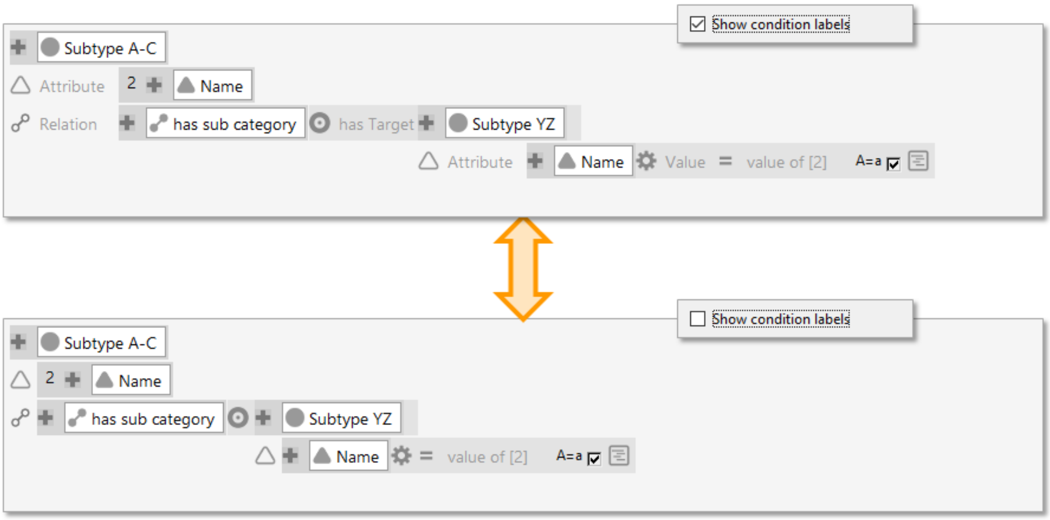
□ Always show finder numbers

□ Show access rights checks

☑ Show messages for the search definition

OK

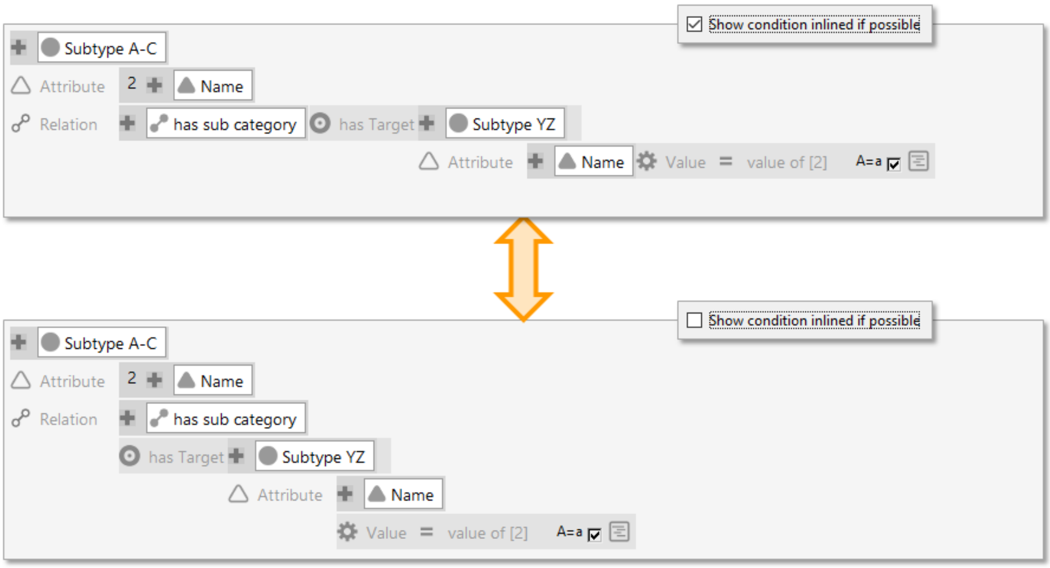
- **Show condition labels:** If activated, query labels for properties are shown additionally to the symbol:



☑ Show condition labels

□ Show condition labels

- **Show condition labels inlined if possible:** If enabled, relation targets and attribute values are shown inline to their property types:

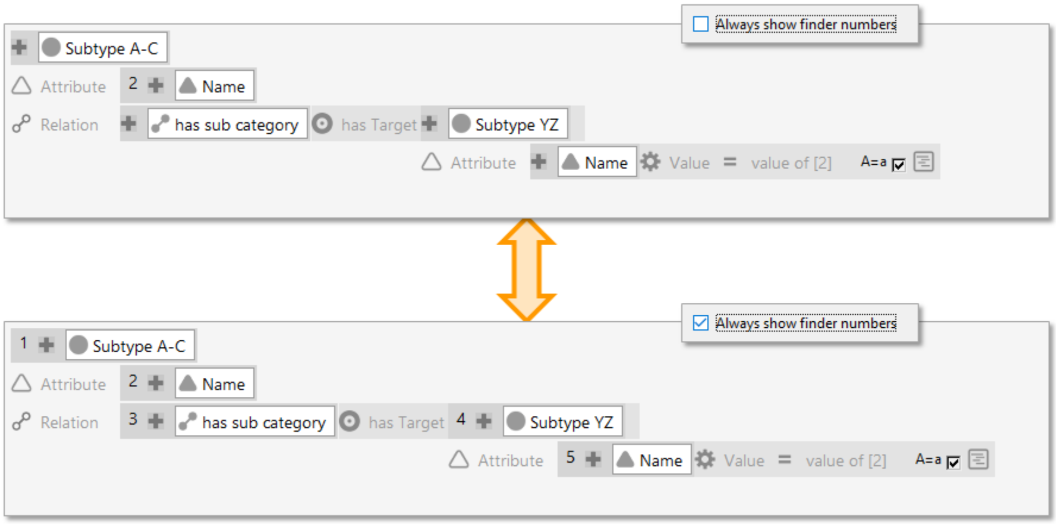


☑ Show condition inlined if possible

Subtype A-C
Attribute 2 Name
Relation has sub category has Target Subtype YZ
Attribute Name Value = value of [2] A=a

☐ Show condition inlined if possible

- **Always show finder numbers:** Within structured queries, all elements are identified by means of an inherent numbering system. However, the numbers only will be shown when needed for building the query or when adding result columns to the results list. When this option is enabled, the numbering will keep persistent:



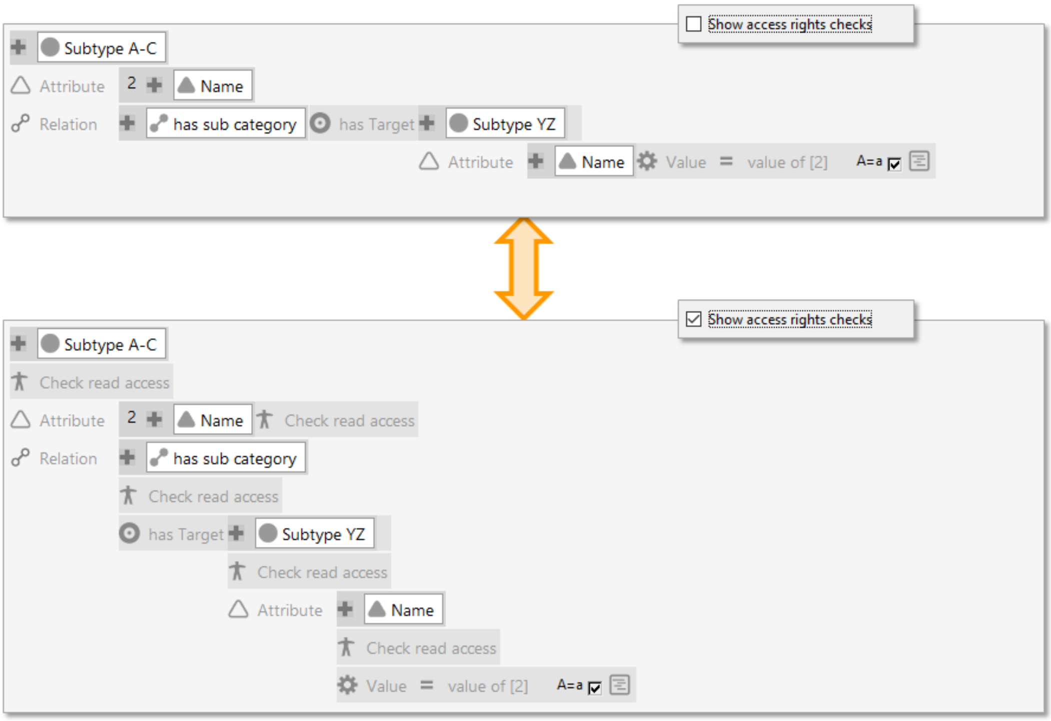
☐ Always show finder numbers

Subtype A-C
Attribute 2 Name
Relation has sub category has Target Subtype YZ
Attribute Name Value = value of [2] A=a

☑ Always show finder numbers

1 Subtype A-C
Attribute 2 Name
Relation 3 has sub category has Target 4 Subtype YZ
Attribute 5 Name Value = value of [2] A=a

- **Show access rights checks:** Shows additionally the associated access rights concerning the particular property.

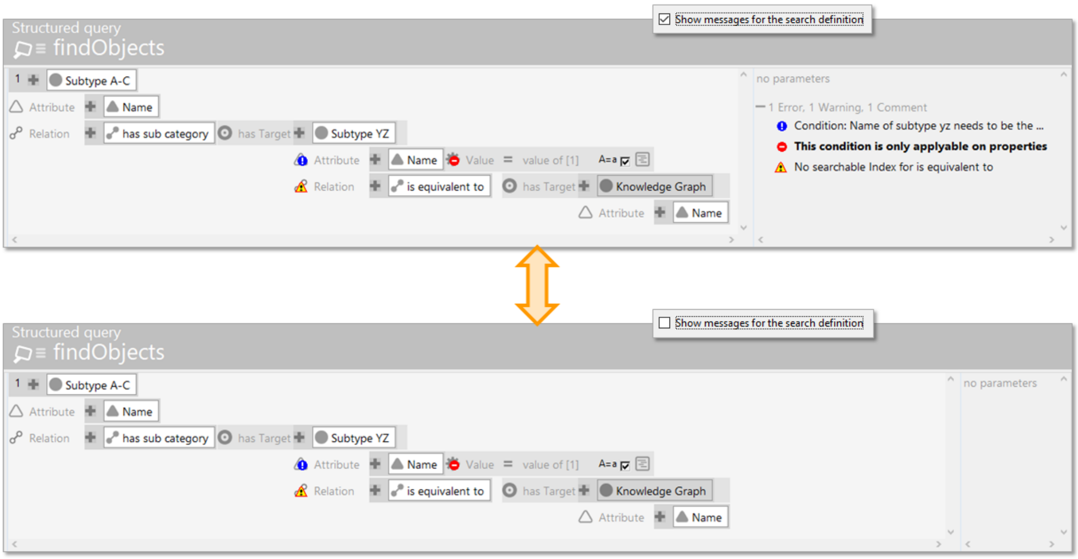


☐ Show access rights checks

☑ Show access rights checks

- **Show message for the search definition:** This option enables messages for comments, warnnigs and errors to be shown at the right side legend of the structured query.

NOTE Besides that, the local option "Suppress warning" is available via context menu for each query label.



☑ Show messages for the search definition

no parameters

1 Error, 1 Warning, 1 Comment

Condition: Name of subtype yz needs to be the ...

This condition is only applyable on properties

No searchable Index for is equivalent to

☐ Show messages for the search definition

1.1.2.5. Graph

The graph options are for the graph editor within the Knowledge Builder. For settings about the graph in forms of the net navigator component, see chapter 3 "vcm-plugin-net-navigator".

Some of these settings can be overridden in the graph editor. This will only apply to that specific graph editor window.

Show tooltips with details

If true, hovering a node will display a tooltip listing the objects properties.

Max. number of lines for tooltips

Defines at what point the tooltip will be cut off.

Auto hide nodes

Automatically hides surplus nodes as soon as the number of desired nodes is exceeded and shown. The number can be set in the input field "max. new nodes" in the toolbar.

Auto layout nodes

Automatically implements the layout function for newly displayed nodes.

Node alignment

If enabled, nodes will snap to the x or y coordinates of close nodes when moving them.

Use Cairo library to display the graph

Can only be activated if the cairo library can be found by the KB executable. If true, the library will be used to display certain things.

Default zoom

The zoom level at which a graph editpr will be openend.

Max. new nodes

If a node / an object has many adjacent objects, it often doesn’t make much sense to display all of them when clicking on the drag point. This defines how many new nodes can be added at once without being prompted to choose specific nodes.

Max. label length

Defines after how many symbols the node labels will be cut off.

Node size

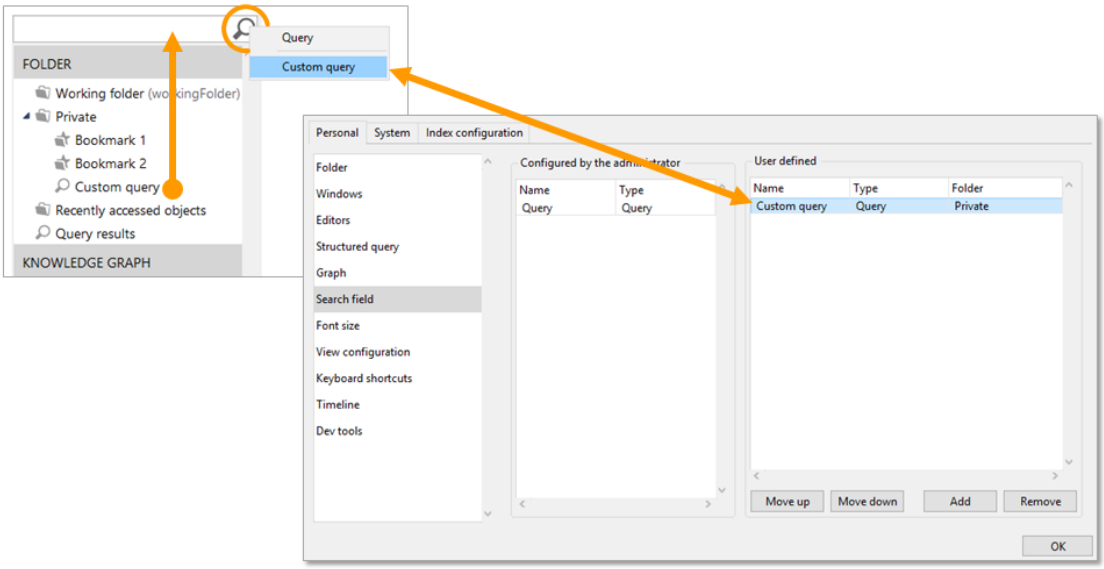
Defines at which size a node is added to a graoh.

Legend configuration

Here you can define types, which will always be displayed in the legend when opening a graph editor.

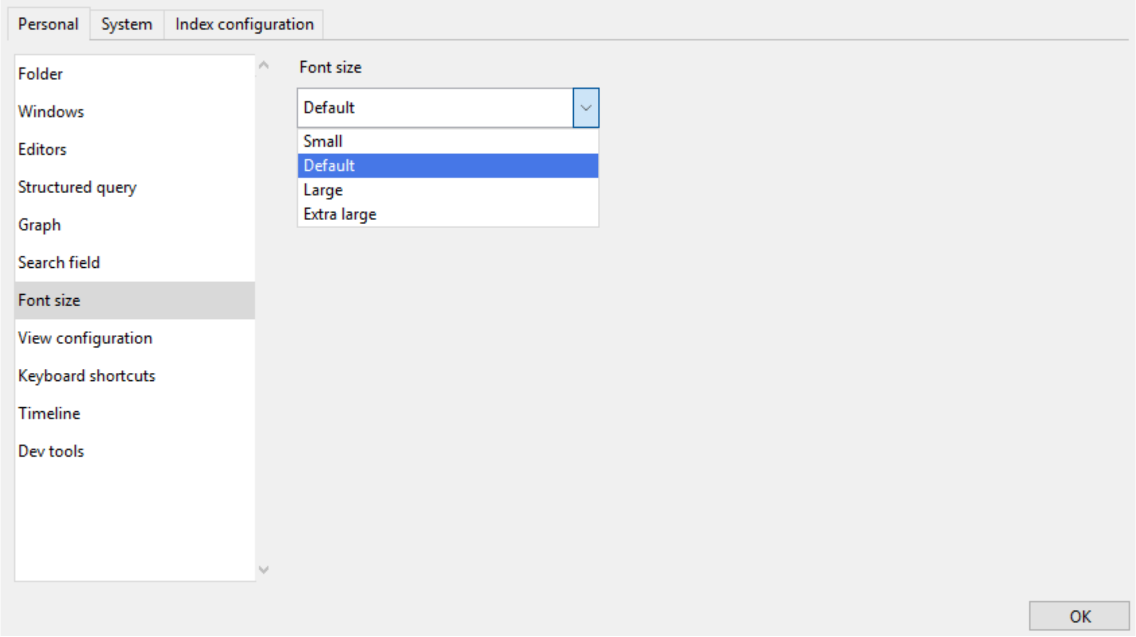
1.1.2.6. Search field

For the Knowledge Builder search, queries from within the working folder or the private folder can be added by drag&drop. To administer the added queries (e. g. removing them), the search field settings are used. Added queries are available in a dropdown selection when clicking on the query-button next to the search field.



1.1.2.7. Font size

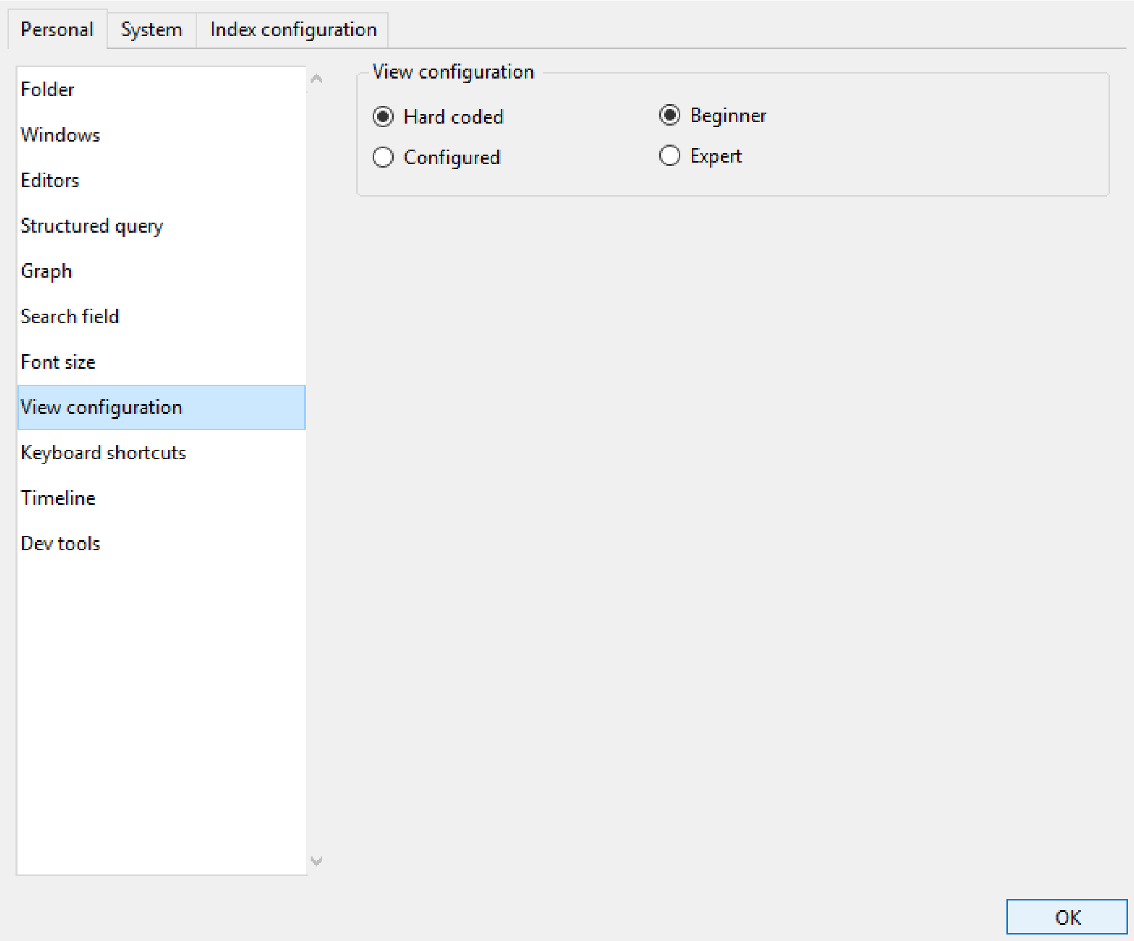
This option allows the permanent setting of the font size within the Knowledge Builder. When changing the font size, an example text is shown. Changes only take effect after restart of the Knowledge Builder.



1.1.2.8. View configuration

The view configuration options take effect on the behaviour of the Knowledge Builder view configurations exclusively. Options for the view configuration of the web frontend are configured by means of the viewconfiguration mapper settings.

- **Hard coded / Configured:** For the folder structure within the organizer of the Knowledge Builder, type-dependent view configurations can be specified. The options "Hard coded" and "Configured" therefore allow switching between the default Knowledge Builder view configuration and the customized view configuration. If certain types have a customized view configuration which are defined for both the detail view and the folder structure, the folder structure view will have priority when the view confguration is switched to "Configured".
- **Beginner/Expert:** Concerning the viewconfiguration mapper, two kinds of user oriented views of the viewconfiguration mapper itself can be selected: "Beginner" splits up the configuration tabs of the detail editors into "Configuration" and "Extended", "Expert" shows all configuration options at once.



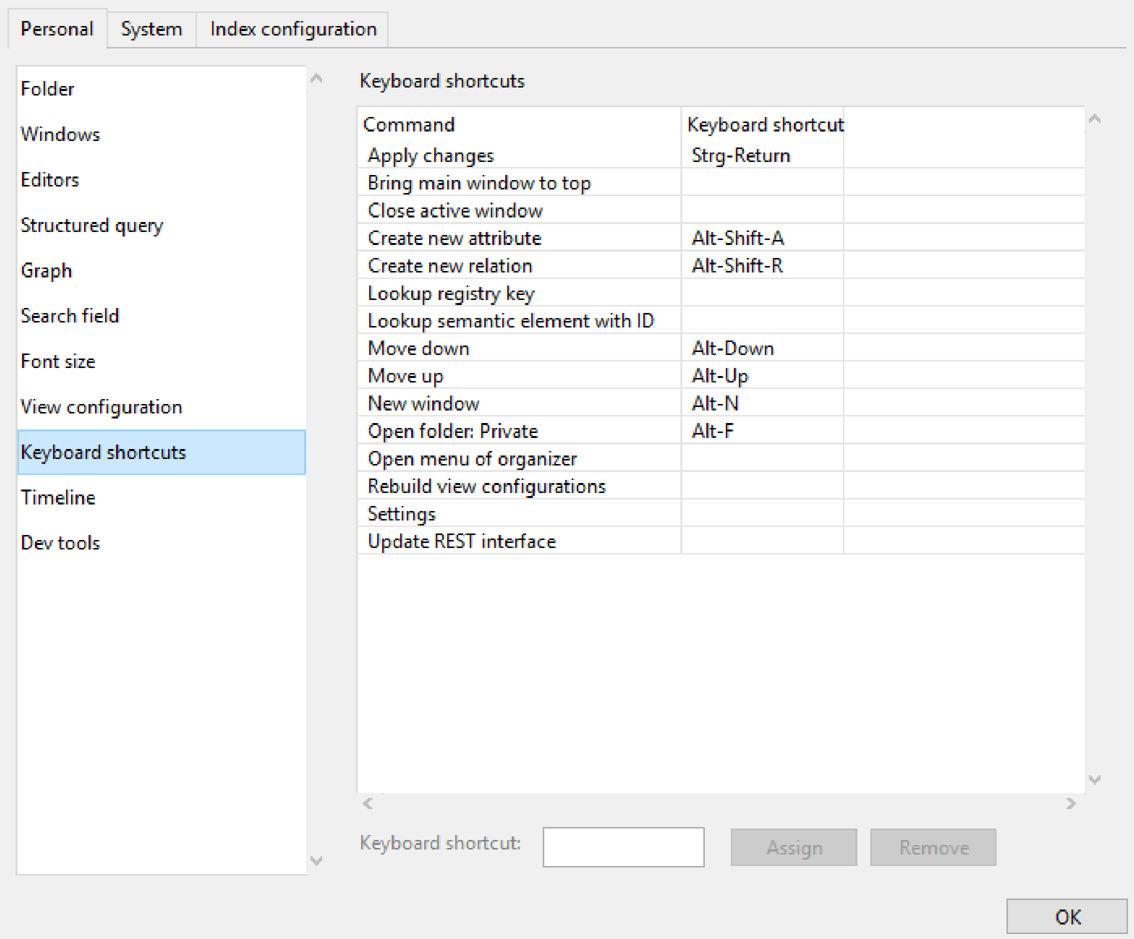
View configuration

● Hard coded ● Beginner

○ Configured ○ Expert

1.1.2.9. Keyboard shortcuts

For the ease of use, custom shortcuts can be defined for the actions as shown in the following image:



Command	Keyboard shortcut
Apply changes	Strg-Return
Bring main window to top	
Close active window	
Create new attribute	Alt-Shift-A
Create new relation	Alt-Shift-R
Lookup registry key	
Lookup semantic element with ID	
Move down	Alt-Down
Move up	Alt-Up
New window	Alt-N
Open folder: Private	Alt-F
Open menu of organizer	
Rebuild view configurations	
Settings	
Update REST interface	

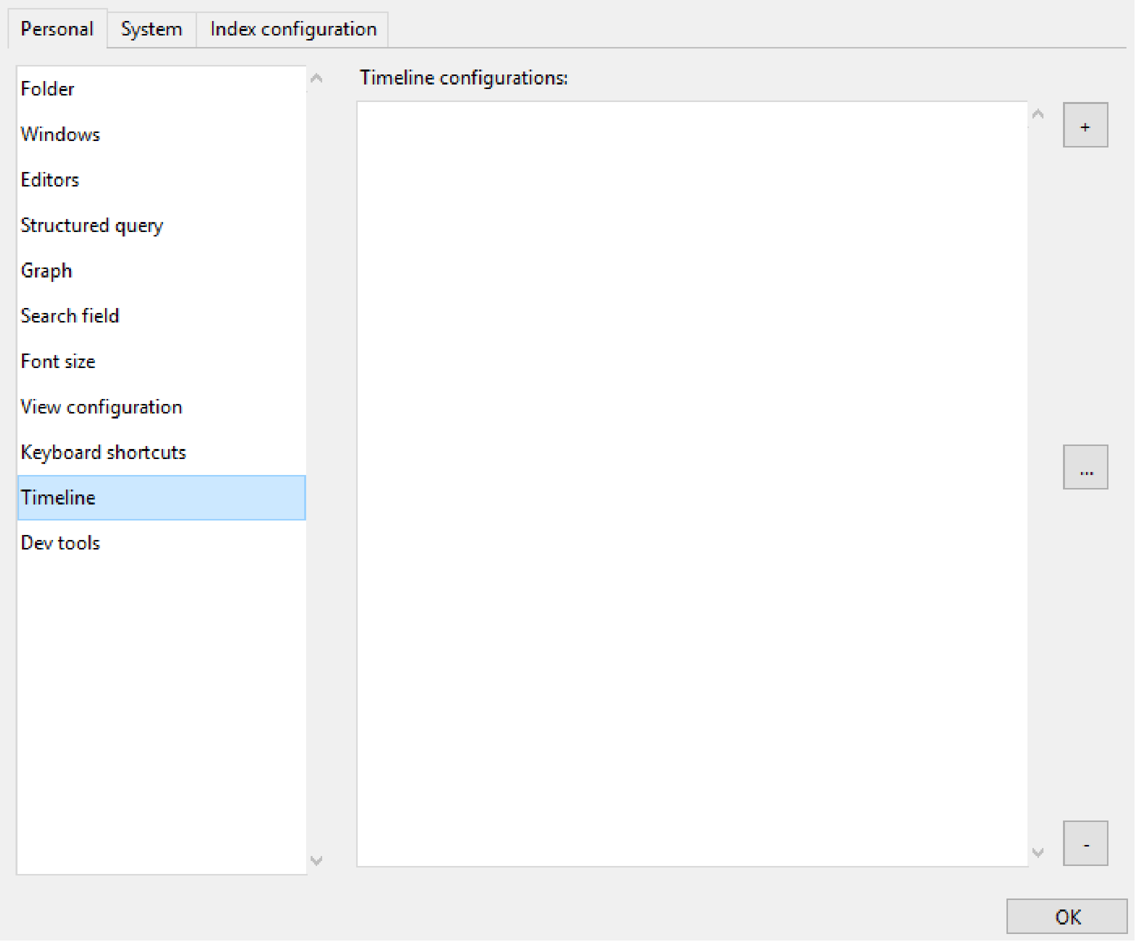
Often there are also inherent shortcuts available. If applicable, these shortcuts are described in forms of a **Shortcut** note in the relevant chapter.

An overall principle of shortcuts within the Knowledge Builder: The combination of Ctrl + Click removes items (e. g. elements from structured queries or properties in the detail editor) or makes them draggable (e. g. drag&drop of semantic elements from the Knowledge Builder into the graph editor).

Within JavaScripts, elements assigned by internal names can be invoked with Ctrl + o (provided a registry key or a configuration name has been given to the element so that it actually can be referenced). Equivalent terms within one script can be browsed easily by marking the term through double-clicking it and then by pressing Ctrl + g.

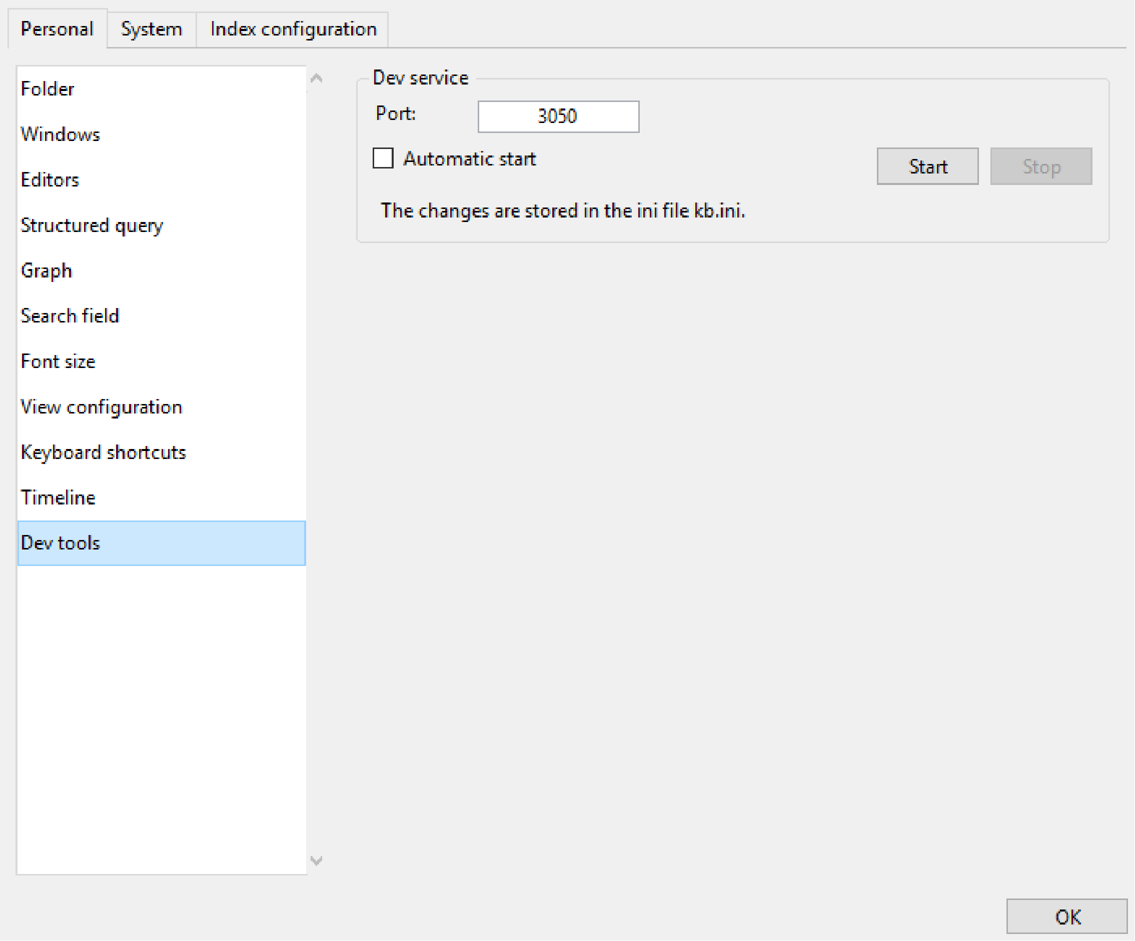
1.1.2.10. Timeline

The timeline feature allows configuring a timeline view for the Knowledge Builder by means of a structured query. For the timeline, several element types can be chosen as a dimension for the timeline for their instances to be displayed according date values, flexible time values or time intervals. The timeline view then needs to be configured as view configuration for the Knowledge Builder to be applicable.



1.1.2.11. Dev tools

These options allow the setting of the port used for Dev services and if the Dev service is to be started automatically when the Knowledge Builder is started. When using several Knowledge Builder at the same time, the corresponding Dev services only can be used in parallel when they are given different ports.



Port: 3050

☐ Automatic start

The changes are stored in the ini file kb.ini.

1.1.3. System settings

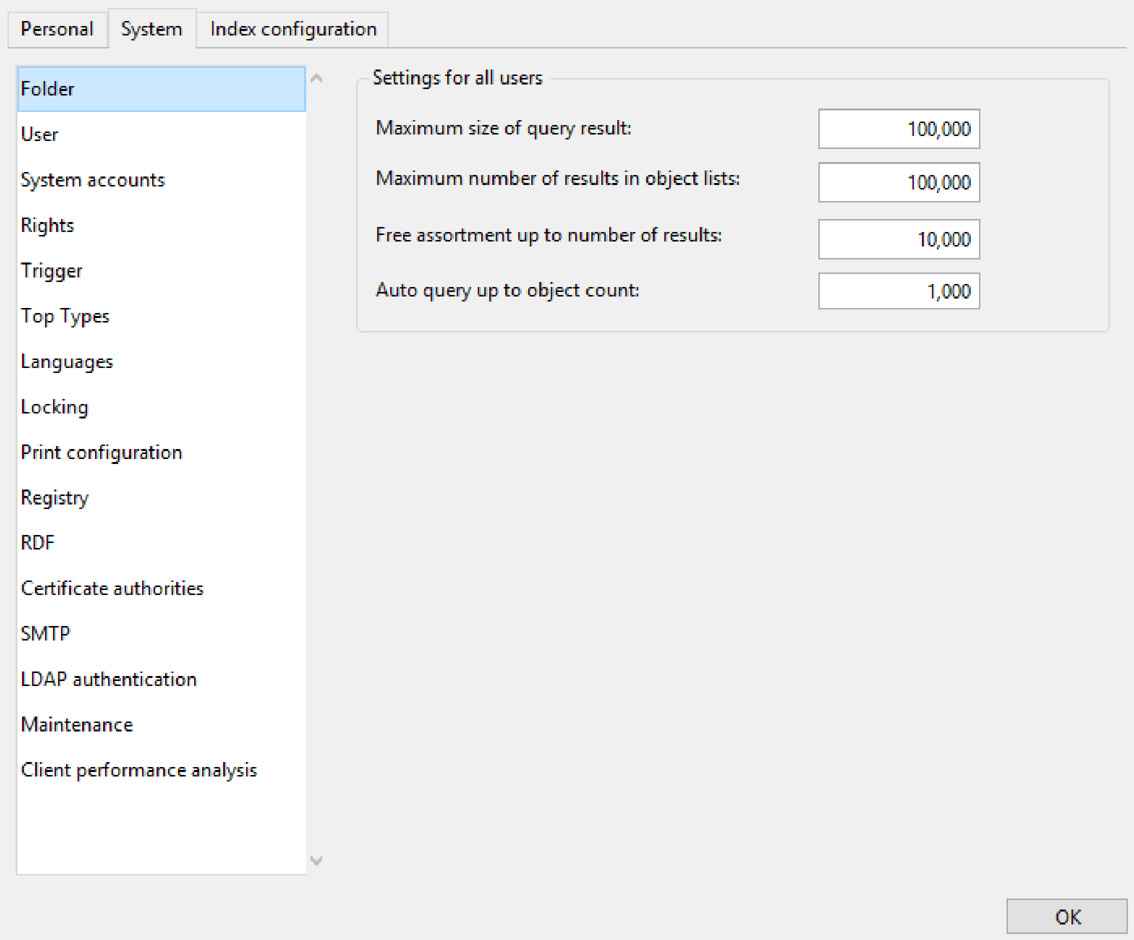
The system settings are available for accounts with administrator status only and allow overall configuration of system-wide settings for the Knowledge Builder.

1.1.3.1. Folder

The folder options are for optimizing the list views according specific use cases when dealing with large amount of data - thus improving usability by limiting additional features which otherwise are active by standard.

- **Maximum size of query result:** Determines the maximum amount of hits that will be processed and rendered for query results.
- **Maximum number of results in objects lists:** Determines the maximum amount of objects that is displayable for an object list. If the amount exceeds the limit, a message will be shown accordingly in the objects list instead of the listed objects.
- **Free assortment up to number of results:** The entries of object lists can be assorted by means of the column header actions and filtering options. For large amounts of objects, the assortment can be disabled to prevent unnecessary load.

- **Auto query up to object count:** Determines the amount of list objects up to which the table queries the list results automatically. If the number of objects to be shown in the object list exceeds the given limit, the query for rendering the table will only start if the search button is clicked by the user. Additionally, for object lists within the KB everey table configuration has separate options for activating the query automatically (tab "KB").



Settings for all users

Maximum size of query result: 100,000

Maximum number of results in object lists: 100,000

Free assortment up to number of results: 10,000

Auto query up to object count: 1,000

1.1.3.2. User

This option category administers the backend users that have access to the Knowledge Graph via the Knowledge Builder.

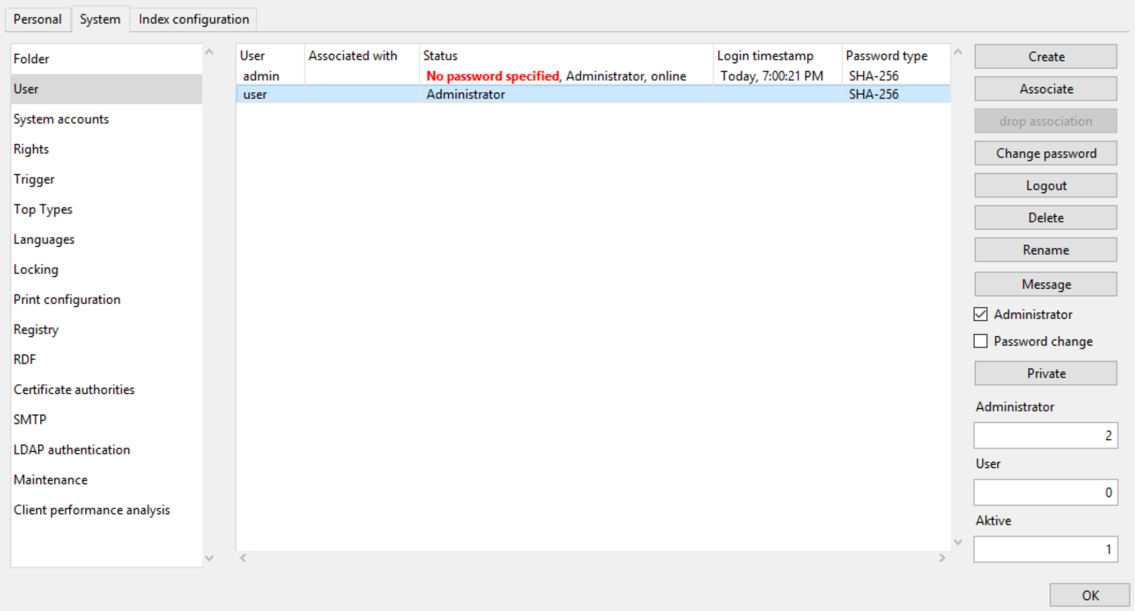
- **Create:** Creates a backend user for the Knowledge Builder.
- **Associate:** Associates the backend user to a frontend user account object.
- **Drop association:** Removes the association of the frontend user account object from the backend user account.
- **Change password:** Allows changing the own password or resetting the password of another backend user. Additionally a password change can be enforced for the first/next login.
- **Logout:** Logs out the selected user.
- **Delete:** Deletes the selected user. **Caution:** Deleting the own user is also possible, leading to an

immediate deletion and logout!

- **Rename:** Renames the selected user.
- **Message:** Sends a message to the selected user, similar to sending a message via the community section on the lower left corner of the Knowledge Builder. If the person is not available because not logged in, a message can be sent here nevertheless. The message will be displayed to the user at next login.
- **Administrator:** Determines if the selected user is an administrator.

NOTE In order to enable non-administrative access to the Knowledge-Builder, a dedicated KB folder structure has to be configured in advance which provides access to the relevant content and functions.

- **Password change:** Enforces a password change for the selected user at next login.
- **Private:** Shows the content of the private folder of the selected user account.
- **Administrator:** Shows the amount of user accounts with administrator status.
- **User:** Shows the number of user accounts with user status.
- **Active:** Shows the number of currently logged in users/administrators.



User	Associated with	Status	Login timestamp	Password type
admin		No password specified, Administrator, online	Today, 7:00:21 PM	SHA-256
user		Administrator		SHA-256

☑ Administrator

□ Password change

Administrator: 2

User: 0

Aktive: 1

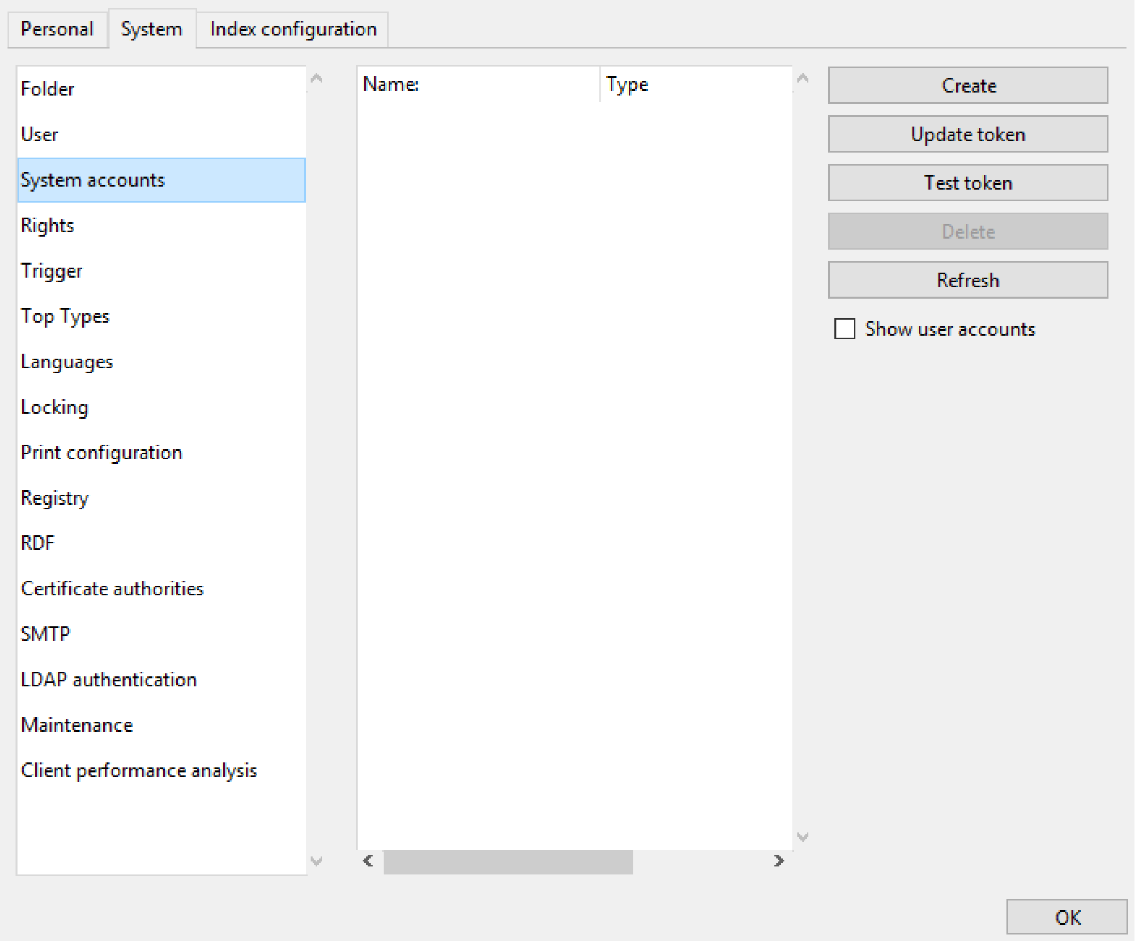
1.1.3.3. System accounts

System accounts are needed for authentication of external services that communicate via TCP/IP and services that communicate via the REST interface (e. g. bridges for webfrontend).

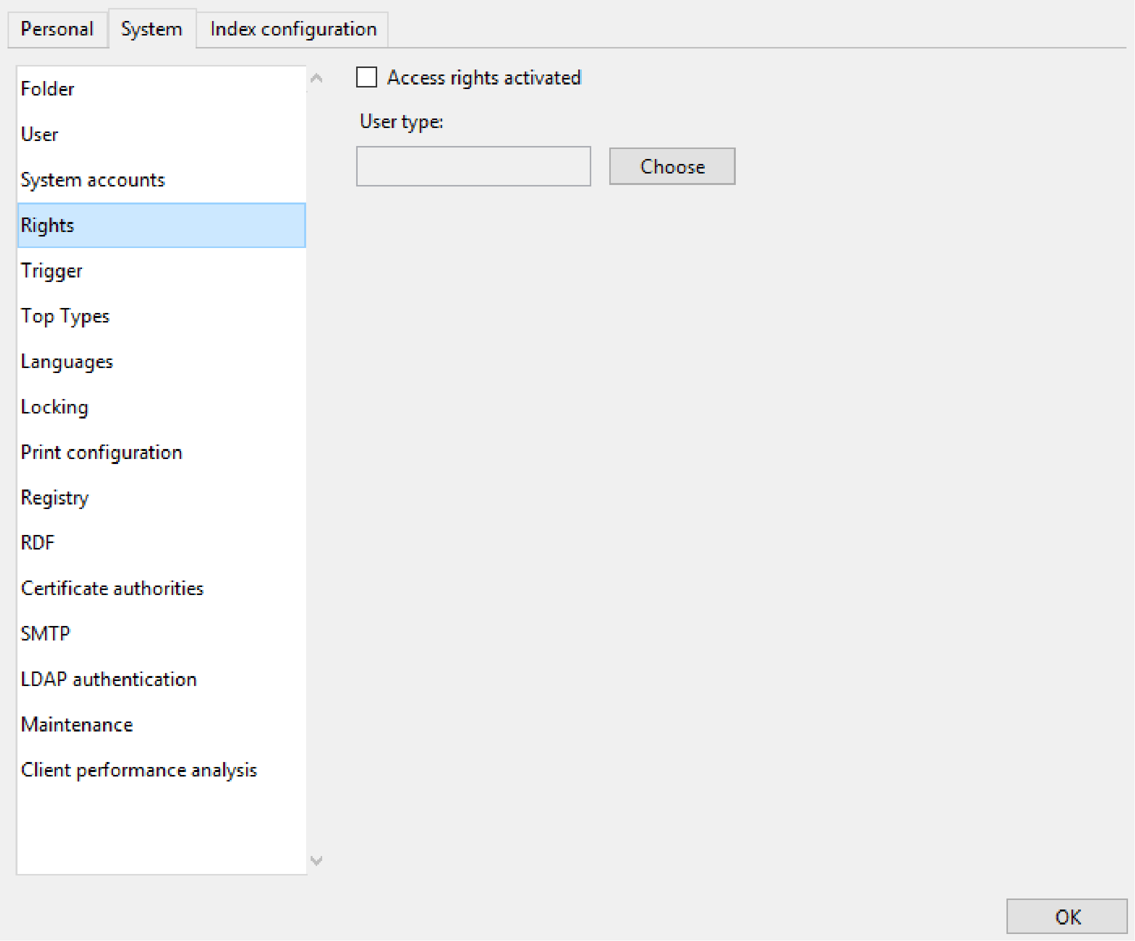
- **Create:** Creates a system account; after specifying a name, a token will be shown only once for copying it for further usage (e. g. for bridge *.ini files).
- **Update token:** Updates a token and shows a suggestion once. Here a token value can also be

entered manually.

- **Test token:** Allows testing if a token string is valid.
- **Delete:** Deletes the selected system account.
- **Refresh:** Refreshes the current system account view.
- **Show user accounts:** Shows the user accounts additionally to the system accounts.



1.1.3.4. Rights



□ Access rights activated

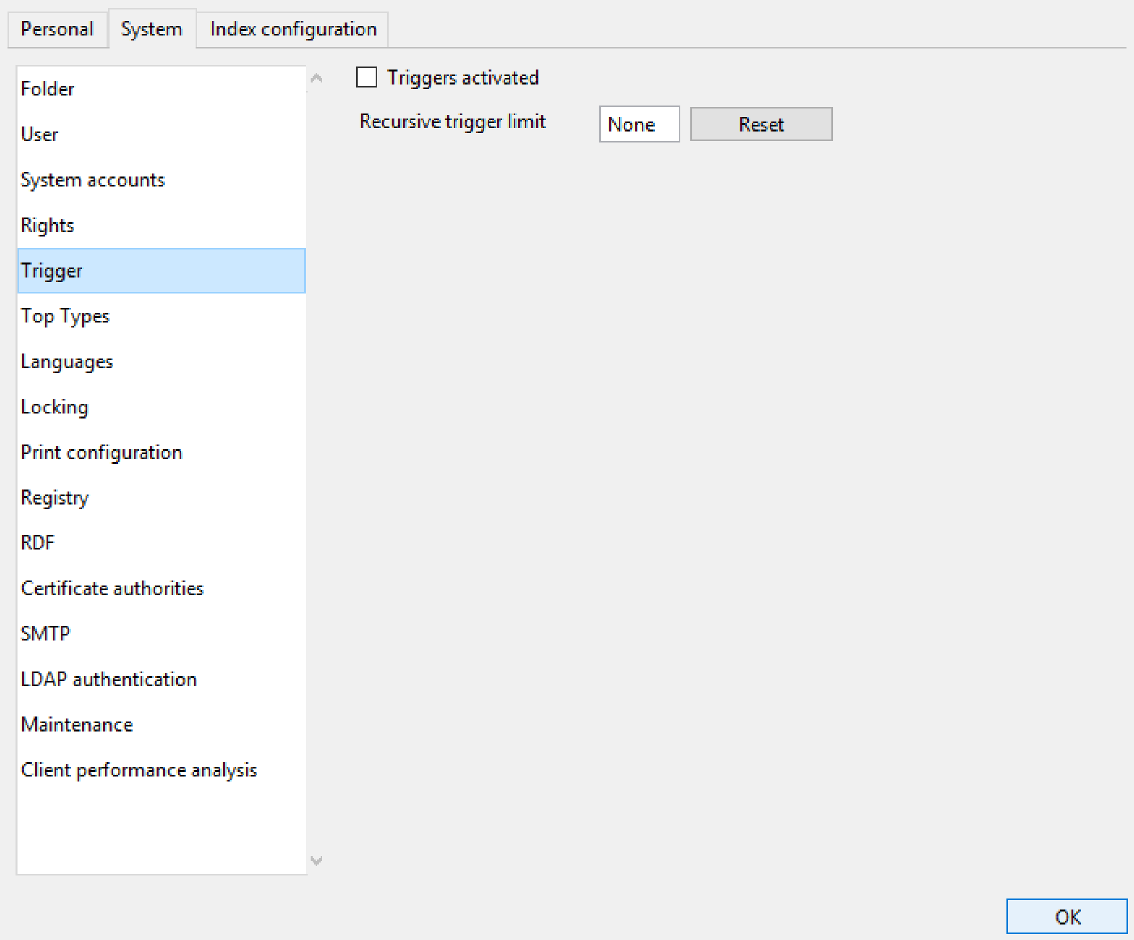
User type:

- **Access rights activated:** The access rights system and its access rights checks are only activated if this option is enabled. The access rights system comprises the access check of web-frontend users.
- **User type:** Specifies which type is used for access rights checks. Objects of this type can be assigned as account-instances to a backend users in the administration section "User".

1.1.3.5. Trigger

This option enables/disables the trigger system.

NOTE The trigger section only is available within the TECHNICAL part when triggers are activated via this option in the global settings.



□ Triggers activated

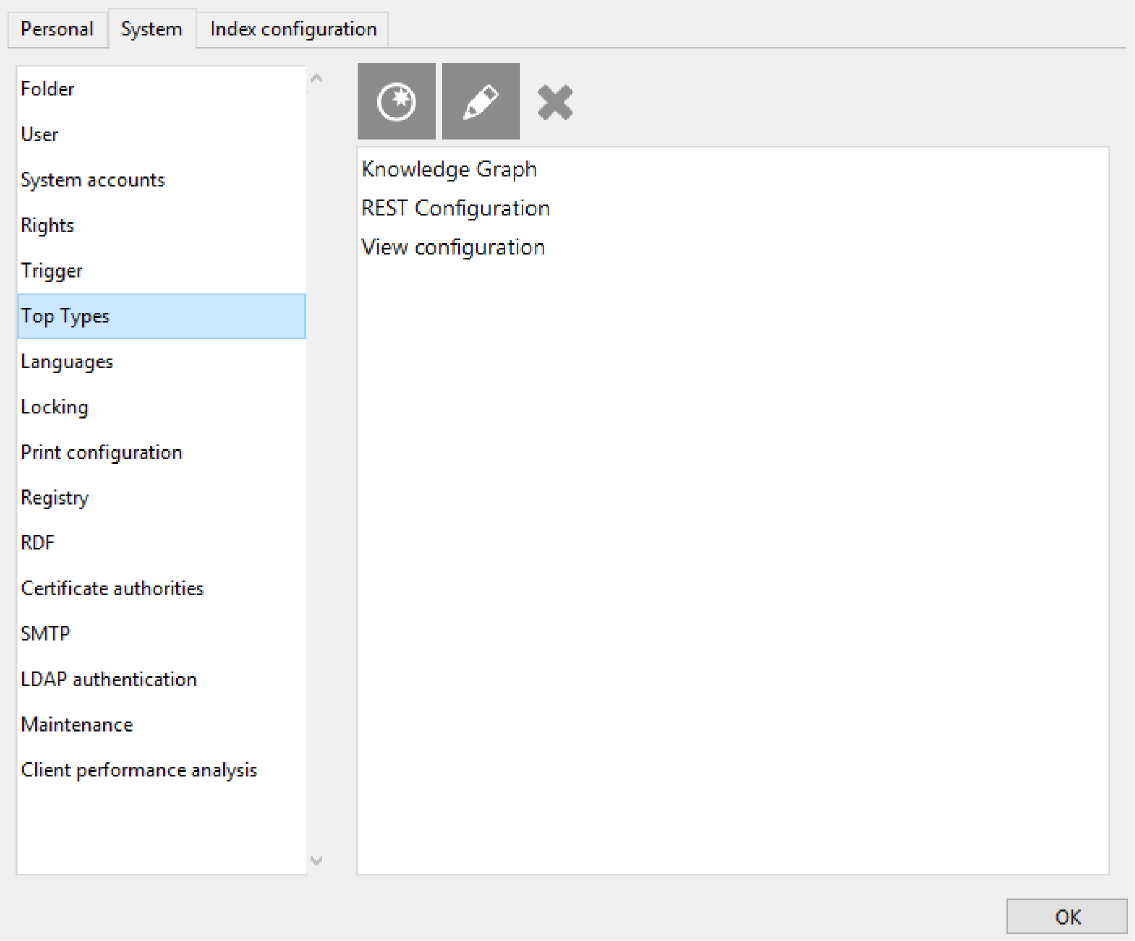
Recursive trigger limit None

1.1.3.6. Top types

Top types can be administered her. Each top type comprises one separate Knowledge Graph within the Knowledge Builder, shown as separate entry in the organizer of the Knowledge Builder.

By standard, properties are handled separately for each top typ and isolated from one top type to another, but can be accessed by queries nevertheless.

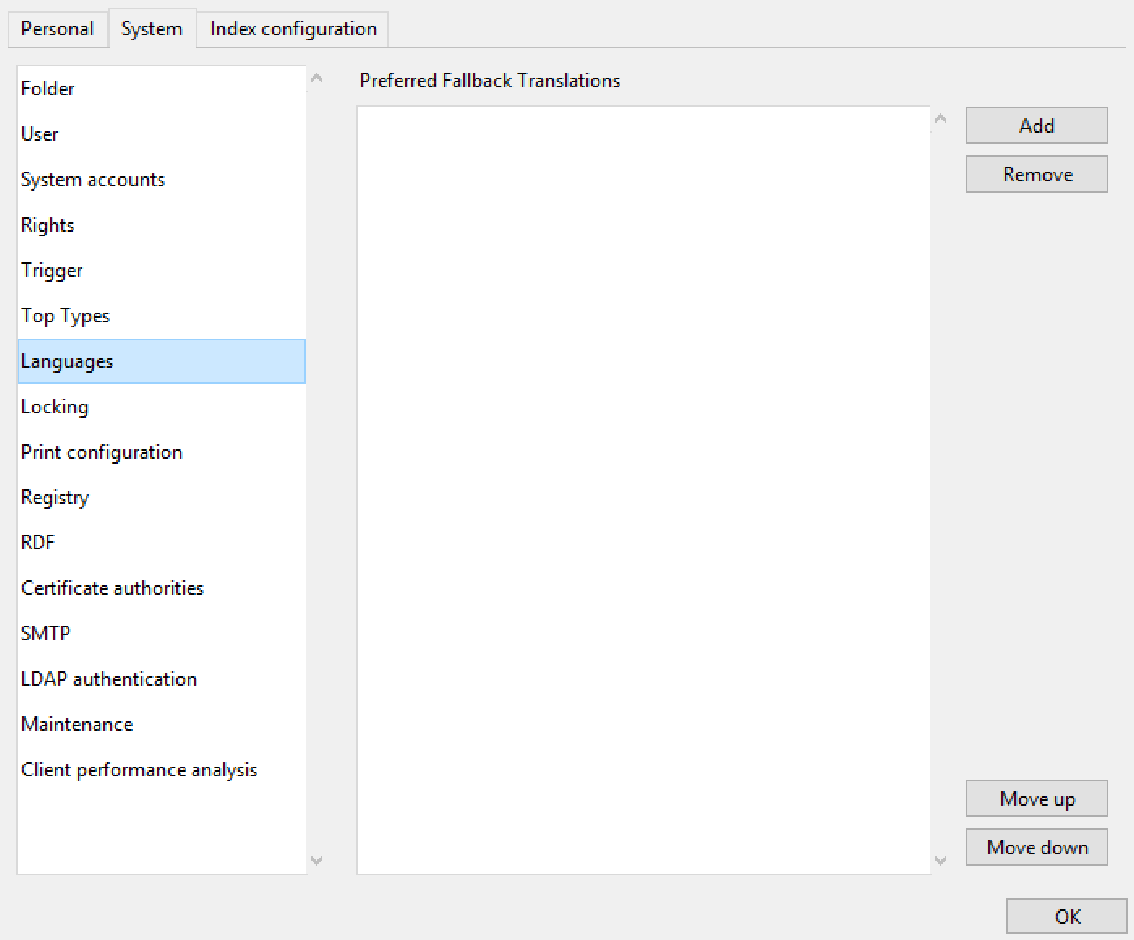
Each top typ is a subtype of the overall "Top-level type" from the core Knowledge Graph.



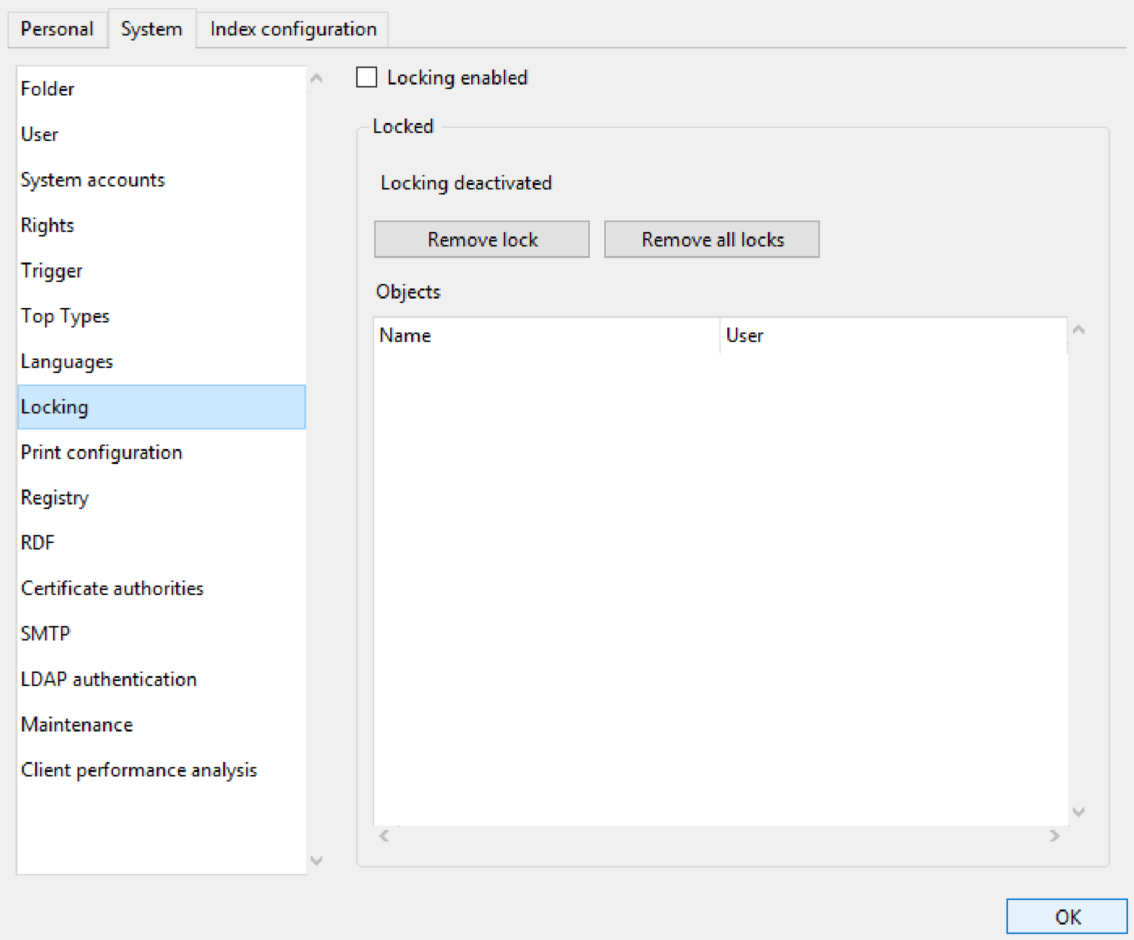
Knowledge Graph
REST Configuration
View configuration

1.1.3.7. Languages

When the value of a given translated attribute is not present in the sessions' current language, this list defines the order of languages which are to be used as replacement values.



1.1.3.8. Locking



□ Locking enabled

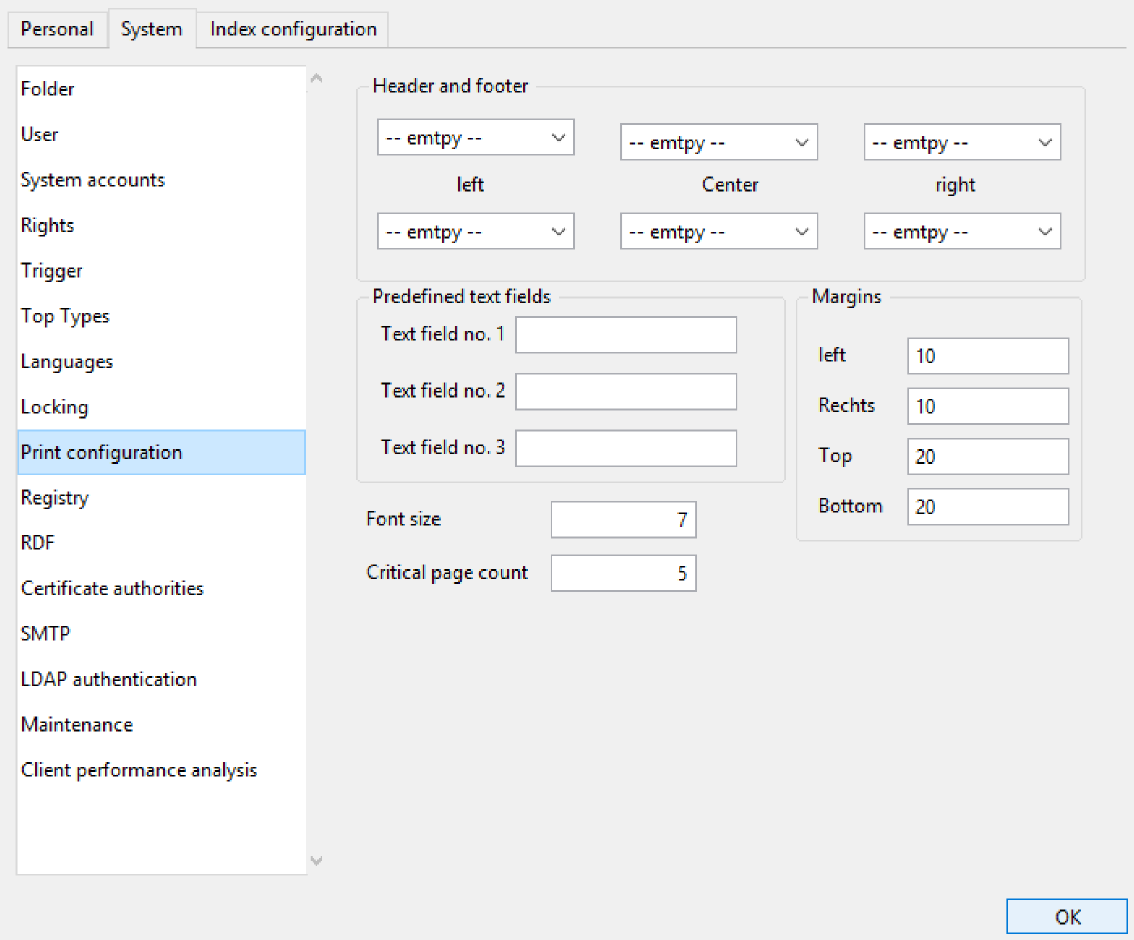
Locked

Locking deactivated

Objects

Name	User

1.1.3.9. Print configuration



Header and footer

	left	Center	right
Header	-- emtpy --	-- emtpy --	-- emtpy --
Footer	-- emtpy --	-- emtpy --	-- emtpy --

Predefined text fields

Text field no. 1

Text field no. 2

Text field no. 3

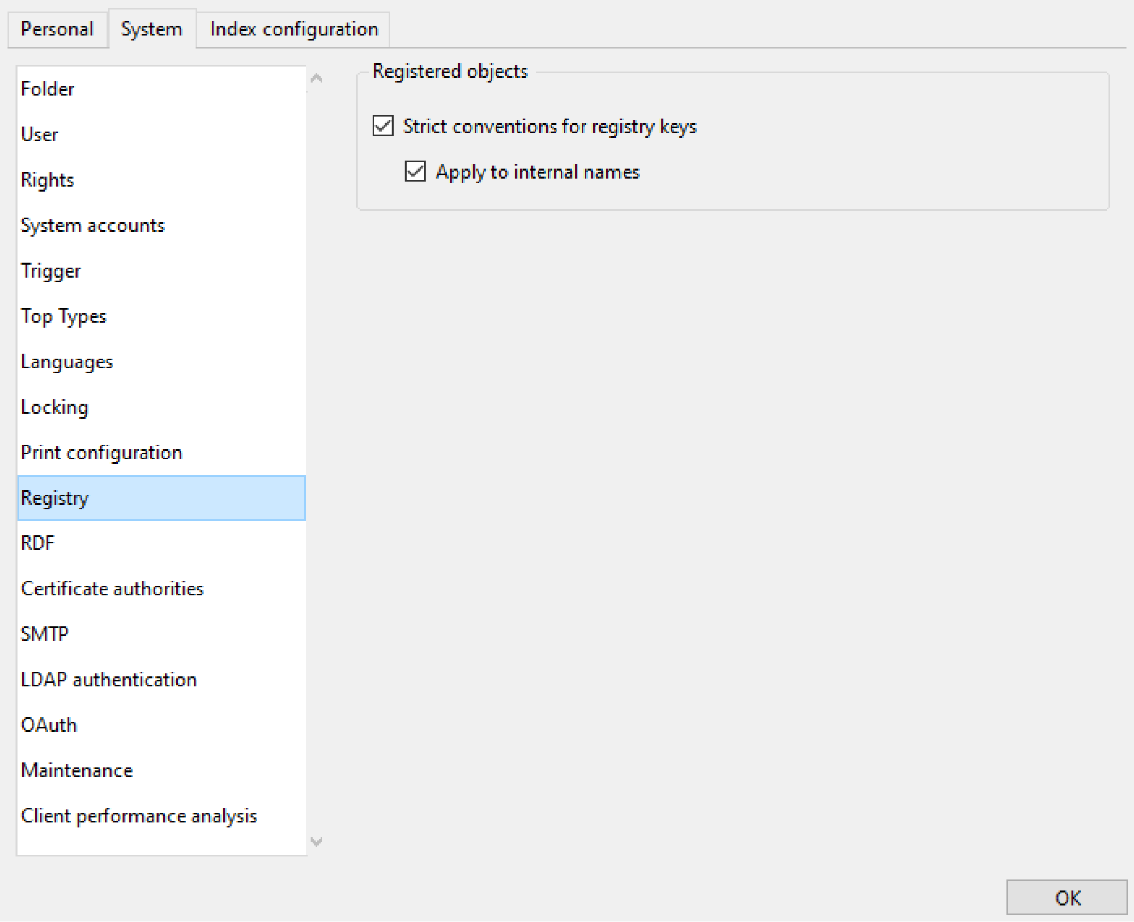
Margins

left	10
Rechts	10
Top	20
Bottom	20

Font size 7

Critical page count 5

1.1.3.10. Registry



Registered objects

☑ Strict conventions for registry keys

☑ Apply to internal names

Strict conventions for registry keys: The conventions apply when creating a registry key and e. g. in case of an XML schema transfer between volumes by means of the admin tool. The strict conventions are as follows:

- All 26 letters of the ASCII code table (small letters and capital letters as well)
- Signs period ".", underscore "_" and dash "-"
- The first sign should be a letter

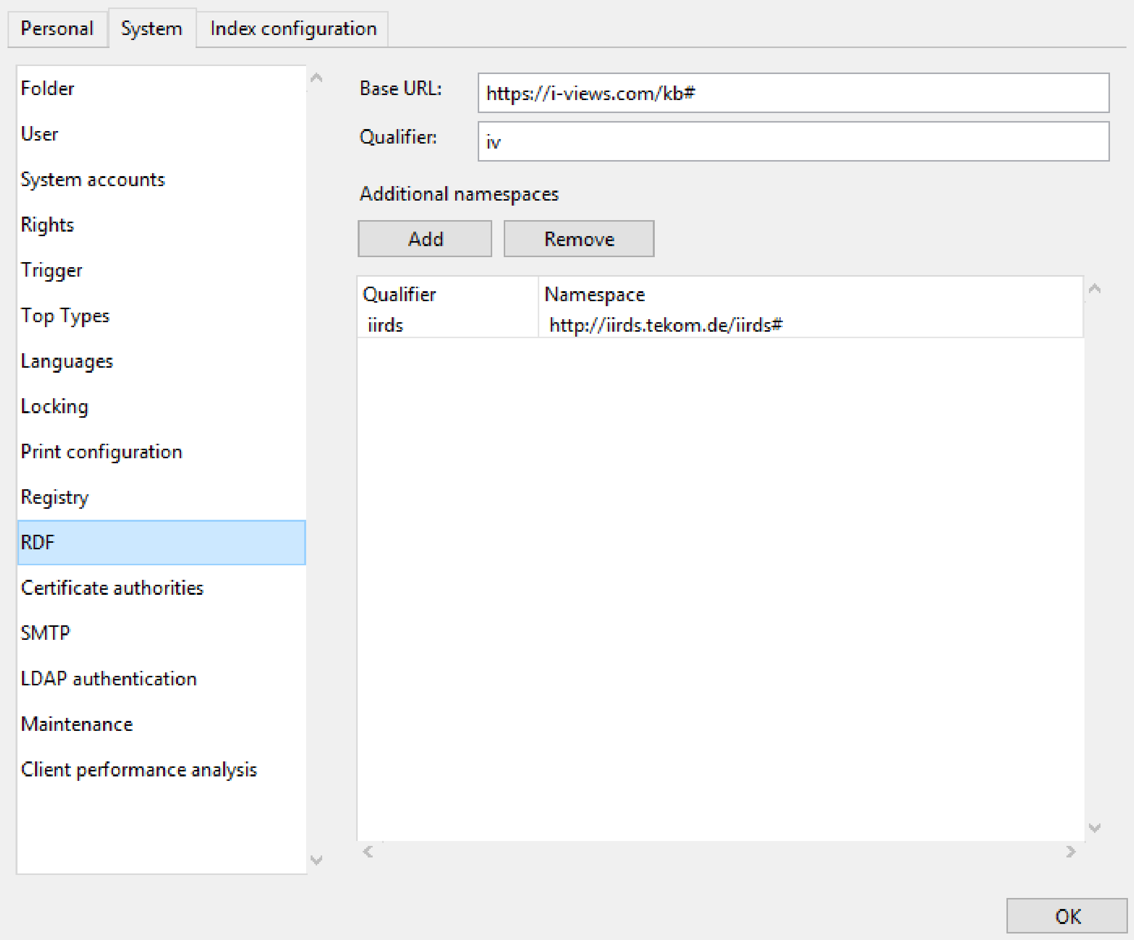
NOTE The conventions are **case insensitive** , which means that a distinction of registry keys by small letters and capital letters is not possible. Example: "myVolume.myQuery1" and "myVolume.MYQuery1" cannot be used within the same volume. This also applies to the XML schema transfer from one volume to another.

Apply to internal names: If enabled, the conventions also apply to internal names.

1.1.3.11. RDF

The RDF options comprise the settings for base URL, qualifier and additional namespaces that come into account for identification entry nodes when importing or exporting RDF files.

NOTE The additional namespaces are for export only. For more information, see chapter "RDF-import and -export" of the users' manual.

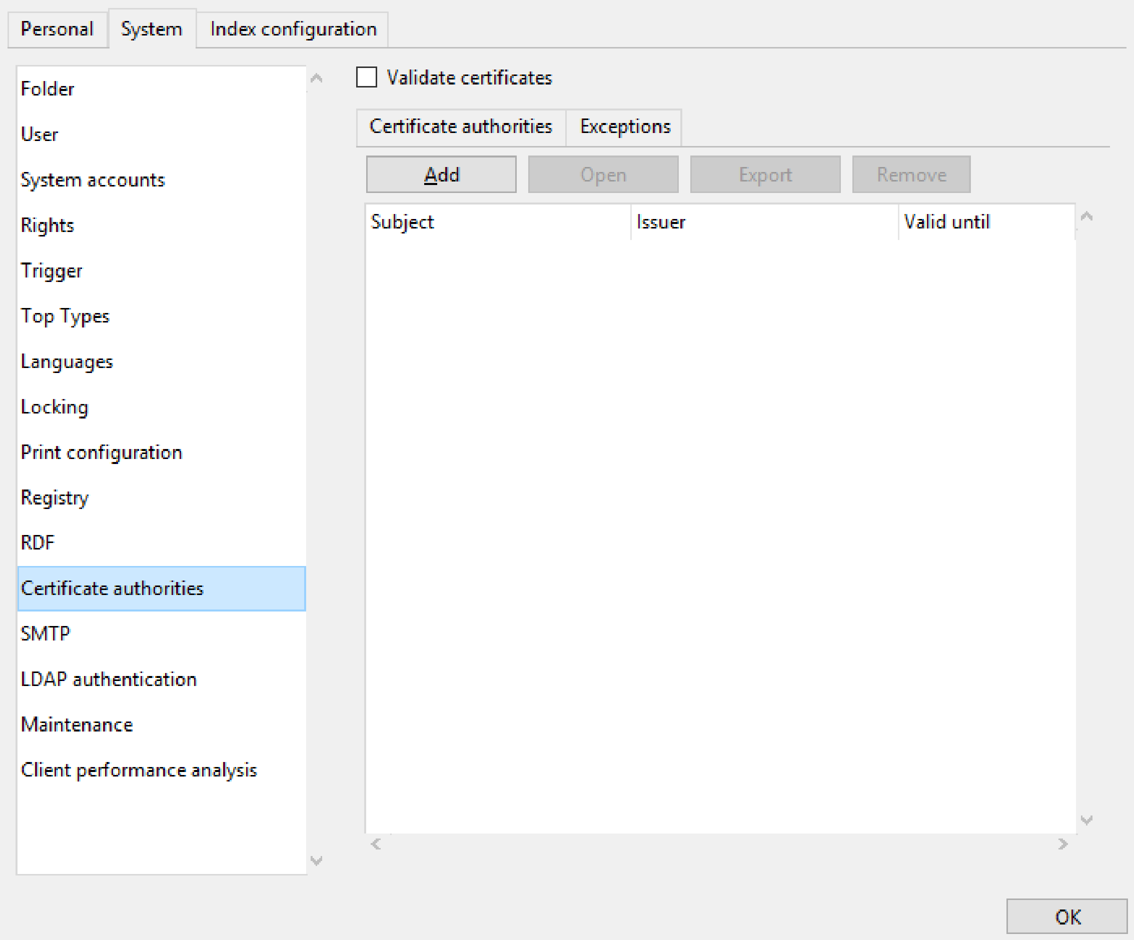


Base URL: https://i-views.com/kb#

Qualifier: iv

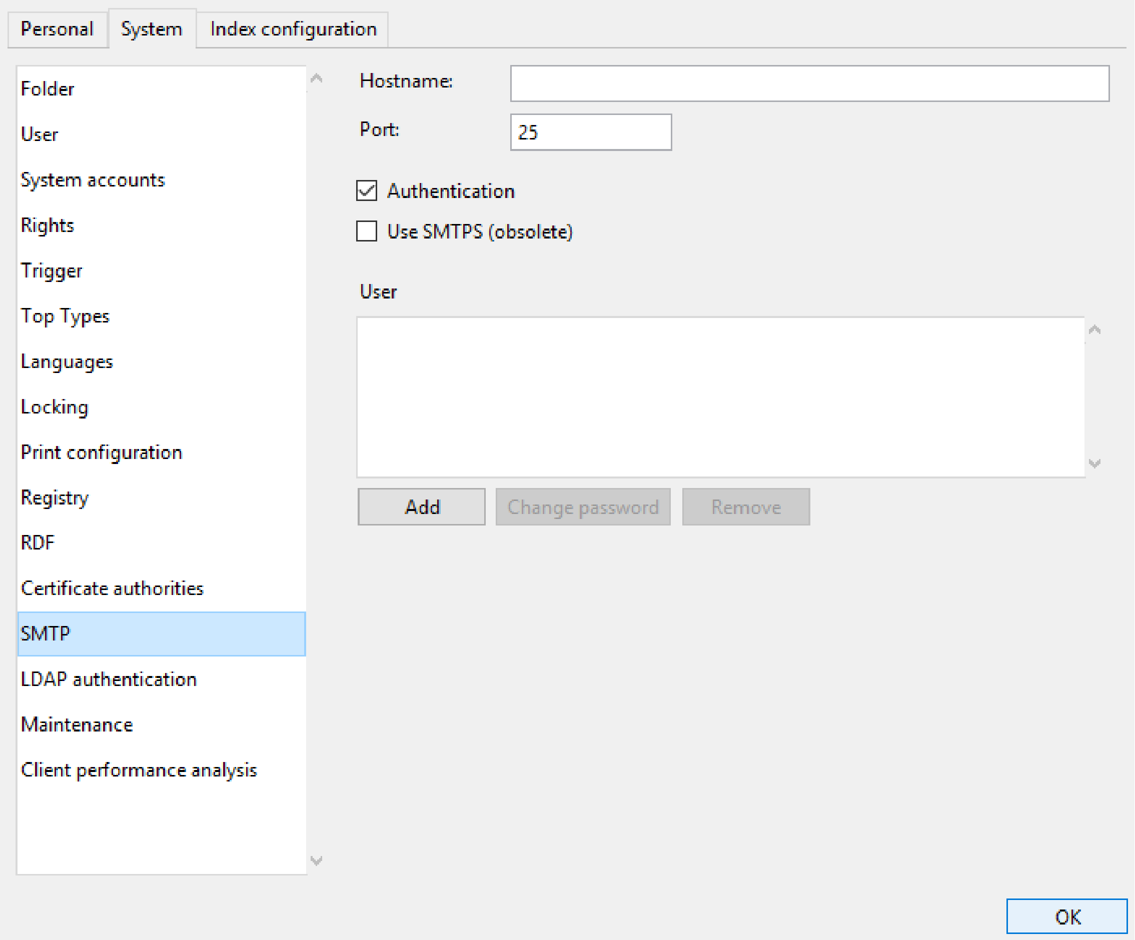
Qualifier	Namespace
iirds	http://iirds.tekom.de/iirds#

1.1.3.12. Certificate authorities



□ Validate certificates

1.1.3.13. SMTP



Hostname:

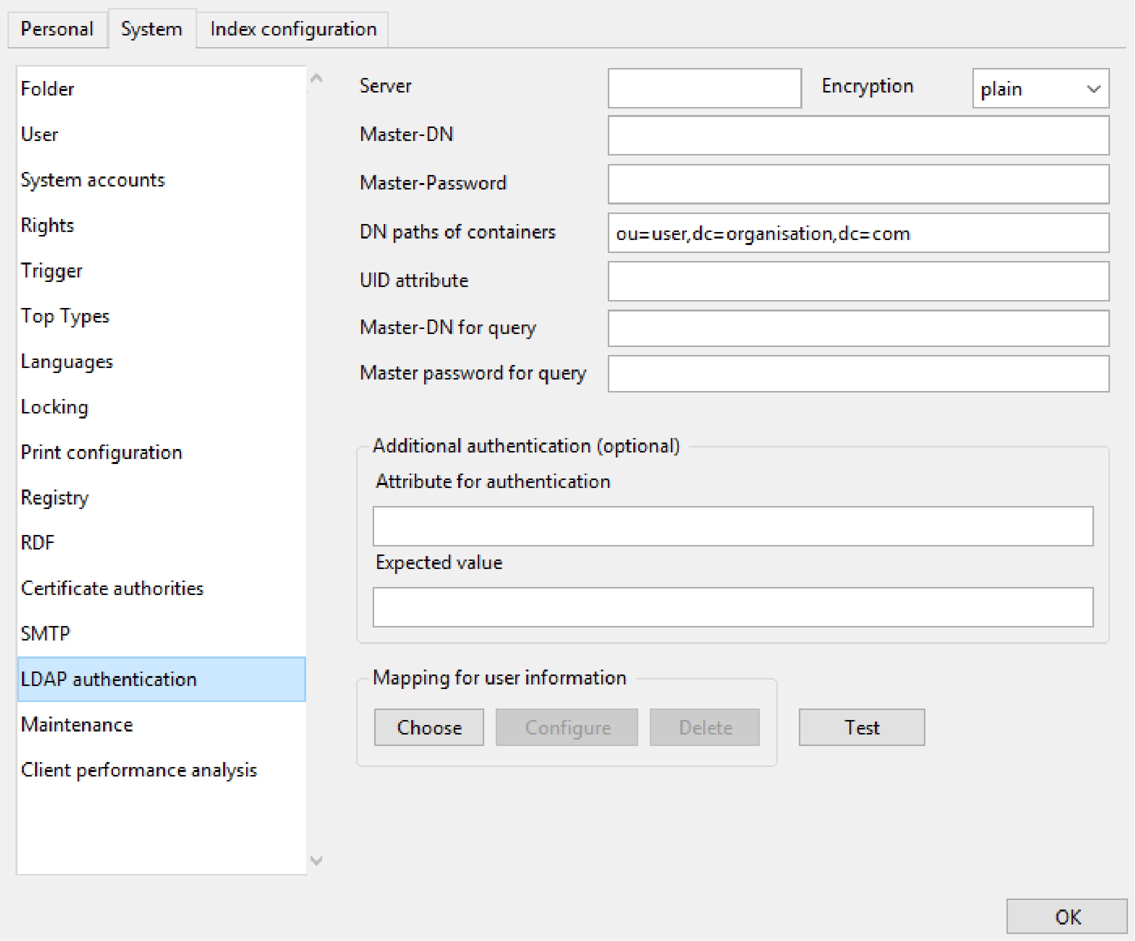
Port: 25

☑ Authentication

□ Use SMTPS (obsolete)

User

1.1.3.14. LDAP authentication



Server

Encryption: plain

Master-DN

Master-Password

DN paths of containers: ou=user,dc=organisation,dc=com

UID attribute

Master-DN for query

Master password for query

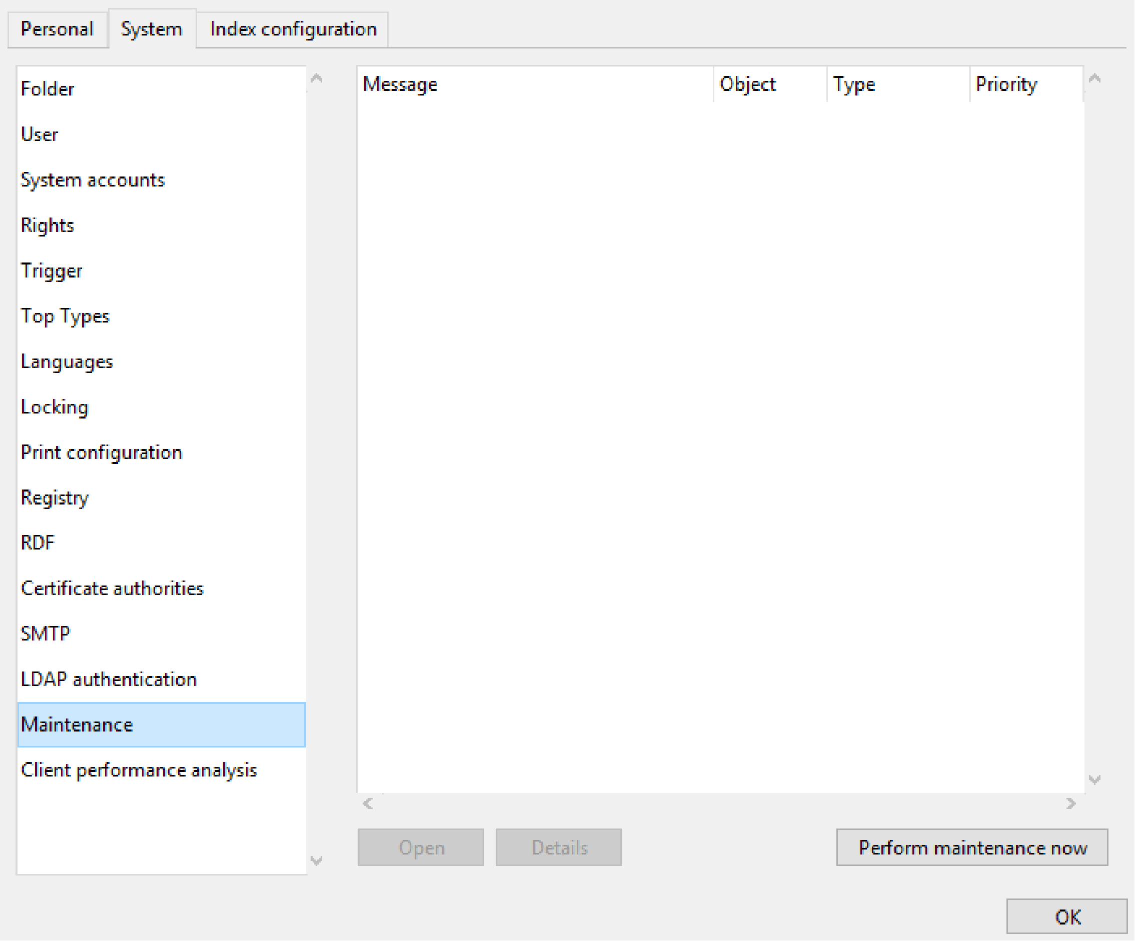
Additional authentication (optional)

Attribute for authentication

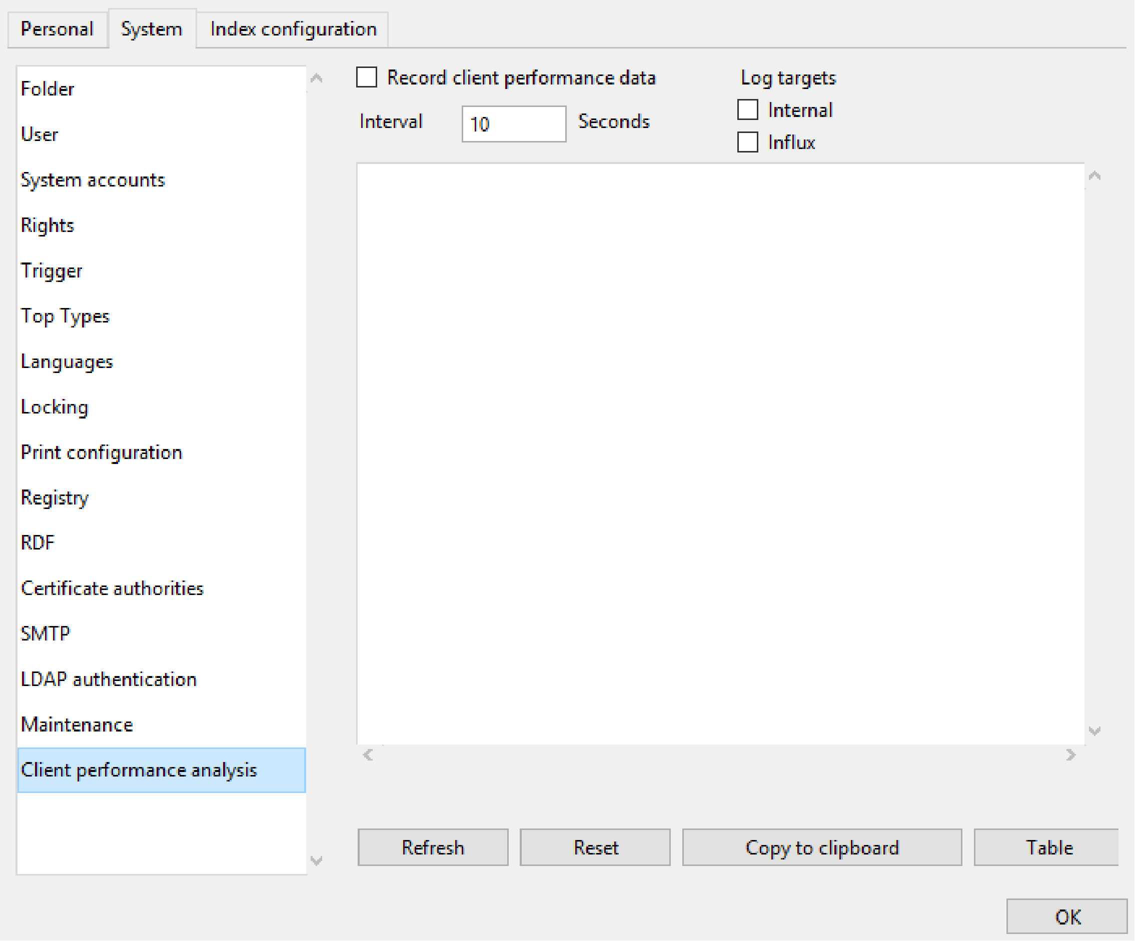
Expected value

Mapping for user information

1.1.3.15. Maintenance



1.1.3.16. Client performance analysis



□ Record client performance data

Interval 10 Seconds

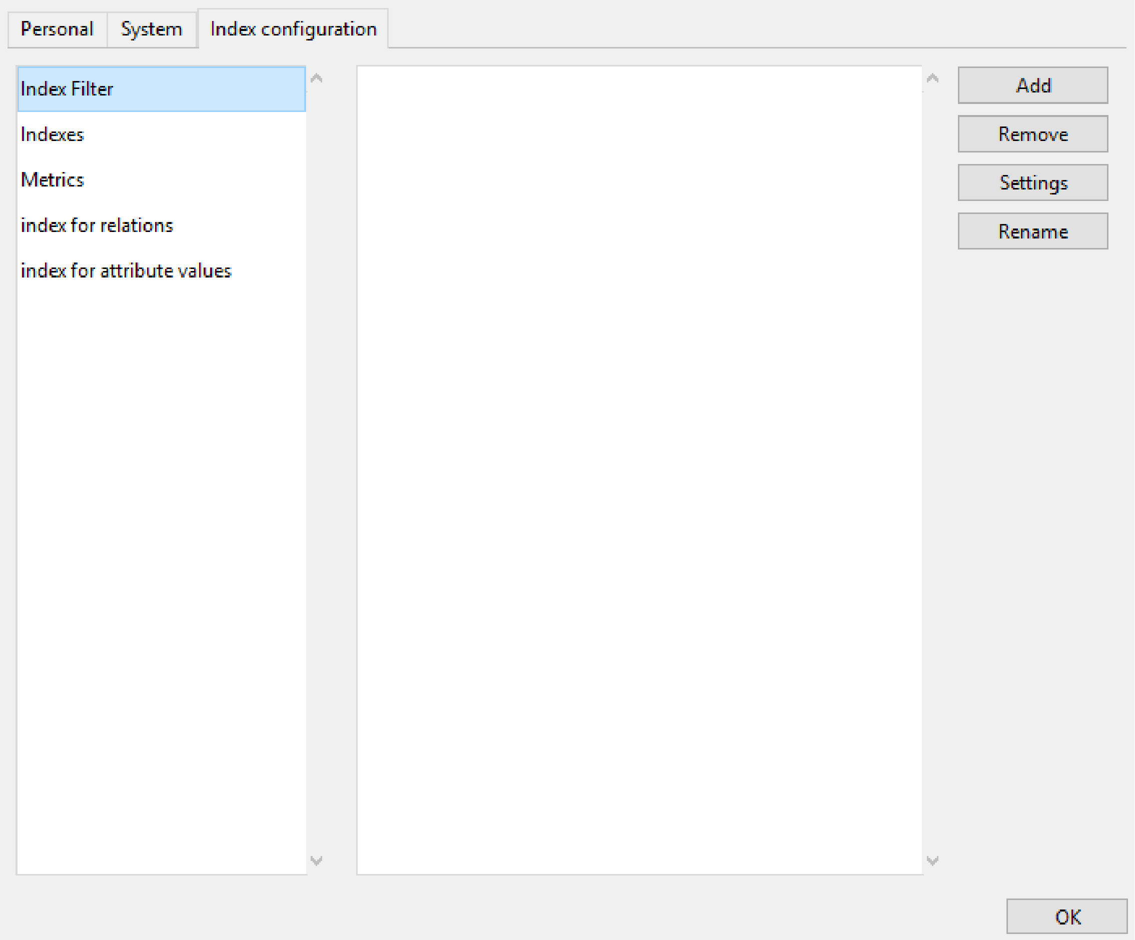
Log targets
□ Internal
□ Influx

1.1.4. Index configuration

The configuration of indexes for semantic elements within the Knowledge Builder can be specified here. Furthermore, already configured indexers can be applied for each kind of element via the detail editor of the respective element type.

Index filter

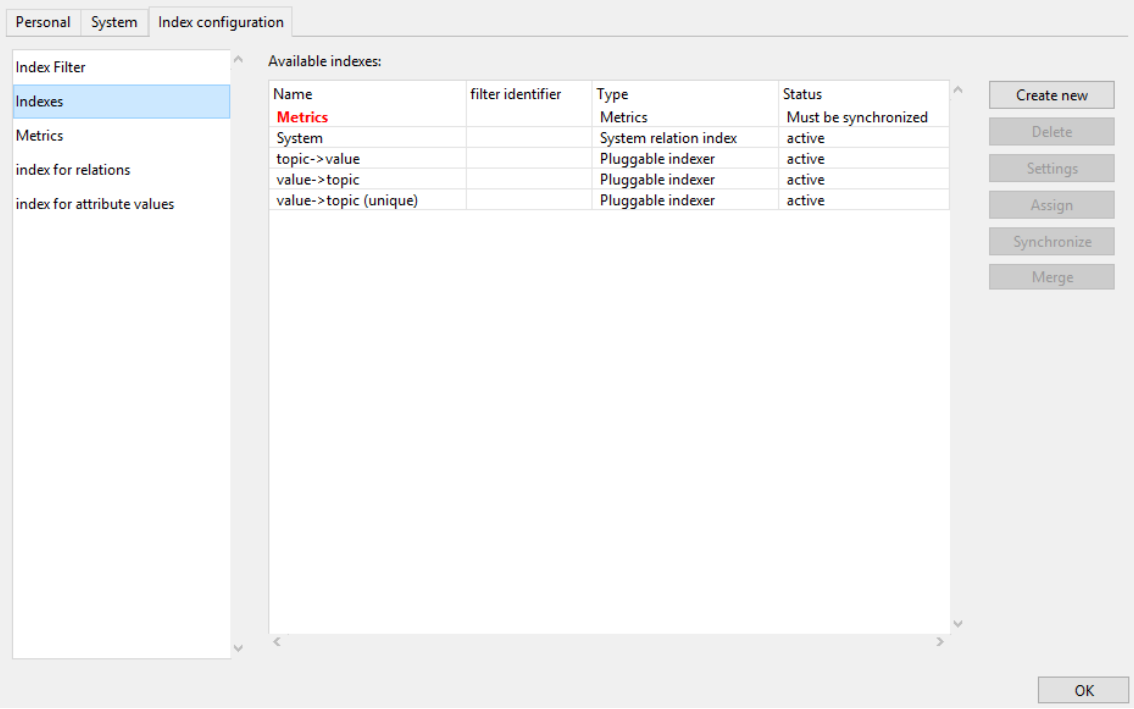
Index filters are needed for fulltext query indexes, comprising the settings for tokenizing, filtering and splitting query strings.



Index Filter
Indexes
Metrics
index for relations
index for attribute values

Indexes

- **Metrics:** The metrics comprise classified entries about the amount of objects, leading to performance improvement in queries. Dependent from the extent of changes within the Knowledge Graph (creating/removing semantic elements), the metrics have to be synchronized from time to time.
- **System:** The system index is reserved for system properties (relation and attributes for core functionalities); they are persistent and cannot be changed.
- **Further indexes:** In most cases, these are the pluggable indexes which can be built up according individual needs.



Name	filter identifier	Type	Status
Metrics		Metrics	Must be synchronized
System		System relation index	active
topic->value		Pluggable indexer	active
value->topic		Pluggable indexer	active
value->topic (unique)		Pluggable indexer	active

1.1.4.1. The index-report

The index-report analyses which indexings are necessary. Comparing this "demand" to the actual indexings will show missing and unnecessary ones. The index-report analyses structured queries, search-configurations, view-configurations and scripts. Since scripts are only analyzed regarding their references, it is impossible to determine the usage of the referenced elements.

NOTE It is possible for structured queries to need rarely used properties because of inheritance. These properties do not need to be indexed but the structured query itself will still show a warning.

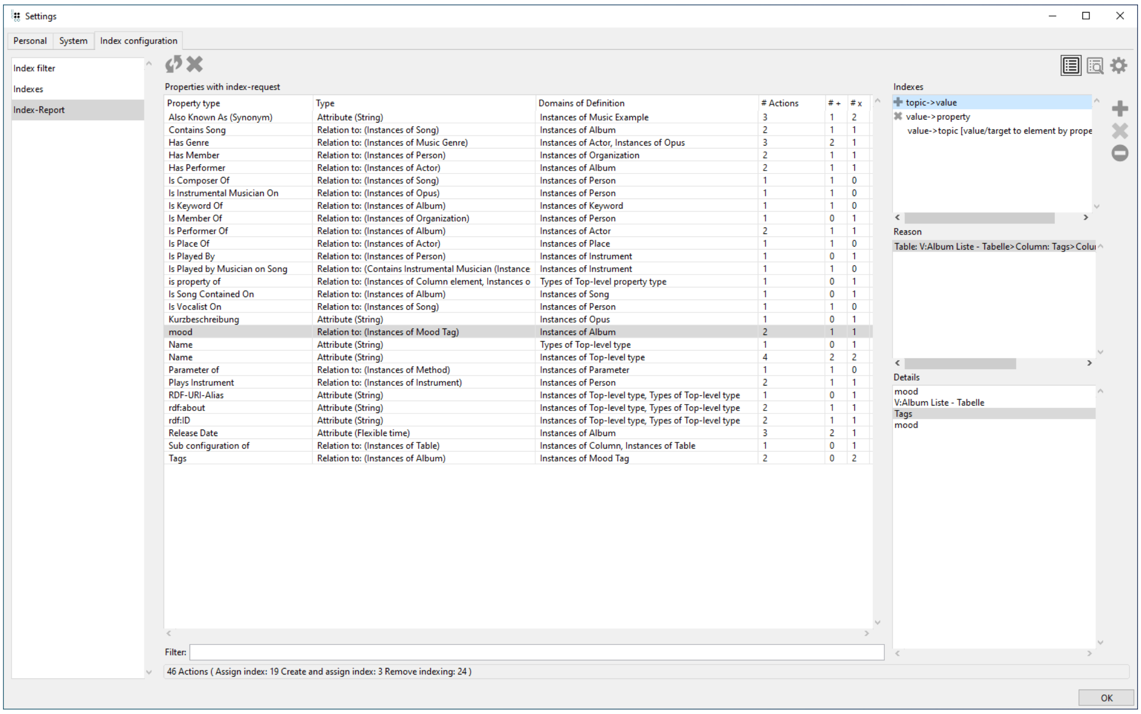
Where to find the index-report

Admins can find the index-report in the kb settings under "Index configuration". At first you can see the "simple view". Using the buttons in the top right, you can switch to the "detailed view" and the settings page.

When opening the index-report it immediately starts its analysis. It will show which area is currently being analysed and how it got to which property.

NOTE For larger graphs the analysis may take a while.

1.1.4.1.1. The simple view



Properties with index-request

Property type	Type	Domains of Definition	# Actions	# +	# x
Also Known As (Synonym)	Attribute (String)	Instances of Music Example	3	1	2
Contains Song	Relation to: (Instances of Song)	Instances of Album	2	1	1
Has Genre	Relation to: (Instances of Music Genre)	Instances of Actor, Instances of Opus	3	2	1
Has Member	Relation to: (Instances of Person)	Instances of Organization	2	1	1
Has Performer	Relation to: (Instances of Actor)	Instances of Album	2	1	1
Is Composer Of	Relation to: (Instances of Song)	Instances of Person	1	1	0
Is Instrumental Musician On	Relation to: (Instances of Opus)	Instances of Person	1	1	0
Is Keyword Of	Relation to: (Instances of Album)	Instances of Keyword	1	1	0
Is Member Of	Relation to: (Instances of Organization)	Instances of Person	1	0	1
Is Performer Of	Relation to: (Instances of Album)	Instances of Actor	2	1	1
Is Place Of	Relation to: (Instances of Actor)	Instances of Place	1	1	0
Is Played By	Relation to: (Instances of Person)	Instances of Instrument	1	0	1
Is Played by Musician on Song	Relation to: (Contains Instrumental Musician (Instance	Instances of Instrument	1	1	0
is property of	Relation to: (Instances of Column element, Instances o	Types of Top-level property type	1	0	1
Is Song Contained On	Relation to: (Instances of Album)	Instances of Song	1	0	1
Is Vocalist On	Relation to: (Instances of Song)	Instances of Person	1	1	0
Kurzbeschreibung	Attribute (String)	Instances of Opus	1	0	1
mood	Relation to: (Instances of Mood Tag)	Instances of Album	2	1	1
Name	Attribute (String)	Types of Top-level type	1	0	1
Name	Attribute (String)	Instances of Top-level type	4	2	2
Parameter of	Relation to: (Instances of Method)	Instances of Parameter	1	1	0
Plays Instrument	Relation to: (Instances of Instrument)	Instances of Person	2	1	1
RDF-URI-Alias	Attribute (String)	Instances of Top-level type, Types of Top-level type	1	0	1
rdf:about	Attribute (String)	Instances of Top-level type, Types of Top-level type	2	1	1
rdf:ID	Attribute (String)	Instances of Top-level type, Types of Top-level type	2	1	1
Release Date	Attribute (Flexible time)	Instances of Album	3	2	1
Sub configuration of	Relation to: (Instances of Table)	Instances of Column, Instances of Table	1	0	1
Tags	Relation to: (Instances of Album)	Instances of Mood Tag	2	0	2

Filter:

46 Actions (Assign index: 19 Create and assign index: 3 Remove indexing: 24)

Indexes

- ✚ topic->value
- ✖ value->property
- value->topic [value/target to element by prope

Reason

Table: V:Album Liste - Tabelle>Column: Tags>Colu

Details

mood
V:Album Liste - Tabelle
Tags
mood

All property types that were scanned by the analysis are listed here. The table shows the name (Property type), the kind of property type (Type), the domains (Domains of Definition) and the number of suggested actions (# Actions) consisting of adding indexings (# +) and removing indexings (# x). A property type can be opened by double clicking its row.

All indexings of a property type can be seen on the right. Suggested changes have a "" or "x" in front of the index name. "" means that this index should be assigned to this property type and "x" means that this indexing can be removed as it is unnecessary. Using the corresponding buttons on the right, selected indexings can be added or removed.

In the settings page you can change when an index should actually be added when clicking the "+".

Below this list, you can see where the analysis found missing indexings of a selected index that is not already assigned to the property type (meaning those with a "+" in front). Upon selecting one of these usages the details of how the analysis got there is shown below and every step can be opened by double clicking it.

The list of property types can be filtered using the input field below. The filter applies to the name of the property type, the type and the defined domains. You can also use +/x/# a number to filter for property types with that number of suggestions to add or remove indices or total suggestions.

If no action should be suggested for an index, property type or usage, you can create an exception by clicking the "No entry" (-) button.

Creating an exception

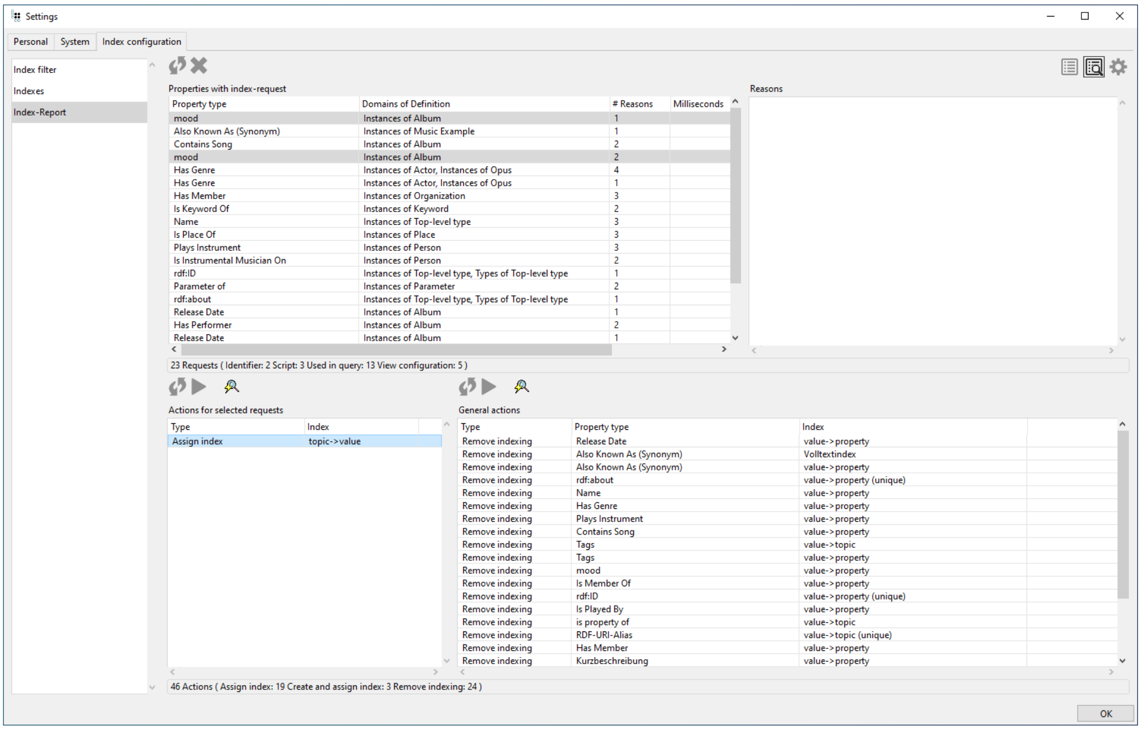
Exceptions define which Property types, Indexes or referencing elements should be ignored by the analysis.

When creating an exception you can choose wether the usage of the property type or index should be ignored. Additionally you can choose specific steps. A few examples for exceptions are: "No indexings for queries in script xyz", "Ignore property abc", "Ignore index ijk" or all at once "in script xyz ignore index ijk for property abc". Exceptions can also prevent suggestions are made for specific property types. Additionally suggestions for a whole index can be prevented or a property type and index can be removed from the list of suggestions. Obviously there are no steps for unused indexings.

Every exception needs a comment describing its intention.

Exceptions also prevent the check wether an indexing is necessary.

1.1.4.1.2. The detailed view



Properties with index-request

Property type	Domains of Definition	# Reasons	Milliseconds
mood	Instances of Album	1	
Also Known As (Synonym)	Instances of Music Example	1	
Contains Song	Instances of Album	2	
mood	Instances of Album	2	
Has Genre	Instances of Actor, Instances of Opus	4	
Has Genre	Instances of Actor, Instances of Opus	1	
Has Member	Instances of Organization	3	
Is Keyword Of	Instances of Keyword	2	
Name	Instances of Top-level type	3	
Is Place Of	Instances of Place	3	
Plays Instrument	Instances of Person	3	
Is Instrumental Musician On	Instances of Person	2	
rdf:ID	Instances of Top-level type, Types of Top-level type	1	
Parameter of	Instances of Parameter	2	
rdf:about	Instances of Top-level type, Types of Top-level type	1	
Release Date	Instances of Album	1	
Has Performer	Instances of Album	2	
Release Date	Instances of Album	1	

23 Requests (Identifier: 2 Script: 3 Used in query: 13 View configuration: 5)

Actions for selected requests

Type	Index
Assign index	topic->value

General actions

Type	Property type	Index
Remove indexing	Release Date	value->property
Remove indexing	Also Known As (Synonym)	Volltextindex
Remove indexing	Also Known As (Synonym)	value->property
Remove indexing	rdf:about	value->property (unique)
Remove indexing	Name	value->property
Remove indexing	Has Genre	value->property
Remove indexing	Plays Instrument	value->property
Remove indexing	Contains Song	value->property
Remove indexing	Tags	value->topic
Remove indexing	Tags	value->property
Remove indexing	mood	value->property
Remove indexing	Is Member Of	value->property
Remove indexing	rdf:ID	value->property (unique)
Remove indexing	Is Played By	value->property
Remove indexing	is property of	value->topic
Remove indexing	RDF-URI-Alias	value->topic (unique)
Remove indexing	Has Member	value->property
Remove indexing	Kurzbeschreibung	value->property

46 Actions (Assign index: 19 Create and assign index: 3 Remove indexing: 24)

This view answers why a specific index is suggested for a property. Instead of properties the main table contains the individual index requests. Additionally it has new columns: The number of causes because of which an index is suggested, milliseconds (as measured by the performance analysis) and a tag that can have the values "Used in query" (used in registered query), "Used in script" (used in registered script), "Used in Mapping" (used in registered table-mapping), "View configuration" (used in the view configuration) and Performance (used in the performance analysis). If a property was found via a starting point, that was configured in the options, the tag is "Query", "Script" or "Identifier" depending on the type of the starting point. "Identifier" is used for RDF-system-properties as well.

All references regarding a request are listed under "Reasons" on the right.

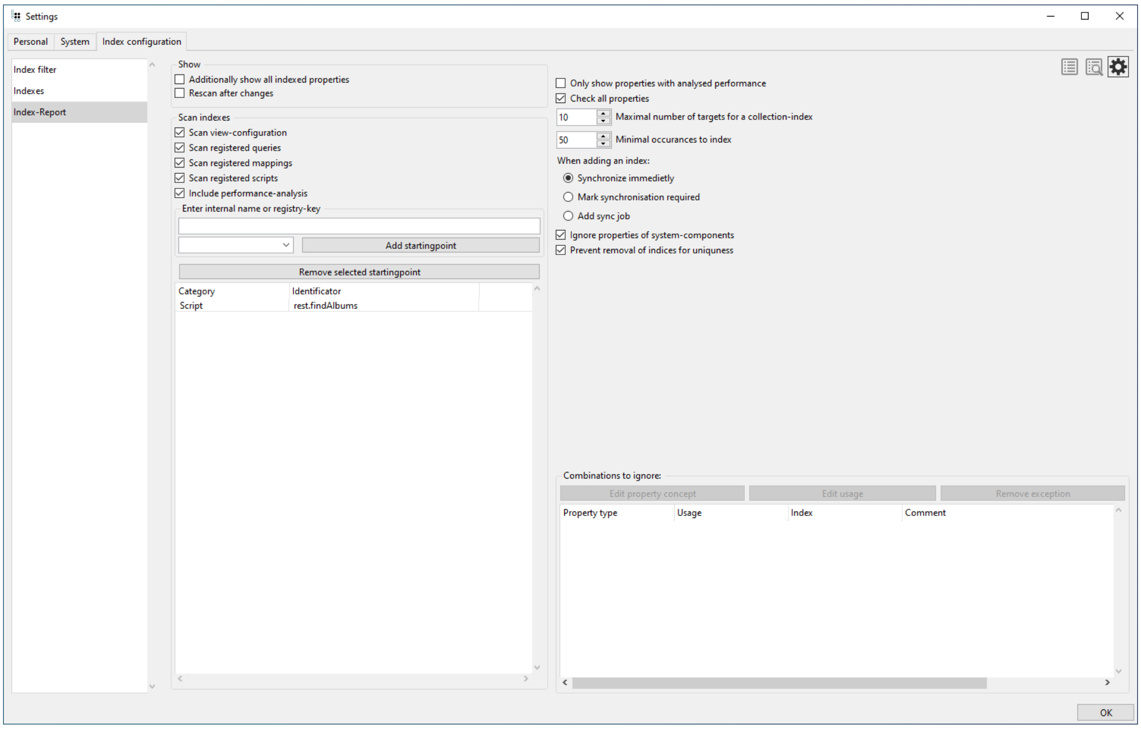
All suggested actions of a request are listed in the bottom left.

Unnecessary indexings are listed in the bottom right under "General actions".

In both of the tables of actions you can select one or more actions and execute them via the "Play"-button above the tables.

1.1.4.1.3. Settings

The settings can be used to configure what gets analyzed and what is suggested as a result.



Show
- ☐ Additionally show all indexed properties
- ☐ Rescan after changes

Scan indexes
- ☑ Scan view-configuration
- ☑ Scan registered queries
- ☑ Scan registered mappings
- ☑ Scan registered scripts
- ☑ Include performance-analysis

Enter internal name or registry-key

Category	Identificator
Script	rest.findAlbums

- ☐ Only show properties with analysed performance
- ☑ Check all properties
- 10 Maximal number of targets for a collection-index
- 50 Minimal occurances to index

When adding an index:
- ● Synchronize immedietly
- ○ Mark synchronisation required
- ○ Add sync job
- ☑ Ignore properties of system-components
- ☑ Prevent removal of indices for uniquness

Combinations to ignore:

Property type	Usage	Index	Comment

Manually configuring startingpoints

If there are scripts that are only used via the batch-tool but should still use indexes, they can be added as a starting point for the analysis: On the left side under "Enter internal name or registry-key" you can specify single or even multiple elements using wildcards (eg. "all*base*"). Additionally you need to specify if the starting point is a script, query or type. Once the definition is done the starting point can be created by pressing the "Add starting point" button. Obsolete starting points can be selected and then deleted by clicking the "Remove selected starting point" button.

Exceptions

The exceptions that were created in the simple view can be managed in the bottom right. They can be removed ("Remove exception") or the configured property ("Edit property concept") or usage ("Edit usage") can be shown.

Miscellaneous Settings

Setting	Description
Additionally show all indexed properties (Default: off)	Additionally shows properties without suggested actions.This setting will only take effect after the next scan.
Rescan after changes (Default: off)	Automatically start a new scan after executing a suggested action.
Scan view-configuration(Default: on)	Analyses the view configuration if present.
Scan registered queries(Default: on)	If turned off, registered queries are not scanned.
Scan registered mappings(Default: on)	If turned off, registered mappings are not scanned.
Scan registered scripts(Default: on)	If turned off, registered scripts are not scanned.
Include performance-analysis(Default: on)	If measurements of the performance-analysis are available (see client-analysis), they will be used as starting points.
Only show properties with analyzed performance (Default: off)	If no measurements are available for a property, ignore its usages.(Can only be activated if the performance-analysis is included.)
Check all properties (Default: on)	Also takes indexings into account, that were not scanned by the index-report, regarding their necessity.
Maximal number of targets for a collection-index(Default: 10)	Relations with few targets should have a specific index. Usually this affects relations to catalog-values. This integer defines the threshold for such indexes.
Minimal occurrences to index(Default: 50)	Properties with few occurrences don’t need an index. This integer defines the threshold for suggesting an index.
When adding an index:(Default: Synchronize immediately)	Defines what actually happens when adding a indexing.Options: * Synchronize immediately * Mark synchronisation required * Add sync job
Ignore properties of system-components (Default: on)	Indicates wether to scan properties of system-components as well. Only visible to developers.
Prevent removal of indices for uniqueness(Default: on)	Prevents the removal of indices for uniqueness. Only visible to developers.

1.1.5. Configuration file kb.ini

As for every i-views product, an *.ini file can be created for the Knowledge Builder. In the following, exemplary excerpts for the Knowledge Builder configuration file are listed:

```
; pre-fill corresponding fields in the login window
host=demo-server.empolis.com
user=peter
volume=demo
; configure logging if needed
logTargets=kb-log
; activate and configure file caching
; file caching speeds up data loading in subsequent sessions
cacheDir=cache
; maxCacheSize sets the size limit of the file cache (in MB)
; default is 50 (MB)
maxCacheSize=200
; the language parameter forces the kb to use the given language
; without this setting, the language is specified by the OS
; possible values are "eng" und "ger"
; fallback is "ger" if the OS language is unsupported
;language=eng

[kb-log]
type=file
file=kb.log
```

1.2. Access rights and triggers

This attribute handles the checking of access rights and triggers:

- **Access rights** regulate which operations on the Knowledge Graph may be executed be specific user groups. They are defined in the rights system in i-views. The rights system is located in the section *Technical > Rights* .
- **Triggers** are automatic operations that are triggered on a certain event and execute the corresponding actions. The Trigger section is located under *Technical > Trigger* .

The rights system and triggers are initially not activated in a newly created Knowledge Graph. These areas have to be activated before they can be used.

The procedure for creating rights and triggers is basically identical. Filters are required that check if certain conditions are met or not. If these conditions are met, the rights system grants or denies access, and a log entry is made or a script is executed for triggers. In the rights system, the arrangement of filters is referred to as rights tree while that for triggers is called trigger tree.

NOTE

For straight-on success in creating adequate query filter conditions based upon the operation, please check the table in chapter 1.2.5 "Operations". In principle, the operations filter work in an "AND" logic, leading to the requirement that all conditions of an operation filter and all conditions of the sub-components of the operation filter have to be fulfilled. Therefore, it is recommended to choose the most precise condition.

1.2.1. Checking of access right

We use rights to regulate user access to the data in the Knowledge Graph. The two basic objectives enabled by the rights system are:

- **Protection of confidential data:** Users or user groups may only see data that they are allowed to read. This ensures that secrecy and confidentiality restrictions are applied.
- **Work-specific overview:** Certain users only need a section of the data of a model for their work with the system. The rights system enables them to display only those elements that they need in order to complete their tasks.

The i-views rights system is very flexible. It can be configured precisely for different requirements of a project. By defining rules in a rights tree, consisting of individual filters and deciders, a graph-specific configuration of the rights system is created. There are many options for compiling these rules for the rights system, which generates even more differentiated rights. It is not possible to list all possible combinations of configurations; this requires consulting in individual cases.

How does the rights system work?

Access rights in the system are always checked when a user executes an operation on the data. The basic operations are:

- *Read* : An element is supposed to be displayed.
- *Modify* : An element is supposed to be changed.
- *Generate* : A new element is supposed to be generated.
- *Delete* : An element is supposed to be deleted.

If the access right is supposed to be changed in a certain access situation, the **Rights tree** is processed until a decision for or against access can be made in this situation. The Rights tree consists of conditions that are checked against the access situation. To check the conditions, **filters** are used which filter the elements of the Knowledge Graph and operations. **Deciders** are located at the end of a subtree of filters in the rights tree. These deciders either allow or prohibit access.

In relation to the access situation, aspects are selected which are used as the condition for allowing or prohibit access. In access situations, the following aspects are often used for the decision:

- The operation (generate, read, delete or modify)
- The element that is supposed to be accessed
- The current user

It is possible that only one aspect of the access situation is selected as a condition but it is also possible to query a combination of the aspects listed.

Example: "Person A [user] is not allowed to delete [operation] descriptions [element]”.

1.2.1.1. The activation of the rights system

In a newly created Knowledge Graph the rights system is deactivated by default. Before it can be used, it has to be activated in the settings of the Knowledge Builder.

Instructions for activation of the rights system

1. In the Knowledge Builder, call up the *Settings* menu and select the *System* tab. Select the *Rights* field there.
2. Place a checkmark in the *Rights system activated* field.
3. In the User type field, specify the object type whose objects are the users of the rights system. This is usually the “Person” object type. (Type must not be abstract.)

Before activation of the rights system, the folder is called *Rights (deactivated)* . Once the rights system has been activated, the folder is called *Rights* . When the rights system is deactivated, checks of the access rights are no longer performed. However, the rules defined in the rights tree are retained and used again after renewed activation of the rights system.

NOTE

If you access an element from the web front-end without special log-in, the person specified under *Standard web user* is used. It is common to create a fictitious person called “anonymous” or “guest” here.

To ensure the rights system also functions in the Knowledge Builder, the user accounts of the Knowledge Builder must be linked to an object from the Knowledge Graph. The user account can only be linked to objects of the type for which activation of the rights system was specified in the user type field.

The link is generally required for using the operation parameter *User* in query filters, or for using the access parameter *User* in structured queries when the rights system or the search is not executed in an application, but rather in the actual Knowledge Builder.

Instructions for linking Knowledge Builder users to objects of the person type

1. Open the *Settings* menu in the Knowledge Builder and select the *System* tab. Select the field *User* there.
2. Select the user who is to be linked. *Link* can be used to link the user to a person object that is not yet linked to a Knowledge Builder account. The *Unlink* function results in the Knowledge Builder account link to the person object is canceled.

NOTE The user currently logged in cannot be linked.

In general, users with administrator rights may perform all operations, regardless of which rights were defined in the rights system. The definition as administrator is also implemented in the *Settings* menu in the *User* field on the *System* tab.

1.2.1.2. The rights tree

Traversing the rights tree

The rights tree is comprised of rules that are defined in a tree. The branches of the tree, also referred to as a subtree, are comprised of the conditions that should be checked. The conditions are defined in the system as filters that are nested in each other. The system works through the tree from the top to bottom when the evaluation occurs. When a condition matches the access situation, then the check continues with the next filter in the subtree. This filter is, in turn, checked. This is implemented until the end of the subtree, when there is an access right or denial. If a condition does not match the access situation, then a switchover to the next subtree occurs. When the system encounters an access right or denial when working through the rights tree, the rights check ends with this result. The branches (subtrees) of the tree are therefore worked through successively, and the tree is “traversed“ until a decision can be made.

Filters and deciders are nested in each other in the form of folders, so that a tree construction is produced that is comprised of different subtrees. A folder can have several subfolders (several successor filters on one level), which produces branching in the rights tree. Folders that are defined on one level are worked through successively (from top to bottom).

Structure of the rights tree

When creating the rights tree, it is important to group the rules in a sensible way because once a decision as to whether access is allowed or denied has been made, no further rules are checked.

Hence, exceptions should be defined ahead of global rules.

The two main cases that you have to distinguish are:

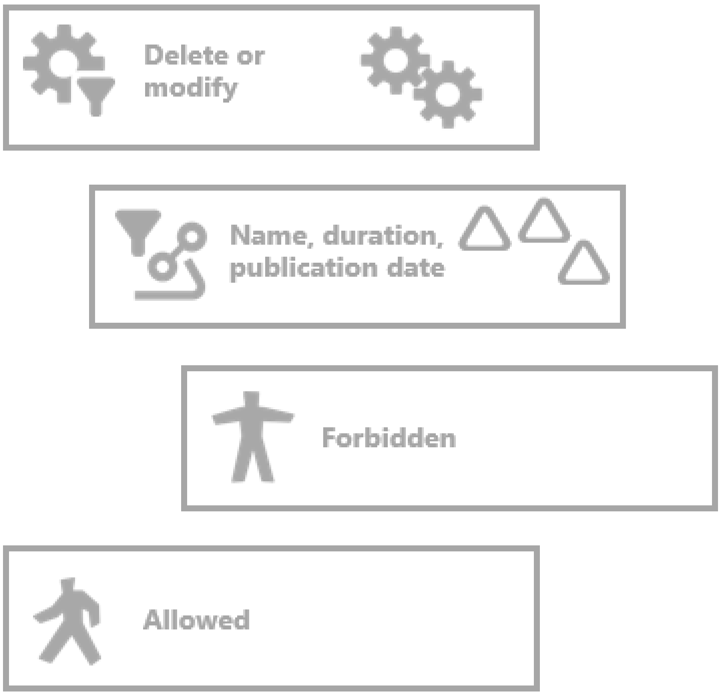
- **Negative configuration** : Everything is allowed at the lowest subtree; denials are defined above it.
- **Positive configuration** Everything is prohibited at the bottom, except for what is allowed above.

The order of the subtrees is therefore crucial when creating the rights tree. The order of the conditions in a subtree in contrast (whether we check the operation first and then the property or vice versa) can be chosen freely.

You don't necessarily have to define all filter types to define a subtree of a rights tree. A subtree consists of at least one filter and one decider. An exception is the last subtree which generally consists of a decider only, which allows all remaining operations (which have not been prohibited in the rights tree beforehand) or which prohibits all remaining operations (which have not been allowed in the rights tree beforehand).

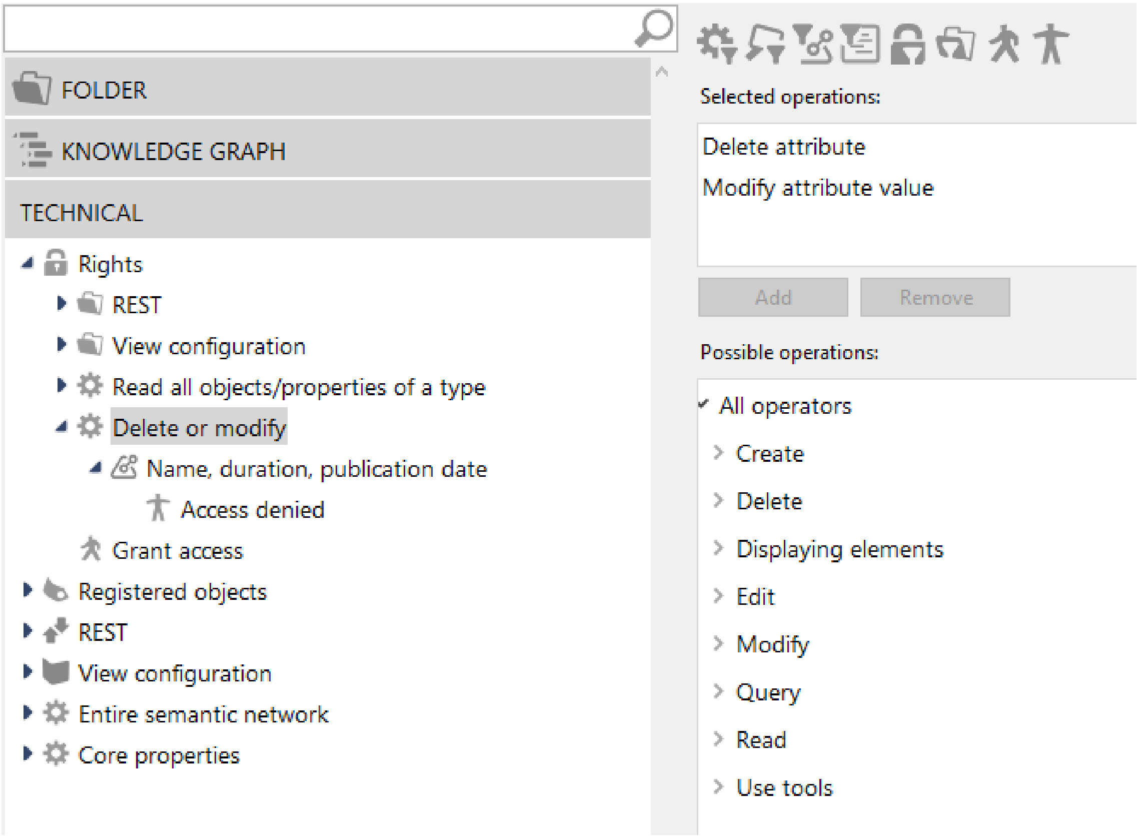
Example: rights tree

This basic example shows a rights tree consisting of a rights tree part and a default decider that allows everything:



In the rights branch, the deletion or modification of the attributes name, duration and publication date is prohibited. To do this, an operation filter is used that has the operations delete or modify as *the condition. Only these operations are let through by the operation filter. The next filter is property filter that filters on certain properties. In this case, the attributes Name, Duration and Publication date are filtered irrespective of the object or property on which these are stored. The last node of the rights branch is the decider "Forbidden", which prohibits all access operations that match the two preceding filters. If one of the two conditions does not apply to the access situation, the default decider "Allowed" is executed.*

This simple rights tree would look as follows in i-views:



FOLDER

KNOWLEDGE GRAPH

TECHNICAL

- Rights
  - REST
  - View configuration
  - Read all objects/properties of a type
  - Delete or modify
    - Name, duration, publication date
      - Access denied
  - Grant access
- Registered objects
- REST
- View configuration
- Entire semantic network
- Core properties

Selected operations:

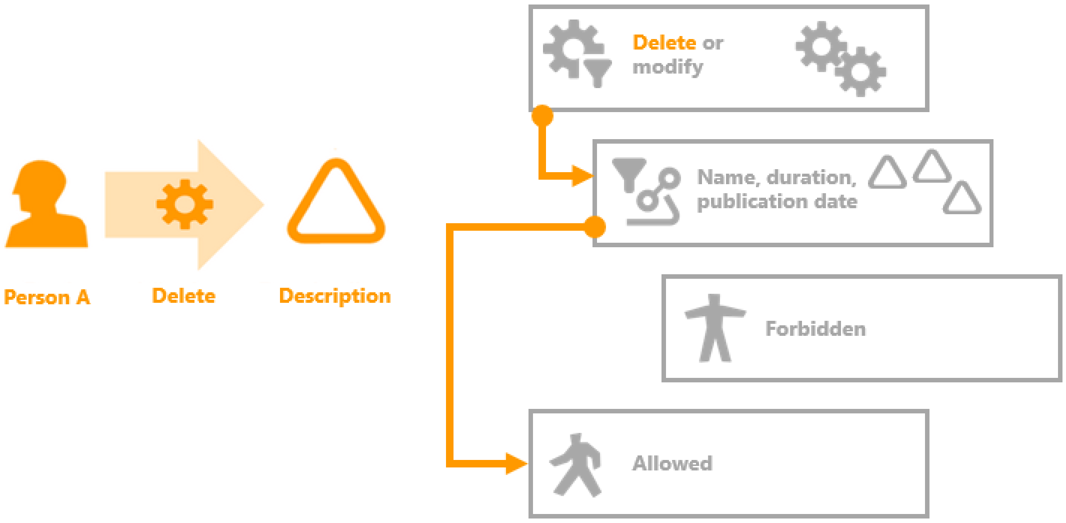
Delete attribute
Modify attribute value

Add | Remove

Possible operations:

- All operators
  - Create
  - Delete
  - Displaying elements
  - Edit
  - Modify
  - Query
  - Read
  - Use tools

Checking an operation using the rights tree example:



The left side shows the operation to be checked: Person A wants to delete the Description attribute. The rights tree is depicted on the right side. The check of the condition of the first filter returns a positive result because Person A wants to execute the operation Delete. In the rights tree, the next filter of the rights sub-tree is executed. This is the property filter of the attributes, Name, Duration and Publication date. The check of the filter returns a negative result because the Description is not one of the filtered properties. Processing of the subtree is terminated. The next subtree of the rights tree is processed next. This is already the default decider “Allow” which allows everything that is not explicitly prohibited in the rights tree.

1.2.1.3. Decision maker in the right tree

Deciders are always at the last position of a rights sub-tree. The combination with filters is used to determine access situations in which access is explicitly allowed or denied. If a decider is reached while traversing the rights tree, the check of rights is answered with this decision. The operation to be checked is then either allowed or rejected. The rights tree is then not checked any further.

Symbol	Access right	Description
🚶	*Access granted*	Access is granted in the access situation to be checked.
🧍	*Access denied*	Access is not granted in the access situation to be checked.

In general, there are two different deciders, a positive one called "Access granted" and a negative one called "Access denied". **

NOTE

Like all labels of the rights tree, "Access granted" and "Access denied" are standard labels which can be modified if needed.

Instructions for creating a decider

1. In the rights tree, choose the position at which you want to create the decider.
2. Use the buttons and to create new deciders as subfolders of the currently selected folder.
3. Assign a name to the folder.

1.2.1.4. Composing rights

To define rights, filters and deciders are combined in the rights tree. The Filters chapter explains the different filter types and how they can be used. The deciders *Grant access* or *Deny access* each represent the last node of the subtree of the decision tree. If the decider is reached, this decision terminates the traversing of the rights tree.

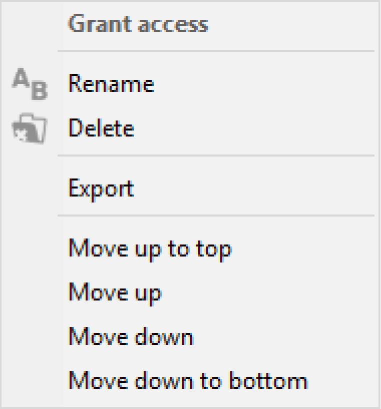
The following functions are available for defining rules in the rights system:

Symbol	Function	Description
	New operations filter	A new operation filter is generated.
	New query filter	A new query filter is generated.
	New property filter	A new property filter is generated.
	New script filter	A new script filter is generated.
	New lock filter	A new lock filter is generated.
	New organizing folder	A new organizing folder is generated.
	Grant access	A positive decider that grants access is generated.
	Deny access	A negative decider that denies access is generated.

Organizing folders can be used to structure rights in a meaningful way. They do not affect the traversing of the rights tree. Their only purpose is to group large numbers of rights into subtrees of the rights tree that have related content.

Changing the arrangement of folders in the rights tree

In order to sort the filters and deciders in the rights tree into the right order, right-clicking opens a context menu:



The filter or decider can be renamed, deleted and exported in this context menu, and its position in the rights tree can be changed. If two folders (filters or deciders) are on the same level, the *Upward* or *Downward* function can be used to shift the folder further to the front or the back in the rights tree. *To the top* and *To the bottom* shifts the folder to the first or last position of the level in the rights tree accordingly.

If folders are to be nested in each other, meaning the level in the decision tree be changed, this can be done using Drag&Drop.

Assembly of rights

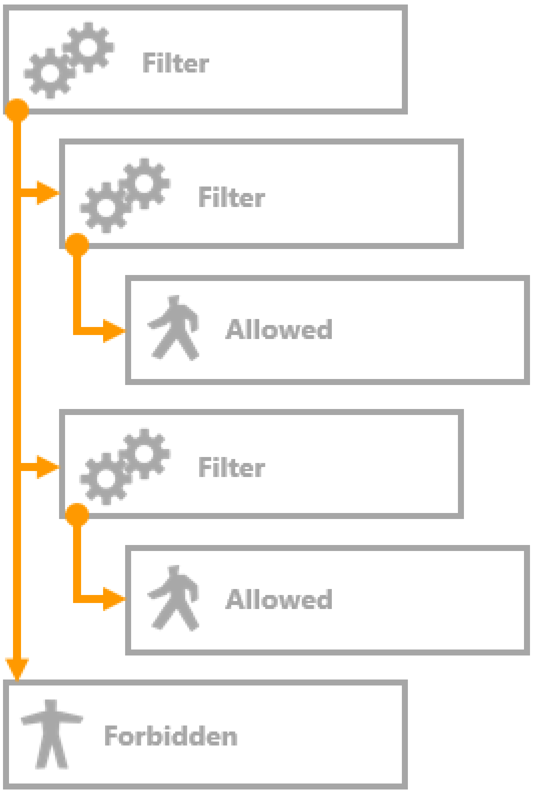
Assembling filters and deciders in the rights tree creates a large number of possible combinations for defining rights. By principle, there are 3 different procedures for defining rights:

- Definition of rights for every possible access situation
- Positive configuration
- Negative configuration

Because defining access rights for every possible access situation is a very complicated procedure, one of the two other means of configuration is generally used. They are explained in the following two sections.

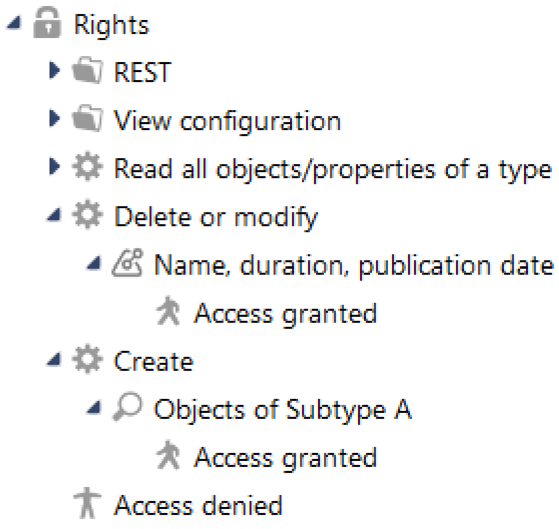
1.2.1.4.1. Positive configuration of rights

If rights are defined in the rights tree which only allow specific accesses and deny all other accesses about which nothing is specified, then this is referred to as a positive configuration of the rights tree. Rules are defined in each subtree of the rights tree, which allow specific operations. All operations to be checked traverse the rights tree: If the operation to be checked does not match the conditions of the subtrees, it is rejected at the end of the rights tree.



Example: Positive configuration

This example shows how a positively formulated rights tree might look like in the Knowledge Builder:

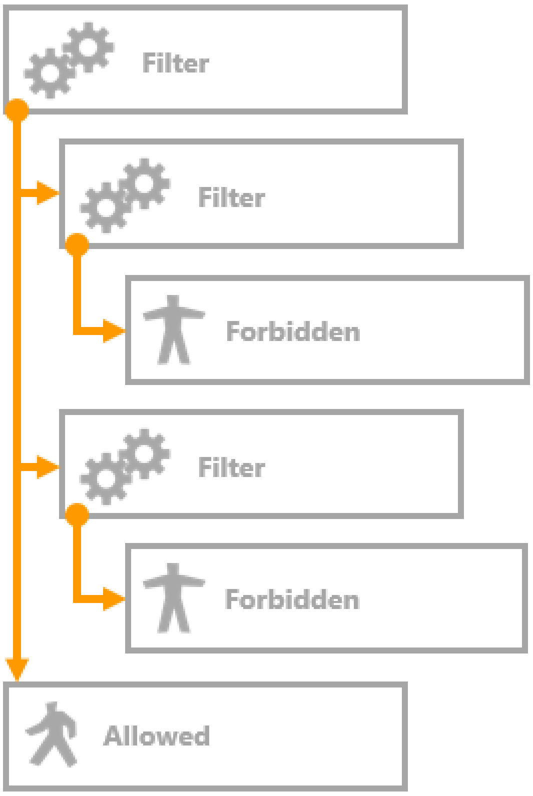


```
◢ Rights
    ▸ REST
    ▸ View configuration
    ▸ Read all objects/properties of a type
    ◢ Delete or modify
        ◢ Name, duration, publication date
            Access granted
    ◢ Create
        ◢ Objects of Subtype A
            Access granted
    Access denied
```

The first rights subtree defines read access to the attributes name, duration and publication date. The read operation is allowed for these attributes. The second rights subtree allows new objects of the type song to be created. All other operations are generally denied at the end of the rights tree.

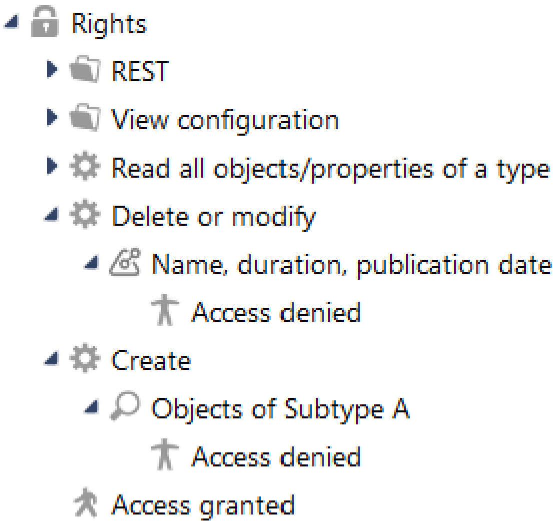
1.2.1.4.2. Negative configuration of rights

When rules are defined in a rights tree to reject specific operations and permit all the operations that, after a check, are identified as not matching those operations, this process is described as a negative configuration. Specific operations are prohibited in the subtrees of the rights tree. If one of the operations to be checked does not match the conditions of the subtrees, the operation is permitted at the end of the rights tree.



Example: Negative configuration

This example shows how a negatively formulated rights tree might look like in the Knowledge Builder:



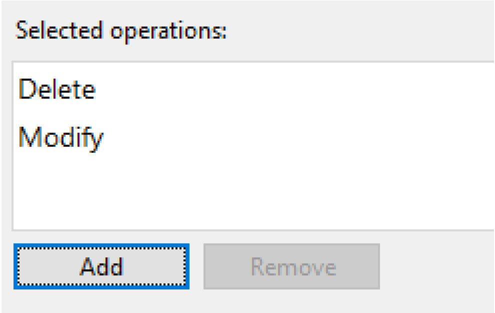
- Rights
  - REST
  - View configuration
  - Read all objects/properties of a type
  - Delete or modify
    - Name, duration, publication date
      - Access denied
  - Create
    - Objects of Subtype A
      - Access denied
  - Access granted

Unlike with a positive configuration, for example, the first rights subtree rejects the access rights for deleting and modifying the Name, Length and Publication date attributes. The second rights subtree prohibits deletion of the relation that links the songs to the album they are contained in. All other operations may be executed.

1.2.1.4.3. Example: Each user is allowed to change and delete items that he has created himself

What do you need to define this right in i-views? On the one hand, you need an operation filter since this is about changing and deleting elements. On the other hand, the connection between the user and the element on which the user wants to execute an operation must be defined, which is only possible by means of query filters.

Operation filter



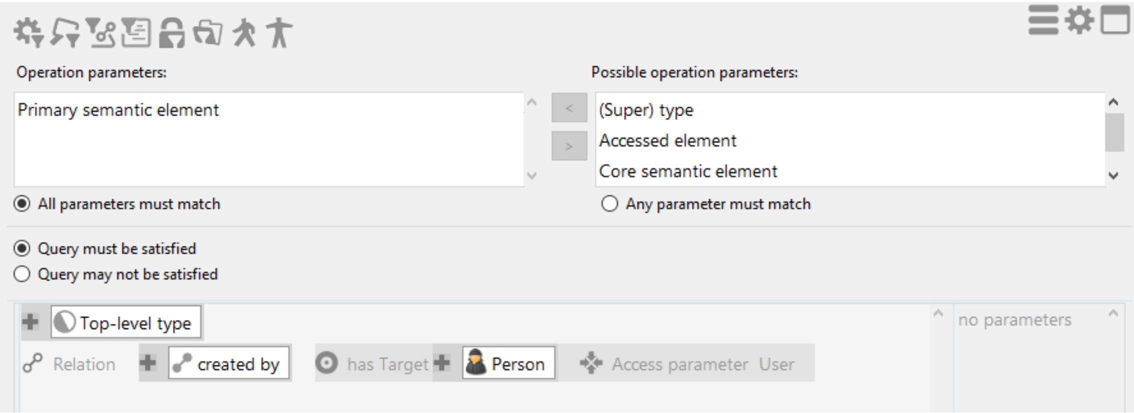
Selected operations:

Delete

Modify

In the operation filter, the operations Delete and Modify were selected.

Query filter



Operation parameters:

Primary semantic element

● All parameters must match

Possible operation parameters:

(Super) type
Accessed element
Core semantic element

○ Any parameter must match

● Query must be satisfied
○ Query may not be satisfied

Top-level type

Relation created by has Target Person Access parameter User

no parameters

In the query filter, “Relation created by” is selected with relation target “Person.” On the relation target Person, the access parameter User was specified. The settings All parameters must apply and Search condition must be met are selected. In this case, the operation parameter “Primary Knowledge Graph element” was selected.

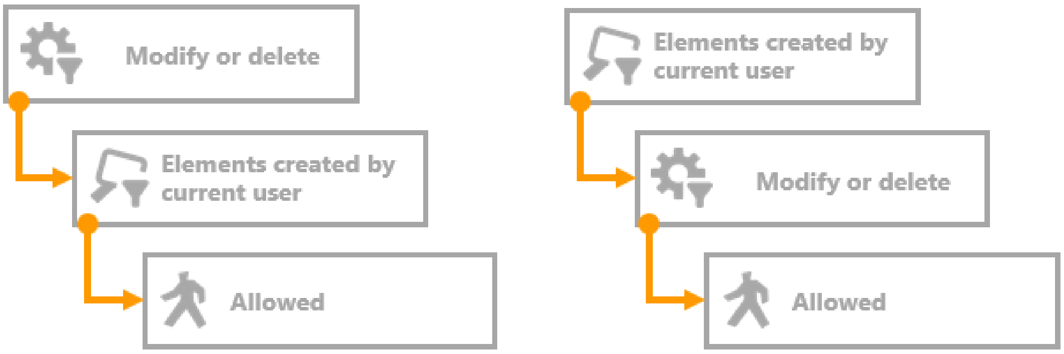
A question relating to the schema is: On which elements is the relation *was created by* defined? There are different options for implementing this relation in a Knowledge Graph:

1. Definition on objects and types: The relation is only used on objects and types.
2. Definition on all elements: The relation is used on all objects, types, extensions, attributes and relations.

In the first case, it makes sense to use the operation parameter “Primary Knowledge Graph element” or “Superordinate element.” If you define the right using the superordinate element, this does not apply only to the object itself but to all properties stored on the objects that were created by the user. If you use the operation parameter “Primary Knowledge Graph element,” the right also applies to all meta properties of the object. In the second case, the operation parameter “Accessed element” is used because only elements may be changed on which the relation *was created* occurs with the corresponding relation target, the user.

Compiling the right in the rights tree

There are two different variants for combining the filters. If there are no branches in the rights subtree, the order of the subtrees is not relevant.

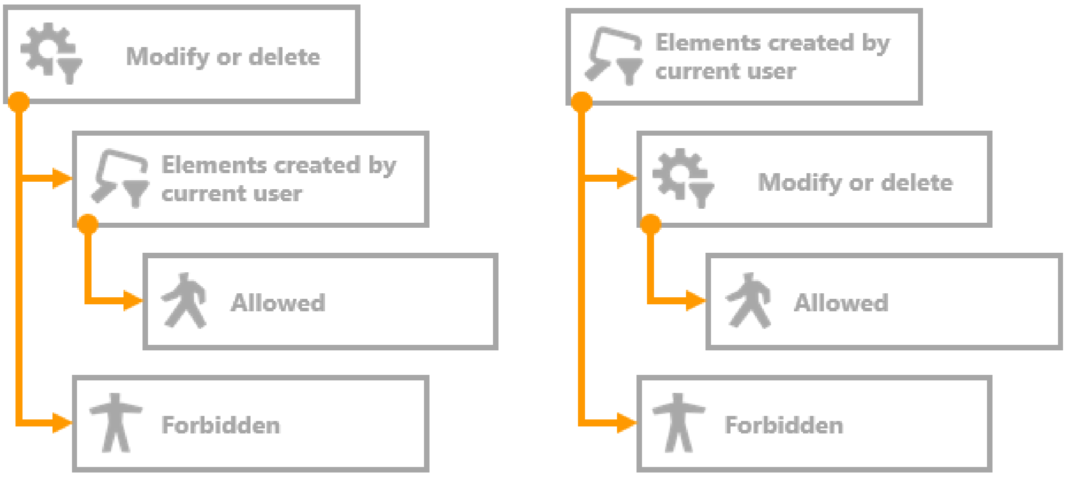

Modify or delete
Elements created by current user
Allowed
Elements created by current user
Modify or delete
Allowed

The graphic illustrates the two possible combinations: Version 1 (left) first operation filter, then query filter, version 2 (right) first query filter then operation filter, in both cases the decider “Allowed” then follows last.

Recommendation: It makes sense to have the operation filter in the first position, which makes it possible to create underneath it all other rights that filter on the same operation. This creates a more simple, traceable structure in the rights tree.

Advanced right: Elements that were not created by the user may not be changed or deleted

The right implies the denial for all elements that were not created by the user but we have not yet expressed this in the definition of rights. To do that, we have to take into account the Access denied decider during the creation of rights. If you look at both versions of rights and combine these with a negative decider, this results in the following variants. However, the two variants have different effects in the rights system.


Modify or delete
Elements created by current user
Allowed
Forbidden
Elements created by current user
Modify or delete
Allowed
Forbidden

If you add one decider Denied to each of the combination options presented above, the two versions are created: Version 1 (left) first operation filter, then query filter and decider “Allowed.” The operation filter is also followed by a decider Denied in a second subtree. Version 2 (right) first query filter then operation filter, and decider “Allowed.” In the version, the query filter is followed by a second subtree with the decider “Denied.”

Effects on the different versions on the rights system

Version 1 (left)

- Allows modification and deletion of elements created by users themselves.
- Prohibits modification and deletion of all other elements.
- No statement is made in relation to all other operations.

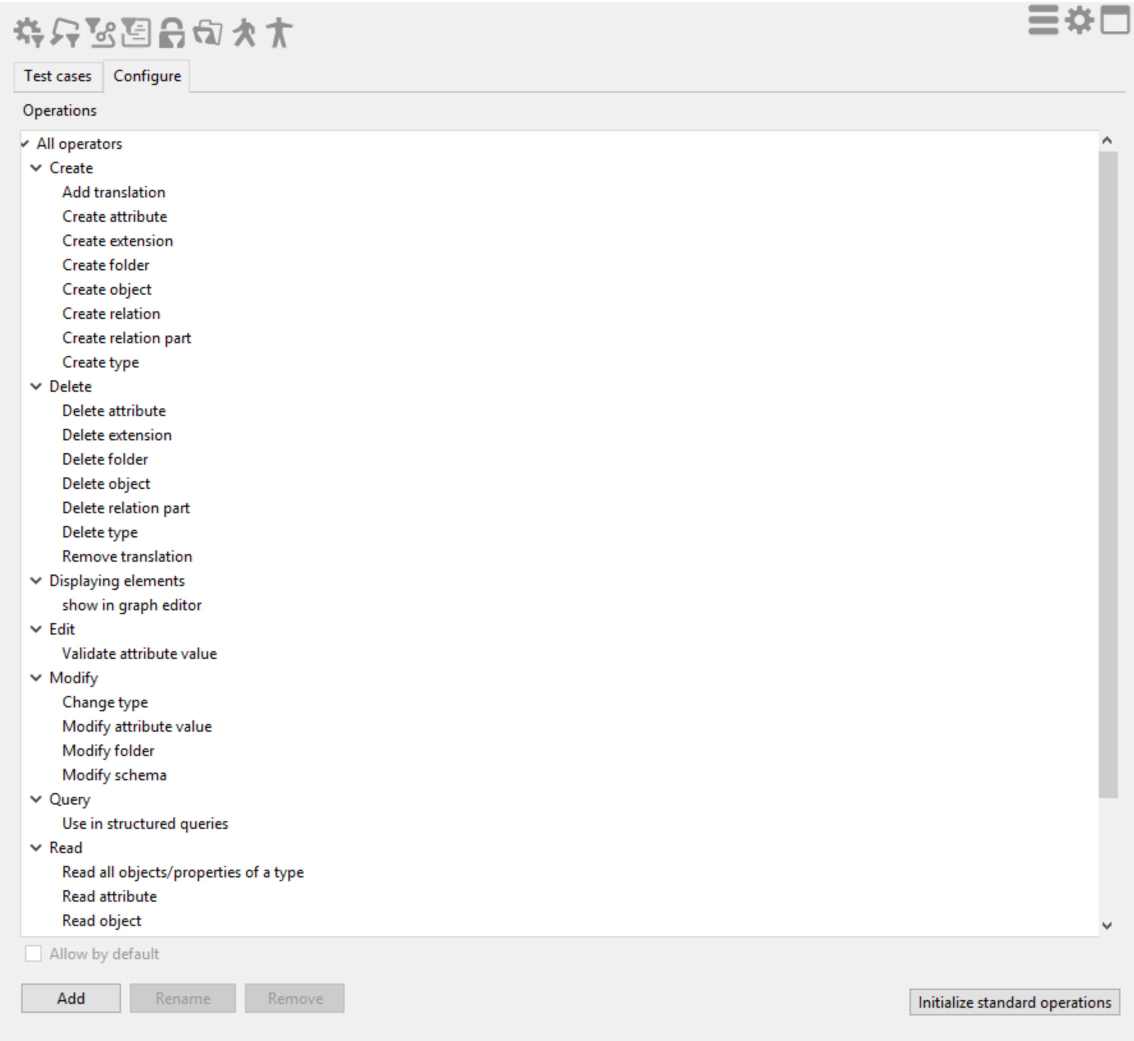
Version 2 (right)

- Allows modification and deletion of elements created by users themselves.
- Prohibits all other operations on elements created by users themselves (e.g. read).
- No statement is made in relation to all other elements.

The items show that version 2 does **not** express the requested access right. Only version 1 formulates the desired access right: All users can modify or delete elements they have created themselves and elements that were not created by the users may not be modified or deleted.

1.2.1.5. Configuration of own operations

When the *Rights* folder is selected in the *System* area, the *Saved test cases* and *Configure* tabs are available in the main window. A number of operations can be configured in the *Configure* tab.



The configuration of custom operations is generally only used when the Knowledge Builder is used with other applications. A number of operations are application-specific operations that should be checked together. This is a matter of checking a chain of operations, and not just an operation.

Instructions for the configuration of custom operations

1. In the Knowledge Builder, select the *Rights* folder in the *System* area.
2. Select the *Configuration* tab in the main window.
3. Click on *Add* to create a new operation.
4. In the windows that follow, enter an internal name and a description for the new operation.
5. The new operation is added as a *user-defined operation* .
6. User-defined operations can be deleted again using *Remove* .

1.2.2. Trigger

Triggers are automatic operations that are executed in i-views when a specific event occurs. They help support work flows by automating steps that always remain unchanged.

Examples for the use of triggers:

- Sending emails due to a specific change
- Editing of documents in a specific order by specific persons
- Marking jobs as open or done on the basis of a specific condition
- Creating objects and relations when a specific change is performed
- Calculating values in a previously defined way
- Automatically generating the name attribute for objects (e.g. combining properties of the object)

How do triggers work?

Triggers are closely related to the rights system. They use the same filter mechanisms in order to determine when a trigger is initiated. The filters are arranged in a tree, the trigger tree, which is structured like the rights tree. It consists of filters that are used to define conditions for the execution of a trigger action. If an access situation occurs because an operation is performed, and that access situation matches the defined conditions, the corresponding trigger action is executed.

Trigger actions are in most cases scripts that, depending on the elements of the access situation, use them to execute operations. This makes it possible to automate steps that remain unchanged or perform intelligent evaluations on the basis of specific constellations in the Knowledge Graph. Scripts can be used to execute any operations on elements that are dependent on complex evaluations, and thereby ensure situation and application-specific requirements for the Knowledge Graph. Most triggers are therefore usually project and Knowledge Graph specific; a consultation should be performed for each individual case.

1.2.2.1. Activate trigger

In order to be able to work with triggers, the trigger functionality must first be activated in the Knowledge Builder.

Instructions for the activation of triggers

1. Call up the *Settings* for the Knowledge Builder.
2. Select the *System* tab there, and the *Trigger* field.
3. Place a checkmark in the *Trigger activated* field.

A *Limit for recursive triggers* can be specified here. The default setting is “None”. Triggers that call themselves are referred to as recursive triggers. This occurs when even operations in the Knowledge Graph are implemented in the trigger script that, in turn, themselves match the filter definition of the trigger.

Before activation of the trigger functionality, the Trigger folder in the technical area of i-views is called *Trigger (deactivated)* . The folder is renamed *Triggers* by the activation.

NOTE

If the current user is used in triggers (e.g. in query filters or using the corresponding script function) and the user does not execute operations in an application, but rather in the actual Knowledge Builder, then the Knowledge Builder user account must be linked to a person object. The chapter Activation of the rights system explains how a link like this is created.

1.2.2.2. The trigger tree

The trigger tree has the same structure as the rights tree. It is comprised of branches (subtrees), which are comprised of filters and triggers. The filters are the conditions that must be checked for the trigger to be able to be executed at the end of the subtree when all conditions to be checked beforehand have been satisfied.

The trigger tree is queried for the data when each operation is performed – the tree is “traversed”. If a subtree applies to the access situation, then the trigger is executed. If the condition of a filter does not apply to the access situation, then a switchover to the next subtree occurs. Once the trigger action has been executed, traversal of the trigger tree continues, in contrast to the rights system, which stops being worked through when an decider is reached. In order to define that no other filters should be checked in the trigger tree after execution of an action, the *Trigger no other triggers* button is used:

Symbol	Function	Description
⊖	Trigger no other triggers	The traversal of the trigger tree is ended.

At the end of a subtree, no decider is available, in contrast to the rights system, but rather actions are available.

Symbol	Function	Description
✯	Define trigger	A new trigger action is created.

The available trigger actions are:

- *Enter log* : A log entry is written.
- *Execute script* > *JavaScript* : A script file in JavaScript is executed.
- *Execute script* > *KScript* : A script file in KScript is executed.

Structure the trigger tree

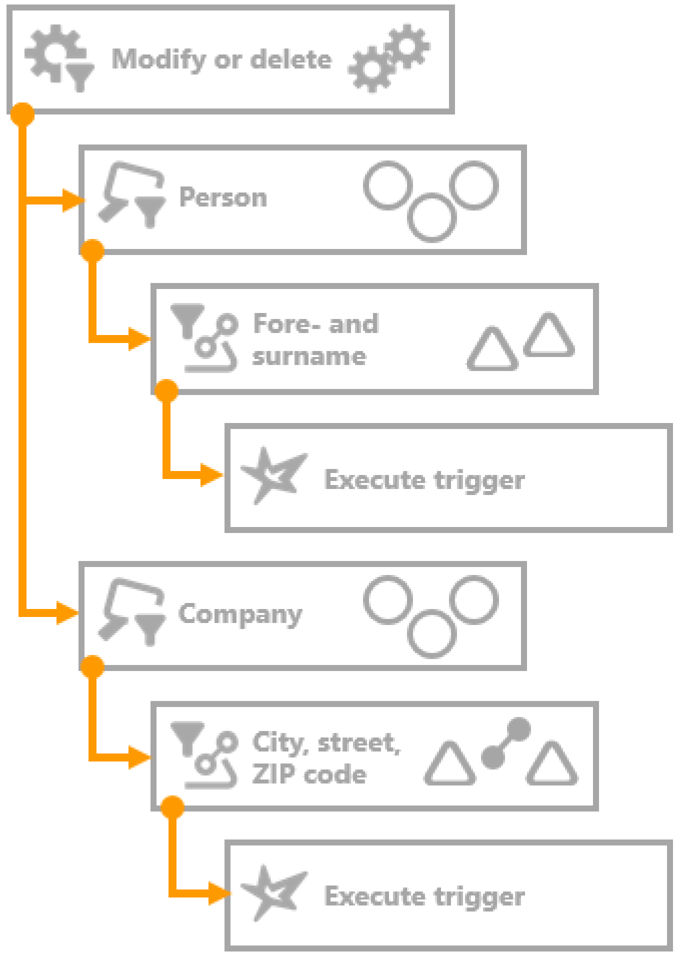
The order in which you define the triggers when designing the trigger tree usually has no effect on the performance of i-views. There are design recommendation for the rights tree, but these cannot be applied to the trigger tree, as the trigger tree is further traversed after a trigger action has been executed.

To provide a clearer structure for triggers, they can be collected in organizing folders. The organizing folders themselves do not affect the traversing of the trigger tree.

Symbol	Function	Description
📁	Organizing folder	Organizing folder for grouping subtrees

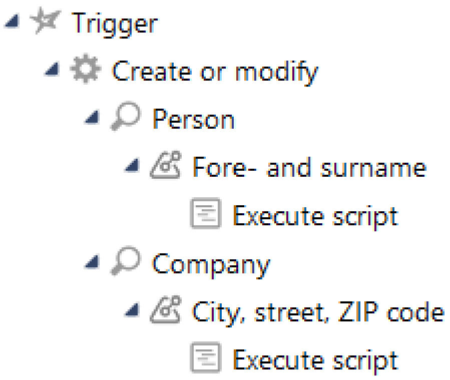
Example: trigger tree

This example shows a trigger tree that combines the names of persons and concerts automatically from properties of the objects:



This simple trigger tree begins with an operation filter and splits into two separate subtrees after the operation filter. If either the modify or the create operation is executed, it is let through by the operation filter. The persons subtree filters operations that are performed on attributes and relations of person type objects. If the operation affects either the first name attribute or the last name attribute, it is let through by the property filter. The corresponding script that compiles the name attribute of a person from their first and last name is executed. The second subtree also applies to the modify or create operation filter. However, it filters attributes and relations that are *saved in company type objects. The property filter only lets operations through if they are performed on the attributes or relations of the city, the street or the ZIP code. If these conditions apply, the corresponding script that compiles the complete address string of the company is executed.*

This is what this trigger tree would look like in i-views:



```
Trigger
  Create or modify
    Person
      Fore- and surname
        Execute script
    Company
      City, street, ZIP code
        Execute script
```

1.2.2.3. Create trigger

As described in the Trigger tree section, triggers consist of filters and trigger actions. These are combined in such a way that a specific trigger action is executed only when it is required.

The following functions are available in the trigger area:

Symbol	Function	Description
⚙	*New operation filter*	A new operation filter is generated.
🔍	*New query filter*	A new query filter is generated.
🔗	*New property filter*	A new property filter is generated.
✖	*New delete filter*	A new delete filter is generated.
📁	*New organizing folder*	A new organizing folder is generated.
★	*New trigger*	A new trigger action is created.
⛔	*Trigger no other triggers*	A new “Stop” folder is created. It ends the traversing of the trigger tree.

When creating triggers, you should consider two fundamental properties of the trigger mechanism:

- Execution of a trigger script can cause further triggers to be triggered. This occurs if operations in the Knowledge Graph are executed in the trigger script itself.
- After a trigger action has been executed, traversal of the trigger tree continues. All trigger actions of the subtrees that apply to the access situation are executed.

1.2.2.4. Trigger actions

Trigger actions are used to perform intelligent operations in the Knowledge Graph, which, for example, automate or support work flows. However, they are only executed when the access situation and the links in the Knowledge Graph assume a specific state defined by the filter.

Instructions for the creation of trigger actions

1. Select the position in the trigger tree at which the trigger action is to be created.
2. Used the button to create a new trigger.
3. Select the action type from the list: Enter the log or execute the script (if you wish to execute a script, select the script language).
4. The trigger is created as a subfolder of the currently selected folder.

Logging actions

In principle, there are three different possibilities for logging changes that have been initiated by the trigger system:

- **Log trigger:** Special logging element that is used additionally to the respective trigger element in order to log the trigger action itself. Advantage: The log trigger can be added quickly to any script trigger, but an initialization file (*.ini) needs to be configured before. The log trigger is described in the sub chapter "Log trigger".
- **Script trigger with output in forms of "$k.log()":** Within any trigger script, entries for logging can be added by means of the $k.log method. Advantage: The log output can be defined in a highly customized manner, restricted by the scope of the JavaScript API only. The log information is output within the "Script messages" dialog and/or in the respective logfile as configured by the initialization file. For more information, see the JavaScript API documentation.
- **changeLog trigger:** A predefined registry key for a string attribute in combination with a JavaScript method can be used for logging. Advantage: Log entries will be created in forms of a "changeLog" attribute directly attached to the respective semantic element on which the changes take effect, depending on the definition range of the changeLog attribute type. The changeLog trigger is described in the last sub chapter.

1.2.2.4.1. Script trigger

An operation parameter must be output for the script to be executed. In contrast to query filters, only one operation parameter can be specified. Execution of the script starts on the element contained in the operation parameter.

Time/type of execution

- Before the change: The trigger is executed before the operation is performed.
- After the change: The trigger is executed immediately after the operation has been performed.

- End of transaction: The trigger is executed only at the end of the shared transaction.
- Job-Client: The Job-Client determines the time of execution.

NOTE

Triggers that are executed for delete operations should preferably use *before the change* as their time, as the element to be deleted will no longer be available otherwise. For other operations, a more suitable time is *after the change* or *end of transaction* , as it is then possible, for example, to add a property to the newly created element or automatically generate the name from various properties of an object if one or more properties were changed.

The import chooses the order in which the properties will be imported in i-views. *Therefore a trigger that is initiated during* the import should not rely on the properties being available in full.

Execute once only per operation parameter

If this setting is selected, the element selected in operation parameter is executed no more than once per transaction. If this setting is chosen, the time of execution should be set to *end of transaction* so that the final state of the element is used in the script.

Example: For persons, the name of the object is meant to consist of the first name and last name. With this setting, the trigger is executed only once if the first and last names are changed at the same time.

Execution does not initiate trigger

This setting specifies that the operations executed within a trigger cannot initiate any further triggers. This setting can be used to avoid endless loops.

Continue to execute script in case of script errors

If this setting is active, an attempt is made to restart after an execution error and continue with the execution of the script. This setting is predominantly useful for scripts that are supposed to execute instructions that are independent of each other, and not for scripts that build on previous steps of the script.

Abort transaction if trigger fails

This setting defines the termination behavior in the event of script errors. If an error occurs while the script is being executed and this setting is active, all actions of the transaction are reversed. If this setting is not active, all actions are executed apart from the ones affected by the error. The original action that led to the trigger being called is nevertheless written to the Knowledge Graph.

Execution during data refactoring

The term data refactoring describes operations for restructuring the Knowledge Graph, e.g. **Change type** or **Choose new relation target** .

Caution: Data refactoring operations can, in some circumstances, initiate unwanted trigger actions and, in some cases, even generate errors during execution of the script. For this reason, it is possible to set for each trigger whether it is to be executed during data refactoring.

Example for data refactoring: Reengineering to single-sided relation. Changing a relation type from a double-sided relation into a single-sided relation causes a re-saving of relation targets. Although this is not a factual change, this can trigger the execution of a trigger script that originally was intended to react on relation target changes only.

Following processes are considered as data refactoring:

In the Knowledge Builder:

- "Choose new semantic element for property" (for attribute)
- "Choose new relation target" (at relation)
- *Copying*
- "Change subtypes into objects" (context menu "Reengineer")
- "Merge" (of nodes in Graph Editor)
- *Relocating relations*
- Change relation source/target in Graph Editor by means of Drag&Drop
- Converting relations from/to one-way relations

In general:

- Changing data storage of file attributes
- Changing relation source/target by RDF import

Deprecated:

- Behavior function "adsorbRelationTarget" (not needed anymore)
- Change relation source/target in edit view in web ui (pre version 5.4)

The function body for script triggers is created automatically.

The script has three parameters:

parameter	$k.SemanticElement / $k.Folder	The selected parameter
access	object	Object with data of the change (new attribute value etc.)
user	$k.User	User who triggered the change

The following example sets the attributes with the internal name “changedOn" / “changedBy.” “Primary semantic core object" should be selected as the parameter here.

```
/**
 * Perform the trigger
 * @param parameter The chosen parameter, usually a semantic element
 * @param {object} access Object that contains all parameters of the
access
 * @param {$k.User} user User that triggered the access
**/

function trigger(parameter, access, user)
{
    parameter.setAttributeValue("modifiedAt", new Date());
    var userName = $k.user().name();
    if (userName)
        parameter.setAttributeValue("modifiedBy", userName);
    else
        parameter.attributes("modifiedBy").forEach(function(old) { old
.remove });
}
```

The parameter "access" may contain the following properties (varies in each operation):

Property	Description
accessedObject	Accessed element
core	Core object
detail	Detail
inversePrimaryCoreTopic	Primary relation target
inverseRelation	Inverse relation
inverseTopic	Relation target
operationSymbol	“read," "deleteRelation" etc.
primaryCoreTopic	Primary semantic core object
primaryProperty	Primary property
primaryTopic	Primary semantic element
property	Property
topic	Superordinate element
user	User (identical to “user” parameter of the function)

1.2.2.4.2. Log trigger

If the user would like to monitor or document the trigger functionality for when which trigger was triggered and which operators were executed in the Knowledge Graph, log triggers are suitable. The log is written to the respective log file (bridge.log, batchtool.log etc.) in the application environment that the operation that triggered the trigger is performed in.

Lines of the log entry	Current state of the Knowledge Graph
pre	before triggering
post	after triggering
end	at the end of the transaction
commit	when the transaction has been processed successfully

Log entries are used to retrace whether a trigger was executed in a specific access situation that actually occurred, and what it did. In contrast to this, a test can be performed in the test environment to determine whether a trigger would be triggered or not in a specific access situation, without the specific access situation being performed.

The operability of the log trigger feature actually depends on how the logging is configured by the respective *.ini file (kb.ini, mediator.ini, jobclient.ini).

Example: For logging the trigger actions when using a local Knowledge Graph wihout mediator, a "kb.ini" file is needed with a minimum set of configuration parameters:

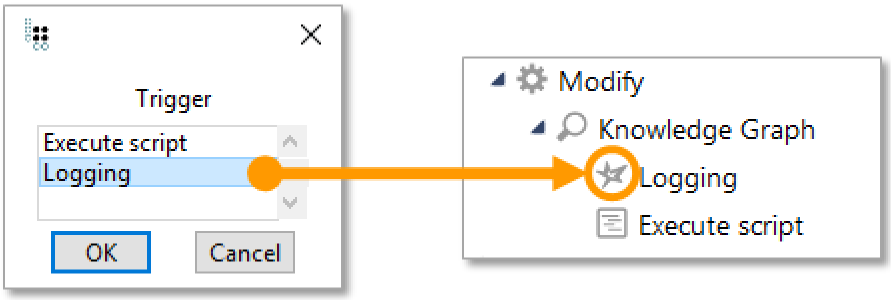
```
[Default]
logTargets = kblog

[kblog]
type = file
format = plain
file = kb.log
```

This initialization file creates a logfile called "kb.log" in the Knowledge Builder folder.

For more information about different configuration files, see the respective chapter about "i-views services".

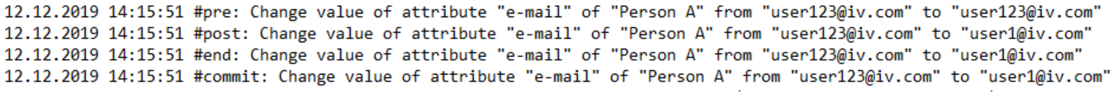
Instructions for the creation of log triggers

1. Select the trigger script that is to be logged in the trigger tree.
2. Using the ★ button to create a trigger of type *Logging* in the trigger tree directly above the



script trigger.

Example:



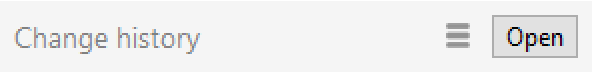
```
12.12.2019 14:15:51 #pre: Change value of attribute "e-mail" of "Person A" from "user123@iv.com" to "user123@iv.com"
12.12.2019 14:15:51 #post: Change value of attribute "e-mail" of "Person A" from "user123@iv.com" to "user1@iv.com"
12.12.2019 14:15:51 #end: Change value of attribute "e-mail" of "Person A" from "user123@iv.com" to "user1@iv.com"
12.12.2019 14:15:51 #commit: Change value of attribute "e-mail" of "Person A" from "user123@iv.com" to "user1@iv.com"
```

Log entry that documents the change of the attribute e-mail using a trigger.

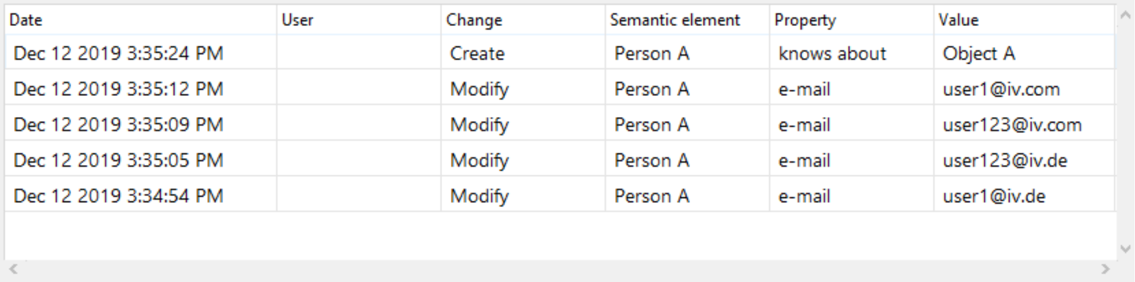
1.2.2.4.3. ChangeLog Trigger

If you want to monitor the activities that users perform on objects, you should set up a changeLog trigger, also referred to as a change history.

For this purpose, you must first define a string attribute with the internal name “changeLog.” This changeLog attribute must be defined for all elements for which it is to document user activities.



Click “Open” to open the table showing who made the change, when they did so, what the change is, to which semantic element it applies, and which value was used.



Date	User	Change	Semantic element	Property	Value
Dec 12 2019 3:35:24 PM		Create	Person A	knows about	Object A
Dec 12 2019 3:35:12 PM		Modify	Person A	e-mail	user1@iv.com
Dec 12 2019 3:35:09 PM		Modify	Person A	e-mail	user123@iv.com
Dec 12 2019 3:35:05 PM		Modify	Person A	e-mail	user123@iv.de
Dec 12 2019 3:34:54 PM		Modify	Person A	e-mail	user1@iv.de

NOTE

Since operation filters like "create relation", "create relation half" or "delete relation half" only apply to the relation origin (the semantic element itself), logging of changing relation targets cannot be triggered. For this purpose, the trigger script can be used if specified accordingly.

Modifications in attribute values will be logged only when they are created (simultaneously when the attribute itself is created), but not when the attribute value is deleted.

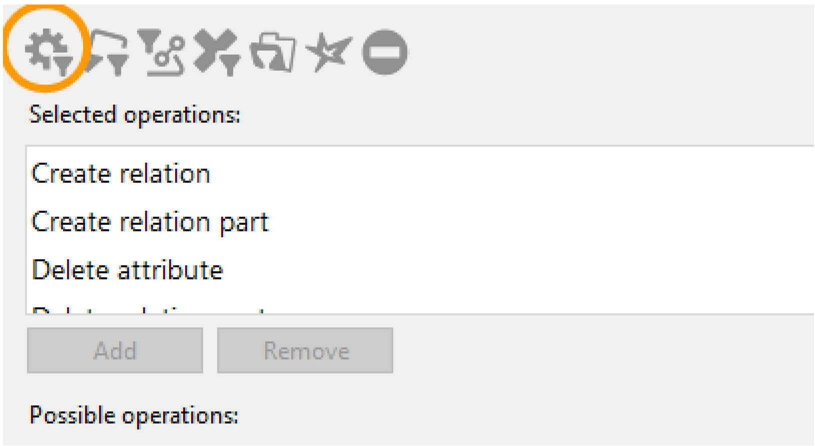
The trigger must contain the operation filters that will log the change history, and the elements where the attribute is to be visible.

The trigger script looks like this:

```
/**
 * Perform the trigger
 * @param parameter The chosen parameter, usually a semantic element
 * @param {object} access Object that contains all parameters of the
access
 * @param {$k.User} user User that triggered the access
**/

function trigger(parameter, access, user) {
     $k.History.addToChangeLog(access,parameter);
}
```

Example

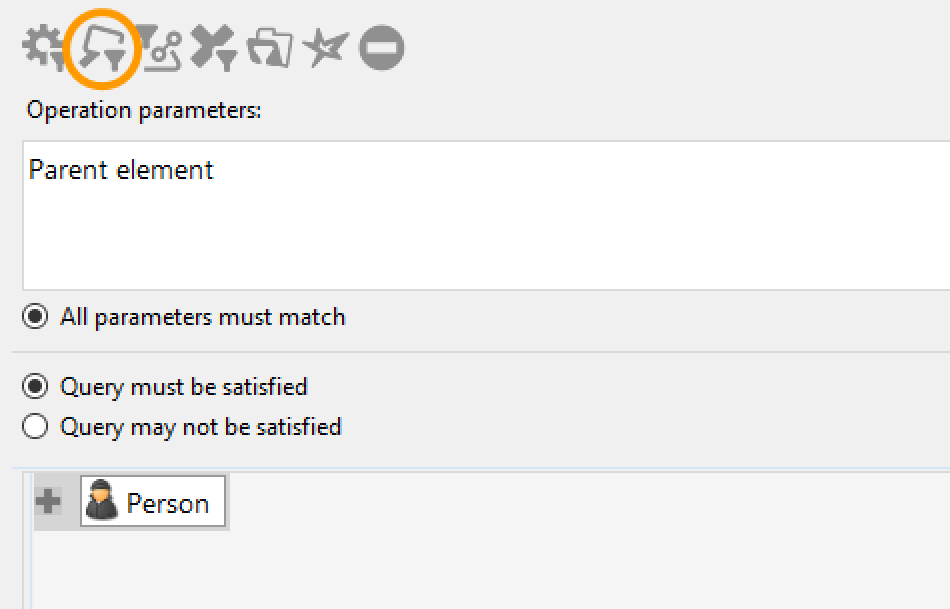
A change log is to be saved in all objects in a Knowledge Graph. The aim is to log the modification, creation and deletion of properties in the objects. First, an operation filter is created that reacts to the operations “Delete attribute”, “Modify attribute value”, “Create relation”, “Create relation part” and “Delete relation part”.


Selected operations:

Create relation
Create relation part
Delete attribute

Possible operations:

In the next step, a query filter was defined to determine the Knowledge Graph on which operations are performed.



Operation parameters:

Parent element

● All parameters must match

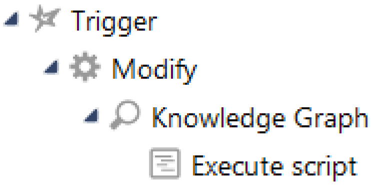
● Query must be satisfied

○ Query may not be satisfied

Person

The “Superordinate element” operation parameter was added to the trigger script, because it corresponds to the query filter.

The trigger rules (operation filter, query filter and trigger script) are located in the hierarchy tree as follows due to their checking sequence:



- Trigger
  - Modify
    - Knowledge Graph
      - Execute script

1.2.3. Filter types

With the aid of filters, the conditions are defined in the rights tree or in the trigger tree to allow access situations to be restricted when a decider or trigger should be executed. New filters are created under the node currently selected in the tree. This way, they are nested in each other.

The three filter types operation filter, query filter and property filter are available in the rights system. In addition to the three basic filter types, the trigger area provides a specific filter – the deletion filter.

There are different types of filters – when do we use which filter?

Symbol	Filter	Description
	Operation filter	Filters the operations; selection from list
	Query filter	Filters elements by means of structured query
	Property filter	Filters relations and attributes; selection from list

Symbol	Filter	Description
✖	Delete filter	Filters the deletion of elements

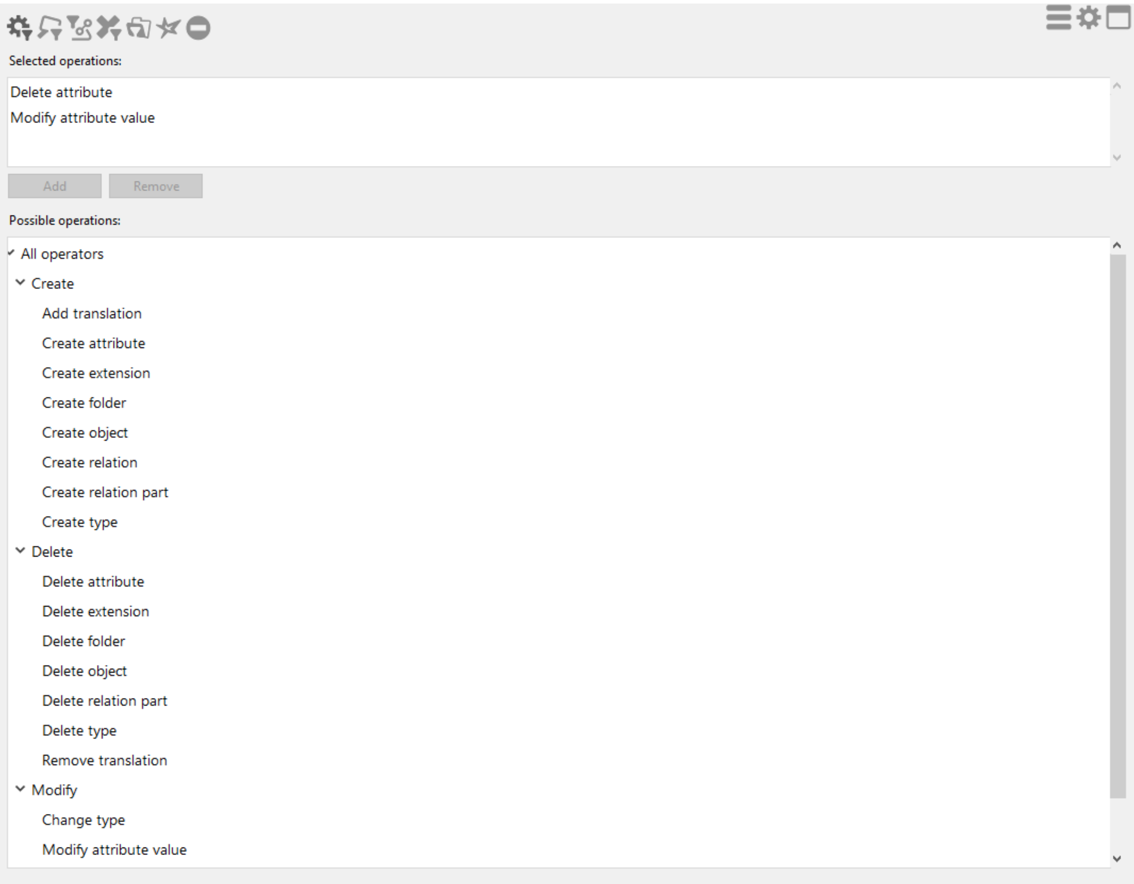
Operations can only be determined using an operation filter. Users can only be determined using a query filter. Properties can be determined using either query filters or property filters. The use of property filters makes sense when properties should be filtered regardless of other properties in the Knowledge Graph such as relations to the user. Above all, when large sets of properties are to be filtered, it is more straightforward and clearer to do so in a list instead of in a structured query. If relations to the accessed element or to the user are to be factored in, then a query filter must be used.

Instructions for creating a filter

1. In the rights or trigger tree, choose the position at which you want to create a new filter.
2. Use the buttons ⚙, , or ✖ to create a new filter.
3. The filter is created in the tree as a subfolder of the currently selected folder.
4. Assign a name to the folder.

1.2.3.1. Operation filter

To specify the operations for which an access right should apply or a trigger should be executed, operation filters are required. By selecting the required operation it is possible to add it to or remove it from the filter.



Selected operations:

Delete attribute
Modify attribute value

Possible operations:

- All operators
  - Create
    - Add translation
    - Create attribute
    - Create extension
    - Create folder
    - Create object
    - Create relation
    - Create relation part
    - Create type
  - Delete
    - Delete attribute
    - Delete extension
    - Delete folder
    - Delete object
    - Delete relation part
    - Delete type
    - Remove translation
  - Modify
    - Change type
    - Modify attribute value

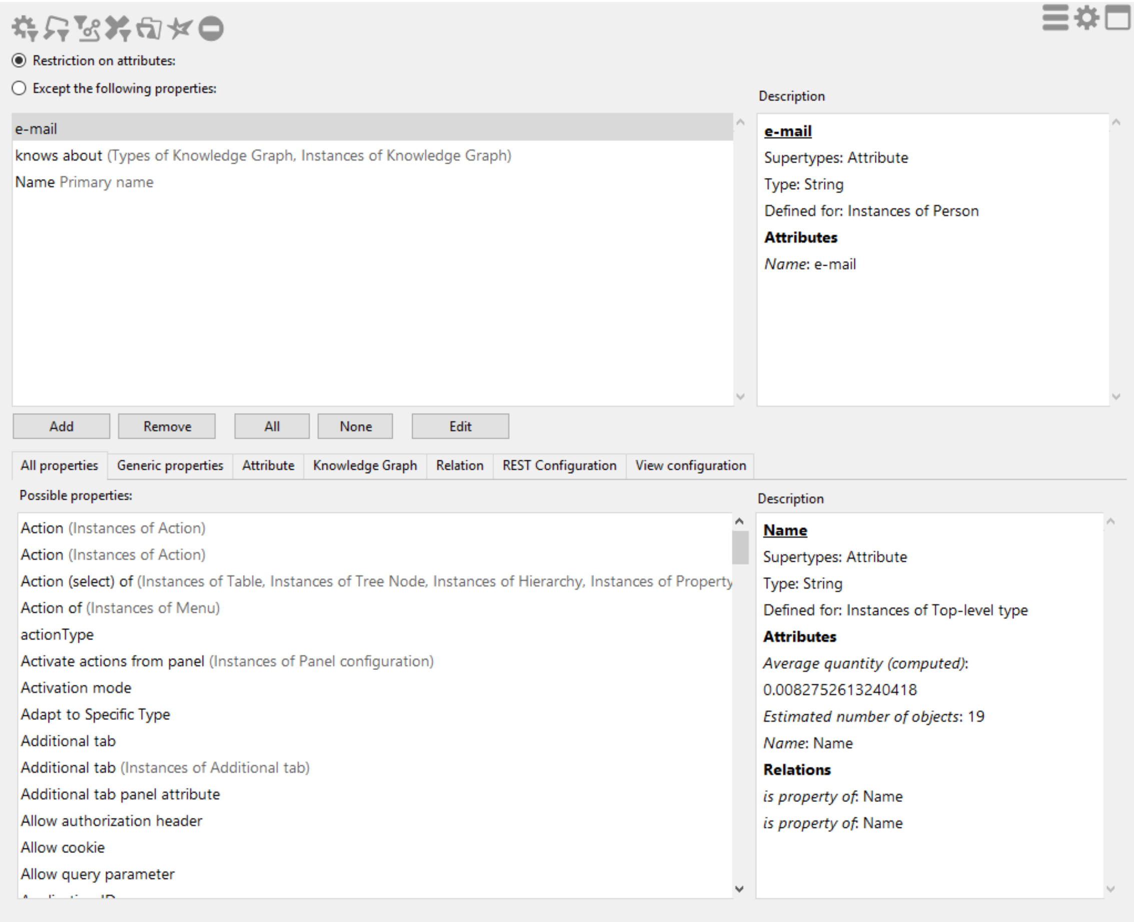
The operations are divided into groups. When you select the higher-level node of a group, all lower-level operations are included in the filter. If, for example, you choose the *Create* operation, the filter considers the operations *Create attribute* , *Create extension* , *Create folder* , *Create relation* , *Create relation half* , *Create type* and *Create translation* .

The Operations chapter lists all available operations and also specifies which operation parameters can be used in combination. The various operation parameters are explained accordingly in the Operation parameters chapter.

1.2.3.2. Property filter

You can use property filters to filter attributes and relations. There are two different procedures for using a property filter:

- *Restriction on properties* : Specify the properties to which the condition is supposed to apply. Subsequent filters or deciders of the subtree are only executed if the access property matches the selected property.
- *Exclude the following properties* : Specify the properties to which the condition is not supposed to apply. If the access property matches one of the selected properties, subsequent filters, deciders or triggers are not executed.



● Restriction on attributes:
○ Except the following properties:

e-mail
knows about (Types of Knowledge Graph, Instances of Knowledge Graph)
Name Primary name

Description

e-mail
Supertypes: Attribute
Type: String
Defined for: Instances of Person
Attributes
Name: e-mail

Possible properties:

Action (Instances of Action)
Action (Instances of Action)
Action (select) of (Instances of Table, Instances of Tree Node, Instances of Hierarchy, Instances of Property
Action of (Instances of Menu)
actionType
Activate actions from panel (Instances of Panel configuration)
Activation mode
Adapt to Specific Type
Additional tab
Additional tab (Instances of Additional tab)
Additional tab panel attribute
Allow authorization header
Allow cookie
Allow query parameter

Description

Name
Supertypes: Attribute
Type: String
Defined for: Instances of Top-level type
Attributes
Average quantity (computed): 0.0082752613240418
Estimated number of objects: 19
Name: Name
Relations
is property of: Name
is property of: Name

You can use *Add* and *Remove* to select the properties listed below. All properties below can be selected using *All* . *None* removes all selected properties. You can use the *Edit* field to call up the Detail editor of the attribute or relation that is selected in the top selection field. The tabs *All properties* , *Generic properties* , *Attribute, Relation* , *View configuration* and *Knowledge Graph* are designed to help users find the filtering properties more quickly. The *Knowledge Graph* tab shows all relations and attributes that the user has created.

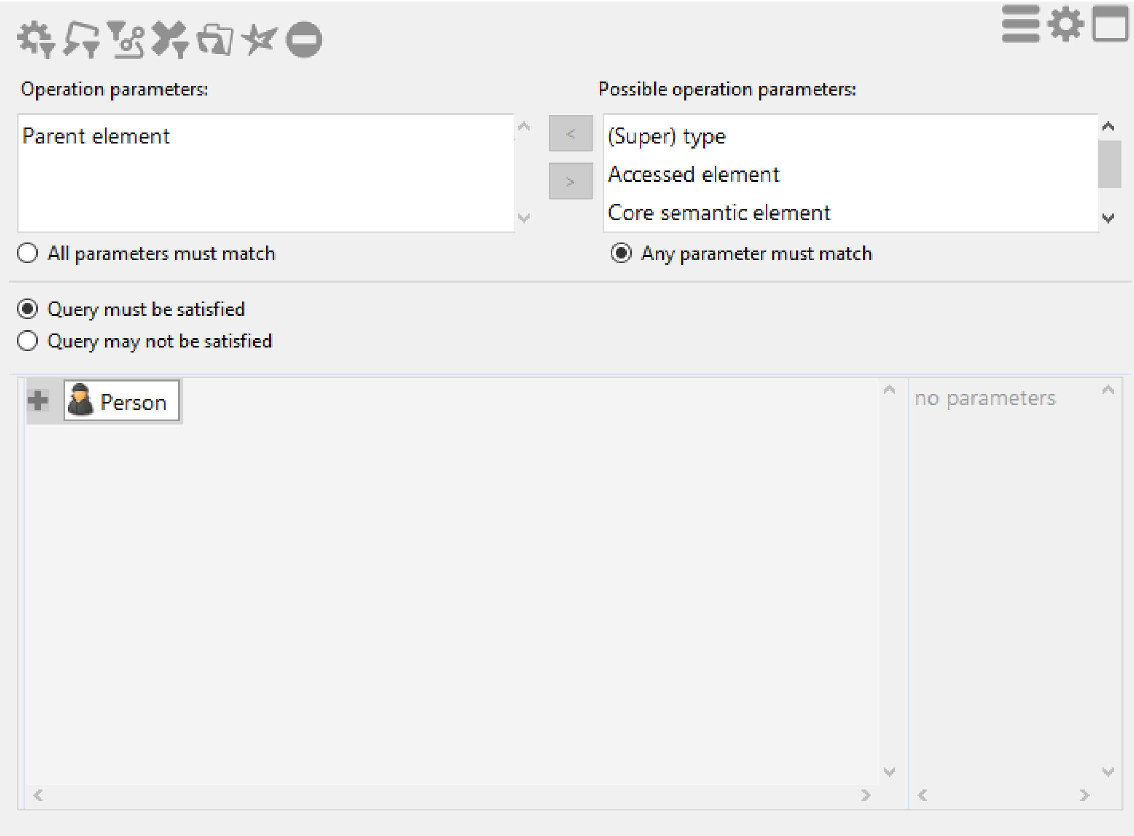
1.2.3.3. Query filter

Query filters make it possible to include elements in the environment of the element that is to be accessed. This allows not only individual properties, but also relationships between objects, properties and attributes to be included in the rights or trigger definition. When using query filters, it is necessary to specify an operation parameter to which the result of the structured query is compared. All available operation parameters are explained in the Operation parameters chapter.

There are two ways to define query filters:

- *Search condition must be met* : This setting is selected initially. If the search result of the structured query matches the operation parameter, the condition of the filter is met and subsequent filters, deciders or triggers are executed.
- *Search condition must not be met* : If the structured query returns the same element as the

access parameter as its result, the condition is not met and the check of the rights or trigger tree switches to the next subtree. If the result of the structured query differs from the result of the access parameter, the condition is met and the subsequent filter, decider or trigger is executed.



Operation parameters:

Parent element

○ All parameters must match

Possible operation parameters:

(Super) type
Accessed element
Core semantic element

● Any parameter must match

● Query must be satisfied
○ Query may not be satisfied

Person

no parameters

The objects of the type at the top left that match the search condition are the result of the structured query. These are compared to the element that is transferred by the operation parameter. It is possible to use access parameters in the structured query. They can be used, for example, to include the user, accessed element etc. in the query.

During selection of the operation parameter it is possible to configure whether

- all selected parameters must apply (*All parameters must apply*)
- or only one parameter must apply (*One parameter must apply*).

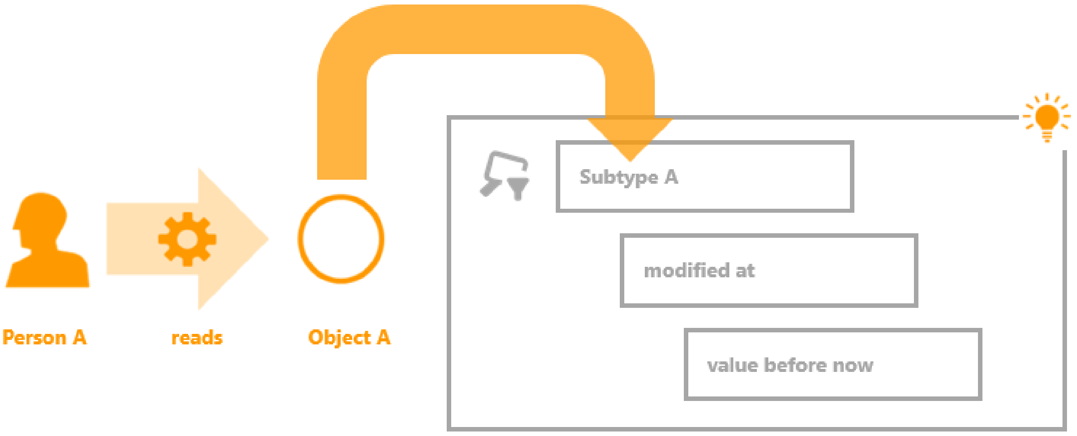
NOTE

Initially, the setting *All parameters must apply* is selected. If, for example, the operation parameters *Accessed element* and *Primary semantic element* are selected, the condition is met only if the result of the structured query is both the accessed element and the primary semantic element of the operation to be checked.

Example 1: Query filter in the rights system

A right should be defined that determines that already modified object may be read by everyone;

unmodified objects, in contrast, may not.

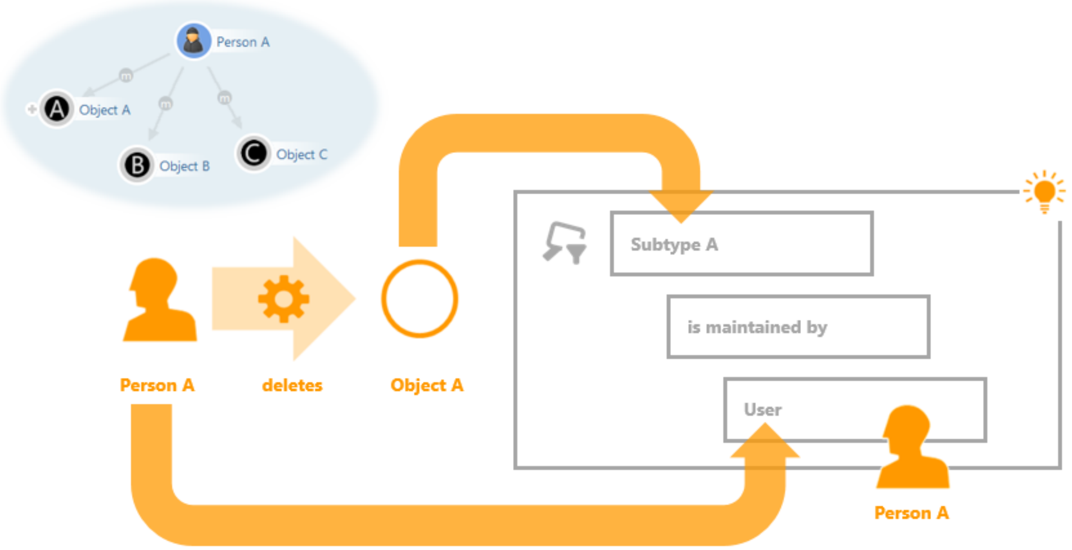


In this example, the user "Person A" would like to read "Object A". This operation is now checked by the rights system. A query filter has been defined in the rights system which checks whether the object has already been modified. The structured query of the query filter searches of objects of the "Subtype A" type, with the restriction that the attribute "modification date" is in the past. The structured query delivers all objects that meet this condition. If "Object A" is one of them, then the check by the filter returns a positive result and the folder that follows the query filter (with a filter or decider) is executed.

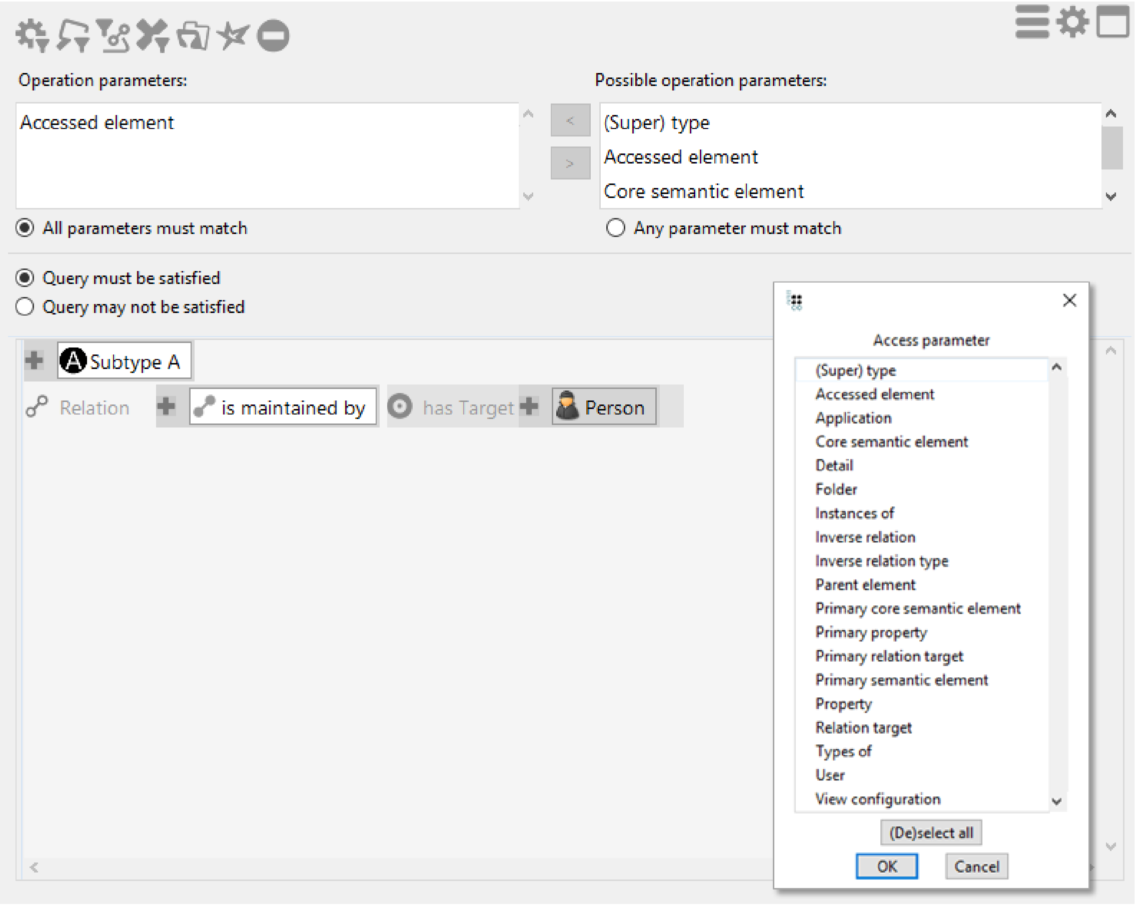
In the case of the query filter, the search condition settings must be met, and "All parameters must apply" must be selected.

Example 2: Query filter in the rights system

In most cases, there is a connection between the user who wants access and the objects and properties that the user wants to access. An example of this would be: "Employees of a department who look after a branch may edit all customers of this branch." Another version of this example that is illustrated below would be: "Users who maintain an object may edit and delete this object."



The left side shows a section of the Knowledge Graph: The object "Person A" is linked to the objects "Object A", "Object B" and "Object C" via the relation "maintains". The inverse relation of “maintains” is “maintained by,” which exists between the objects Object A, Object B and Object C and the object Person A, and is queried in the query filter. This relation in the Knowledge Graph represents that one person maintains object data relating to "Subtype A".



Operation parameters:

Accessed element

Possible operation parameters:

(Super) type
Accessed element
Core semantic element

● All parameters must match   ○ Any parameter must match

● Query must be satisfied
○ Query may not be satisfied

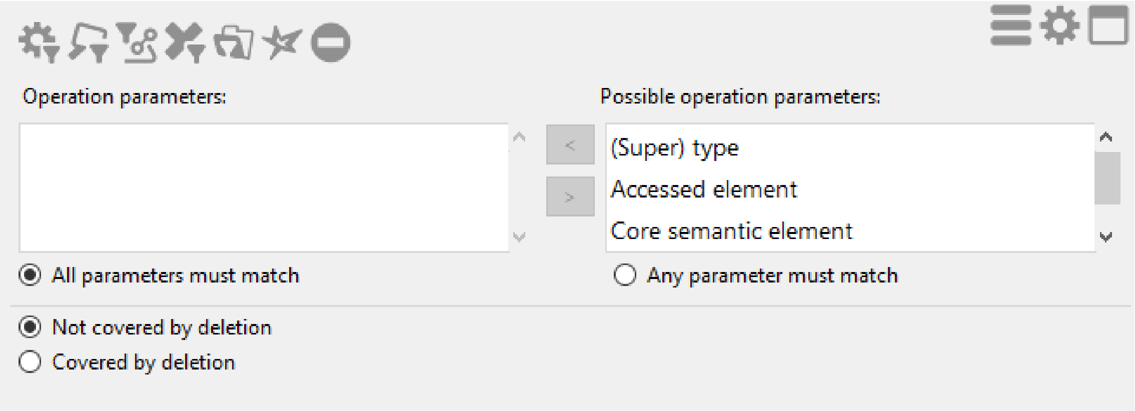
Subtype A
Relation   is maintained by   has Target   Person

In this example, user "Person A" wants to delete "Object A". The corresponding query filter delivers all objects of "Subtype A" that were maintained by a certain user as the query result. The current user is transferred to the structured query as an access parameter. The “Structured query” chapter explains access parameters in structured queries. Hence the search in this access situation returns all objects that were maintained by Person A. Since Object A is one of them, the query filter check returns a positive result.

In this example, the access situation adds two aspects to the query filter: the object to be deleted and the user. The query filter can thus be defined in two different ways. The object is either transferred to the query filter as an accessed element and the user is used as the access parameter in the structured query. Or the user is transferred to the query filter as the operation parameter “User” and the object is used as the access parameter “Accessed element” in the structured query.

1.2.3.4. Delete filter

Delete filters are only available for defining triggers. They are used for testing in a deletion situation whether the higher-level element is also affected by the delete operation. For example, you want a trigger to not be executed if an object including all its properties is deleted but a deletion filter must be used if a certain property of the object is deleted.



Operation parameters:

Possible operation parameters:

(Super) type
Accessed element
Core semantic element

● All parameters must match ○ Any parameter must match

● Not covered by deletion
○ Covered by deletion

When defining a delete filter, at least one operation parameter must be specified which determines which deletion of an object is to be tested.

- *All parameters must apply* : All specified operation parameters must apply. For example, if two operation parameters are specified (accessed element and primary element), then it is checked whether the delete operation applies to both the accessed element and the primary element. This can only be the case if the primary element is also the accessed element.
- *One parameter must apply* : Only one of the specified operation parameters has to apply.

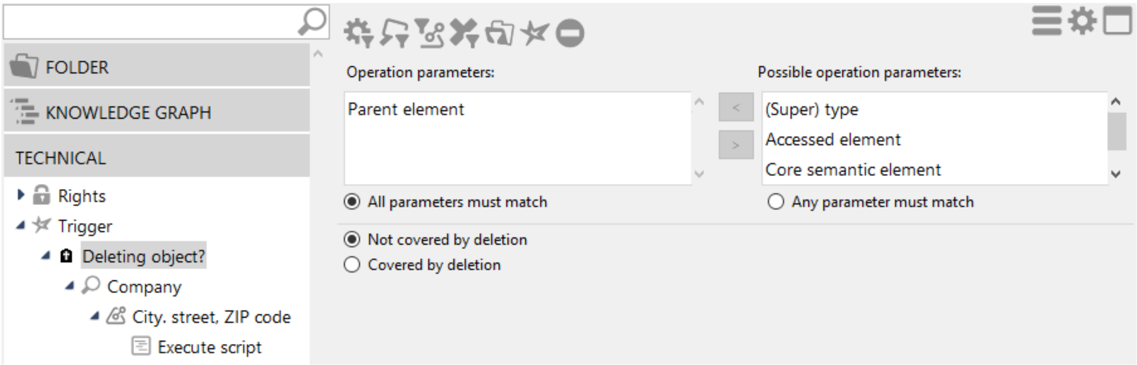
NOTE

In most cases, the operation parameter offers a superordinate element or primary object because a check is to be performed as to whether only the property is deleted or if the property is deleted because the entire object has been deleted.

- *Not affected by the delete operation* : The condition of the filter is positive if the element transferred in the operation parameter is not deleted in this transaction.
- *Affected by the delete operation* : The condition of the filter is thus positive if the element transferred in the operation parameter is deleted in this transaction.

Example: Delete filters in triggers

In this example, a trigger is only to be executed if the city, street or ZIP code of a company is modified or deleted, but not if the object containing the properties is deleted. The setting *Not covered by deletion* is used for this purpose. If the delete operation affects the superordinate accessed element, which in this case is the company object itself, then the checking of the subtree is aborted because the filter has returned a negative result.



Operation parameters:

Parent element

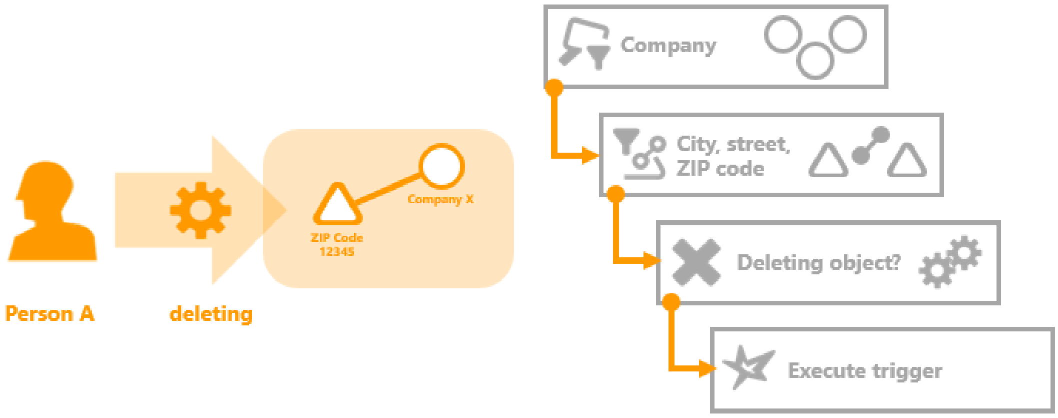
Possible operation parameters:

(Super) type
Accessed element
Core semantic element

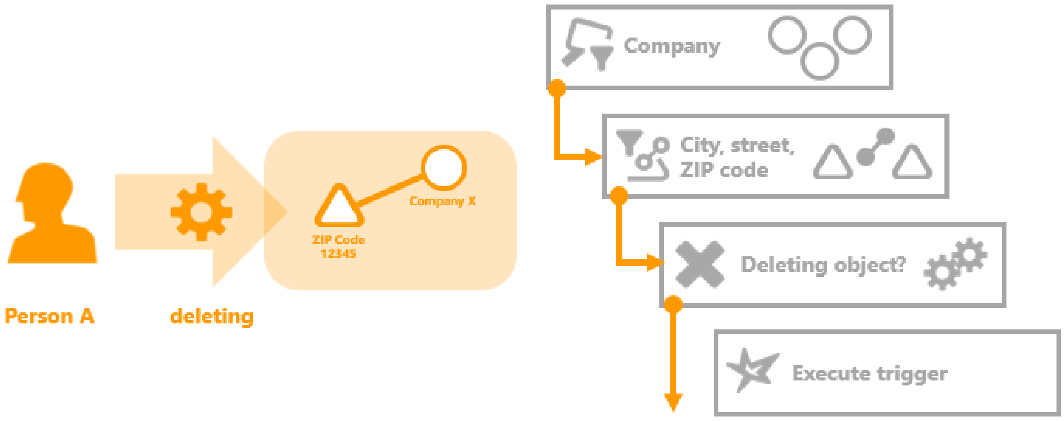
● All parameters must match ○ Any parameter must match

● Not covered by deletion
○ Covered by deletion

The superordinate element operation parameter is used along with the Not affected by the delete operation setting.



In this example access situation, the ZIP code attribute with the value “12345” in the “Company X” object is deleted. The object itself is not deleted. The “Company” query filter, which is defined by the “Superordinate accessed element” operation parameter, and the “City, street and ZIP code” property filter receive a positive response. The subsequent delete filter also returns a positive response, as the object containing the property (superordinate accessed element) is not affected by the delete operation – in line with the “Not covered by deletion” setting of the delete filter.



In this access situation the "Company X" object is deleted by user Person A. Deleting the object automatically deletes all properties of the object – and thus all attributes of the object as well. The check of the trigger tree is executed for the deletion of both the object and the attribute. The "Company" query filter and the "City, street and ZIP code" property filter are fulfilled for the delete process of the attribute in the check of the trigger tree. The delete filter itself is not fulfilled in this situation, as the "Company X" object containing the "ZIP code 12345" property is deleted.

Use of delete filters makes sense, for example, if the trigger script compiles the name of the object from its properties. As a result, the name of the object is not modified several times when the properties of the object are deleted; instead, the object and all related properties are deleted without the script for compiling the name being executed. This usually saves unnecessary calculation times and can make sense in specific application scenarios, e.g. if the trigger sends an email notification that an object has been renamed (and this avoids sending numerous superfluous emails regarding the name change).

1.2.4. Operation parameters

Operation parameters control the element to which the result of the structured query for the condition check should be compared in query filters. In the simplest case, the result is compared to the element that is to be used to execute the operation to be checked. Operation parameters can be used to modify the transferred element. You can choose the current user or elements from the element environment that will be used as the comparison element for the query filter.

They are also used, among other things, in delete filters and script triggers. Based on the element to which access is executed, they specify there the element on which the script is to be executed, or on which the deletion of elements (and which elements) is to be filtered.

When is this useful? It can be essential if you cannot use an element from the environment of the affected object instead of the object itself for comparison: when, for example, you want to check access rights for creating new objects or types. It is not possible to define a structured query that returns the object that has not been created yet. In this case, the query filter must be compared to something else, i.e. the type of object to be created and, in case of object types, to the super-type of the type to be created.

Operation parameter	Description
(Super) type	In the case of types, the (super) type is the super-type of the type. In the case of objects, the (super) type is the type of the object type. In the case of attributes or relations, the (super) type is the type of the property.
Accessed element	The *accessed element* is the element affected by the operation.
Application	Objects of the type "application" (to be found within TECHNICAL > View configuration > Object Types > Application).
Core semantic element	If the higher-level element is an extension, then the *core semantic element* is the object on which the extension is stored. Otherwise, the *core semantic element* is identical to the accessed element.
Folder	The *Folder* operation parameter is the folder affected by the operation.
Inverse relation	If the property affected by the operation is a relation, the parameter contains the inverse relation half.
Inverse relation type	The *inverse relation type* is the type of the inverse relation. This can be used for the generation of relations.
Parent element	The *parent element* is the object, the type or the extension affected by the operation. In the case of properties, the *parent element* is the object, the type or the extension on which the property is saved. If the accessed element is a meta property and the parent element is a relation, the following needs to be obeyed: • Due to the symmetric storage of meta properties at relation halves, the returned direction of the relation is not unique for double-sided relations (= conventional relations) or symmetric relations. In this case, the required relation half needs to be determined by means of a script or a structured query. • In case of single-sided relations, the parent element is the real relation half (meaning: not the virtual relation half).
Primary core element	If the primary element is an extension, then the *primary core element* is the core element of the extension. Otherwise, the *primary core element* is identical to the core semantic element.
Primary element	If the superordinate accessed element is a property, the *primary element* is the object, the type or the extension on which the property is stored (transitive). Otherwise, the *primary element* is identical to superordinate element.

Operation parameter	Description
Primary property	In the case of meta properties, the *primary property* is the property closest to the object, type or extension. Otherwise, the *primary property* is identical to property.
Primary relation target	The *primary relation target* is the primary semantic element of the relation target.
Property	The *property* is the property that the operation affects (attribute or relation). If the operation is performed on an object, type or extension, the operation parameter *property* is blank.
Relation target	If the property affected by the operation is a relation, the *Relation target* parameter contains the relation target of the relation half. (The source of the relation would be the higher-level element in this case.)
User	The *user* is the object of the users which executes the operation.

1.2.4.1. Operation parameter (Super) type

The “(super) type” parameter is used, for example, if operations that create new elements are to be checked in the rights system. When elements are created, the query filter cannot be defined so that it finds elements that have not been created yet. The query filter must work on the super-type or type of the element to be created. During the creation of objects, attributes and relations, the type of the objects, attribute or relation is used. For types, the super-type of the type to be displayed is used.

Accessed element	(Super) type
Object or extension	The type of object or extension
Type	The super-type
Property	The type of property

1.2.4.2. Operation parameter Accessed element

The accessed element is the element of the Knowledge Graph that is currently being accessed. For query filters in the rights system, for example, the accessed element is the element that is to be accessed by an operation. When checking an access situation, the element is then transferred to the query filter on which the operation is supposed to be executed. The query filter then compares the accessed element to the result of the structured query.

1.2.4.3. Operation parameter Application

The operation parameter "Application" refers to the application context within which the element is actually being accessed. Examples for applications are the Knowledge Builder or the Viewconfiguration mapper.

Accessed element	Application
Object, type or extension	Object of the currently used application

1.2.4.4. Operation parameter Core semantic element

The core element is used when work is done with extensions. Instead of the extension, the core element delivers the object to which the extension is saved.

Accessed element	Core object
Object, type or property	The actual accessed element
Extension	The object to which the extension is saved

1.2.4.5. Operation parameter Folder

If a folder from the *Folder* area of the Knowledge Graph is to be transferred to the search as a parameter, the Folder operation parameter must be used.

Accessed element	Folder
Folder	The actual accessed element
Object, type, extension or property	Blank

1.2.4.6. Operation parameter Inverse relation

The inverse relation is the “opposing direction” of a relation half. If the relation half is considered as directed graphs, then there is a relation between two opposing graphs (the “forward direction” and the “reverse direction” of the relation) that is attached between two elements. The inverse relation is therefore the opposing relation half. The inverse relation has the relation source of the relation half as its relation target and vice-versa.

Accessed element	Inverse relation
Relation half	The inverse relation half
Object, type, extension or attribute	Blank

1.2.4.7. Operation parameter Inverse relation type

The inverse relation type is the type of the inverse relation.

Accessed element	Inverse relation type
Relation half	Type of inverse relation half

Accessed element	Inverse relation type
Object, type, extension or attribute	Blank

1.2.4.8. Operation parameter Parent element

The semantic element is used if the direct properties of an element are to be retrieved.

Accessed element	Superordinate element
Object, type or extension	The actual accessed element
Property	Object, type or extension on which the property is stored
Meta-property	Property on which the meta-property is stored

1.2.4.9. Operation parameter primary core element

If you want the corresponding object or type to be addressed for an accessed element, you must use the primary core element. In contrast to the primary element, no extensions are addressed/permitted when using the primary core element. In case of extensions as accessed element, the core object is output.

Accessed element	Primary core element
Extension	The object to which the extension is saved
Object or type	The actual accessed element
Property or meta-property of an extension	The object to which the extension is saved
Property or meta-property of an object or type	Primary semantic element - object or type to which the property is saved (transitive)

1.2.4.10. Operation parameter primary element

The core semantic element always delivers an object, type or extension. If the core semantic element is executed on meta properties, the properties are processed transitively until the object, type or extension to which the properties are appended is found.

Accessed element	Core semantic element
Object, type or extension	The actual accessed element
Property	Object, type or extension on which the property is stored
Meta-property	Object, type or extension on which the property is stored on which in turn the meta-property is stored (transitive)

1.2.4.11. Operation parameter Primary property

The primary property is always a property. It resembles the primary semantic element in that it transitively processes meta properties. However, it delivers the last property that precedes the primary semantic element, that is, the property stored directly on the primary semantic element.

Accessed element	Primary property
Property	The actual accessed element
Meta-property (or meta-property of a meta-property)	The property that is closest to the object, type or extension
Object, type or extension	Blank

1.2.4.12. Operation parameter Primary relation target

In contrast to the primary semantic element of a relation half, the primary relation target is not the object, type or extension on which the relation half is located but the object, type or extension to which the inverse half of the relation is connected.

Accessed element	Primary relation target
Relation half	The primary semantic element of the relation target (object, type or extension on which the inverse relation half is stored)
Relation half whose relation target is a property or meta-property	The primary semantic element of the relation target (object, type or extension of the meta-property or property on which the inverse relation half is stored)
Object, type, extension or attribute	Blank

1.2.4.13. Operation parameter Property

Attributes and relations are understood to be properties. The operation parameter contains the attribute or the relation on which the operation is performed. If the operation is performed on an object or type, the operation parameter property is blank.

Accessed element	Property
Attribute or relation	The actual accessed element
Object, type or extension	Blank

1.2.4.14. Operation parameter Relation target

The relation target is not the source, but rather the “target” of a relation half. It can also be considered the inverse relation half.

Accessed element	Relation target
Relation half	The relation target is the relation source of the inverse relation
Object, type, extension or attribute	Blank

1.2.4.15. Operation parameter User

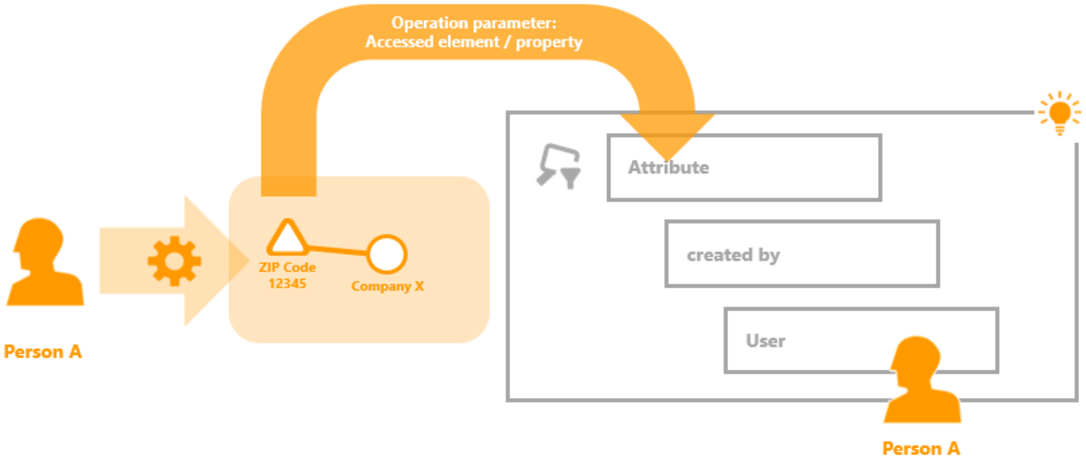
The “User” parameter is always the user object of the user who is currently logged in, regardless of the accessed element. For this purpose, the Knowledge Builder account must be linked to a Knowledge Graph object. The chapter on activation of the rights system describes how this link is created.

Accessed element	User
Object, type, extension or property	Object of the user who is currently logged in

1.2.4.16. Examples: The use of operation parameters

Example 1: Accessed element and property in the rights system

The example below shows the access situation on the left side and the corresponding query filter on the right side.



Access situation: Person A wants to change the attribute ZIP Code of company X.

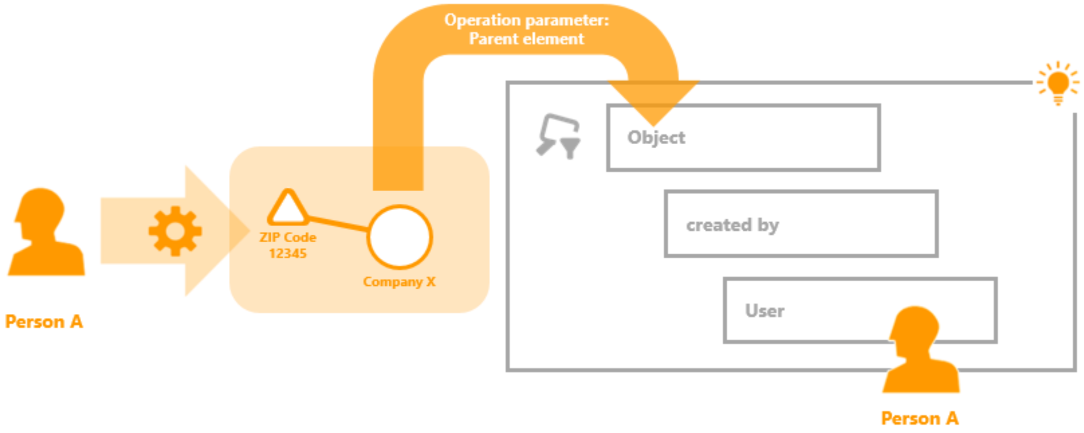
Query filter: All attributes created by a certain user are filtered. In the structured query, the access parameter “User” is used, which restricts the objects of user to the person who wants to execute the operation. This corresponds to all attributes that were created by Person A.

Checking the access rights: To check the access rights, the attribute (accessed element/property) on which the operation is to be executed is transferred to the query filter. If this attribute is included in the set of search results, the query filter check returns a positive result.

Operation parameter: The attribute Duration is transferred to the query filter. In this case, both the operation parameter “Accessed element” and the property can be used because the attribute “ZIP Code” is actually a property and represents the accessed element of the operation.

Example 2: Superordinate element and primary semantic element in the rights system

This example shows the access situation on the left side and the corresponding query filter on the right side.



Access situation: Person A changes the Zip Code attribute, which currently has the value 12345 and is part of the Company X object.

Query filter: The query filter is defined in such a way that it searches for all objects that were created by a specific user; the currently logged-in user is the accessed element. Accordingly, the query filter finds all the objects created by Person A.

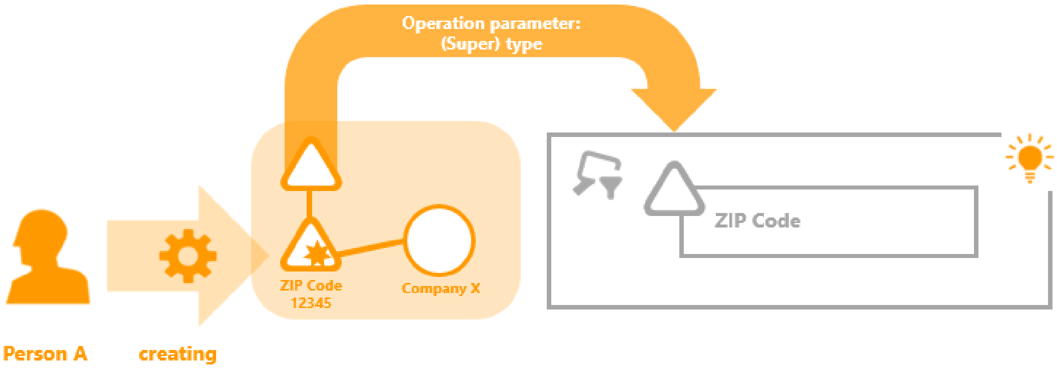
Checking the access rights: If the result set of the query filter contains Company X, the following folder (filter or decider) is executed.

Operation parameter: Use of the “Superordinate element” operation parameter has the effect that, instead of the “Zip Code” attribute to be changed being transferred to the query filter, the object in which it was defined is transferred to the query filter. This is the case for Company X.

In addition to the superordinate element, it would also be possible to use the “Primary semantic element” operation parameter in this case. The “Superordinate element” operation parameter would have the result that all properties and the object itself are rated positive by the filter. In addition, the “Primary semantic element” operation parameter would also permit meta properties of the object, no matter how many properties are between the object and the meta property.

Example 3: (Super) type in the rights system

The example shows the access situation on the left-hand side and the query filter applied in this situation on the right-hand side.



Access situation: Person A wants to create the attribute Zip Code on the object Company X. The value is to be 12345.

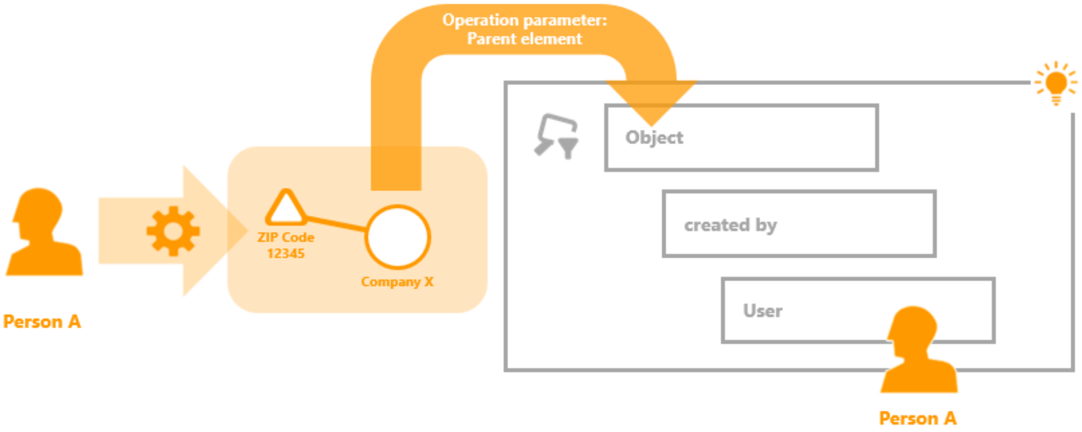
Query filter: The query filter returns the attribute type “ZIP Code”.

Checking the access rights: If the attribute to be created has the “ZIP Code” type, the check of the query filter returns a positive result.

Operation parameters: When creating elements, it is not possible to define a query filter that returns the element to be created and is thereby able to check the access rights. This means that a different operation parameter must be chosen as the accessed element when creating elements. The “(super) type” operation parameter is suitable in these situations. In this example, the attribute type is used, which is the ZIP Code attribute type.

Example 2: Superordinate element and primary semantic element in the rights system

This example shows the access situation on the left side and the corresponding query filter on the right side.



Access situation: User Paul changes the Length attribute, which currently has the value 02:30 and is part of the Song X object.

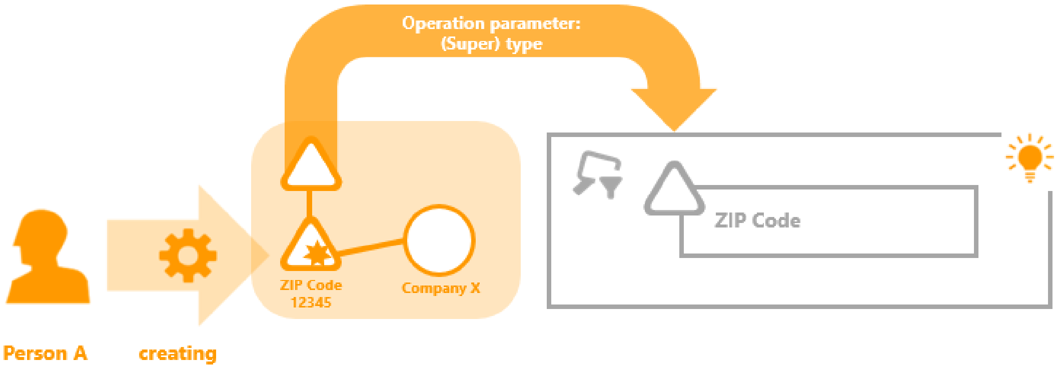
Query filter: The query filter is defined in such a way that it searches for all objects that were created by a specific user; the currently logged-in user is the accessed element. Accordingly, the query filter finds all the objects created by Paul.

Checking the access rights: If the result set of the query filter contains Song X, the following folder (filter or decider) is executed.

Operation parameter: Use of the "Superordinate element" operation parameter has the effect that, instead of the "Length" attribute to be changed being transferred to the query filter, the object in which it was defined is transferred to the query filter. This is the case for Song X. In addition to the superordinate element it would also be possible to use the "Primary semantic element" operation parameter in this case. The "Superordinate element" operation parameter would have the result that all properties and the object itself are rated positive by the filter. In addition, the "Primary semantic element" operation parameter would also permit meta properties of the object, no matter how many properties are between the object and the meta property.

Example 3: (Super) type in the rights system

The example shows the access situation on the left-hand side and the query filter applied in this situation on the right-hand side.



Access situation: User Paul wants to create the attribute Length on the object Song X. The value is to be 02:30.

Query filter: The query filter returns the attribute type "Length."

Checking the access rights: If the attribute to be created has the "Length" type, the check of the query filter returns a positive result.

Operation parameters: When creating elements, it is not possible to define a query filter that returns the element to be created and is thereby able to check the access rights. This means that a different operation parameter must be chosen as the accessed element when creating elements. The "(super) type" operation parameter is suitable in these situations. In this example, the attribute type is used, which is the Length attribute type.

1.2.5. Operations

Operation filters can be used to specify operations that are then permitted in the filter process of operation filters. If a different operation is executed in the access situation than specified in the operation filter, the system switches to the next subtree when traversing the rights or trigger tree.

The general operations *Create* , *Read* , *Modify* and *Delete* consist of multiple individual operations. If one operation group is prohibited, that means that all the operations it contains are also not permitted; vice versa, if an operation group is permitted, all the operations it contains are automatically permitted as well.

The table shows an overview of all available operations that can be applied in operation filters. Depending on the operation, only specific operation parameters can be used in query filters. These are specified in the "Operation parameters" column.

NOTE Derived operation parameters such as primary semantic elements or primary semantic core objects, for example, can be used whenever the parameter from which they are derived can be used.

Special features of triggers No read operations can be used for triggers. In addition, the operation groups Display of objects (operation: Display in graph editor) and Edit (operation: Validate attribute value are not available for triggers.

In addition, the "Accessed element" operation parameter is available for triggers in the "Create" operations if the time/type of execution is set to *After the change* or *End of transaction* .

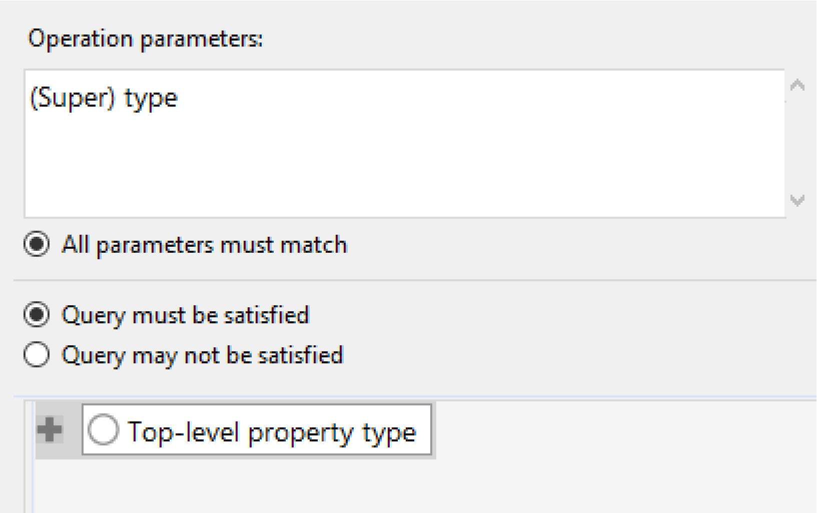
Operation group	Operation	Operation parameter
Display of objects	Display in graph editor	Accessed element
Edit	Validate attribute value	Accessed element, property, superordinate element, (parameter to be checked: attribute value)
User-defined operation		
Create	Create attribute	(Super) type, superordinate element
	Create extension	(Super) type, superordinate element, core object
	Create object	(Super) type
	Create folder	Folder
	Create relation	(Super) type, superordinate element, relation target, inverse relation type

Operation group	Operation	Operation parameter
	Create relation half	(Super) type, superordinate element, relation target
	Create type	(Super) type
	Add translation	Accessed element, property, superordinate element
Read	Read all objects/properties of a type	(Super) type
	Read attribute	Accessed element, property, superordinate element
	Read object	Accessed element, superordinate element
	Read relation	Accessed element, superordinate element, property, inverse relation, relation target, inverse relation target
	Read type	Accessed element, superordinate element
Delete	Delete attribute	Accessed element, superordinate element
	Delete extension	Accessed element, property, superordinate element
	Delete object	Accessed element, superordinate element
	Delete folder	Folder
	Delete relation half	Accessed element, inverse relation, property, superordinate element, relation target, inverse relation target
	Delete type	Accessed element, superordinate element
	Remove translation	Accessed element, property, superordinate element
Modify	Modify attribute value	Accessed element, property, superordinate element
	Modify folder	Folder

Operation group	Operation	Operation parameter
	Modify schema	Accessed element, superordinate element
	Change type	Accessed element, superordinate element
Use tools	Export	is no longer evaluated
	Import	is no longer evaluated
	Edit/execute script	is no longer evaluated

Read object The operation *Read object* is used to display objects for the corresponding object type on the Objects tab. The operation does not prevent the display of the object when it is called up using a linked object. In this case, the operations for properties *Read attribute* and *Read relation* then apply.

Read all objects/properties of a type This operation specifically controls the access rights check when processing a structured query. A structured query checks all intermediate results by default. A search for all employees with a wage greater than €10,000 would therefore not result in any hits when the wage cannot be read, even if the corresponding employee objects could be read. This response is often preferred, however is seldom performant. In the case of an extensively configured rights system in particular, processing of which requires a lot of processor capacity, we recommend using a control that does not require intermediate results of a structured query to be checked because a check of the final results is sufficient. In most Knowledge Graphs, permission can be issued for all property types ("top-level type for properties").



Operation parameters:

(Super) type

● All parameters must match

● Query must be satisfied

○ Query may not be satisfied

○ Top-level property type

To examine which intermediate results are checked, this information can be made to appear in a structured query. This is done using "Settings→Personal→Structured query→Show access rights checks".

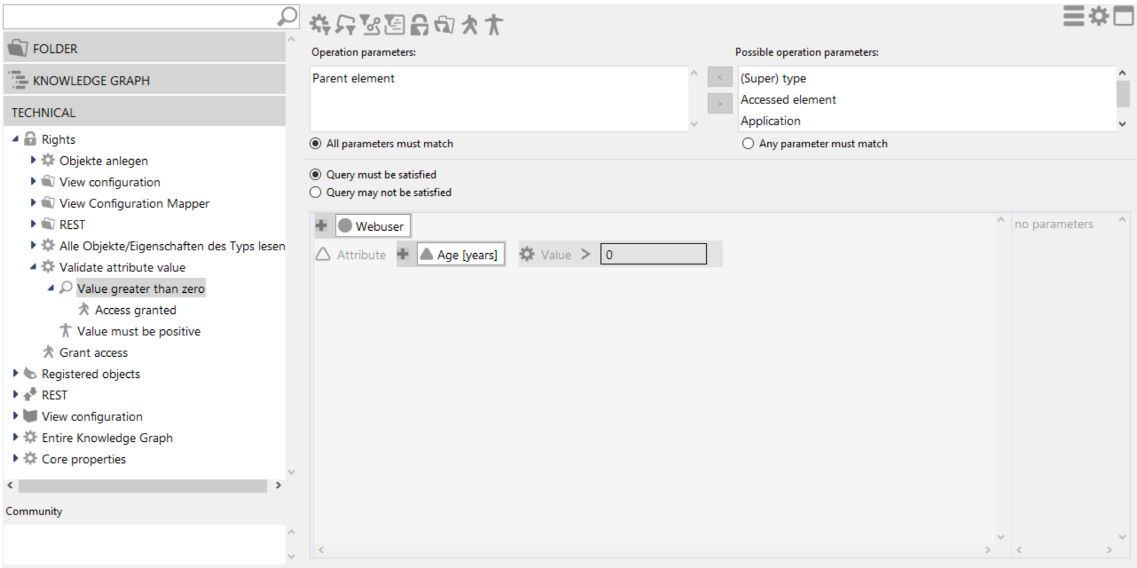
Validate attribute value The operation *Validate attribute value* is used when the attribute value to be set must satisfy certain conditions. The definition of the condition for the attribute value is made in a structured query.

Case example of a configuration: The entered age of a webuser must be greater than zero.

Configuration by means of the structured query in the rights system: the attribute "Age [years]" contains the condition "value > 0".

Configuration in the rights system: If the query condition is satisfied, then the value can be stored - this is done by configuring a positive stopmarker "Access granted" , for all other cases the storage is denied by using the stopmarker "Access denied" - which in this case is renamed to "Value must be positive" for displaying the validation message.

NOTE The name of the stopmarker (here: "Value must be positive") is going to be displayed by the validation mechanism in the web frontend.



Operation parameters: Parent element

Possible operation parameters: (Super) type, Accessed element, Application

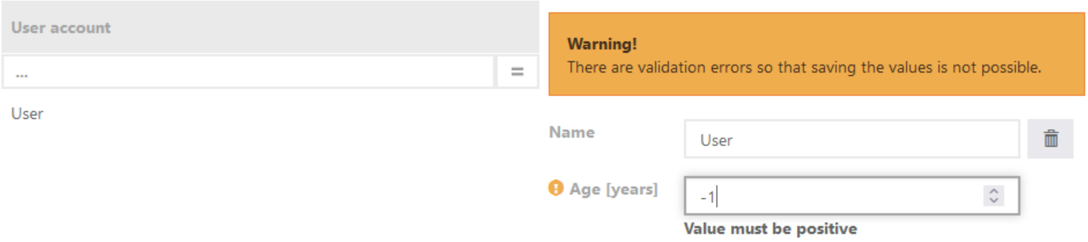
● All parameters must match ○ Any parameter must match

● Query must be satisfied
○ Query may not be satisfied

Webuser
Attribute Age [years] Value > 0

no parameters

Display in the web frontend: when entering a wrong value, the validator returns a yellow warning message with the name of the stopmarker directly underneath the related property edit input field:



Warning!
There are validation errors so that saving the values is not possible.

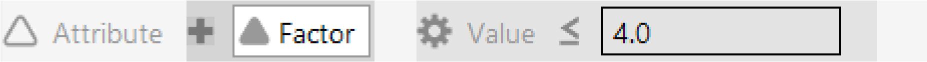
Name: User

Age [years]: -1

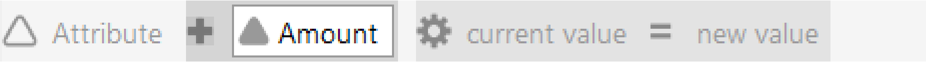
Value must be positive

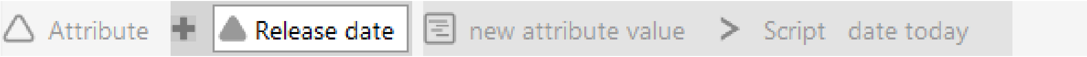
Two possible definitions are available there for validation of the attribute value:

- *Condition for the attribute value to be set* : The new value of the attribute can be validated by a comparison with a specified value in the structured query.


Attribute + Factor Value ≤ 4.0

Example: The attribute value may only be less or equal to 4.0.

- *Compare with the attribute value to be set* : This compares the current value with the new value.
  
  Attribute + Amount current value = new value

  Example: The new value of the attribute age may only be greater in this case. Smaller values are not permitted.
- Compare the value to be set with the result of a script: This initially determines a comparative value by means of a script.
  
  Attribute + Release date new attribute value > Script date today

  The script is called using a parameter object that contains the following properties:

Property	Value
attributeValue	Value to be set
property	Property to be changed (attribute)
topic	Element of the property
user	User who sets the value

Different comparative operators are available for the validation, which can be used to check the attribute value to be set with another value. If the new value does not match the defined condition, the filter check produces a negative result when the initial setting *Search condition must be satisfied* has been selected.

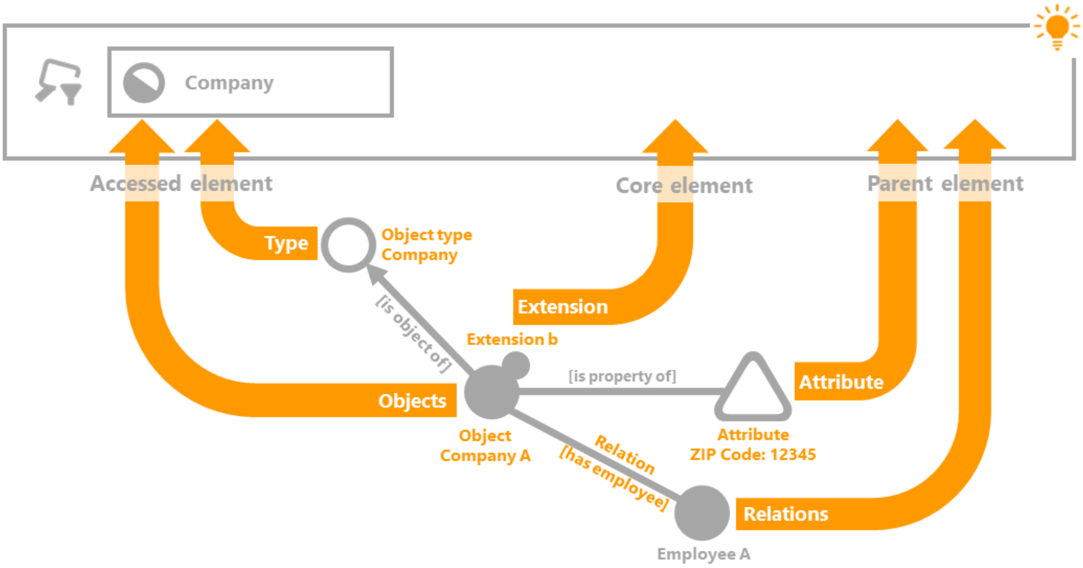
Modify schema The modify schema operation concerns changes to the definition area of relations and changes to the type hierarchy (*is a subtype of* and *is a super-type of* relations).

1.2.5.1. Example: The use of operation groups in the right system

This example shows how groups of operations (read, generate, modify, delete) can be used sensibly when defining rights. All operations are to be prohibited for the Company type and its objects. This includes the following actions:

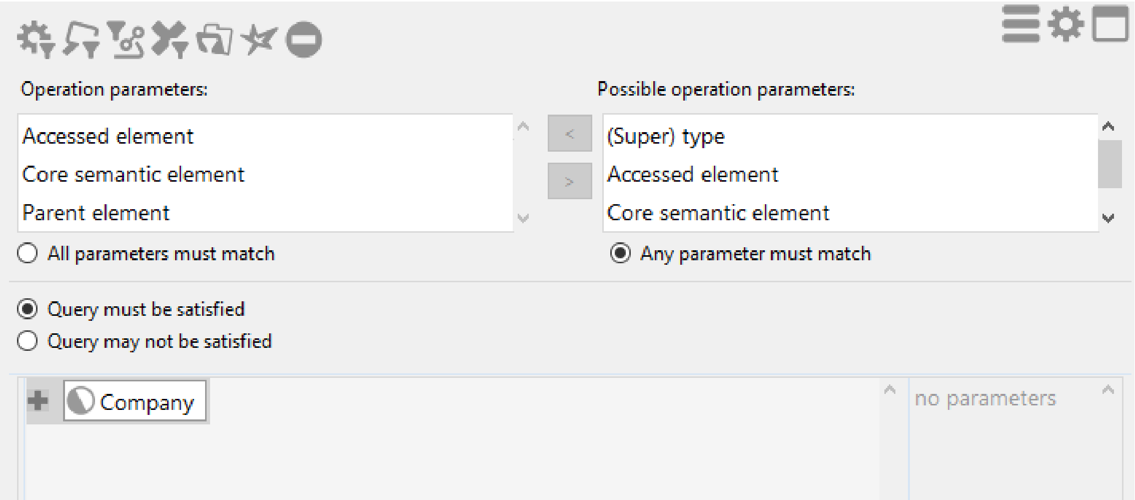
- Deletion of the object type Company
- Deletion of specific company (objects of Company)
- Deletion of attributes that occur on a company
- Deletion of relations that occur on a company (relation target and source)
- Deletion of extensions that extend objects of Company
- Deletion of attribute and relation types that have objects or subtypes of Company as their definition area

For example, if all delete operations for an object and the corresponding type are to be prohibited, you have to ensure you cover all delete operations by means of the corresponding parameters when selecting the operation parameters in the query filter of the right:



The only condition of the query filter used is the object type Company, for which the setting Instances and Subtypes is selected. The operation parameter "Accessed element" covers the object type "Company" and all objects that belong to this type. The parameter Core object covers the extension objects that belong to copanies. Attributes and relations are covered by the operation parameter "Parent element."

In the rights tree, the operational filter for the delete operation comes first. This is followed by the query filter depicted below and finally the decider "Access refused."



Operation parameters:

Accessed element
Core semantic element
Parent element

○ All parameters must match

Possible operation parameters:

(Super) type
Accessed element
Core semantic element

● Any parameter must match

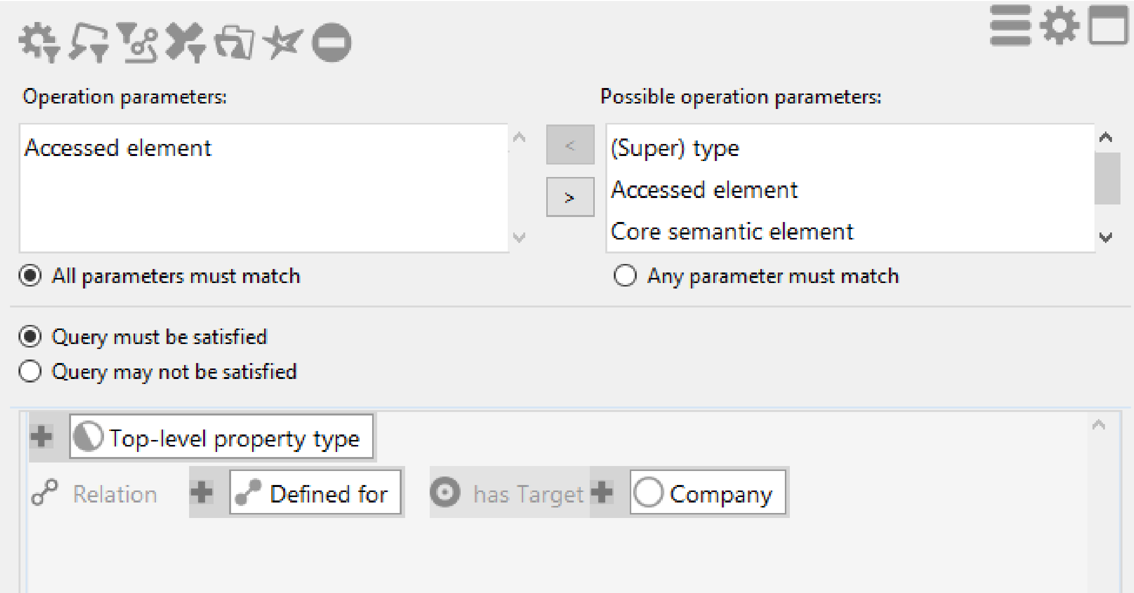
● Query must be satisfied
○ Query may not be satisfied

Query filters used in the example: "Core object," "Superordinate element" and "Accessed element" have been selected as operation parameters. The settings used are "One parameter must apply" and "Search condition must be met."

Extension of the right with attribute and relation types

A thus defined right covers all but one of the above described requirements on the right. Only the deletion of attribute and relation types that have been defined for objects and subtypes of songs are not taken into account in this definition of rights.

The definition of rights is extended with the following filter:



Operation parameters:

Accessed element

Possible operation parameters:

(Super) type
Accessed element
Core semantic element

● All parameters must match
○ Any parameter must match

● Query must be satisfied
○ Query may not be satisfied

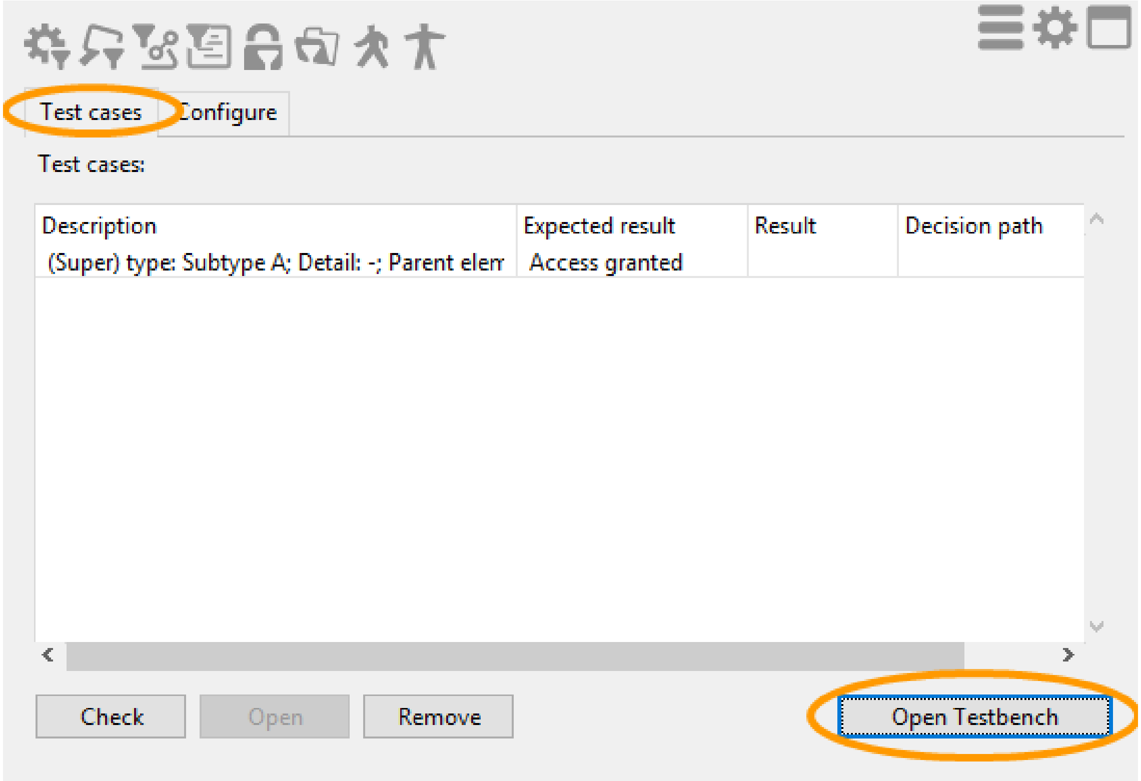
Top-level property type
Relation — Defined for — has Target — Company

The query filter includes all property types (attribute and relation types) that have been defined for objects or subtypes of company. In the query filter definition, the parameter “Accessed element” and the setting “Search condition must be met” are used.

1.2.6. Testbench

When the *Rights* folder is selected in the *System* area, the *Saved test cases* and *Configure* tabs are available in the main window. The test system area is found in the *Saved test cases* tab. The test system for triggers is called in the *Triggers* folder by means of the *System* area.

Saved test cases can be tested again here. The test interface in which the test cases can be defined can be called using the *Open testbench* button.



Test cases:

Description	Expected result	Result	Decision path
(Super) type: Subtype A; Detail: -; Parent elem	Access granted		

In addition to the functionalities that are described in the following chapters, Testing an access situation and Defining test cases, there is the option of testing access rights directly on an object or type. Select the access rights function using the context menu (right click). The following menu items can be selected there:

- **Object:** All operations (modify, delete, read and display in graph editor) are tested on the object and their result is output.
- **All:** All operations (modify, delete, read and display in graph editor) are tested on the object and all their properties (attributes and relations) are tested.
- **Rights system test environment:** The test environment for checking rights opens.

1.2.6.1. Test the access right situation

Two areas are relevant for testing the rights system and the trigger functionality:

- The actual test environment: The test environment offers the option to test the access rights or when a trigger is executed for a certain test case.
- The *Saved test cases* tab: This lists the test cases and makes them available for subsequent checking.

Instructions for opening the test environment

1. Select the folder *Rights* or *Triggers* in the *Technical* area in the Knowledge Builder.
2. If you are working in the rights system, select the *Saved test cases* tab in the main window.

3. Click *Open test environment* (bottom right) so that the test environment opens in a new window.

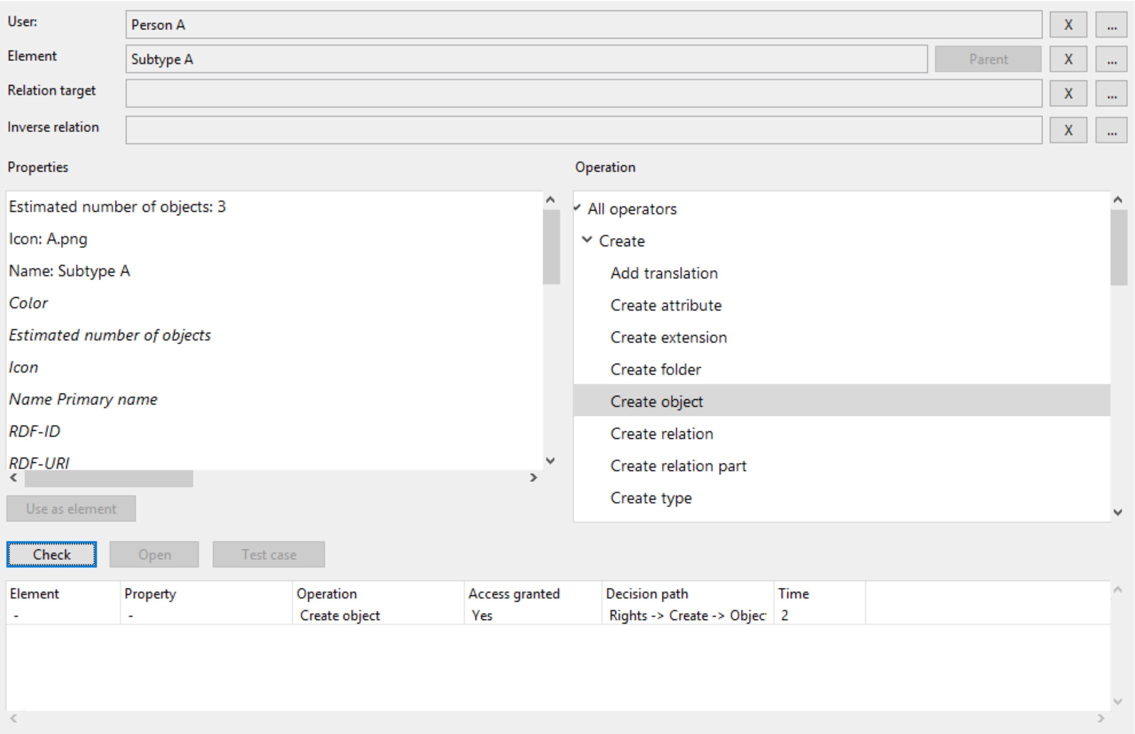
The test environment is comprised of several areas: The user and the element to which the property that is to be checked is attached is defined in the upper area. The elements can be an object, a type or a property (when this is transferred as an element).

The *properties* area lists all properties of the selected element. Non-italic properties are specific properties that are already on the object or the property. Italic properties, in contrast, are properties that can be created based on the schema, but have not yet been created. If creation of a new property is to be tested, the property in italics must be selected.

The operation that is to be tested can be selected in the *Operation* window. Depending on the parameters selected, checking rights either is possible or not.

NOTE

If a property of a property, this being a meta-property, is to be tested, then the property must be marked in the property window and the *As element* button must be selected. In the case of relations, for example, the specific relation between two objects or properties is selected as an object. All properties of the specific relation are now available in the properties window. (This can also be done with attributes.) The *Sem. element* button can be used to reverse this step.



User: Person A

Element: Subtype A

Relation target:

Inverse relation:

Properties

Estimated number of objects: 3
Icon: A.png
Name: Subtype A
Color
Estimated number of objects
Icon
Name Primary name
RDF-ID
RDF-URI

Operation

Element	Property	Operation	Access granted	Decision path	Time
-	-	Create object	Yes	Rights -> Create -> Objec	2

The result of the test is displayed in the bottom window. The *Check* button must be selected for this. The results window displays all tested cases.

- *Element* : the object, the type or the property on which the property is defined.

- *Property* : the specific property that is to be tested (is blank when italic properties are tested)
- *Operation* : that operation that is to be tested
- *Access allowed* : the result of the test in the test case
- *Decision path* : the corresponding folder which leads to the test result
- *Time* : the time required for the rights check

NOTE When testing relations, the relation, the inverse relation and the both relations halves are generally tested separately.

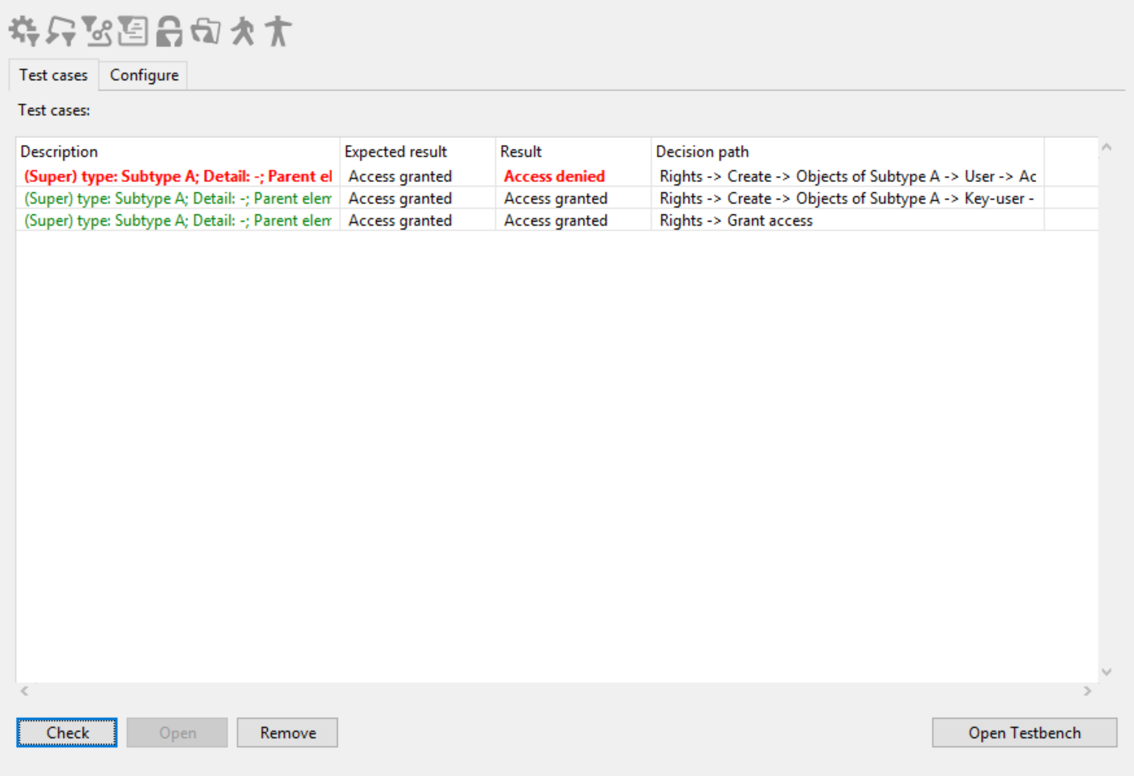
1.2.6.2. Define test cases

In order to monitor the functionality of the rights system, it is possible to save test cases. This is particularly important if changes are made to the rights system and you want to check afterwards whether the new result still matches the expected result. All saved test cases are displayed on the *Saved test cases* tab. There it is possible to check all test cases at the same time.

Instructions for defining a test case

1. In the test environment, select the element and the property you wish to check.
2. Select the operation to be tested.
3. Press the *Check* button. Now the access rights are tested for the delivered parameters.
4. In the results output, choose the test case you want to save. (You can only ever save one operation as a test case.)
5. Press the Test case button. The selected test case is saved and is available for future checks.

Test multiple test cases simultaneously



Test cases:

Description	Expected result	Result	Decision path
(Super) type: Subtype A; Detail: -; Parent el	Access granted	Access denied	Rights -> Create -> Objects of Subtype A -> User -> Ac
(Super) type: Subtype A; Detail: -; Parent elem	Access granted	Access granted	Rights -> Create -> Objects of Subtype A -> Key-user -
(Super) type: Subtype A; Detail: -; Parent elem	Access granted	Access granted	Rights -> Grant access

Screenshot with saved test cases, the second test case is displayed in red.

All test cases whose test result matches the expected test result are displayed in green. If a test case is displayed in red, the result of the check differs from the expected test result. The expected test result is determined by the fact that the check of the test case was performed for the first time during the definition of the test case. The result of this first check is displayed as the expected result during later checks of the test case. In the test system, the expected result is either *Access permitted* or *Access refused* ; for triggers, the expected result is either *Execute script* or “nothing happens” in the form of a hyphen.

Saved test cases can be deleted with *Remove* . If you want to edit a test case, you can use the *Open test environment* button to do so. In that case, all the parameters of the test case are transferred to the test environment.

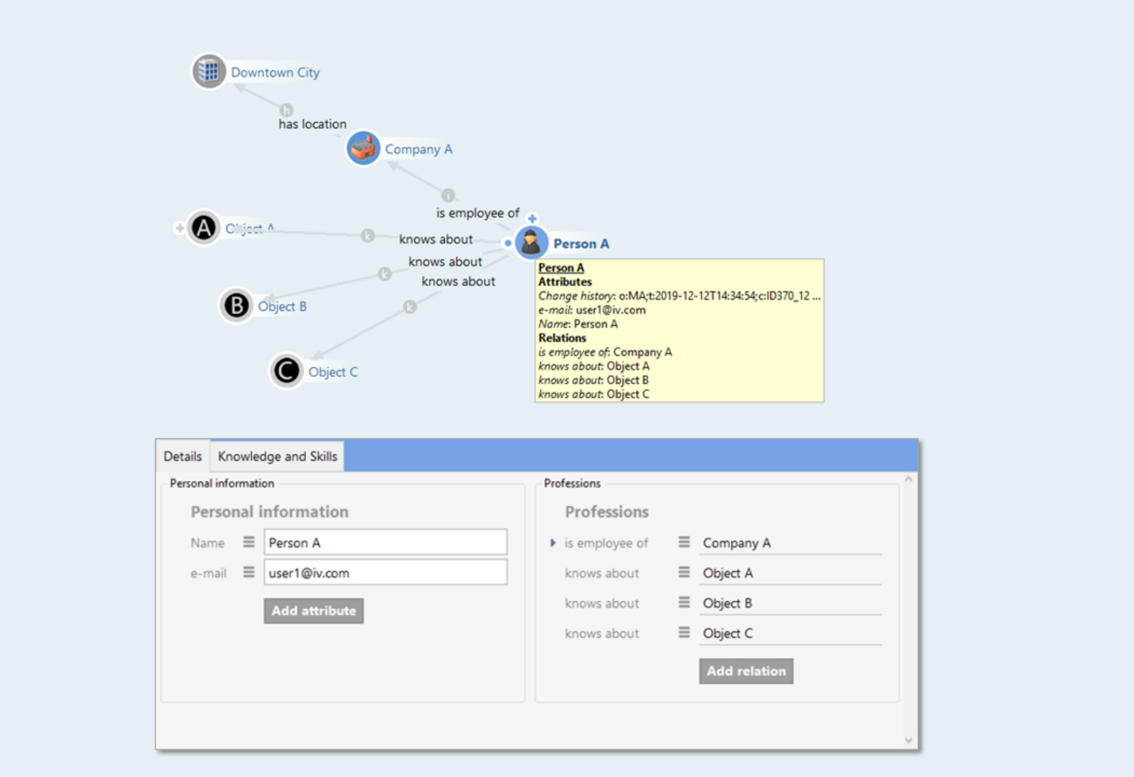
1.3. View Configuration

The view configuration makes it possible to configure various views of the data in i-views. The configured views are deployed in applications. It is possible, for example, to display sections of the Knowledge Graph or create specific compilations of data (e.g. in forms, tables, results lists etc.).

This allows us to answer the following questions, for example, and create the required views with view configurations:

- How should the properties of specific objects be displayed?
- In what order should the properties be displayed?
- When we create a new object, which attributes and relations should be displayed in such a way that they cannot be overlooked and thus not filled out?
- What should the list of objects for a type look like?
- Should it even be a simple list, or should the objects be displayed in tables?
- Which elements should be displayed in the individual columns?
- Should relation targets be displayed directly? Or only specific attributes?
- Should we define different tabs that summarize properties and attributes that go together? …

Example: Specific persons have the properties Name, Age, Gender, Address, Phone number, Email, Cell number, Fax, *knows* , *is friends with* and *is a colleague of* . Now we can use the view configuration to create more structure for the data view by defining a tab with the heading “General information”, which contains the name, age and gender; a tab with the heading “Contact data”, which contains the address, phone number, email, cell number and fax; and a tab with the heading “Contacts”, which contains the *knows* , *is friends with* and *is a colleague of* properties.



Person A
Attributes
Change history: o:MA;t:2019-12-12T14:34:54;c:ID370_12 ...
e-mail: user1@iv.com
Name: Person A
Relations
is employee of: Company A
knows about: Object A
knows about: Object B
knows about: Object C

Personal information

Personal information

Name: Person A
e-mail: user1@iv.com

Professions

Professions

is employee of: Company A
knows about: Object A
knows about: Object B
knows about: Object C

Example of a view configuration. Upper part of screenshot: Unconfigured section of an object in the graph view with all its properties. Lower part of screenshot: Configured view of the same object, where the properties that go together have been grouped, unimportant relations have been left out, and similarity relationships are displayed directly.

One special case of view configuration is the configuration of the data view in the Knowledge Builder, because the Knowledge Builder is also an application which allows various data views. This is helpful if we want to use the Knowledge Builder as a preview in order to try out specific configurations. The view configuration in the Knowledge Builder can be configured so that important properties that need to be added can be requested in a clearly visible way, for example the detail pages for objects. This is particularly helpful if data are to be collected systematically.

1.3.1. Concept

The concept of i-views is that semantic elements can be used for configuration. The views in the Knowledge Builder are generated with the help of a preset view configuration.

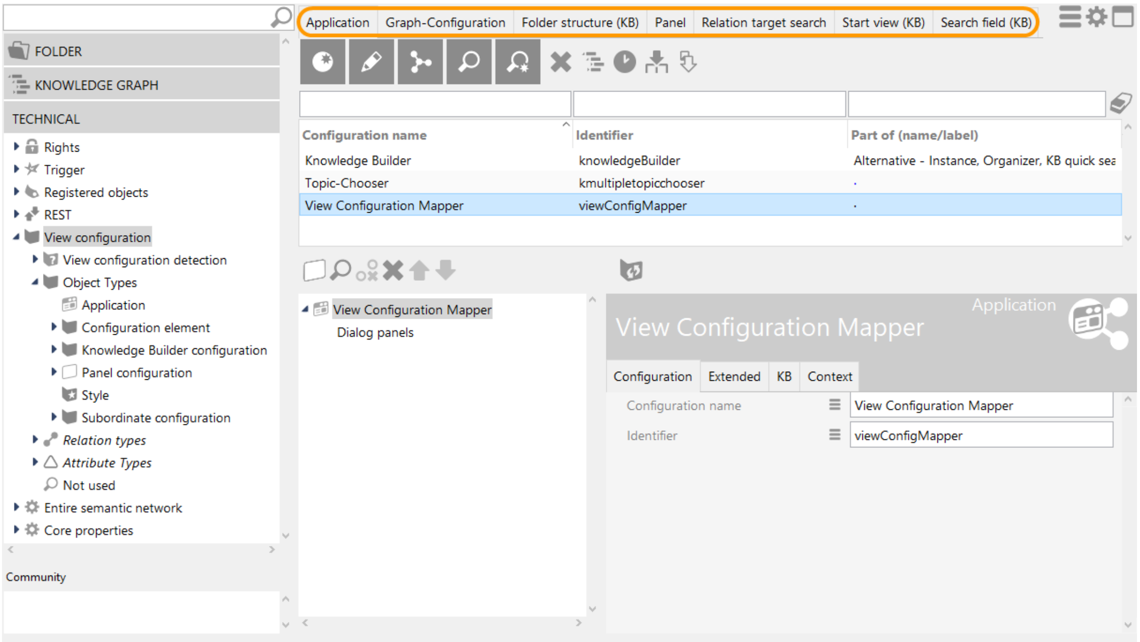
1.3.1.1. View Configuration

The purpose of the view configuration is to format the data of the Knowledge Graph for applications in such a way that it can be displayed either in Knowledge Builder or as an application in the web front-end via a bridge.

In the Knowledge Graph, special “view configurations” can thus be created for use in Knowledge Builder and for applications such as the ViewConfiguration Mapper.

The view configuration in Knowledge Builder contains the following categories:

- Applications
- Graph configuration
- Configuration of the KB folder structure
- Panel
- Relation target search
- Start view (KB)
- Search field (KB)



Configuration name	Identifier	Part of (name/label)
Knowledge Builder	knowledgeBuilder	Alternative - Instance, Organizer, KB quick sea
Topic-Chooser	kmultipletopicchooser	·
View Configuration Mapper	viewConfigMapper	·

View Configuration Mapper

Configuration name: View Configuration Mapper

Identifier: viewConfigMapper

For more information, see the “Context/using view configurations” chapter.

1.3.1.2. View Configuration Mapper

The view configuration mapper is used to map the preconfigured views of the view configuration to the web front-end of the browser.

The structure of the view configuration mapper is generally structured in hierarchical fashion and contains the panels for building the layout (= content arrangement) of the web front-end. To display the contents, a panel needs a sub-configuration, which is referred to as a “view” (= prepared content).

In concrete terms, the view configuration mapper contains one main window panel and any number of dialog panels. The main window panel reflects the entire display area of the website in the web front-end and contains the following panels, for example:

- Window title panel

- Panel with defined view
- Panel with flexible view
- Panel with linear layout
- Panel with changing layout

Please note that the view configuration mapper is a single-page application; this means it is not the visibility of panels over several pages that is controlled, but the visibility of the elements featured in the permanent panels.

1.3.1.3. Create and update the view configuration

Create

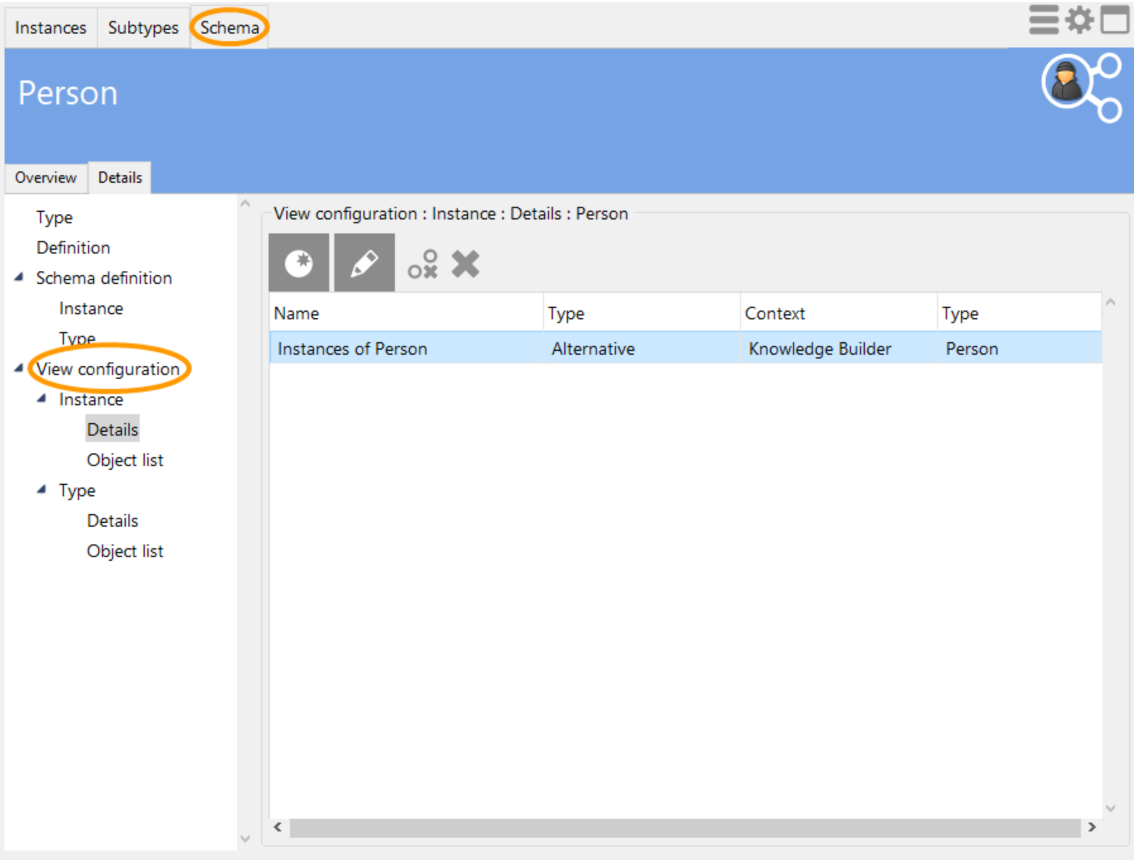
In Knowledge Builder, there are two places where you can create a new view configuration:

1. Semantic element-oriented configuration

The first place makes sense if a view configuration is to be generated for a certain object type: On the “Details” tab, you can edit the view configuration for details views and lists.

The displayed hierarchy has the sub-item “View configuration” with four additional subitems.

- Object → Details: This is where you can configure the details view for objects.
- Object → Object list: This is where you can configure the object list that shows the objects of the selected type in Knowledge Builder.
- Type → Details: This is where you can configure the details view for types.
- Type → Object list: This is where you can configure the object list of subtypes of the selected type that can be seen in Knowledge Builder.



Name	Type	Context	Type
Instances of Person	Alternative	Knowledge Builder	Person

You can create view configurations for this type or objects of this type on the objects type on the "Details" tab.

Click on "New" ✱ to create a new view configuration. For object lists you automatically create a new view configuration of the table type. For details, a dialog opens in which you can select the desired view configuration element (on this subject, see the "View configuration elements" chapter).

By clicking on the Edit button or double-clicking on the selected view configuration, open the editor with which you can configure the view.

NOTE On the "Context" tab of the respective configuration, the entry "use in" specifies in which application the configuration is to be displayed:

Application context "apply in"	Result
Knowledge Builder	The details view or the list for a type or object in Knowledge Builder is displayed.
View configuration mapper	The details view is used for the web front-end.

If there is no entry for the application context and the view does not receive an application content through inheritance from a higher-level element (view or panel), the view is not assigned and therefore deactivated.

Special case: Hierarchy + object list

A possible use case for the details view of the Knowledge Builder is to display a domain-specific hierarchy with object details. In this case, “Knowledge Builder” must be entered for the application context in the “Knowledge Builder” hierarchy view, and to configure the details, the configuration name must be entered in the hierarchy view. Assigning a different application context in this constellation can lead to an endless cycle in the view configuration.

2. View-oriented configuration

The second position presents itself if an application is to be generated from scratch the many view configurations are to be created at once. To this end, *Technical > View configuration > Object types* contains all view configuration elements that are in use in the Knowledge Graph or for which a new view configuration can be created.

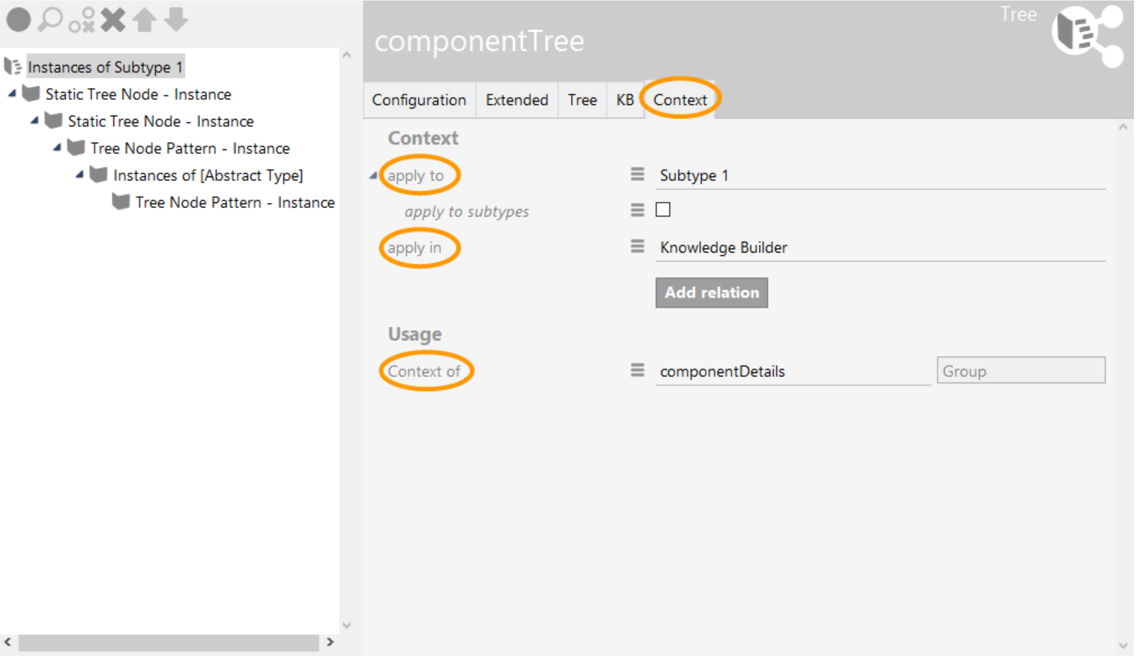
To configure a web front-end, use the panel configuration *Technical > View configuration > View configuration mapper* . For more information, see chapter 3 “View configuration mapper.”

Update

To ensure that changes to the view configuration are copied to the application, you have to update the view configuration in Knowledge Builder by clicking on the “View configuration update” button. This button is always located in the respective View configuration menu bar.

1.3.1.4. Context / Use of view configurations

The context in which a view configuration element is used is shown in the properties editor under the “Context” menu tab.



Instances of Subtype 1
Static Tree Node - Instance
Static Tree Node - Instance
Tree Node Pattern - Instance
Instances of [Abstract Type]
Tree Node Pattern - Instance
Tree
componentTree
Configuration
Extended
Tree
KB
Context
Context
Usage
Add relation

apply to: Subtype 1

apply to subtypes: □

apply in: Knowledge Builder

Context of: componentDetails | Group

Context

The context area is used to define the semantic elements for which the view configuration applies, and to define where, i.e. in which applications or in which other view configurations, it is displayed:

- **“Apply to”:** The semantic element for which the view is being used must be specified here. If the view configuration is defined by the object type, the object type is entered automatically. Additional object types can be specified as necessary

  Example: If the view is a node category of the Net-Navigator, then the object type for which the objects are shown can be specified under “Apply to.”

- **“Apply to subtypes”:** This is selected to show the type itself, and its subtypes, using the application.
- **“Apply in”:** Specifies the application context, i.e. which application (mostly: ViewConfiguration Mapper or Knowledge Builder) or configuration the view is applied in.

If no application has been entered for using the view configuration, then the view configuration will not be shown, apart from the following exceptions. View configurations are defined as a tree structure in which the principle of inheritance applies. This is why the application does not have to be specified separately for sub-configurations. They are shown as part of the top-level configuration. A property configuration is shown, for example, when this is part of a layout for which its use was specified. A view configuration is also shown when it is part of a panel which, in turn, is defined in an application.

The following applications are available from the start:

- **Graph editor:** The configurations have an impact on the display in the graph editor. The graph editor is used for visualizing the semantic elements and their relationships.
- **Knowledge Builder:** The view configurations are used in the actual Knowledge Builder. Along with the detailed configurations, the object list configurations are also available here.
- **Net-Navigator:** This is used for visualizing semantic elements. In contrast to the graph editor, which is part of the Knowledge Builder, it can be used in the Knowledge Builder and ViewConfiguration Mapper applications.
- **Topic chooser:** It allows relation targets to be selected in a window.
- **ViewConfiguration Mapper:** The ViewConfiguration Mapper is an intelligent front-end which uses view configurations. It can be used to create straightforward and fast views of the data.

Moreover, it also allows any individual applications to be defined, which can be linked to the view configuration at this point.

References

“References” refers to the reuse and continued use of a view configuration within another view configuration:

- **“Is included in panel”:** Indicates which higher-level panels there are in the view configuration hierarchy

- **“Has sub-panel”:** Indicates which panels there are in subordinate hierarchy levels
- **“Order”:** Determines the order of the panel when the higher-level panel has a linear layout (horizontal or vertical)
- **“Sub-configuration”:** Refers to a subordinate configuration that contains the view (= specific display of the content)
- **“Activate actions from panel”:** Indicates that an action in this panel is influenced by the action in another panel (for example: Display of the search result in one panel is influenced by the search input in another panel)
- **“Show result from action”:** Determines that the action by another panel causes a result to be displayed in a defined form in this panel (for example: Net-Navigator shows the elements for the object that was clicked in another panel’s search result field)
- Other relations (“Table of”, “Context of”, “Configuration for meta properties of”, “Action of”, etc.) show the contexts in which a view configuration is used. A view configuration can be used in any number of view configurations.

1.3.1.5. The validity of view configurations

The chapter *Using the view configurations* already noted that the application in which and the objects and types for which the view is displayed are decisive for view configurations. Nonetheless it is possible that the view configuration is not displayed in the selected application. This question is: When is a view configuration valid? And for which object or type is the view configuration valid?

Inheritance of view configurations

In relation to inheritance, view configurations respond like properties. Subtypes or objects of subtypes inherit view configurations.

Application of the most specific view configuration

The subtypes use the super-types according to the inheritance principle as long as they don’t have their own view configurations. The most specific view configuration is always used: This is the configuration that is defined directly on the type. If that is not the case, it is checked whether there is a view configuration on the super-type. If that is not the case, the next level up in the type hierarchy is checked to determine if a view configuration has been defined. The view configuration that is closest to the object type is then used. If no view configuration is found on the super-types, the default configuration is used for the administrators.

What happens when there are two equivalent view configurations?

If there are two equivalent view configurations, no view configuration is displayed. If the application or object type was not defined for one of the view configurations, this is not considered to be an active view configuration. In this case the other view configuration is used. If you want to display different views for different users, you can define a rule in the detector system. In this case, the view configuration is used in accordance with the defined rule as long as the rule only has one view configuration dependent on the user.

1.3.2. Menus

Menu configurations contain buttons, so-called *actions* , which allow the user to execute a range of functions.

The menus mainly serve two functionalities in the handling of actions. On the one hand, they can be used to structure actions, and on the other, they can be used to specify where the menus are deployed. The Knowledge Builder and ViewConfigMapper contain many locations where the contents of menus are displayed, for example buttons at the head of an editor, or the context menu for an individual property. Currently it is not yet possible to apply menus to all places where menus are theoretically possible.

The next section describes the direct setting options for a menu, as well as the existing menu types and how to use them.

Name	Value
Label	The menu type and the interface handling the display determine whether the label is displayed.
Replaces standard menu	This parameter currently only affects the Knowledge Builder. Some editors, e.g. for a table, display standard menus. These can be switched off with the help of this parameter.
Menu type	The menu type describes the use of the menu in the individual components. The menu types are described further down.

Menu types:

Menu bar

Name	Value
Add standard actions	This icon is only displayed as an entry of the context menu if standard actions can be added, currently for menus of tables and search configurations.The standard actions are applicable for the Knowledge Builder view configuration only and comprise the actions provided as in object list menus:

✸	New
✎	Show (Edit)
⋔	Display graphically
🔍	Search
✖	Delete
🕒	Recently accessed objects
⇩	Refresh view
☰	Show in tree
🖨	Print

Note

- If the parameter *Replaces standard menu* is not set, the actions that are not included in the menus are appended sequentially.
- If the order of the standard actions is supposed to be changed, the parameter *Replaces standard menu* must be set. Following that, standard actions can be added using the *Add standard actions* action. The standard actions can now be sorted in any way you wish and mixed with your own actions.

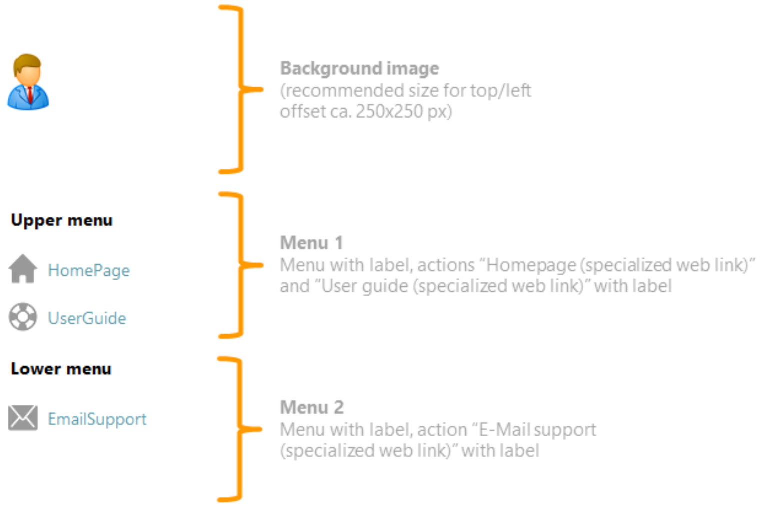
Context menu

Icon	🗐
Knowledge Builder	Currently it is possible to expand or define context menus for a table row and an object editor. **Object configuration:** You can use the *Menu* tab to create menus in any top configuration of an element. You can also switch off the standard menu here by setting the *Replaces standard menu* parameter. **Table configuration:** The context menu contains two sections for a table row. The first relates to the selected element, the second relates to the table. There are two different configuration locations for the two sections. For the first case, the menu for an element must be linked to any configuration, ideally a new one, which in turn is attached via *Apply in* to the table that is to display the context menu. In the second case, the menu can be attached directly to the table.
ViewConfigMapper	*This is currently not used in the ViewConfigMapper.*

JSON	`"label" : “Menu (context)”,`
	`"actions" : [{...}],`
	`"type" : "contextMenu"`

List

Icon	⋮

Knowledge Builder	This is only used in the start screen configuration. The configured actions are displayed in a list. If labels are assigned for the menus, these are also displayed and therefore offer a structuring option.  Upper menu HomePage UserGuide Lower menu EmailSupport Background image (recommended size for top/left offset ca. 250x250 px) Menu 1 Menu with label, actions “Homepage (specialized web link)” and “User guide (specialized web link)” with label Menu 2 Menu with label, action “E-Mail support (specialized web link)” with label
ViewConfigMapper	*This is currently not used in the ViewConfigMapper.*

JSON	`"label" : “Menu (List)",`
	`"actions" : [{...}],`
	`"type" : "listMenu"`

Toolbar

Icon	⋯
Knowledge Builder	The actions contained in the menus are added in sequence. Subdivision by menus and labelling of menus are currently not considered.
ViewConfigMapper	The actions contained in the menus are added in sequence. Subdivision by menus and labelling of menus are currently not considered.

JSON	`"label" : “Menu (toolbar)”,`
	`"actions" : [{...}],`
	`"type" : "toolbar"`

1.3.3. Actions

The actions in i-views are divided into preconfigured action types. These action types are categorized as follows:

- Universal actions (can be used in knowledge and ViewConfiguration Mapper)
- Actions specific to Knowledge Builder
- Actions specific to ViewConfiguration Mapper
- Internal actions (for administrative use only)

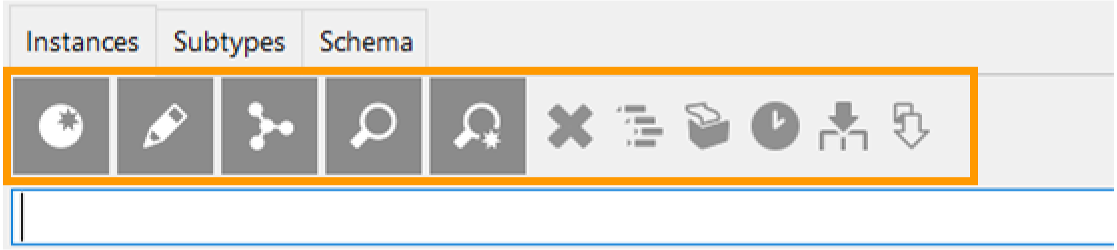
Depending on the action type and application, additional configurations are required, for example creating additional panels for displaying the results of an action.

1.3.3.1. General

Functionalities can be specified in the view configuration using actions.

All the configured actions are displayed in the Knowledge Builder as additional buttons. The script contained in the action is executed when the respective button is clicked.

The actions configured are generally displayed as buttons in the Knowledge Builder or in the web frontend (by means of the ViewConfiguration Mapper). Actions can be summarized in a menu, or be defined directly for a view configuration.



Standad actions on an instances list

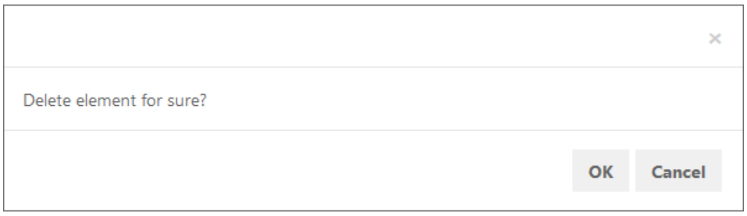
The label is displayed as a tooltip in the Knowledge Builder. The selected symbol (any image file) is scaled to the size of the button.

NOTE	If no symbol is specified, no button is displayed in the Knowledge Builder. For the web frontend, a label or an icon is needed at least. Actions of any type can be attached at a wide range of positions. In most cases, they are also displayed. There is no guarantee that this action can be executed in the content in which it is currently being used. The application area for the actions (Knowledge Builder or web frontend) is described in detail in the following chapters.

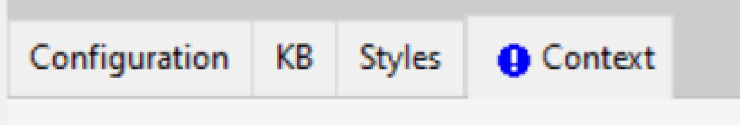
Setting options

Name	Value
Configuration	

Name	Value
Configuration name	The configuration name serves for identification and reuse of the configuration element.
Label	A label can be defined for the button for the action here.
Script for label	A script can be used to specify the button label. This option is only available when no label is specified.
Bookmark path	Bookmark path can be selected or created here. The displayed name is used as path pattern in the same time. The path pattern is used for path pattern construction of the bookmarking resource. For detailed information, see chapter about bookmarking ("Bookmarks and Resource").
Action type	The type of action. The different types are explained further down. A script overwrites the action defined by the action type. Dependent on the action type, only certain types of script might be available. **NOTE** When switching the action type, scripts which are not applicable anymore will be removed; is the script is unregistered, it will be deleted. A dialog informs about the consequences beforehand.
Script (custom)	The script that is to be executed for this action. The script is allowed to modify elements of the knowledge graph and defines the action result.This script is available if one of the following action types has been selected: • Choose relation target • Script • Selection
Script (deprecated)	The script that is to be executed for this action. Deprecated → use "Script (Custom)"
Script (before action)	This script is available only if the action type "Save" has been selected.
Script (ActionResponse) (VCM)	A script specified here executes a so-called *ActionResponse* after the action. This script must not be used for standard VCM-views. Not available for all action types.
Script (after action)	This script is available only if the action type "Save" has been selected.
Script (recall)	
perform by	A view role can be selected or defined here.

Name	Value
Question before execution	For web frontend only. A text can be specified here which is to be shown to the user in a dialog box before the action is executed. The dialog provides the option of canceling or continuing the action (Ok/Cancel/Close).  Delete element for sure?
Script for question before execution	A script can be used here to determine the text for the confirmation dialog for the action. **Caution:** • If a blank string is returned, the dialog does not appear. • If an error occurs within the script, the dialog won't appear as well.
Transaction	This option is only needed for editing purposes in the web frontend: By means of the transaction **begin** , a temporary state/element can be memorized until another action ends the transaction using the transaction **commit** . **Example:** Creating temporary elements in a dialog which then can be written permanently into the Knowledg Graph by means of the acton type " **Save** " and the transaction type " **commit** " or rejecting the creation by means of the action type " **Abort** ", without a transaction type.
Display	
Script (enabled)	A script can be used here to determine whether the button for the action is to be activated, and should therefore be able to be executed.
Script (visible)	A script can be used here to determine whether the button for the action is to be displayed (return value "true" for visibility, "false" for invisibility).
Icon	Icon in forms of a bitmapped graphic which can be selected here that is to be displayed on the button for the action. For the web frontend, vector graphics can be used as well. An action needs at least an icon or a label to be visible in the web frontend.
Tooltip	The content of the tooltip (= mouse-over text) for the action can be defined here, instead of using the text of the label.
Script for tooltip	A script can be used here to determine the content of the tooltip (= mouse-over text) for the action, instead of using the text of the label.

Name	Value
After execution (action)	
Notification	Text shown in a notification that appears after the action.
Script for notification	A script can be used here to determine the content of the notification.
Notification type (VCM)	As a metaproperty of the notification or the script for notification, the notification type can be set to for different message colors in the web frontend: • "Success" (green message) • "Information" (blue message) • "Warning" (yellow message) • "Error" (colorless message)
After execution (panels)	
Show result in panel (VCM)	A panel in which the result of the action is to be displayed.
Activation mode (VCM)	See chapter "View Configuration Mapper" (3.2.1.2)
Script for activation (VCM)	In general, an action can be (re-)used to show content in diverse panels. The script for activation defines, if the respective panel will be activated for showing the content after execution of the action or not - by returning a Boolean value. If no script is used, the panel will be activated in every case. **Example:** The save action of an edit dialog is configured to initiate the creation of a new object. Depending on the type of the recently created object, the new object will be displayed either in sub panel **A** or in sub panel **B** of a flexible layout panel.
Script for target model (VCM)	A script can be used which context / semantic element is to be passed on to the following view after execution of the action.
Close panel (VCM)	Applicable to dialog panels only. After execution of the action, the panel is automatically closed.
KB	
Action type	The action type that is only applicable when using the action within the Knowledge Builder and not for the web frontend.
Use original position	
Styles	
Styles can be used in different ways to influence the appearance of the button or the behavior of the button. See respective chapter.	
Context	

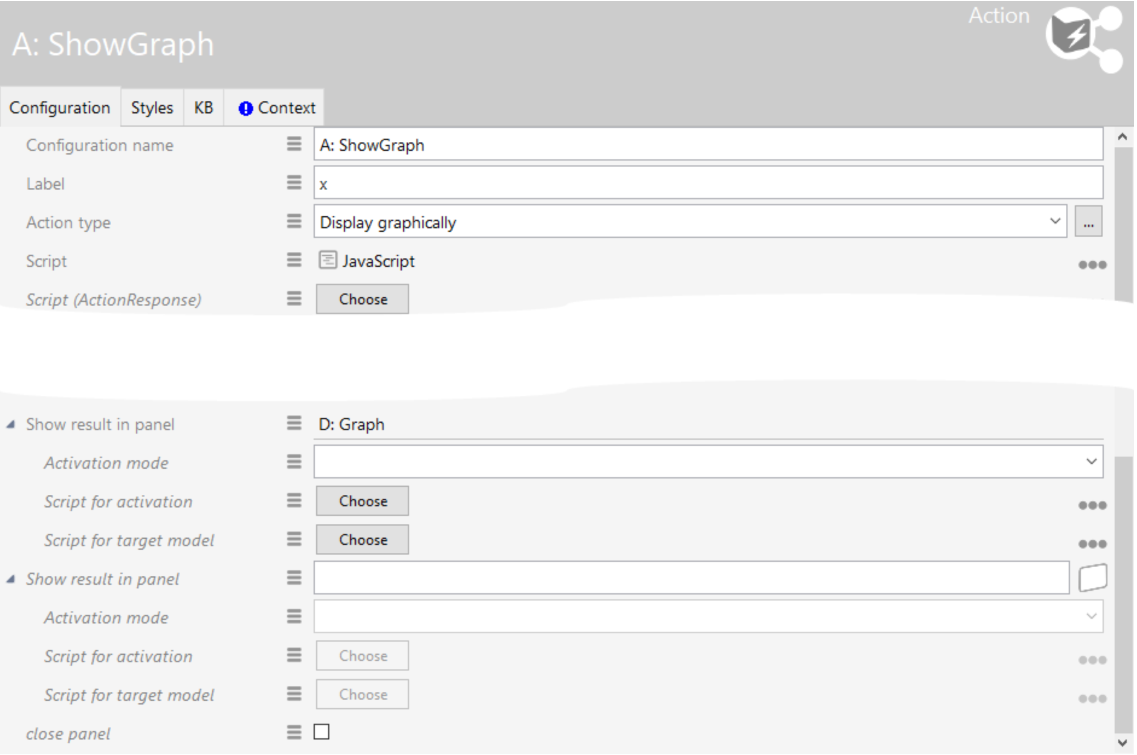
Name	Value
Action of	Describes in which menu the action is currently used. An action can be (re-)used in different menus.
Sort order	Describes the position of the action within the superordinate menu.
Notice	Tells e. g. whether the action is used in more than one configuration. In this case, a blue sign with an exclamation mark appears nearby the context tab: 

1.3.3.2. Universally applicable actions

Universally applicable actions can be used in both the Knowledge Builder and in the web frontend using the ViewConfiguration Mapper. This includes the action types “Display graphically”, “Delete” and “Search”.

1.3.3.2.1. Action type "Display graphically"

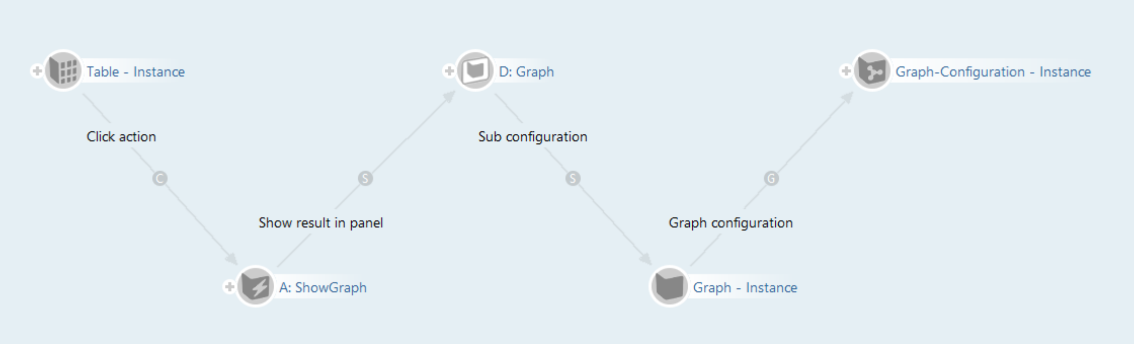
The “Display graphically” action is used in a view configuration to graphically depict object types, relations and objects in the Net-Navigator. Here the configuration is as follows:



A: ShowGraph

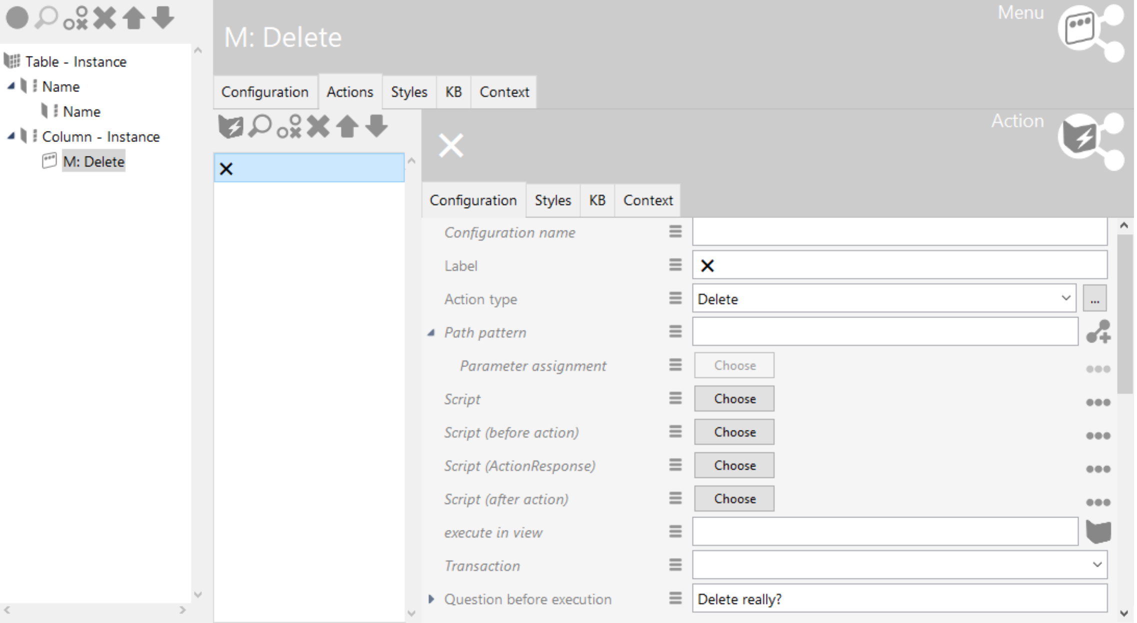
Configuration name	A: ShowGraph
Label	x
Action type	Display graphically
Script	JavaScript
Script (ActionResponse)	Choose
Show result in panel	D: Graph
Activation mode	
Script for activation	Choose
Script for target model	Choose
Show result in panel	
Activation mode	
Script for activation	Choose
Script for target model	Choose
close panel	□

For this purpose, a panel must be specified under “Show result in panel” that contains a graph object as its sub-configuration. The graph object in turn must contain a graph configuration for the definition of the elements to be displayed:



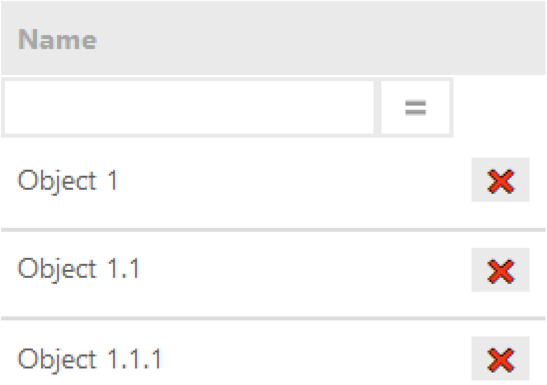
1.3.3.2.2. Action type "Delete"

This action type deletes the respective element.

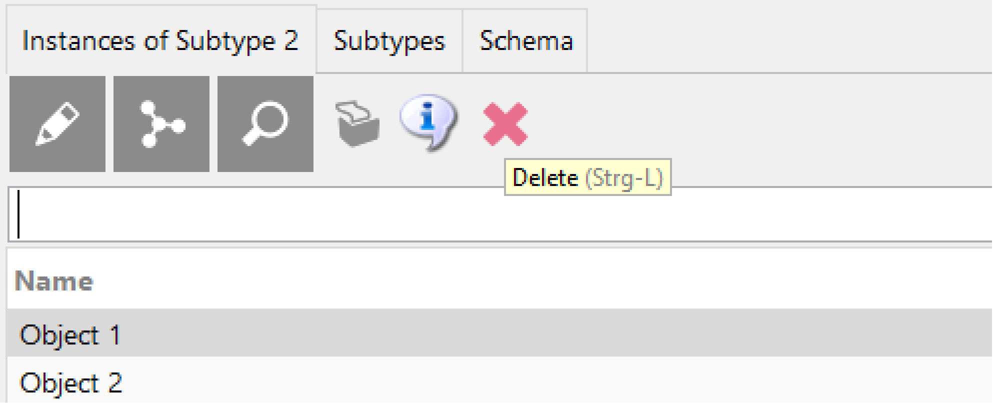


Configuration name	
Label	×
Action type	Delete
Path pattern	
Parameter assignment	Choose
Script	Choose
Script (before action)	Choose
Script (ActionResponse)	Choose
Script (after action)	Choose
execute in view	
Transaction	
Question before execution	Delete really?

For view configuration in the web frontend, the delete action deletes the respective accessed element. For example, a delete action in a menu in the second column element of a table results into a button shown at each row, leading to the row element being deleted when clicked onto.



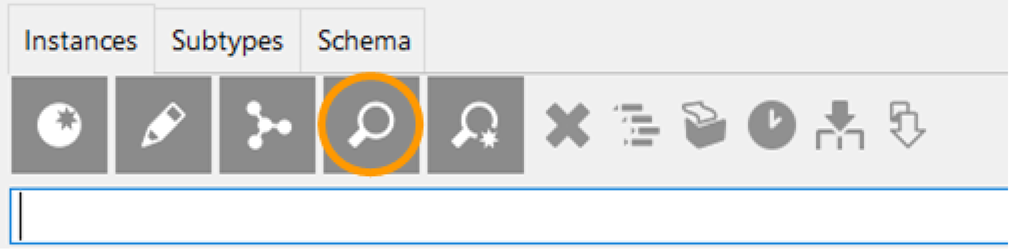
In Knowledge Builder, the "Delete" action type is preconfigured for object lists:



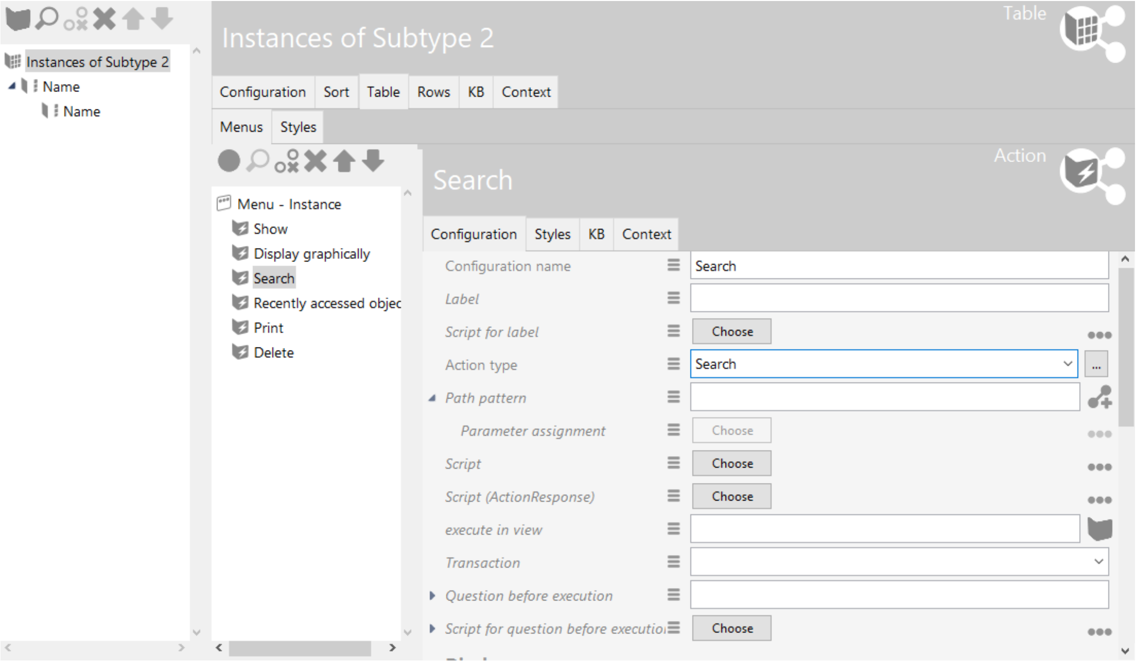
Like any other configuration in Knowledge Builder, the default configuration can be replaced with a customized configuration, containing the specified "Delete" action type.

1.3.3.2.3. Action type "Search"

This action triggers a search. This function has been integrated into the menu bar of object lists in the KB (shortcut Ctrl + S):



When used for the configuration of the web front-end, the action is assigned to an action by means of the drop-down menu under the entry “Action type:”



Configuration name: Search
Label:
Script for label: Choose
Action type: Search
Path pattern:
Parameter assignment: Choose
Script: Choose
Script (ActionResponse): Choose
execute in view:
Transaction:
Question before execution:
Script for question before executio: Choose

Tip:

- If a search function with string input (keyword search) is required for the web frontend, then the search field element or the query element in the view configuration mapper can be used. An input line and search button are preconfigured for search field view and query view as well. For the search field view, the search result can be displayed by means of a search result view which is influenced by the search field view.
- Furthermore, a "Query" view can be used which combines search input field and search result view into one element. As long as the search input field is not required to be situated apart from the search result, using this view is recommended.

1.3.3.3. Actions for the Knowledge Builder

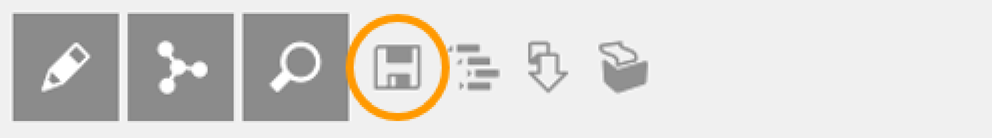
These action types can only be used for configurations in the Knowledge Builder.

NOTE

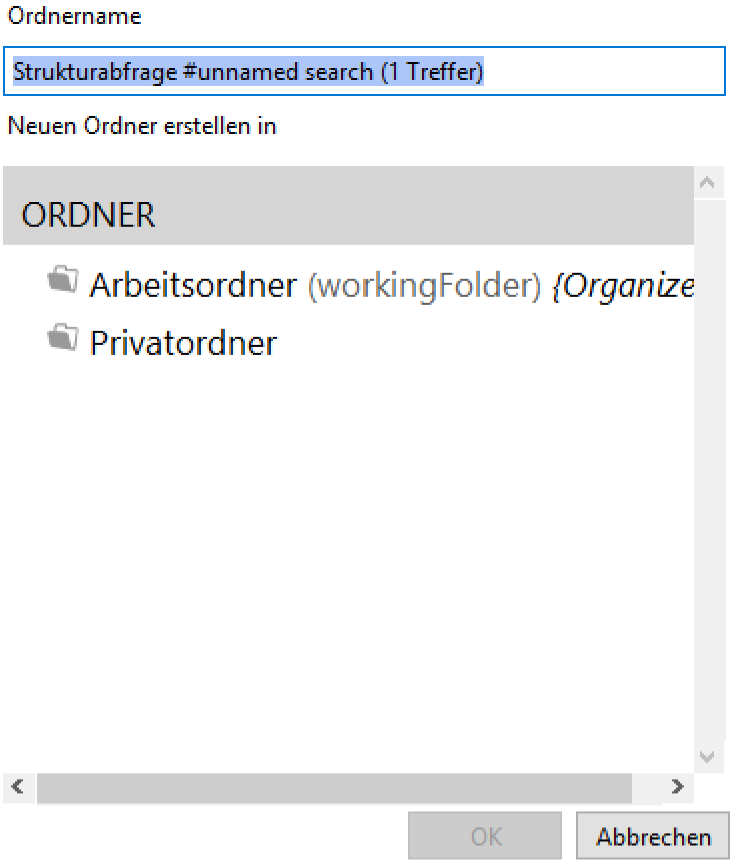
The KB-specific action types are only available in the “KB” tab of an action from KB version 5.2.2 or higher. Since these action types are all used per default for object lists and the Knowledge Builder start page nevertheless, they mainly are for configuring menus with a reduced amount of actions or for completion of customized actions by additionally using the standard action types.

1.3.3.3.1. Action type "Save query results"

If searches are executed in the Knowledge Builder by means of a structured query, you can save the results by clicking the button in the menu bar:



This action saves the query result in a folder you can choose:



Ordnername

Strukturabfrage #unnamed search (1 Treffer)

Neuen Ordner erstellen in

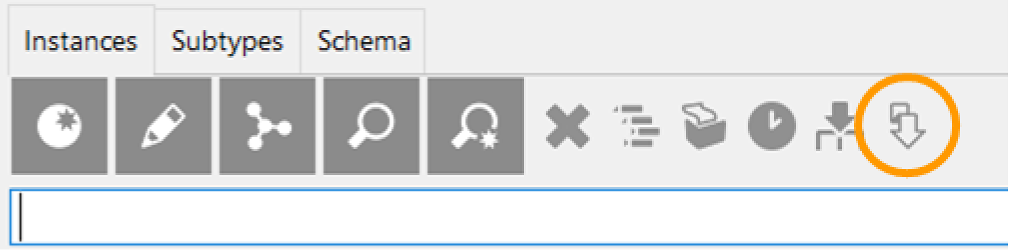
ORDNER

- Arbeitsordner (workingFolder) {Organize
- Privatordner

NOTE The saved search is an object list based on the configuration of a structured query relating to currently existing semantic elements. If changes are made to the relevant elements after the search result has been saved, this will have an effect on the saved results as well: When the relevant element is deleted, it no longer exists in the saved search result.

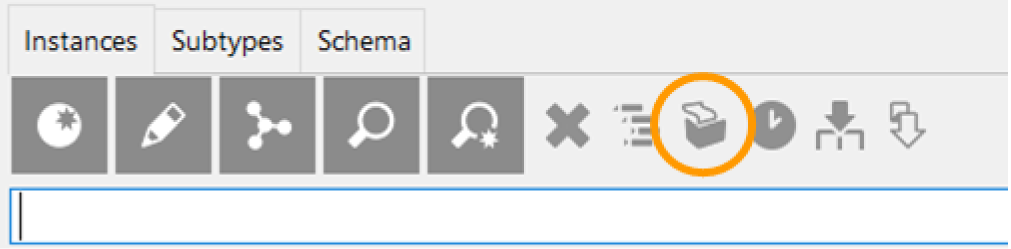
1.3.3.3.2. Action type "Refresh view"

In the KB, an action with the action type "Refresh view" recalculates the visible content of table cells. This option provides a preconfigured action that is available via the “Update” button in the object list menu bar (shortcut: F5).

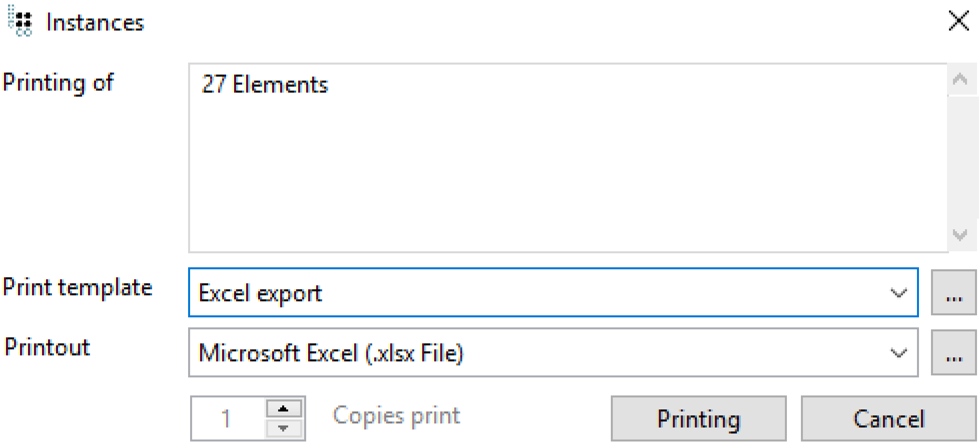


1.3.3.3.3. Action type "Print"

This action is used in the menu bar of list views. The preset configuration can be used to print out object lists or output them in an Excel table, without having to create an export mapping.



The “Print” action opens the Print dialog in Knowledge Builder.


Instances

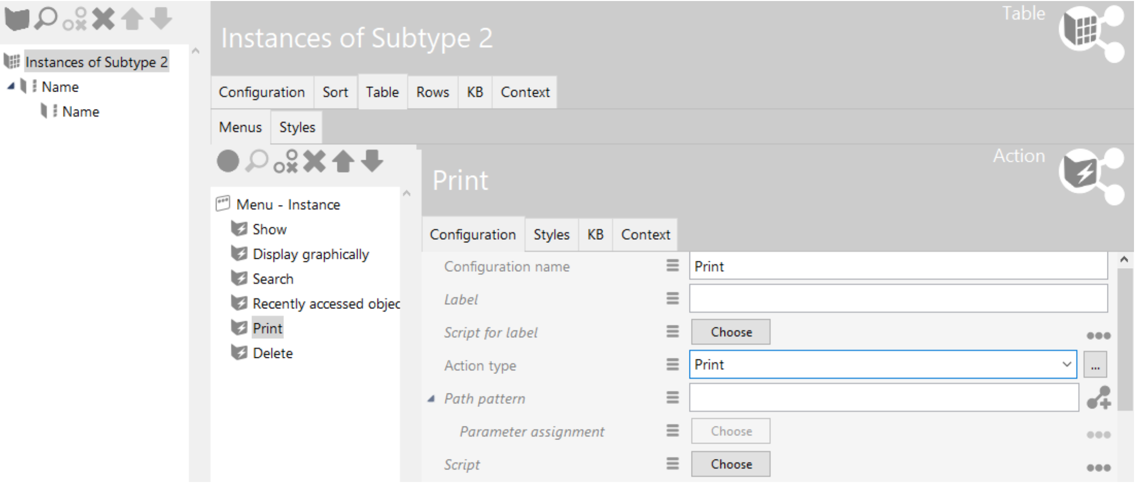
Printing of: 27 Elements

Print template: Excel export

Printout: Microsoft Excel (.xlsx File)

1 Copies print

The Print action is also available in the results lists of structured queries. When configuring individual views in Knowledge Builder, the action must be added to the respective view or configuration element:



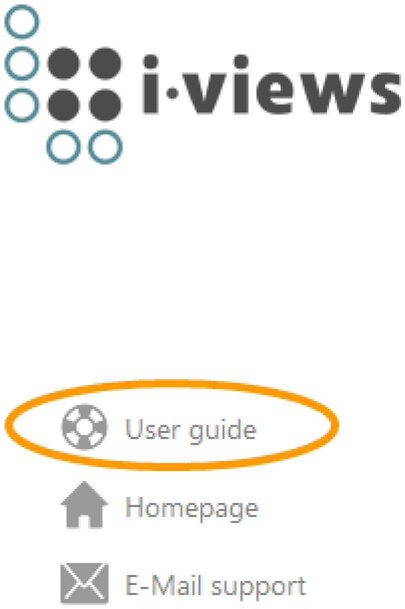
Configuration name: Print
Label:
Script for label: Choose
Action type: Print
Path pattern:
Parameter assignment: Choose
Script: Choose

The prerequisite for being able to use the action type “Print” is that the Printing component exists, which can be installed retrospectively via the Admin tool if necessary.

The configuration of the printing component is available within the "TECHNICAL" part of the Knowledge Graph. There, printing templates can be defined using document templates. For more information, see the respective chapter "Reports and printing".

1.3.3.3.4. Action type "User guide"

The action type "User guide" provides a preconfigured action that opens the i-views web manual in the browser.



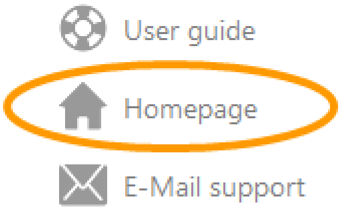
In contrast to the “Web-link” action type, this is a link to a preconfigured address, like the “Homepage” action type.

Setting options

Name	Value
URL	Preconfigured weblink to the i-views manual.

1.3.3.3.5. Action type "Homepage"

This action type can be used for the start view of the KB. The home page is opened in the browser.

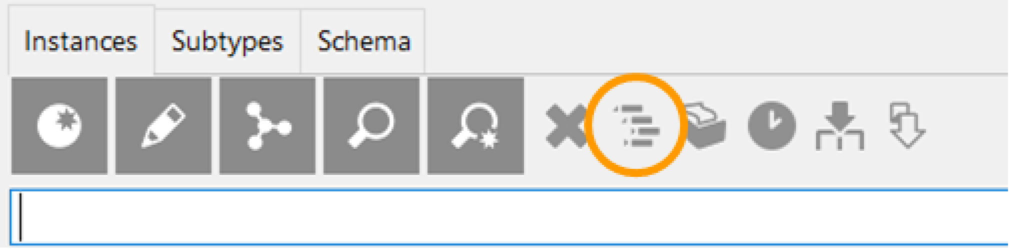


Setting options

Name	Value
URL	Link to a website

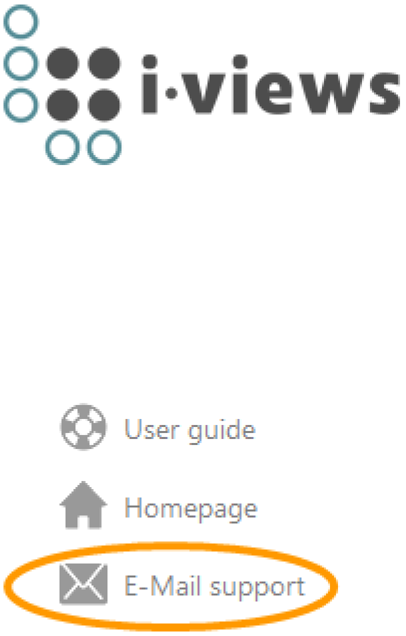
1.3.3.3.6. Action type "Show in tree"

The Show in tree action can be used to display the location of an element from the Knowledge Graph. Executing this action has the effect that the location of an element (e.g. an entry in a list view) appears at the corresponding point in the structure tree of the organizer (left column of the KB) and opens in the details view of the element.



1.3.3.3.7. Action type "E-Mail support"

This action type can be used for the start view of the KB. The actions contained open a dialog in which you can send an email to the configured address.



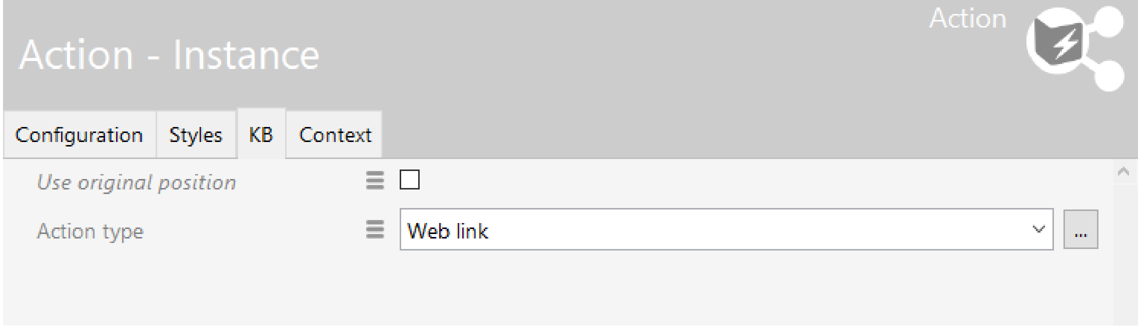
Setting options

Name	Value
URL	Email link

1.3.3.3.8. Action type "Web link"

The “Web link” action type can be used for the start view of the KB. It differs from the “Homepage”, “E-Mail support” or “User guide” action type in that way that you can assign any web address as the hyperlink.

NOTE	In later KB versions (KB 5.2.2) the “Web link” action type is only available on the “KB” tab - see following picture.



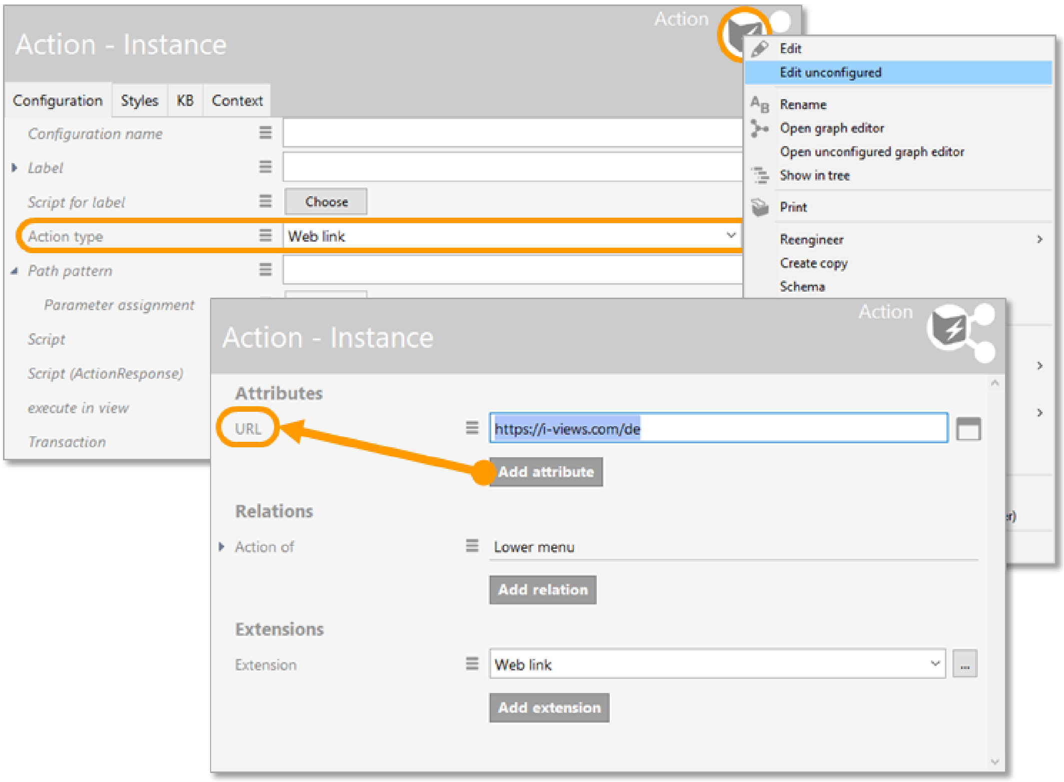
Use original position ☐

Action type: Web link

Setting options

Name	Value
URL	Address of the web link.

NOTE If the URL attribute is not displayed, it can be added by editing the action in an unconfigured editor view.



Action type: Web link

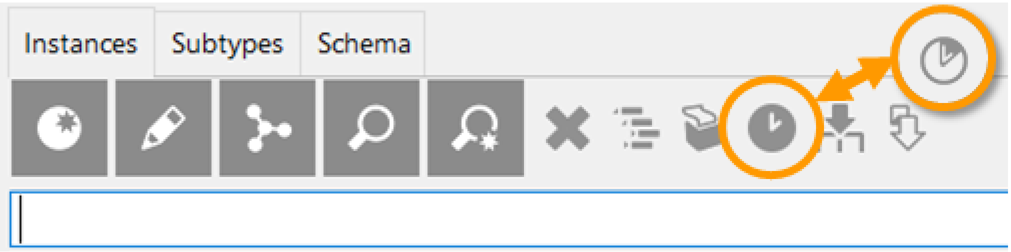
URL: https://i-views.com/de

Action of: Lower menu

Extension: Web link

1.3.3.3.9. Action type "Recently accessed objects"

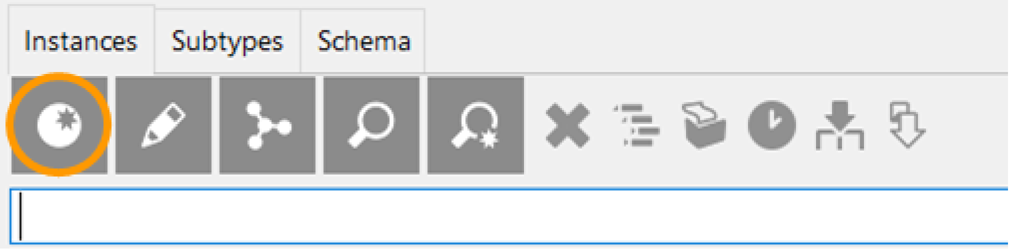
Shows the objects (semantic elements) that were last used in the respective table. Objects might be filtered depending on the definition of the table.



In Knowledge Builder, this action is preconfigured for list views and can be called up using the key combination Ctrl+R.

1.3.3.3.10. Action type "New"

The new action creates new types or new objects in the Knowledge Graph. The new action is, for example, used in the menu bar of object lists in the Knowledge Builder.



NOTE For the web frontend, a script must be used instead of the action type "New". For more information, see the chapter “JavaScript-API”.

1.3.3.4. Actions for the viewconfiguration mapper

The actions for the ViewConfiguration Mapper can only be used for the web front-end and are split into different action types.

1.3.3.4.1. Action type "Cancel"

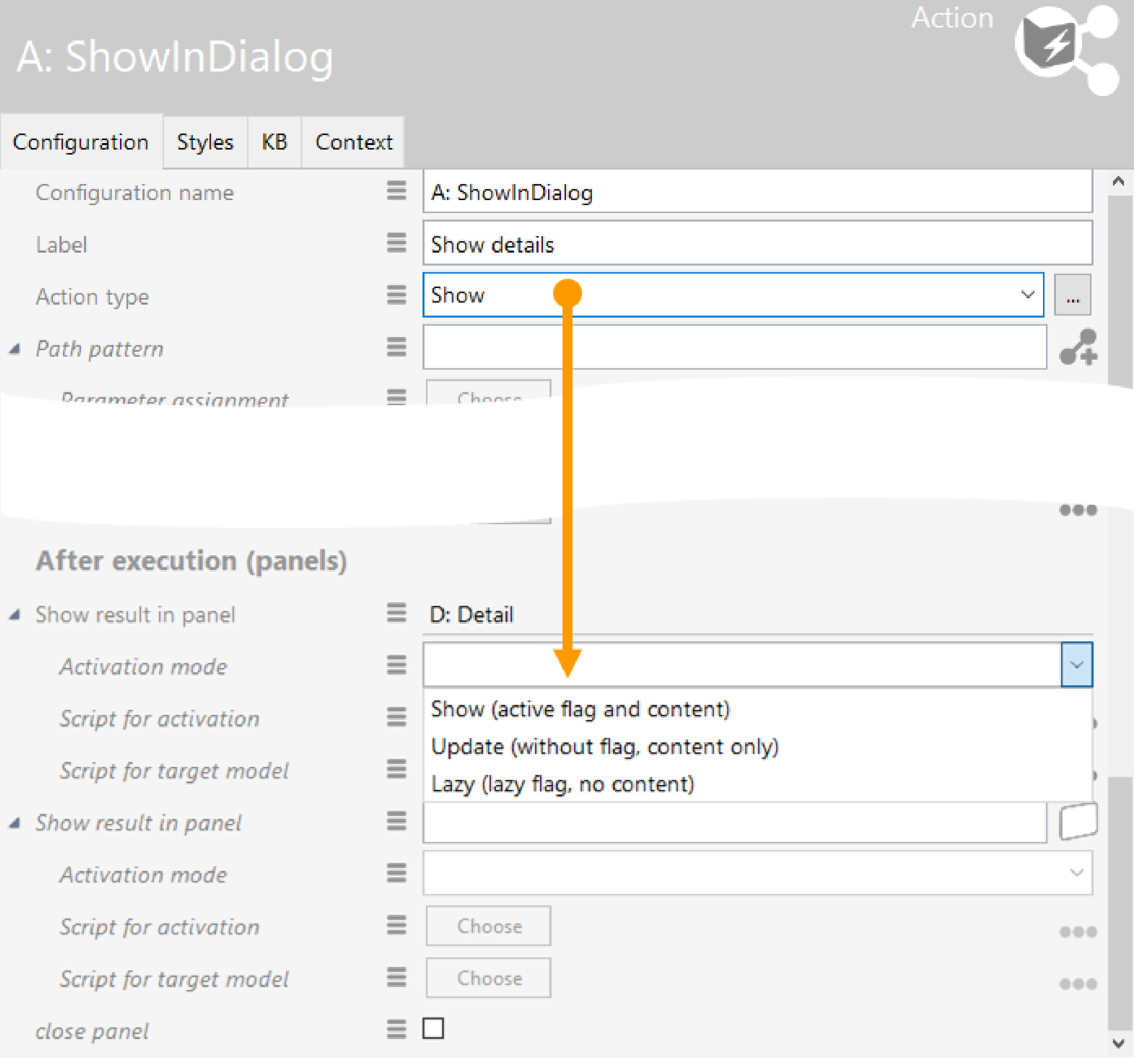
The action type "Cancel" is used in the web frontend to cancel a started transaction.

Example: A menu action with the option "transaction: begin" is configured to create a temporary object for displaying it in a dialog. A subsequent action with the option "transaction: commit" (mostly in combination with the action type "Save") completes the transaction and persists the object, whereas an action of the action type "Cancel" cancels the transaction and rejects the temporarily created object.

1.3.3.4.2. Action type "Show"

This action initiates a re-calculation of a suitable view for the semantic object that is the target of the action. You typically use this action if you want to change the view. The result of the action is the new view.

You can use “Show result in panel” to determine in which panel the view is to be displayed.



A: ShowInDialog

Configuration name: A: ShowInDialog

Label: Show details

Action type: Show

Path pattern

Parameter assignment

After execution (panels)

Show result in panel: D: Detail

Activation mode

- Show (active flag and content)
- Update (without flag, content only)
- Lazy (lazy flag, no content)

Script for activation

Script for target model

Show result in panel

Activation mode

Script for activation: Choose

Script for target model: Choose

close panel □

The “Activation mode” determines the update behavior of the view:

Activation mode	Description
Default	The target panel is activated (= made visible) after the action execution, regardless of whether it was activated before the action or not. If the target panel is connected to other panels with an "influences" relation, those panels will be activated as well and provided with new content. The action result will be the new model of the target panel and its influenced panels. This mode is useful e.g. for showing a dialog panel.If there is no configured activation mode, this mode will be used as default.

Activation mode	Description
Refresh view only	The target panel is only refreshed, if it was already visible before the action execution. Furthermore, no additional panels will be activated through "influences" relations. The action result has no influence on the panel’s content: the same model will be shown, but the view may change due to side effects of the action (e.g. because a query now yields more results or a shown object received new properties).
Refresh model and view	The target panel is only refreshed, if it was already visible before the action execution. Furthermore, no additional panels will be activated through "influences" relations. The action result becomes the new model of the target panel and replaces the previous model.

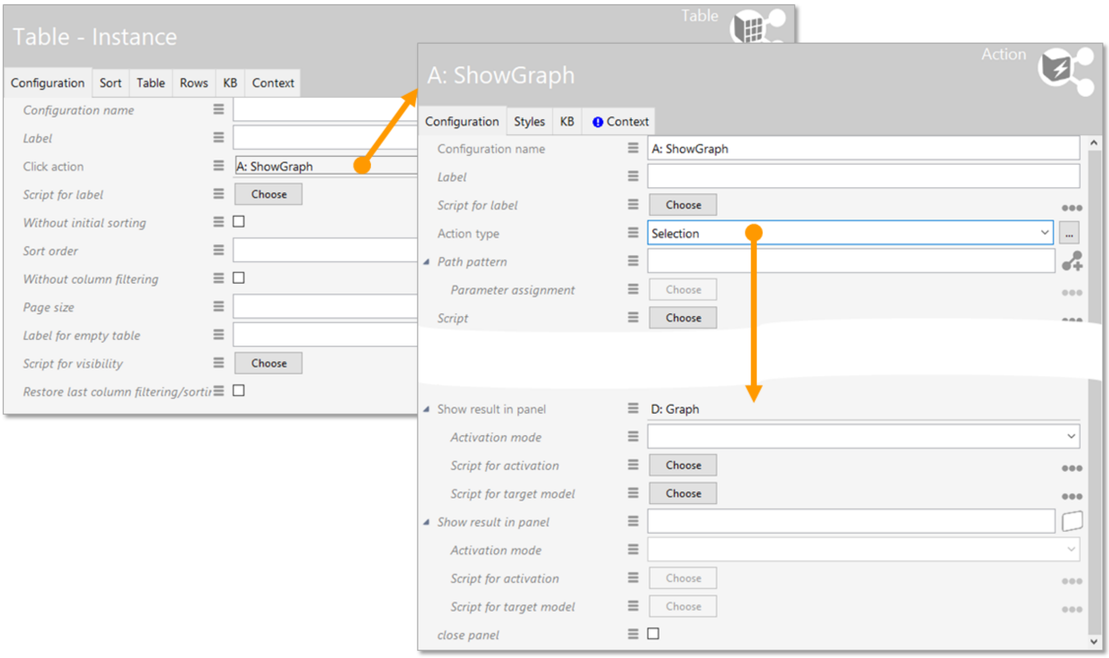
1.3.3.4.3. Action type "Selection"

This action corresponds to the “Display” action, with the only difference being that the action is executed on the parameter “selectionElement,” i.e. on a selected element.

NOTE This effect also applies to any script that might be available.

The “Selection” action is used only (but not necessarily) in order to call up a display from another panel when clicking on a table entry or list entry in a search result. This is often used to display detailed information on a semantic element.

Example



Click action: A: ShowGraph

Configuration name: A: ShowGraph

Action type: Selection

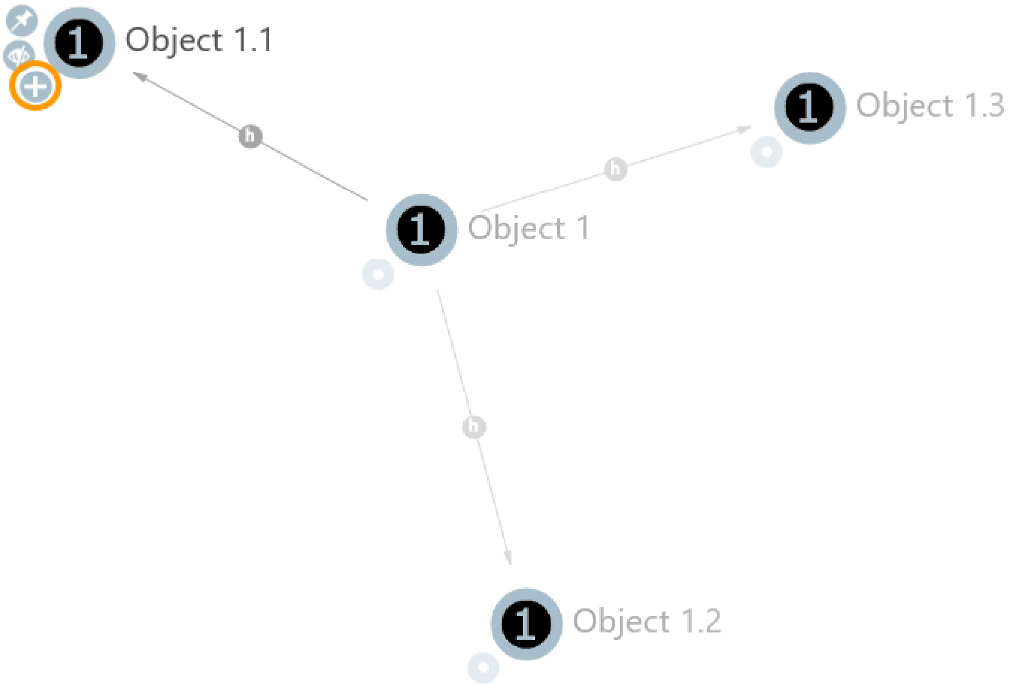
Show result in panel: D: Graph

Keep in mind that the respective “Selection” action specifies the panel that this action is supposed to affect. This is specified under “Show result in panel.”

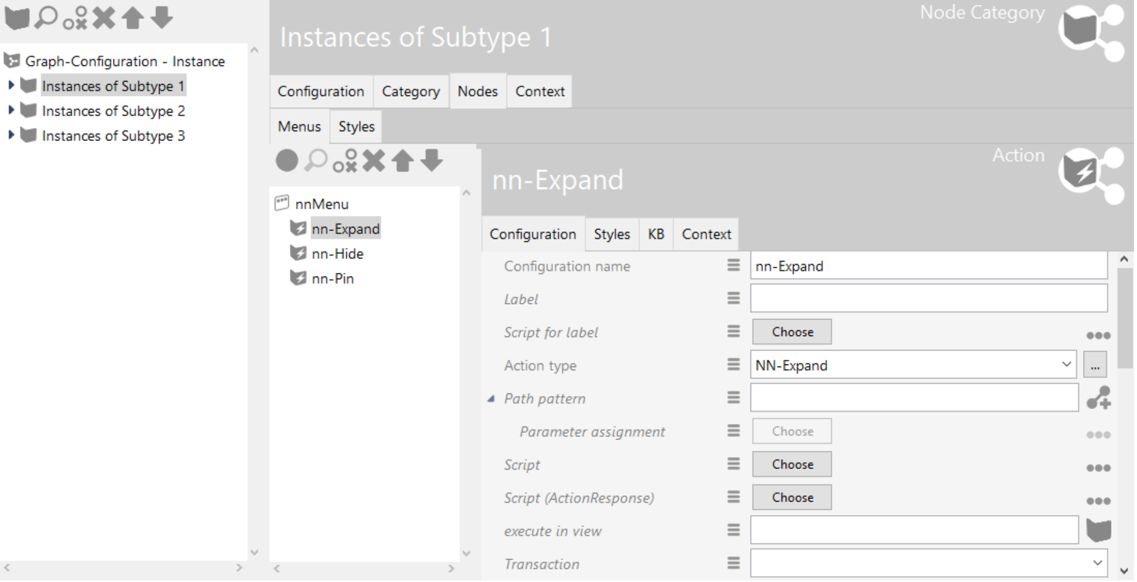
1.3.3.4.4. Action type "NN-Expand"

NN-Expand is an action type that makes it possible to expand a graph node in the Net-Navigator. This means that you can see all the nodes that are connected to this node via a relation and that are permitted by the graph configuration. The affected relations between the nodes are also displayed. Nodes that are already displayed in the Net-Navigator only display the relevant relations in addition.

Display with a plus sign as shown in the image below is the default setting. If you click on the plus button and it involves too many relations, a dialog window appears, and that dialog window has also been configured already. In this dialog you can choose which nodes should be displayed.



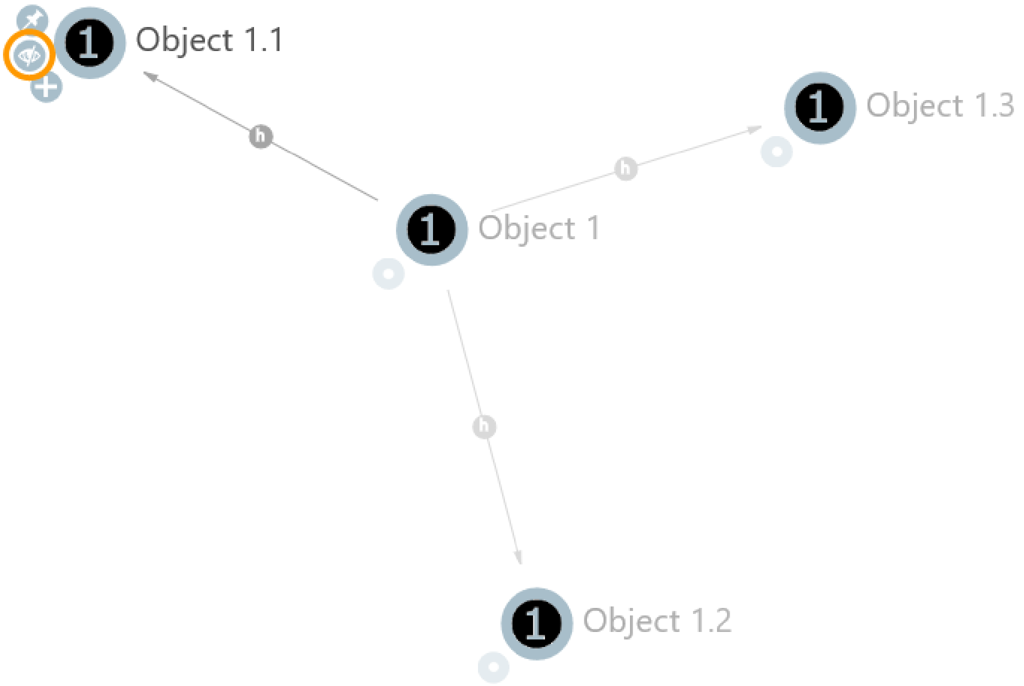
In the graph configuration this action is attached to all node categories that are supposed to be equipped with it. A menu that can contain all NN actions is created on the “Node” tab. In the action itself, it is only necessary to select the “NN-Expand” action type, all other specifications are optional. Further action types are available from the neighboring “...” button.



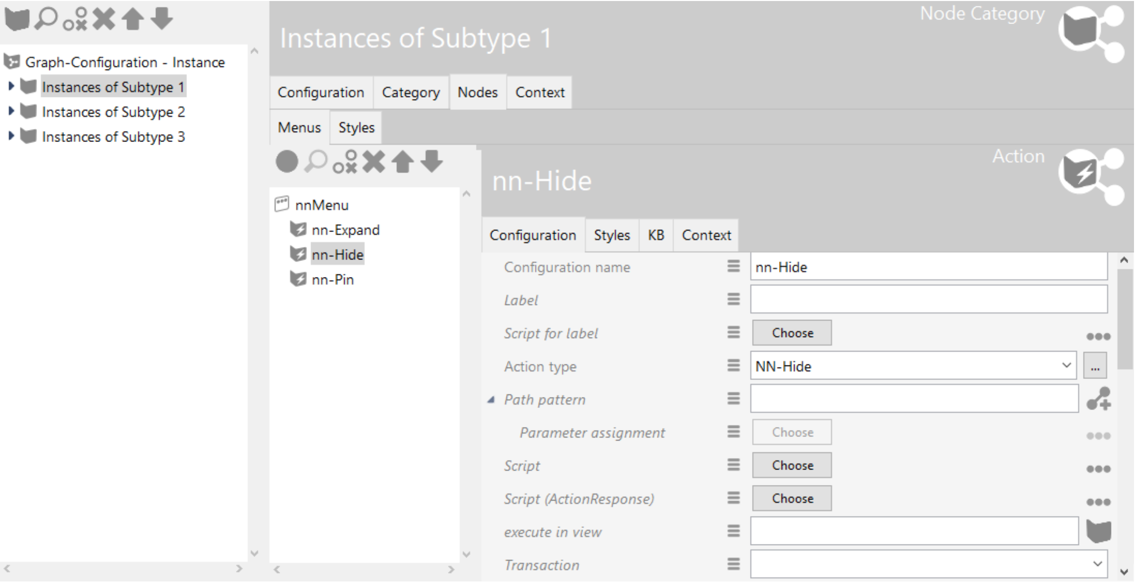
Configuration name	nn-Expand
Label	
Script for label	Choose
Action type	NN-Expand
Path pattern	
Parameter assignment	Choose
Script	Choose
Script (ActionResponse)	Choose
execute in view	
Transaction	

1.3.3.4.5. Action type "NN-Hide"

With the configuration of this action type, a menu button is provided in the graph nodes that hides the selected graph nodes and its displayed relations one time (see crossed-out eye in the image). The node can, for example, be displayed again when another connected node is expanded.



The NN-Hide action is configured like the NN-Expand action, but “NN-Hide” is chosen as the action type instead of “NN-Expand”. In order to configure more than one action type on a node, multiple actions must be created for a menu.



Graph-Configuration - Instance
Instances of Subtype 1
Instances of Subtype 2
Instances of Subtype 3

Instances of Subtype 1 — Node Category
Configuration | Category | Nodes | Context
Menus | Styles

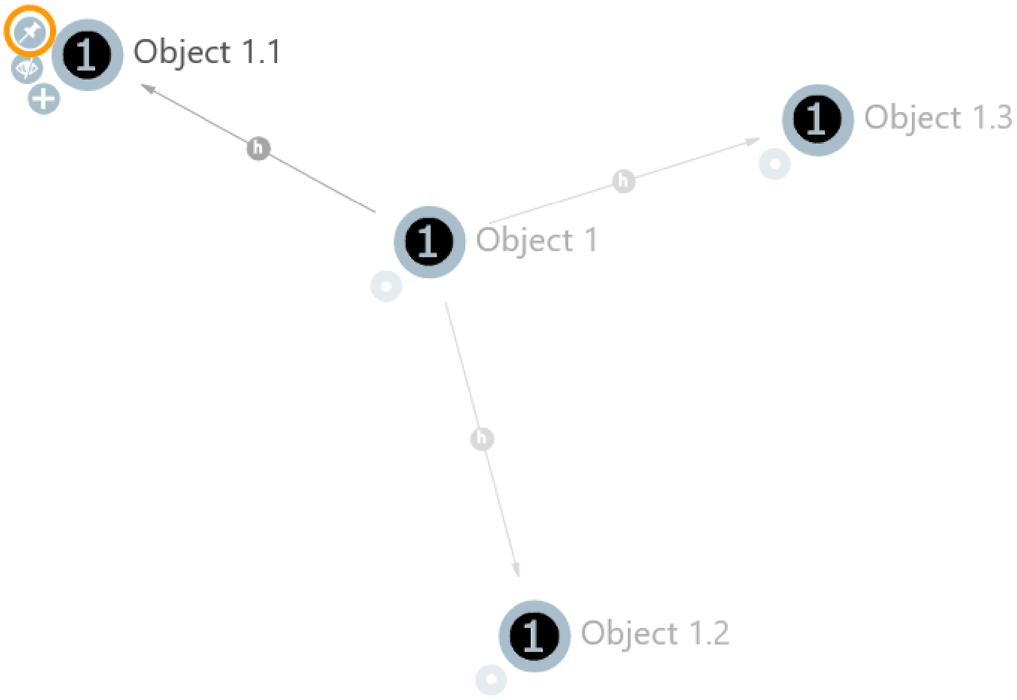
nnMenu
nn-Expand
nn-Hide
nn-Pin

nn-Hide — Action
Configuration | Styles | KB | Context

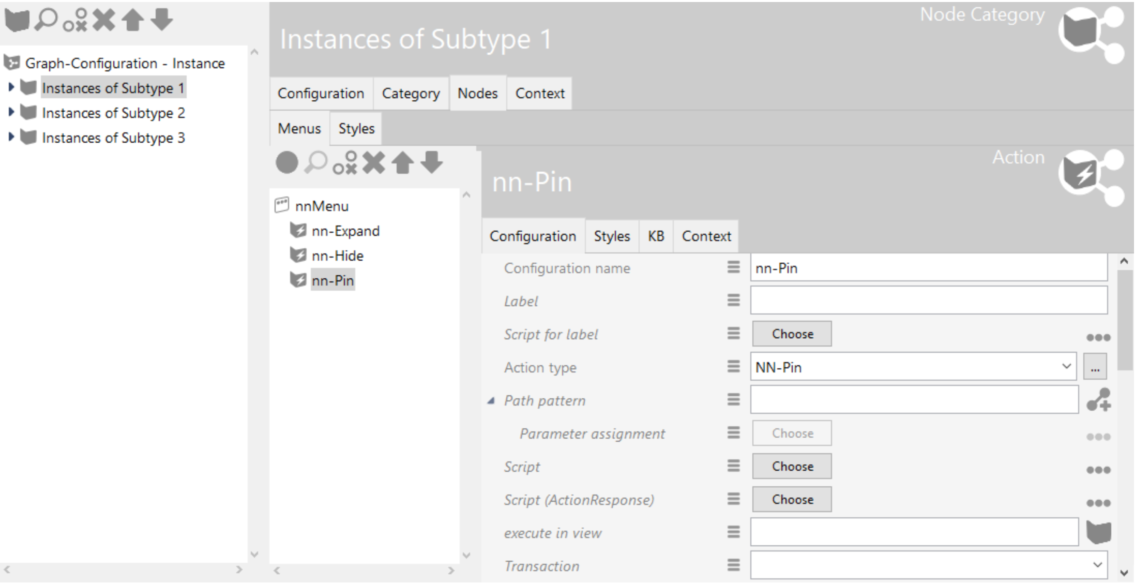
Configuration name: nn-Hide
Label:
Script for label: Choose
Action type: NN-Hide
Path pattern:
Parameter assignment: Choose
Script: Choose
Script (ActionResponse): Choose
execute in view:
Transaction:

1.3.3.4.6. Action type "NN-Pin"

The NN-Pin action is used to configure a menu button that allows a node to be pinned down in the Net-Navigator. When the graph is automatically restructured, for example when expanding another node, the node that was pinned down remains in its position. Despite this, the node can be repositioned manually and the pin is released when the graph is reloaded. Clicking on the pin again also releases it again. The “pinned” status is displayed by a change in the graphic (the pin points downwards instead of lying at an angle).



The configuration of the action type is performed as described in the “NN-Expand action”.

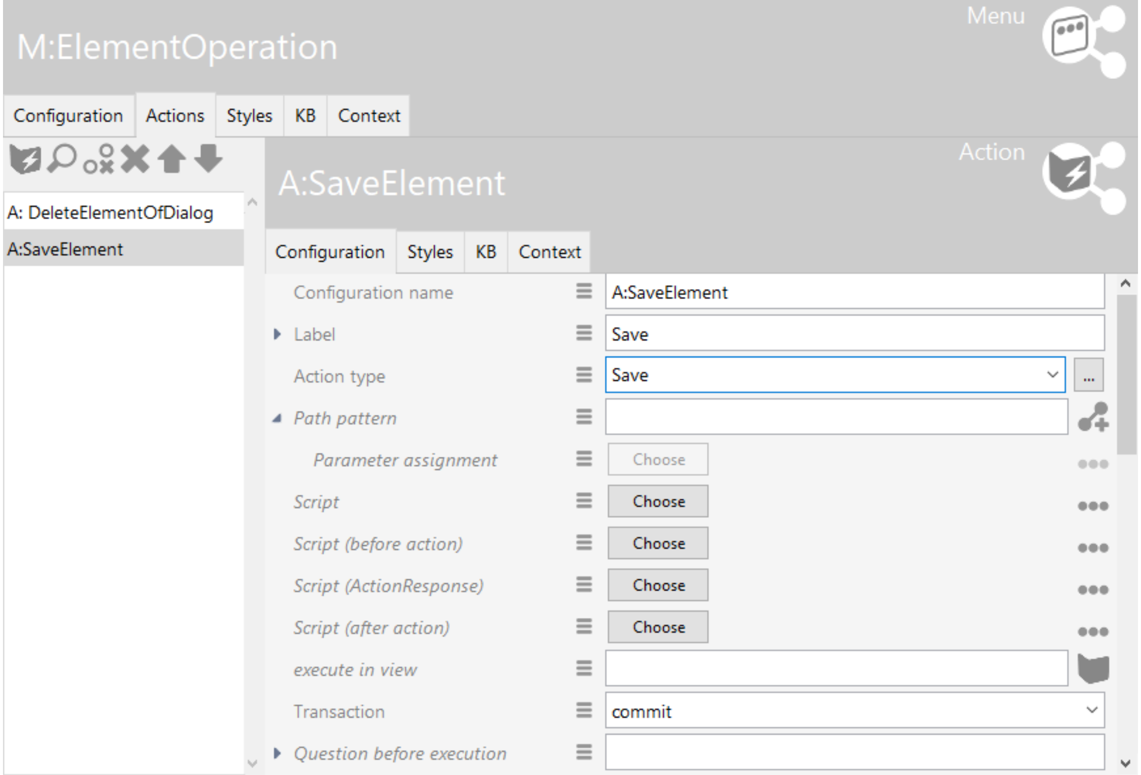


Configuration name	nn-Pin
Label	
Script for label	Choose
Action type	NN-Pin
Path pattern	
Parameter assignment	Choose
Script	Choose
Script (ActionResponse)	Choose
execute in view	
Transaction	

1.3.3.4.7. Action type "Save"

The Save action stores the form data from the web front-end in the Knowledge Graph. The web front-end automatically recognizes the action type and sends it to the configured view. If no view has been defined as the recipient of the action, the web front-end tries to find a suitable view in a neighboring panel.

To do this, the action type “Save” is assigned to the action in a menu:



A: DeleteElementOfDialog
A:SaveElement

A:SaveElement

Configuration name: A:SaveElement
Label: Save
Action type: Save
Path pattern:
Parameter assignment: Choose
Script: Choose
Script (before action): Choose
Script (ActionResponse): Choose
Script (after action): Choose
execute in view:
Transaction: commit
Question before execution:

The Save action can be used, for example, to replace the individual Save buttons in several edit fields in a dialog with a customized Save button.

NOTE

If you want to to use the save-action to do more than just to save (e.g. add another object to the object you just edited), you have to use "Script (after action)" instead of "Script". The reason is that otherwise the save action would be overwritten by the script action.

1.3.3.4.8. Action type "Print"

Like in the Knowledge Builder, the Print action is used for generating documents based on the shown model. The difference is that no configuration dialog is shown to the user. The necessary settings therefore need to be configured at the action configuration. Prerequisite for using the Print action is the availability of the printing component, which can be installed using the admin tool.

Depending on which kind of view is used to execute the print action, the behavior is slightly different:

- Table print: If the action is performed by a table or search view, a table print based on the columns and content of the respective table is executed. If the print action is not connected to a print template, a new .xlsx document is generated, otherwise the table content is embedded in the provided template file. In both cases the filtering and sorting of the table is respected, but all elements are printed regardless the configured pagination of the table.

- Element print: If the action is performed by any other view, the element which is the model of the respective view is used as the basis for the generated document. In this case the configuration of a print template is mandatory. This mode is also used for print actions that are configured in table rows. In that case they refer to the respective row’s element.

For a print action, the desired file name and target format can be configured. A configured target format requires a suitable converter configuration from the template’s source format to the target format. For more information regarding the configuration of print templates and converters, refer to the chapter on "Reports and printing".

1.3.3.5. Internal actions

The use of internal actions requires expert knowledge. If in doubt, please contact i-views support: support@i-views.com.

The actions listed here are only included for reasons of completeness. This includes actions such as:

- Sort action
- Jump action
- Create target action
- Script action: If there is a script on an action, it causes it to be executed automatically, and therefore overwrites the integrated function of the respective action type.

1.3.3.6. Scripts of actions

1.3.3.6.1. Script (custom)

This script is executed when the action is triggered. The script may modify elements of the knowledge graph and compute the result of the action which is usually the model to be visualized in the defined target panel(s).

The script may also read and modify session variables or view state.

```
function customAction(action, actionResult) {
}
```

Arguments

action	$k.Action - The object representing this action.
actionResult	$k.ActionResult - The result of the action.

Parameters of the action can be accessed by the corresponding functions of the action object (see JavaScript documentation).

The view ($k.View or sub-classes) executing the action can be accessed in the script as "this" object.

1.3.3.6.2. Script (actionResponse)

The purpose of this script is to provide a custom response to customized fronted-implementations. For standard ViewConfigMapper this script must not be used.

This script is executed after the action has been executed. Its main task is to prepare the result of the action for the ViewConfigMapper (or other front-ends). The script must return an object of the type $k.ActionResponse.

```
function actionResponse(element, context, resultModel) {
    var actionResponse = new $k.ActionResponse();

    actionResponse.setData(resultModel);
    actionResponse.setFollowup("new");
    actionResponse.setNotification("done","warn");

    return actionResponse;
}
```

Arguments

Argument	Value
element	The semantic element in the context of which the action is executed
context (deprecated)	More predefined variables that describe the context of the action in more detail
resultModel	The result model of the action result.

ActionResponse

The ActionResponse can be supplemented with values for *Followup / Data* and *Notification* . These values can be evaluated by other applications such as the ViewConfigMapper.

In the Knowledge Builder, the following values for *Followup* are possible in tables:

refresh	Renders the current table again without recomputing the list
update	Recalculates the table
show-element	Selects the element in *data* in the table. Alternatively, the “data” element can handle an object by means of {"element": actionResult, "viewMode": "edit} in order to open the result in a new Detail editor.

Followup is not evaluated in detail editors.

1.3.3.6.3. Script (actionVisible)

```
function actionVisible(element, context)  {
    return true;
}
```

The return value is used to decide whether the button is displayed or not.

In the case of actions on the elements, the following function is called up in tables, which transfer an array of elements and expect an array of Boolean values. This can be used to compute the visibility for the elements more efficiently in one go.

```
function actionsEnabled(elements, contexts) {
    return elements.map(function (element, index) {
        return actionVisible(element, contexts[index]);
    });
}
```

1.3.3.6.4. Script (actionEnabled)

```
function actionEnabled(element, context) {
    return true;
}
```

The return value is used to decide whether the button is active.

In the case of actions on the elements, the following function is called in tables, which transfer an array of elements and expect an array of Boolean values:

```
function actionsVisible(elements, contexts) {
    return elements.map(function (element, index) {
        return actionVisible(element, contexts[index]);
    });
}
```

1.3.3.6.5. Script with UI specific actions

The script that implements the action can access UI-specific functions in the Knowledge Builder using *context.ui* .

UI functions should not be executes within transactions when possible, as the display is not updated within the transaction.

```
context.ui.alert(message, windowTitle)
```

Shows a message.

```
context.ui.requestString(message, windowTitle)
```

The user can enter a string.

```
context.ui.confirm(message, windowTitle)
```

Opens a cancel dialog.

```
context.ui.choose(objects, message, windowTitle, stringFunction)
```

Have an object selected from a set.

```
context.ui.openEditor(element)
```

Open the default editor for the object.

```
context.ui.notificationDialog(notificationFunction, parameters,
windowTitle)
```

A wait dialog or notification dialog is opened. Depending on how it is configured, it can be canceled.

Possible parameters:

Parameter	Description	Default value
autoExpand	The dialog display area is opened initially.	true
canCancel	The dialog can be canceled.	true
stayOpen	The dialog remains open after the end of the function.	true

Example:

```
ui.notificationDialog(
    function() {
        ui.raiseNotification("start");
        for (var i = 0; i < 10; i ++)
            ui.raiseNotification("" + i + "*" + i + "=" + (i*i));
        ui.raiseNotification("end");
        return undefined;
    },
    { "canCancel" : false },
    "A wait dialog"
)
```

Messages can be output in the display area using the following *raiseNotification* function.

```
$k.UI.raiseNotification(message)
```

This message is only captured by the *notificationDialog* function, and the message is only output in the display area there.

1.3.3.7. Action sequences

Often we might want to summarize the changes that the user makes to the Knowledge Graph and that are split into several sequential actions.

Example: In one action, a new product is created, and in the next action the properties of the product are described. Aborting the second action would create a product without a description in the Knowledge Graph.

What is required is an “All or nothing” behavior to ensure that either all actions that belong together are executed or that none of them are. You also want to ensure that other users can only see the change to the Knowledge Graph once it has been completed. You can achieve such behavior by encapsulating the actions in a “Transaction”.

In order to summarize a sequence of actions in a transaction, you mark the first action with “Transaction - begin” and the final action with “Transaction - commit”.

Caution: The transaction is started only if the first action actually modifies the Knowledge Graph. When creating new objects in a sequence of actions you also have to ensure that the order of newly created objects is deterministic, so whenever an action script is repeated the creation order is the same as before. If the set of created objects varies dependent on the actual situation, make sure to sort the originating set in a deterministic way before creating the objects (e.g. by idString()).

The transaction commit can also be brought about dynamically via the “setTransactionCommit()” script function.

If the transaction is to be canceled, you can achieve this by means of an action of the “Cancel” type. Canceling means that all previous changes to the Knowledge Graph conducted within the transaction are undone. The “setFailed()” script function can be used to dynamically initiate a cancellation.

As a transaction is always coupled to the duration of a session, a transaction is canceled automatically when the session ends in which the transaction was started. If, for example, you open a dialog at the start of the transaction and the dialog is closed before the transaction was completed, the transaction is canceled automatically. This does not apply to dialogs that are opened while a transaction is already running, because this creates a new session on the session stack. Dialog sequences (one dialog is closed and another dialog is opened immediately) do not interrupt the transaction either.

1.3.4. View configuration elements

A view configuration describes how objects or types are to be shown. The different element types that are available in the view configuration are described in the following.

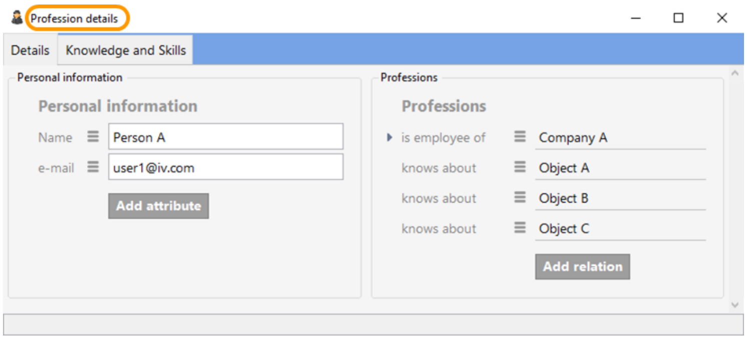
The individual view configuration elements can, in part, be plugged together in any way. The configurations can also be used multiple times as a sub-configuration.

List of the different detail configuration types

Configuration type	Top-level configuration	Can include the following sub-configuration
Alternative	x	any
Property		
Properties	x	property
Layout	x	any
Hierarchy	x	any
Script-generated content	x	
Static text		
Search		Table

Setting options that all detail configuration types have in common

Name	Value
Configuration name	This is not used in the user interface. The user who creates a configuration has the option of assigning a name that is comprehensible for the user in order to be able to find this configuration more easily later on, and to be able to use it again in other configurations.

Name	Value
Script for window title	Only for use in the Knowledge Builder. If an object is, for example, opened by double-clicking in the object list, a window with the properties of this object opens. The title of this window can be determined using a script.  Profession details Details Knowledge and Skills Personal information Personal information Name: Person A e-mail: user1@iv.com Add attribute Professions Professions is employee of: Company A knows about: Object A knows about: Object B knows about: Object C Add relation

NOTE The setting options for the individual configuration types are described in the following sections. The obligatory parameters are printed in bold.

1.3.4.1. Alternative

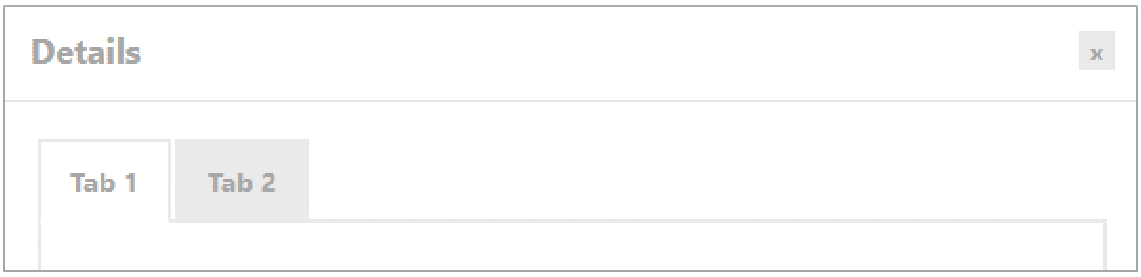
An alternative is used to configure many different alternative views on an object. You can use tabs to switch between the views in the application.

Setting options

Name	Value
Configuration name	The configuration name is used for identification and reuse of the configuration.
Label	A label is only used if this configuration is embedded in another configuration, e.g. an additional *alternative* .
Script for label	The Script for label is used for dynamic computing of the label. This script is only available, if no entry exists for "Label"
Default alternative	The sub-view that is supposed to be selected initially can be specified here.
Script for default alternative	.
Restore last selected alternative	If enabled, the lastly selected tab keeps selected, even if a change of view occurred.
Script for visibility	This script is used to compute dynamically, if the view needs to be visible or not.

Display in an application

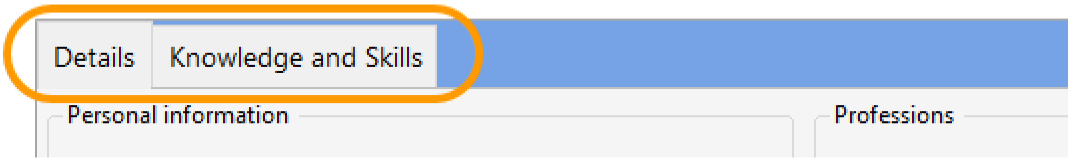
If the views are exported into JSON, the individual sub-views are attached to the *alternatives* KEY in an ARRAY.



Example of an alternative in an application: You can use the tabs to switch between the views “Tab 1” and “Tab 2”.

Display in Knowledge Builder

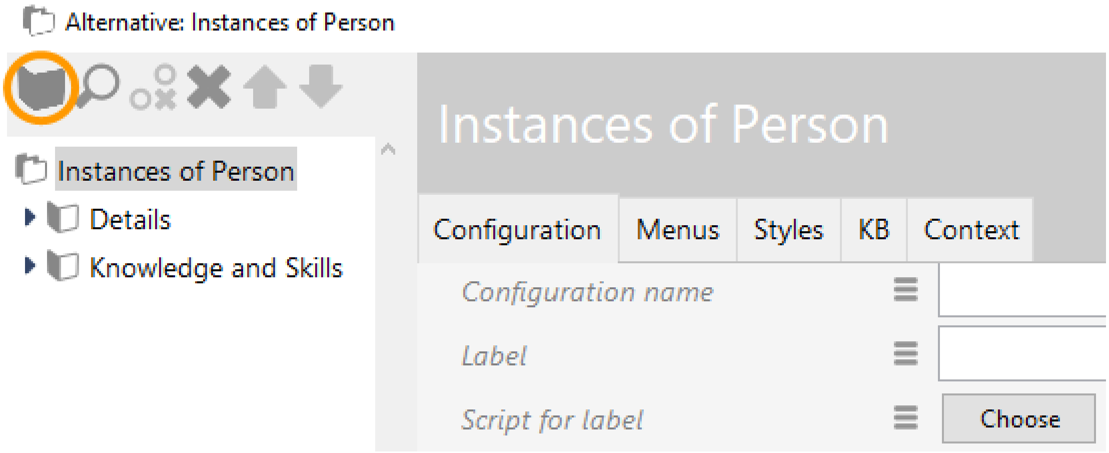
In Knowledge Builder, the various configured views of an object that are linked to the alternative are made available to users by means of tabs



Example of an alternative in Knowledge Builder: You can use the tabs to move between the view “Details” and the view “Knowledge and Skills”

Configuration of tabs

If a view configuration of the “Alternative” type has been created, you can use the button “Create new objects of object configurations” to add a new tab.



It usually makes sense to use the view configuration type “Layout” as the tab as any number of view configurations can be placed therein. The label of the view configuration is also the label of the tab.

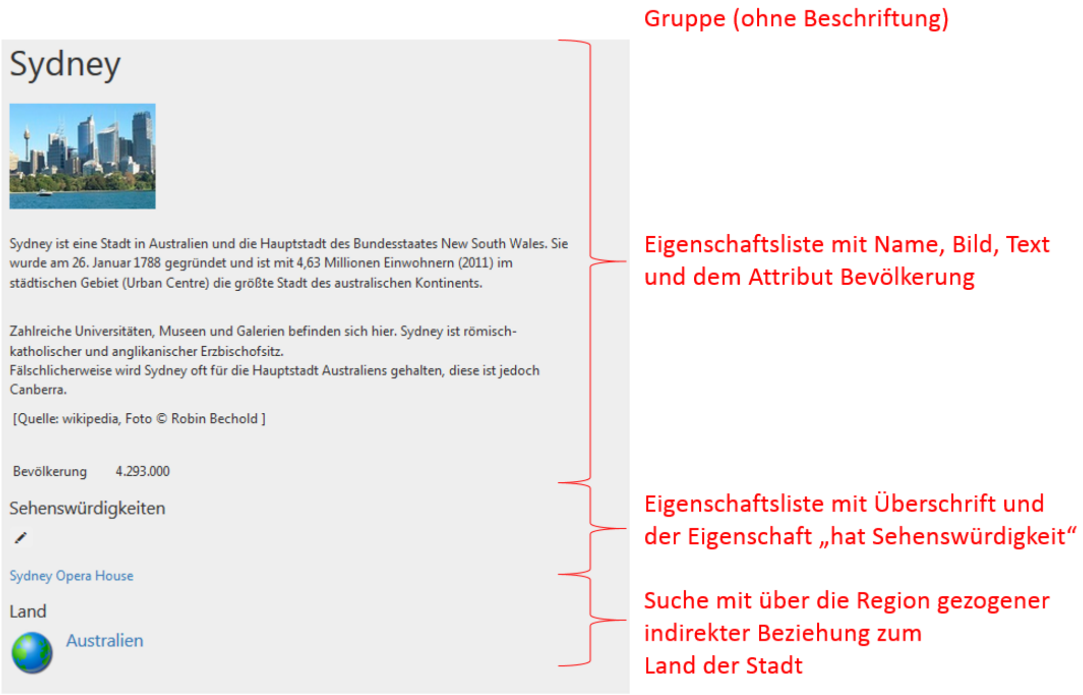
1.3.4.2. Layout

A layout can be used to summarize different sub-configurations in one view. The subelements are then shown in order.

Setting options

Name	Value
Configuration name	The configuration name is used for identification and reuse of the configuration.
Label	A label is only used if this configuration is embedded in another configuration, e.g. *Alternative* .
Orientation	Defines whether subviews are stacked horizontally or vertically. The default behavior is horizontal orientation.
Script for visibility	A script that determines whether the layout is displayed or not.

Display in an application



Sydney

Sydney ist eine Stadt in Australien und die Hauptstadt des Bundesstaates New South Wales. Sie wurde am 26. Januar 1788 gegründet und ist mit 4,63 Millionen Einwohnern (2011) im städtischen Gebiet (Urban Centre) die größte Stadt des australischen Kontinents.

Zahlreiche Universitäten, Museen und Galerien befinden sich hier. Sydney ist römisch-katholischer und anglikanischer Erzbischofsitz.
Fälschlicherweise wird Sydney oft für die Hauptstadt Australiens gehalten, diese ist jedoch Canberra.

[Quelle: wikipedia, Foto © Robin Bechold]

Bevölkerung 4.293.000

Sehenswürdigkeiten

Sydney Opera House

Land

Australien

Gruppe (ohne Beschriftung)

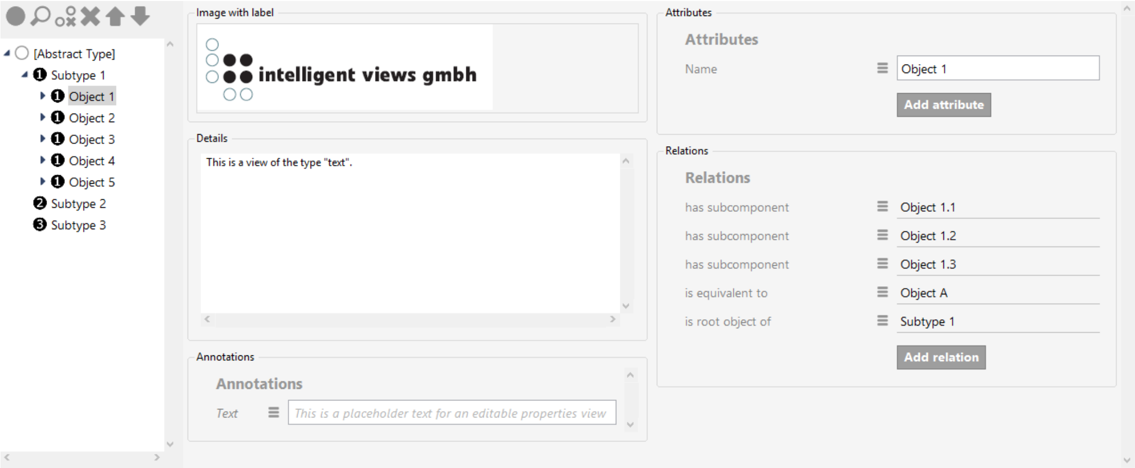
Eigenschaftsliste mit Name, Bild, Text und dem Attribut Bevölkerung

Eigenschaftsliste mit Überschrift und der Eigenschaft „hat Sehenswürdigkeit“

Suche mit über die Region gezogener indirekter Beziehung zum Land der Stadt

Display in Knowledge Builder

A frame is drawn around a layout in the Knowledge Builder. This frame then shows the views of the sub-configurations.



This is a view of the type "text".

Annotations

Text: This is a placeholder text for an editable properties view

Attributes

Name: Object 1

Relations

has subcomponent: Object 1.1

has subcomponent: Object 1.2

has subcomponent: Object 1.3

is equivalent to: Object A

is root object of: Subtype 1

A layout detail view adjacent to the tree view with the following sub-configurations: Image view “Image with label”, text view "Details" and string property "Text", contained in a vertically oriented layout on the left side. On the right side, a second layout with vertical orientation of properties is shown.

1.3.4.3. Hierarchy

The configuration type “Hierarchy” displays elements of a Knowledge Graph as a hierarchy in a tree structure, in which individual branches can be expanded and collapsed.

Either relations or relation targets can be used for work. The hierarchy is structured from the start element of the view configuration, for which all subordinate relations or objects and their subordinates must first be determined. After this, the higher-level relations or objects are determined for each element. This element result set is then shown in the hierarchy.

Setting options

Name	Value
Configuration name	The configuration name is used for identification and reuse of the configuration.
Label	A label is only used if this configuration is embedded in another configuration, e.g. *Alternative* .
Script for label	It is also possible to define a label using a script.
Icon	Sets the icon for all nodes of the hierarchy.
Script for icon	Returns an element icon as blob to be displayed for nodes of the hierarchy.
Show parent banner	Only relevant for the Knowledge Builder: Banner is displayed.
Do not show detail view	Per default, a standardized detail view is displayed besides the hierarchy view which shows the details of the selected element. This option suppresses the detail view from being displayed.

Name	Value
Restore last expanded nodes	If enabled, the last expanded nodes stay expanded for one and the same context element during the whole web-frontend session.
Click action	Reference to an action that is called when a hierarchy element is clicked.
Script for visibility	Determines whether the whole hierarchy view is visible or not.
Generate subelements without name query	When new subelements are generated in the hierarchy, what their name should be is queried by default. A checkmark here generates nameless objects without a name query.
Traversal	
Structured query (down)	Structured query for determining the subordinate element.
Structured query (up)	Structured query for determining the superordinate element.
Script (down)	Script for specifying a relation or a relation target to determine the element node of the lower hierarchy level.See example below.
Script (up)	Script for specifying a relation or a relation target to determine the element node of the upper hierarchy level.See example below.
Relation (down)	Relation half which points downwards.
Relation (up)	Relation half which points upwards.
Output up to depth	.
Sort	
Sort downward	Controls if sorting is in ascending or descending order. If this parameter is not set, sorting occurs in ascending order.
Primary sort criterion	Selection option for the criterion used for sorting the properties: • **Position:** The order defined in the configuration is used (default). • **Value:** The content of the attribute or display name of the relation target is used. • **Script for sorting:** The script saved in the attribute Script for sorting is used for determining the sort criterion.
Secondary sort criterion	Sort criterion for properties which have the same value for the primary sort criterion. The setting options are analogous to those for the *primary sort criterion*.
Script for sorting	Reference to a registered script that returns the sort key for the primary or secondary sort criterion.

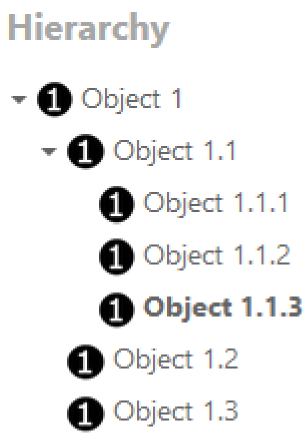
Name	Value
Disallow manual sorting	By default, the user can reattach elements in the Knowledge Builder to the schema by means of Drag&Drop. If this option is activated, this is no longer possible.
KB	
Creating elements without question by name	If enabled, the menu directly above the hierarchy allows creating subelements without asking the user for a name of the new element.

Actions and styles

Actions and styles can be attach for both the entire hierarchy and for the individual nodes. From version 5.2 or higher, style classes can be automatically assigned using a script.

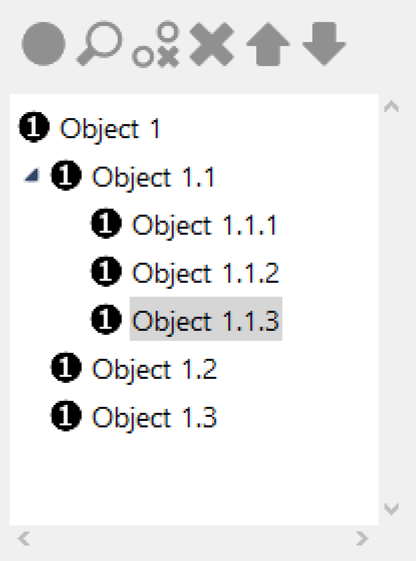
Display in an application

The JSON representation of a configuration of type *hierarchy* is only available from version 4.1 or higher.



Display in Knowledge Builder

A hierarchy appears in the area on the left in the detailed display of an element. The element is displayed with a view configuration without hierarchy in the area on the right. This view configuration must be defined separately and the configuration name of the hierarchy must be specified under *Reference* >> *Apply in* . Alternatively, the sub-configuration can also be specified directly in the hierarchy under *Sub-configuration* .



Object 1
- Object 1.1
  - Object 1.1.1
  - Object 1.1.2
  - Object 1.1.3
- Object 1.2
- Object 1.3

Notes

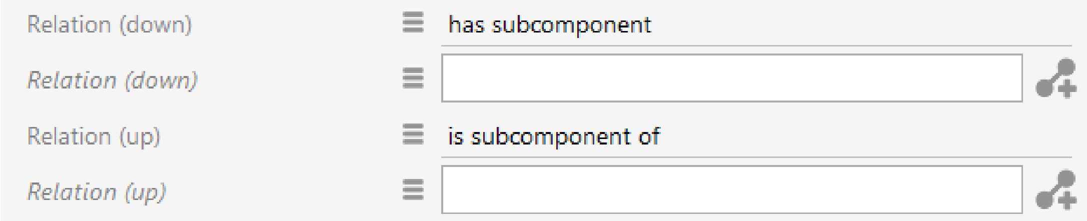
- Elements are not always represented by their name in hierarchies. It is not possible to display anything other than the name, or information supplementing the name, directly in the hierarchy.
- The values of all properties that can be filled out for forming the hierarchy are relations.
- The individual attributes such as *relation - descending* can be assigned multiple times.
- The relation or relations are determined and collected for each attribute type. If different attribute types are specified, the subsets are used to form an intersect.

Example - application case

Hierarchies are typically used to represent supertopic/subtopic relations or part-of relations.

1. Relation that forms a hierarchy

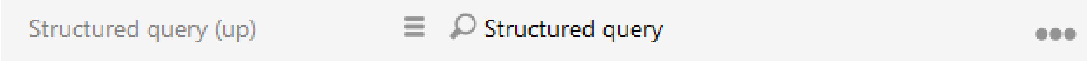
   The most direct variant. The relations that form the hierarchy are entered.



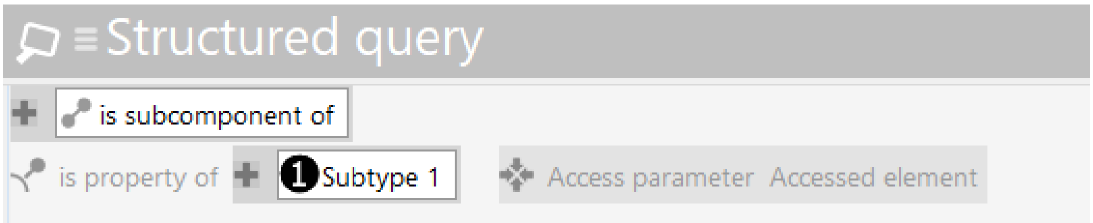
Relation (down)	has subcomponent
Relation (down)	
Relation (up)	is subcomponent of
Relation (up)	

2. Structured query that forms the hierarchy

   The relations can also be determined by means of a structured query.

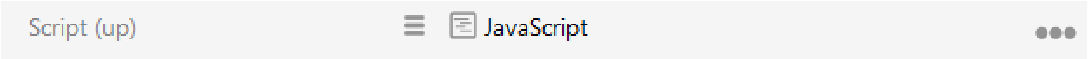


Structured query (up)	Structured query



3. Script that forms a hierarchy

   A script can also be used to collect the relations that potentially form a hierarchy. The current element is passed to it as a parameter, and it must return a set of relations. Instead of working on relations, working on elements is also possible.



Script (up) JavaScript

Script example for relation with internal name *'is SubcomponentOf':*

Option a): Using relations

```
function relationsOf(element)
{
    return element.relations('isSubcomponentOf');
}
function targetsOf (element)
{
    return undefined;
}
```

Option b): Using relation targets

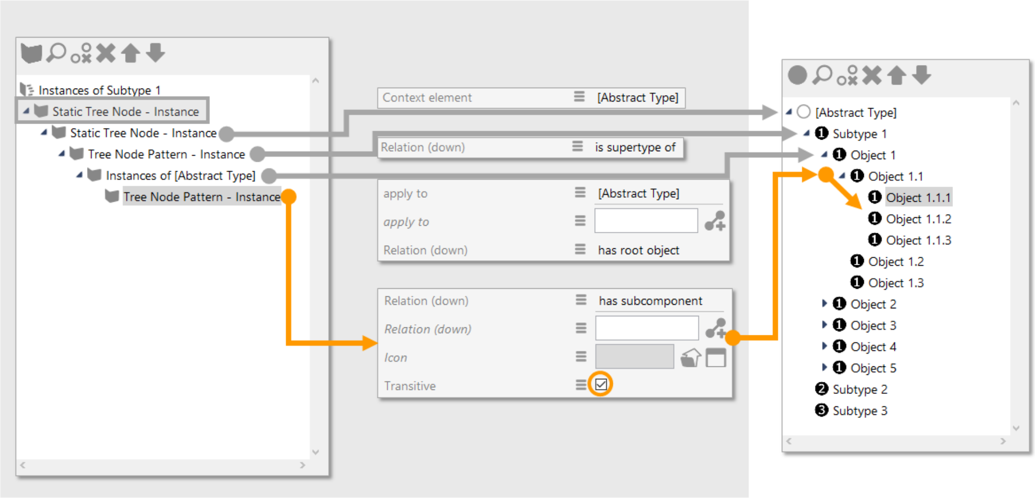
```
function relationsOf(element)
{
    return undefined;
}
function targetsOf (element)
{
    return element.relationTargets('isSubcomponentOf');
}
```

NOTE

Please be aware that only the usage of relations *or* relation targets in one and the same script makes sense; otherwise each hierarchy node will appear twice. The other part of the script keeps unchanged and returns "undefined".

1.3.4.4. Tree

Just like a “hierarchy,” a “tree” is based on the configuration of a hierarchical tree structure. In contrast to a hierarchy, a tree can also include static nodes. Hence, it is possible to create a tree without a Knowledge Graph source element. Another difference is that the sub-nodes of a “tree” can be configured differently whereas all nodes of a “hierarchy” respond in the same way for a given semantic element.



- Instances of Subtype 1
  - Static Tree Node - Instance
    - Static Tree Node - Instance
      - Tree Node Pattern - Instance
        - Instances of [Abstract Type]
          - Tree Node Pattern - Instance

Context element	[Abstract Type]
Relation (down)	is supertype of

apply to	[Abstract Type]
apply to	
Relation (down)	has root object

Relation (down)	has subcomponent
Relation (down)	
Icon	
Transitive	☑

- [Abstract Type]
  - Subtype 1
    - Object 1
      - Object 1.1
        - Object 1.1.1
        - Object 1.1.2
        - Object 1.1.3
      - Object 1.2
      - Object 1.3
    - Object 2
    - Object 3
    - Object 4
    - Object 5
  - Subtype 2
  - Subtype 3

View configuration for tree view

Tree view in Knowledge Builder

A tree configuration generally distinguishes two types of nodes:

- **Static hierarchy node:** Nodes of this type always exist if there is a connection to the root of the tree. The “context element” relation can be used to optionally integrate the node into a semantic element.

  NOTE The top node of a tree is always static and always invisible.

- **Hierarchy node patterns:** This type can map several nodes for each level. A node is formed for each relation target that can be reached from an element of the higher-level node. You can set the property “transitive” to map several levels. You can the property “apply to” to restrict to which element types the node pattern is applicable. Otherwise the node pattern can be applied to all elements that fall into the target validity area of the configured relations. As an alternative to determination via a relation type, sub-nodes can be determined using a structured query. The structured query begins with the element of the parent node. The subordinate nodes are determined by the part of the query that is marked with the predefined identifier “subnode”. If you want to use the “Transitive” option, the corresponding relation in the query must be marked with the predefined identifier “subnodeRelation”.

The sorting of tree nodes can be configured in the same way as that of the “hierarchy.” However, this configuration does not globally apply to the tree but each node configuration applies to the respective sub-nodes.

Finally, the image and label displayed can be configured for each node type, either directly or via script.

Setting options

Name	Value
Configuration name	The configuration name is used for identification and reuse of the configuration.
Label	A label is only used if this configuration is embedded in another configuration, e.g. *Alternative* .
Script for label	Script that returns a string for the label instead of using the label attribute.
Do not show detail view	Per default, a standardized detail view is displayed besides the hierarchy view which shows the details of the selected element. This option suppresses the detail view from being displayed. **NOTE** The standardized detail view can be replaced by configuring a customized view.
Disallow manual sorting	By default, the user can reattach elements in the Knowledge Builder to the schema by means of Drag&Drop. If this option is activated, this is no longer possible.
Restore last expanded nodes	If enabled, the last expanded nodes stay expanded for one and the same context element during the whole web-frontend session.
Script for visibility	Script that returns a Boolean value for whether the view is to be displayed or not.
Sort configuration	
Sort downward	Controls if sorting of subnodes is in ascending or descending order. If this parameter is not set, sorting occurs in ascending order.
Primary sort criterion	Selection option for the criterion used for sorting the subnodes: • **Position:** The order defined by the *Sort order* meta property of involved relations is used • **Script for sorting:** The script saved in the attribute *Script for sorting* is used for determining the sort criterion. • **Value:** The display name of the relation target is used.

Name	Value
Secondary sort criterion	Sort criterion for subnodes which have the same value for the primary sort criterion. The setting options are analogous to those for the *primary sort criterion.*
Script for sorting	Reference to a registered script that returns the sort key for the primary or secondary sort criterion. **Attention**: if there are several different sub-node configurations, the script is potentially called with instances of different types and should be formulated in a correspondingly general way.
KB	
Script for window status	Returns a status label for the window footer when the detail view within the Knowledge Builder is opened in a new window.
Script for window title	Returns a label for the window title when the detail view within the Knowledge Builder is opened in a new window.

1.3.4.5. Properties

The *Properties* configuration is a list of individual configurations. The sub-configurations can be exclusively of the *Property* type, each of which is linked to an attribute or a relation of a Knowledge Graph object or type.

Setting options

Name	Value
Configuration name	The configuration name is used for identification and reuse of this configuration.
Label	Display name of the collection of properties. If no label is specified, the string 'Properties' is used in Knowledge Builder.
Script for label	Alternatively, the display name can also be determined via a script.
Initially expanded	If this configuration in included e.g. as a meta-configuration, this parameter can be used if this is supposed to be expanded already when opening Knowledge Builder. **NOTE** The web frontend does not display the affected meta-property if the checkmark is not set here.
Script for visibility	Control of the visibility of the properties by a script.

Setting options for sorting

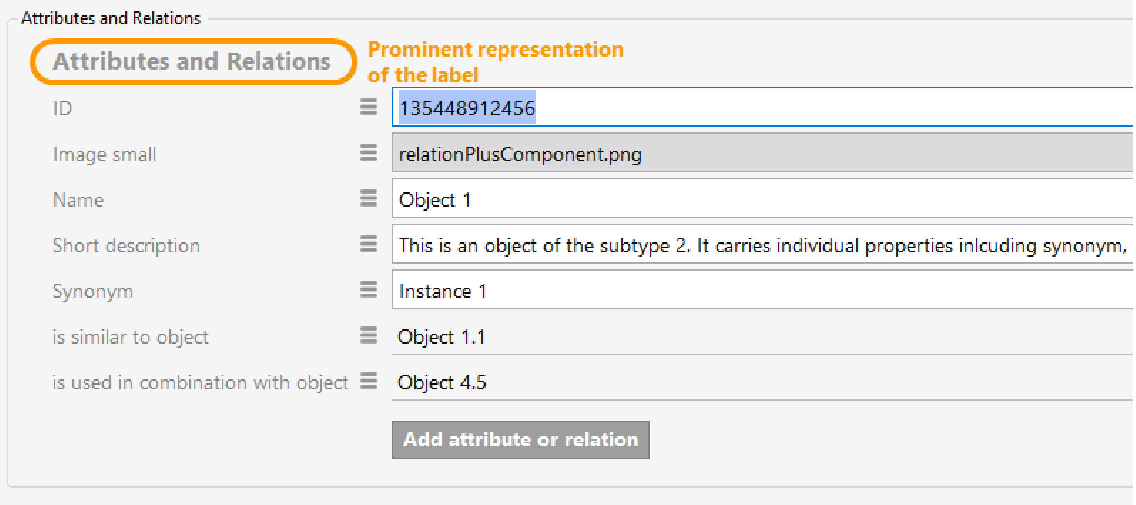
Name	Value
Sort downward	Controls if sorting is in ascending or descending order. If this parameter is not set, sorting occurs in ascending order.
Primary sort criterion	Selection option for the criterion used for sorting the properties: • **Position:** The order defined in the configuration is used (default). • **Script for sorting:** The script saved in the attribute Script for sorting is used for determining the sort criterion. • **Value:** The content of the attribute or display name of the relation target is used.
Secondary sort criterion	Sort criterion for properties which have the same value for the primary sort criterion. The setting options are analogous to those for the *primary sort criterion.*
Script for sorting	Reference to a registered script that returns the sort key for the primary or secondary sort criterion.

Display in applications

The views of the configuration of individual property elements are stored in an ARRAY during output in JSON format and appended with the KEY *properties* .

Display in Knowledge Builder

The label set in the configuration is displayed prominently. This is followed by views of the configured properties.



Attributes and Relations

Attributes and Relations

Prominent representation of the label

ID: 135448912456

Image small: relationPlusComponent.png

Name: Object 1

Short description: This is an object of the subtype 2. It carries individual properties inlcuding synonym,

Synonym: Instance 1

is similar to object: Object 1.1

is used in combination with object: Object 4.5

Note

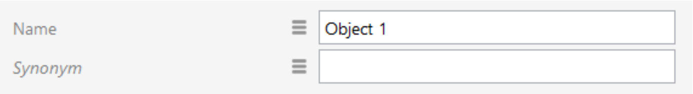
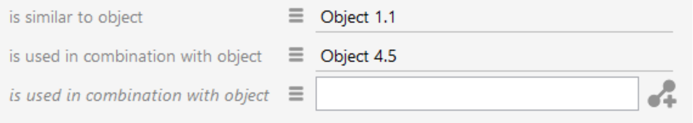
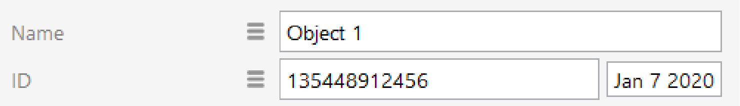
Meta properties are appended using the same process.

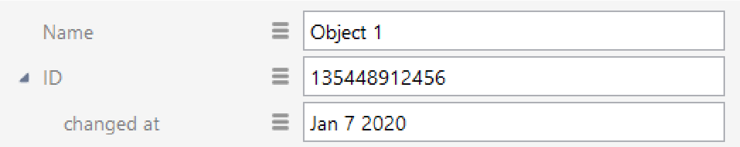
1.3.4.6. Property

The *Property* view configuration can be used to define individual attributes or relations to be displayed in a list of properties. It is also possible to use an abstract property that groups a set of properties.

Setting options

Name	Value
Configuration name	The configuration name is used for identification and reuse of this configuration.
Label	Display name of the property. If no label is specified, the name of the property type is output.
Script for label	The label can be determined by means of a script specified here.
Property	Link to the property type that is to be displayed.
Query for virtual properties	Alternative to 'Property': Instead of defining the property, a query can be used which returns the needed kind of property. This comes in handy when the property is not directly assigned to the object.
Script for virtual properties	Alternative to 'Property': Script for calculating the values to be displayed.If you set the "**Automatic updates**" meta flag, the KB is automatically updated when a value on which a calculation was based is changed. **Caution:** if you set this flag, this can have a significant effect on performance, depending on the script.
Display type	Available in two cases: 1. The property is a relation: Selection option for the display of the label of a relation target. This setting is only available if the *Relation target view* setting has the value *Choice* or *Relation structure* . 2. The property is a file attribute: Selection option for the display of the value in a file attribute. Selection options: • **Icon (topicIcon):** Icon of the relation target / file as an icon • **Icon and string** • **String (name attribute:** Name of the relation target / name of the file

Name	Value
Show filter	Only relevant in the view for editing objects: This option can be used to create a prompt that decides whether this configuration is displayed. The prompt is filled with the object of this property. The property is displayed for editing only if the prompt receives a result.
Show new properties	Only relevant in the view for editing objects.There are following options: • **never:** If this option is set, the respective property is only shown if already assigned. If the property value of a shown property is erased without replacement by another value, the property edit line is faded out. In order to show new properties, this is done by clicking on the button "Add attribute or relation". • **if not available yet:** If this option is set, the property is shown only if the property has not been created yet. This makes it quick and easy to complete and less easy to forget.  Name ≡ Object 1 *Synonym* ≡ • **always:** If this option is set, another property is shown in addition to the property of the same type, so this can be filled quickly and conveniently. It must be permitted for the property to occur multiple times.  is similar to object ≡ Object 1.1 is used in combination with object ≡ Object 4.5 *is used in combination with object* ≡ **NOTE** If no option is chosen, the behaviour equals the "never" option. The formerly available setting "Show additional properties" from previous i-views versions (5.3 and earlier) is incorporated into the option "always".
Configuration for embedded meta properties	Specification of the configuration to be used to display meta properties. The meta properties are embedded, i.e. the property is displayed after the value. The name of the property type is not displayed.  Name ≡ Object 1 ID ≡ 135448912456 Jan 7 2020

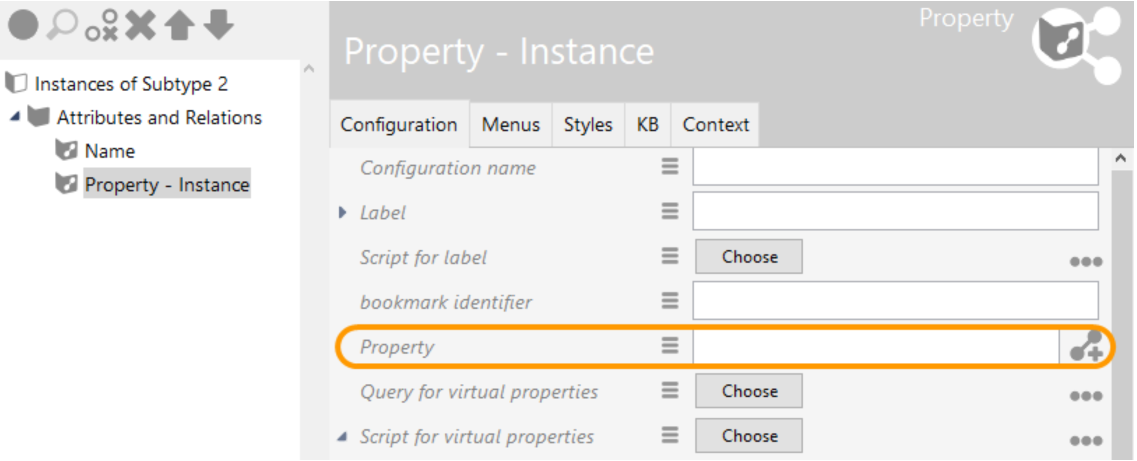
Name	**Value**
Configuration for meta properties	Specification of the configuration to be used to display meta properties. The meta properties are displayed under the value of the property. For display in the web front-end, the properties with the meta properties must be set to “initially expanded.”  Name: Object 1 ID: 135448912456 changed at: Jan 7 2020
Click action	.
Script for visibility	The conditions under which the property is displayed can be defined via JavaScript.
Relation target (only available for relations)	
Relation target view	If a relation is chosen as the property, this parameter can be used to define the view of the relation targets: • **Choice:** All relation targets are listed and displayed with a preceding checkbox. In case of existing relations, the checkbox is equipped with a tick. • **Drop down:** This setting is only useful if the relation may appear only once. A drop-down list is displayed showing all relation targets available for selection. • **Relation structure:** All relation targets are listed in the left area, rather like a hierarchy. The right area then shows the details view for the selected relation target. This view is only effective if the configuration is directly subordinate to a top-level configuration. • **Table:** Table view of the relations. The table view can *not* be applied in the Knowledge Builder. For the table view, the *Table* setting must be filled in. • **Table (relation targets):** Table view of the relation targets. This table can be applied in the Knowledge Builder.
Table	Only available if the *Relation target view* has the value *Table* or *Table (relation targets)* , in which case it is obligatory. The table configuration specified here determines which properties of the relation targets are to be output in table form. For the relation target to be displayed, at least its name must be configured in the table. For configuration of a table, see the Table chapter.
Relation target filter	Query for filtering the relation targets to be shown.
Relation target type filter	Query for filtering the relation targets by their type.

Name	Value
Script for relation target label	Script which returns a dedicated string for the relation target label. If not used, the primary name of the relation target is shown as label.Example: A person belongs to a department with the name 'Dpt. IV'. Using a suitable script, it is possible to change the output for the person from 'Dpt. IV' to 'Darmstadt city administration, Dpt. IV'.
Show relation target	Only available for relations. By default only the name of the relation target is displayed. When you click the name, the relation target opens in another editor. But if you choose the *Show relation target* option, the relation targets are shown directly, which means not just their names, but also all their properties.
Display	
Tooltip	Tooltip which appears when hovering the mouse pointer over the relation target.
Placeholder text	A placeholder text which is shown in light grey when the relevant string attribute has no attribute value yet.
Script for placeholder text	Script which returns a string for the placeholder text instead of a statically configured placeholder text.
Script for tooltip	Script which returns a string for the tooltip instead of a statically configured tooltip.
Sort	
Script for sorting	The script is used to determine a value for sorting. See the example below.
Sort downward	Controls whether the properties are sorted by name in ascending or descending order. If this parameter is not set, sorting occurs in *ascending* order.

NOTE Options either can be set by defining their value or, if available, by an equivalent script. Option value and script cannot be used at the same time.

Configuration of a property

A property can only be configured as part of a list of properties. It is acceptable for the list to contain only one property.

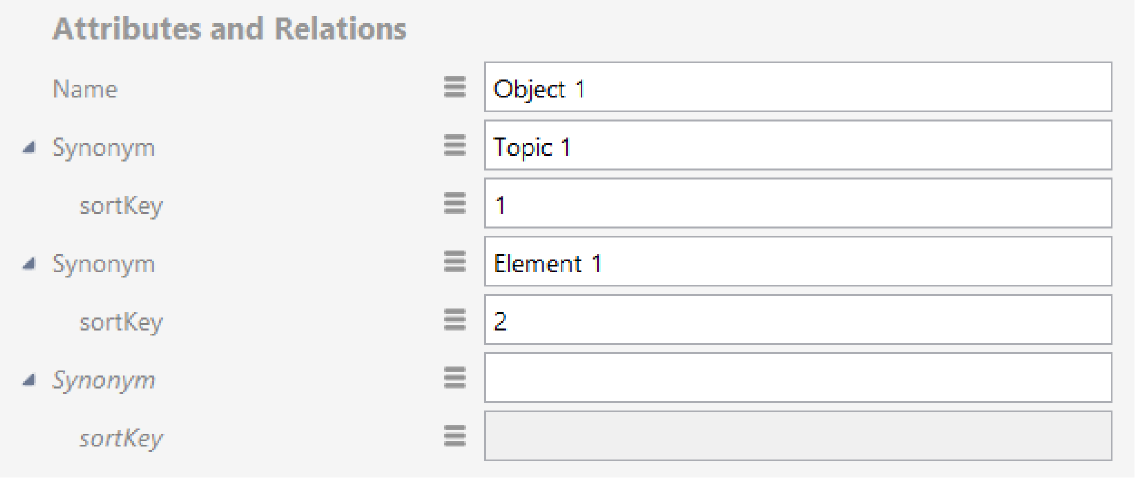


In this example, the properties view configuration already contains the "Name" property. A second property is created by selecting an attribute or a relation for the entry "Property" (marked in orange).

Assorted property display for an object

If an object has several properties of the same type, they will be displayed in alphabetical order by default. If nevertheless the display order of the properties needs to be different (e. g. in order to emphasize preferences for synonyms or for forenames), a dedicated metaattribute can be attached to each property.

The sortKey attribute can be displayed for editing purposes by configuring a meta properties view:

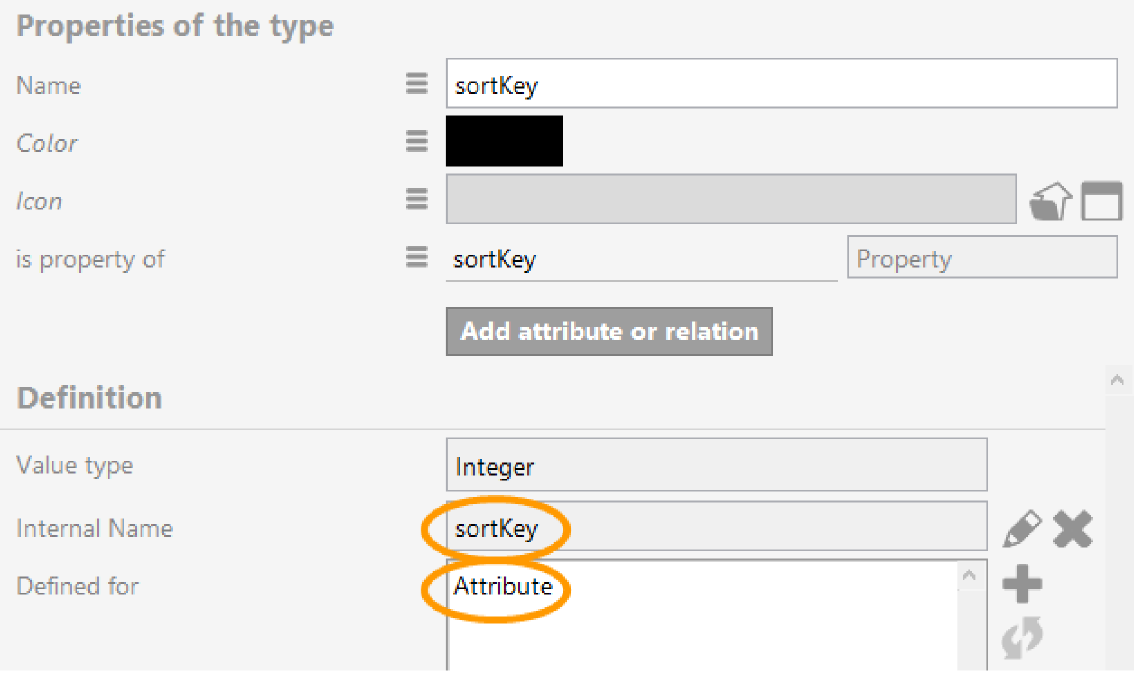


Attributes and Relations

Name	Object 1
Synonym	Topic 1
sortKey	1
Synonym	Element 1
sortKey	2
Synonym	
sortKey	

NOTE In case of the *Synonym* attribute, the value 2 is entered for *sortKey* , so this value is temporarily shown at the end of the list.

For this purpose, an attribute with the internal name '*sortKey*' needs to be defined which can be applied to each individual property:



Properties of the type

Name: sortKey

Color: ■

Icon:

is property of: sortKey | Property

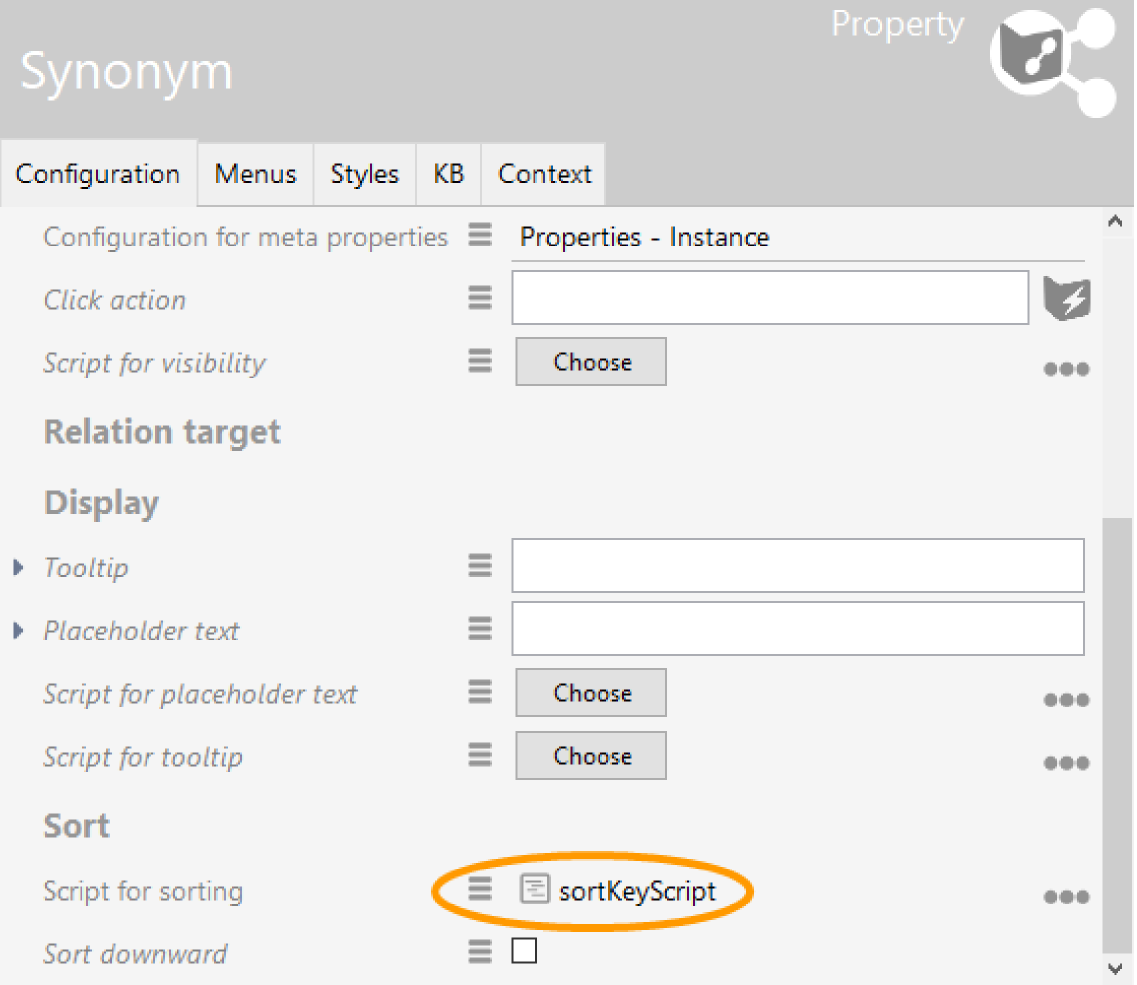
Definition

Value type: Integer

Internal Name: sortKey

Defined for: Attribute

The sortKey attribute is then referenced by a script for sorting which is attached to the property view configuration:



Configuration for meta properties: Properties - Instance

Click action

Script for visibility: Choose

Relation target

Display

Tooltip

Placeholder text

Script for placeholder text: Choose

Script for tooltip: Choose

Sort

Script for sorting: sortKeyScript

Sort downward: □

Example of a *script for sorting*:

```
function sortKey(element)
{
    if (element instanceof $k.Property)
    {
        var attribute = element.attribute('sortKey')
        if (attribute)
        {
            return attribute.value();
        };
    };
    return undefined;
}
```

1.3.4.7. Edit

This configuration type is used to make attributes and relations of a *Properties* configuration editable. For this purpose, it is assigned to the relevant *Properties* element at a higher level. Next to a button for saving changes, a Delete button is displayed next to every property where this is possible.

Setting options

Name	Value
Configuration name	The configuration name is used for identification and reuse of the configuration.
Label	A label is only used if this configuration is embedded in another configuration, e.g. *Alternative*.
Script for label	.
Edit mode switchable	If this option is selected, the properties are first displayed only as a normal list. However, a switch is offered as an addition, which can be used to switch between the normal view and the edit view.
Only custom buttons	If this option is set, the Save button is not displayed.
Script for visibility	.

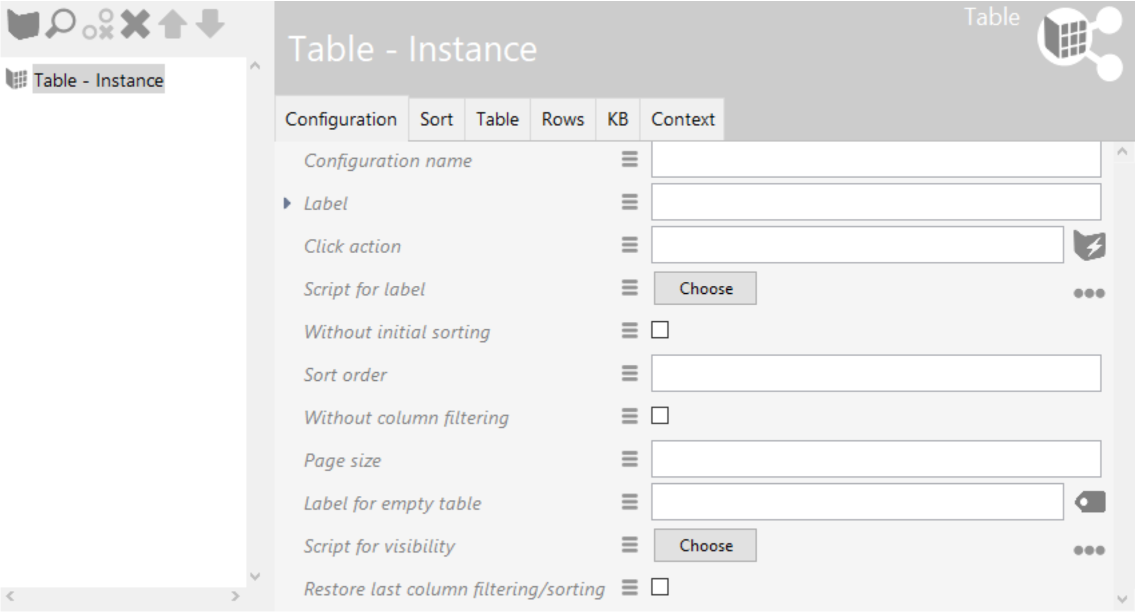
1.3.4.8. Table

Tables can be used as a sub-configuration for displaying results of queries of the configuration type “Query,” or as a separate configuration for displaying the object lists in the Knowledge Builder.

A table lists specific objects, properties or subtypes of a specific type. Whether all objects, properties or subtypes, or only a selection, is displayed, can be managed using the input in the heads of the columns. The values entered are used to execute a structured query according to suitable objects, properties or subtypes and display the result as a table. Moreover, in the case of object lists, a new object, a new property value or a new subtype can be generated with the properties that were filled in after entering values in the heads of the columns.

A subcomponent of the *table* configuration is the *column configuration* . This, in turn, contains *a column element* or a *menu cell* . This layout is used to separate properties relevant to the column (such as order and name of the column in the table) and to assign which contents should be displayed in the column. *Column elements*, in turn, allow the assignment of properties, script modules and structured query modules.

Since version 5.1, not only *column configurations* , but also additional *tables* can be added to a *table* configuration. This provides the option of summarizing frequently used columns in a *table* configuration and add them to another *table* in full. The intermediate tables are removed when determining the overall table. There is only one level of columns.



Configuration name

Label

Click action

Script for label

Without initial sorting □

Sort order

Without column filtering □

Page size

Label for empty table

Script for visibility

Restore last column filtering/sorting □

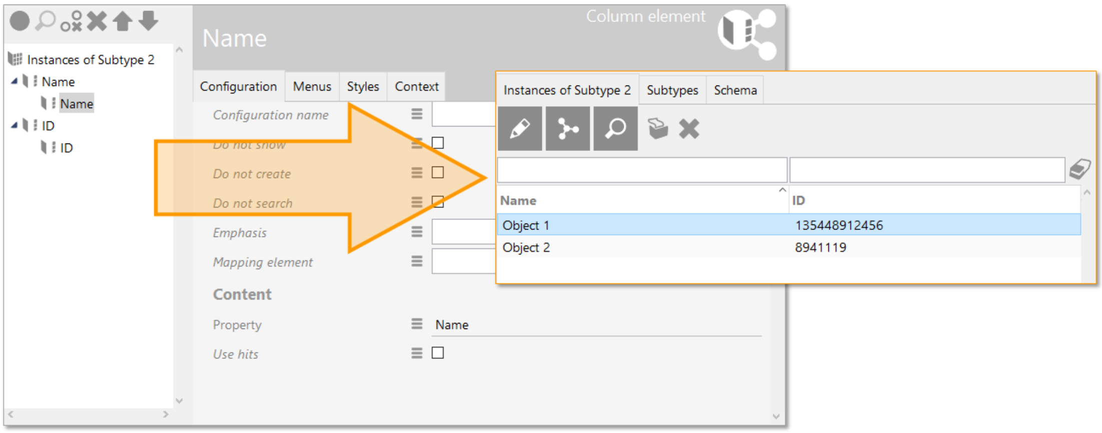
The hierarchical display of all sub-configuration elements in the table configuration exhibits a menu line that is assigned with actions as follows:

◼	Create and link a new subelement.
🔍	Search through all potential subelements that already exist and link (= add) the selected subelement.
○✖	Delete link again. When this occurs, the subelement is retained as an object and can be used again in other configurations.
✖	Delete complete subelement selected. If used in other configurations, a warning will appear before deleting which highlights all existing links.
⬆	Move selected subelement up in the list.
⬇	Move selected subelement down in the list.

NOTE The availability of an action depends on the currently selected table element in the hierarchy on the left side.

Example of a simple table configuration

For a list of objects, certain properties should appear in a table. The name attribute used to represent the objects in the first column should not be forgotten.



Setting options (table)

Name	Value
Configuration	
Configuration name	The configuration name is used for identification and reuse of the configuration.
Label	Defines a static heading for the table.
Click action	Determines an action which is performed when clicking into a table row.
Script for label	Script which returns a string as substitute for the label.
Without initial sorting	No sorting occurs. Default process: the first column is used for sorting.
Sort order	For instance lists in the Knowledge Builder, each table configuration is represented in a separate tab. By specifying an integer, the user can control at which position the tab is displayed, provided that several tables are configured for the same instance type or for the same folder structure element. The tables are sorted using two criteria, which are checked in the following order: 1. Attribute *Sort order* specified: If yes, then this is used as the sort criterion. If no, then the configurations for types are shown first, followed by those for objects. 2. Sorting by display name
Without column filtering (VCM)	Suppresses the indication of column filters in the web frontend. In the Knowledge Builder, column filters are always displayed.
Page size (VCM)	This specifies how many rows (= search result hits) should be display on one page. Default value: 20

Name	Value
Label for empty table (VCM)	A label configuration which is displayed instead of the original label when the table is empty.
Script for visibility (KB)	Script which returns a Boolean for whether the table is visible or not. For instance lists in the Knowledge Builder, the whole tab will not be displayed if visibility is set to false. In the web frontend, this script has no effect.
Restore last column filtering/sorting (VCM)	Restores the recently selected filtering or sort order for the duration of the web frontend session.
Structuring relation	If this table configuration is embedded in another table configuration, all columns in this table refer to the relation targets of the configured structuring relation. If for example the outer table lists persons and the inner table has "owns" as structuring relation, all columns of the inner table refer to the things that a person owns. The configured properties of the relation targets (e.g. all category names of all owned things) are accumulated in the column. If a column element of the inner table determines its values by script or query, the script or query is executed once for every relation target and the results are also accumulated in the column.
Sort	
Column	Column configuration for which the sorting takes effect.
Sort priority	An integer value determines the order by which table column values the assortment of the table rows will be influenced first. **Example:** If an ID is more important for sorting instances than the primary name, the column for ID gets the sort priority 1 and the column for primary name gets sort priority 2. A higher sort priority overrides the sort direction ("Sort downward") of another column.
Sort downward	Determines if the values are sorted upward (alphanumerical order) or downwards.
Table	
Tab "Menus"	For the Knowledge Builder, the menu actions at the top of the table can be configured here. For more information, see chapter "Actions for the Knowledge Builder".
Tab "Styles" (VCM)	For the web frontend, different styles can be applied on the whole table at once.
Rows	
Tab "Styles"	When using a table for the Knowledge Builder, styles can be used for rows of the table for the purpose of character formatting.

Name	Value
KB	
Automatic search	• Automatic search • No automatic search: No automatic search is performed. • Automatic search up to threshold (system settings)
Creating elements without question by name	When this option is enabled, new elements can be created by clicking on the button "New", without a dialog asking for a name before creating the element. As an indication for the missing name, a period "." is shown as name instead.
Script for window title (KB)	A script can be used which returns a string for the window title whenthe table is opened in a separat window.
Script for windowstatus (KB)	A script can be used which returns a string for the bottom line of the Knowledge Builder application or the window (if the table is opened in a separate window).
Without inheritance	If the table is used for instance lists in the Knowledge Builder, only the instances of the currently chosen type are displayed, without instances of subtypes.
Context	
apply to	Restricts the context to the instances of a given element type.
apply to subtypes	Restricts the context to the subtypes of a given element type (instead of instances).
apply in	Application context for within the view is applied. For the table to be displayed within the Knowledge-Builder at all, the application "Knowledge-Builder" must be selected here.
Usage	Within the section "Usage", the "Context of" relation reveals for which view the current element is used as application context. It is the counter part of the relation "apply in" of the other respective element.
Table of	Indicates the superordinate view configuration element within which the table is used.

Actions and styles

Actions and styles can be defined for the entire table, as well as for rows.

Use

The *Context* tab specifies where the table is used.

The object type specified under *Apply to* is the type to which the table should be applied. Tables can be used again in other view configurations. If the table module is a different view configuration, this is displayed under *[inverse] Apply in* .

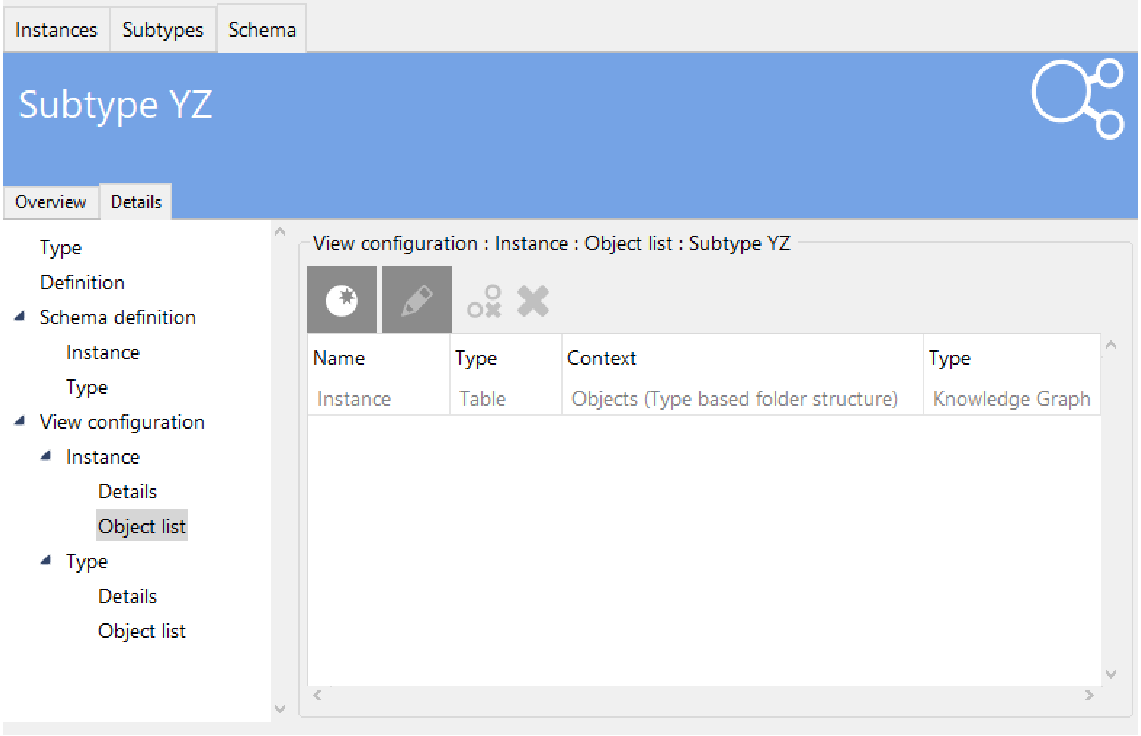
The property *Apply in* refers to an application. Several links are possible.

Examples:

- If the table to the right in the main window in the Knowledge Builder is to be used by the folder structure during navigation, then the table configuration must be linked to the corresponding folder structure element.
- If potential relation targets are displayed as tables in the Knowledge Builder, then the table must be linked with the Knowledge Builder application.

Tables / Object lists in the Knowledge Builder

To configure the way objects or types are displayed in a table in the Knowledge Builder, the *Details* tab contains the section *View configuration* → *Instance/Type* → *Object list* next to the respective type. Creating and maintaining the table configuration is explained using the objects of *Subtype YZ* as an example.



Name	Type	Context	Type
Instance	Table	Objects (Type based folder structure)	Knowledge Graph

No table configuration has yet been linked with this type. The greyed entry shows a standard configuration which is inherited from the upmost type "Knowledge Graph" of the type hierarchy by default. By clicking on the *New* button, a new, blank configuration is generated here. The configuration can then be selected and be edited as needed. As soon as the application context has been specified (e. g. "apply in: Knowledge Builder"), the configuration is applicable after updating the view configuration.

1.3.4.8.1. Column configuration

As mentioned before, *column configurations* contain properties used to define the display and behavior of the column in the table. The column is only displayed once properties are configured in the column elements contained in the column configuration.

Setting options

Name	Value
Configuration	
Configuration name	The configuration name is used for identification and reuse of the configuration.
Label	Displayed in the caption of the column. Please note that the *label* is used for display in the table, but the column configuration also contains the *configuration name* attribute. This name is used only to manage and find the configuration internally and is not displayed or output.
Script for label	As an alternative to the static label text, a script can be used which returns a string for the label.
Bookmark identifier	The bookmark identifier is used to represent a query parameter in forms of an expression within the web frontend URL. It can be used for query views and table column filters and synchronizes parameter value and URL in both directions.
Column width (%)	A percentage value is expected here for the column width (so for 60% you have to enter “60”).
Standard operator	The operator used initially in the search for a search text.
Search string	Preset search text for the column filter.
Do not show	If this value is set, the complete column is hidden. This is used, for example, to sort a search result using hit qualities without displaying them.
Mandatory for query	If this value is set, the column must be filled out for the search to be permitted.
Not sortable	Prevents the table from being sorted when clicking onto the column header.
Script for input field preprocessing	For preprocessing any search text input in the column filter before passed on as parameter for the column element query, a script can be used.
Mapping element	.
Operators	

Name	Value
Configuration name	The configuration name is used for identification and reuse of the configuration.
Symbol	Symbol that will be shown in the dropdown selection of the column filter.
Key	Operator designator that defines which kind of operator is used (e.g. "word" or "containsPhrase"). See the operators explained in the chapter about runtime generated queries.
Label	Tooltip that will be shown in addition to the symbol in case of mouse-over.
Modifier	Name of the indexer string filter.

Menus

For the column, a menu can be configured for the web frontend which is displayed besides the label text at the label (header) of the column.

Styles

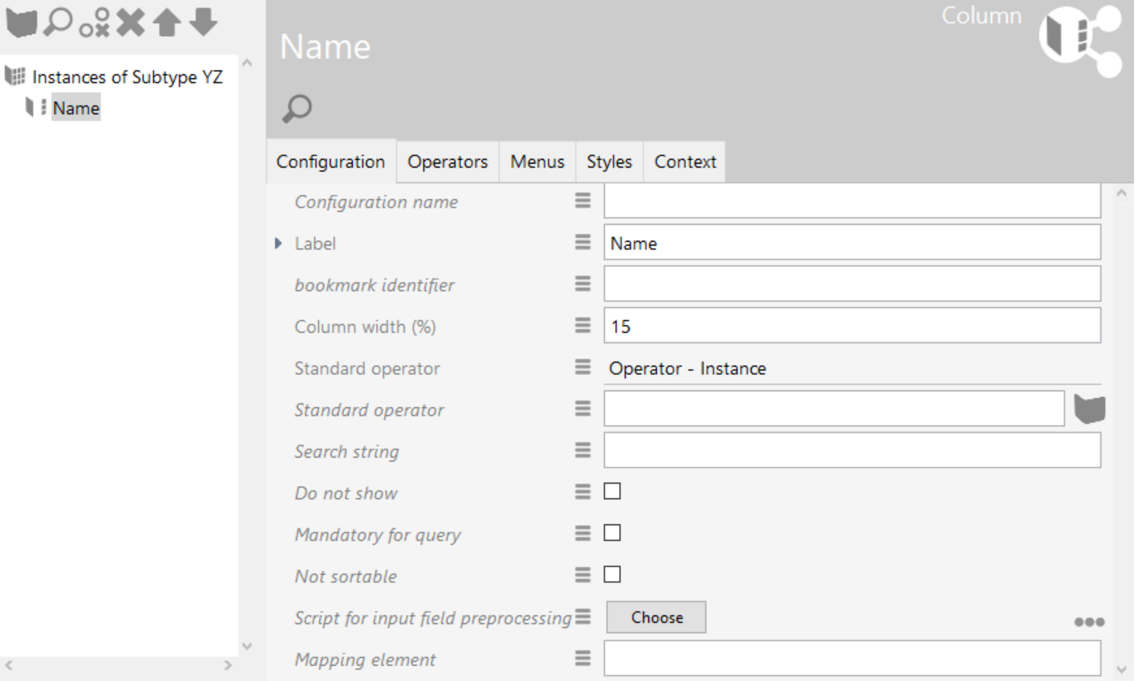
For columns, there are following style settings which can be applied within the view configuration mapper:

hideFilters	Suppresses the column filters from being displayed in the web frontend.
hideLabel	Suppresses the column label from being displayed in the web frontend.

Context

Sub configuration of	Specifies for which table configuration(s) the column configuration is used.
Sort order	Specifies at which order the column is arranged within the table, compared to another column. If there is more than one column with the same sort order, the columns are ordered alphabeticaly by column label.
Sorted column of	Indicates that the column is used for sorting the table content.
Sort priority	Specifies the sort priority of the column used for sorting, compared to other columns used for sorting.

Example



Configuration name	
Label	Name
bookmark identifier	
Column width (%)	15
Standard operator	Operator - Instance
Standard operator	
Search string	
Do not show	□
Mandatory for query	□
Not sortable	□
Script for input field preprocessing	Choose
Mapping element	

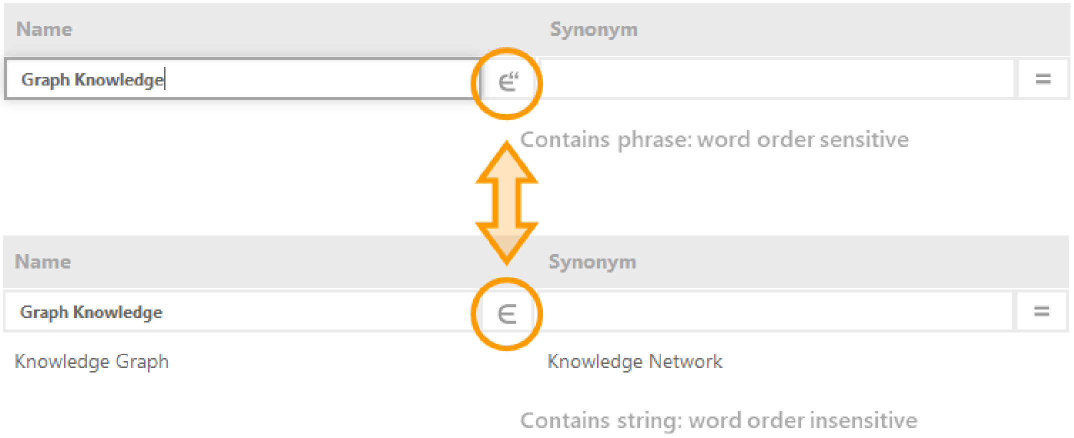
Column configuration for the Name column

1.3.4.8.2. Column operator

The column operator configuration determines which comparison operator can be used in the table view when entering a term into the table filter. In most cases, operators like "equal", "contains phrase" or "contains string" might be needed.

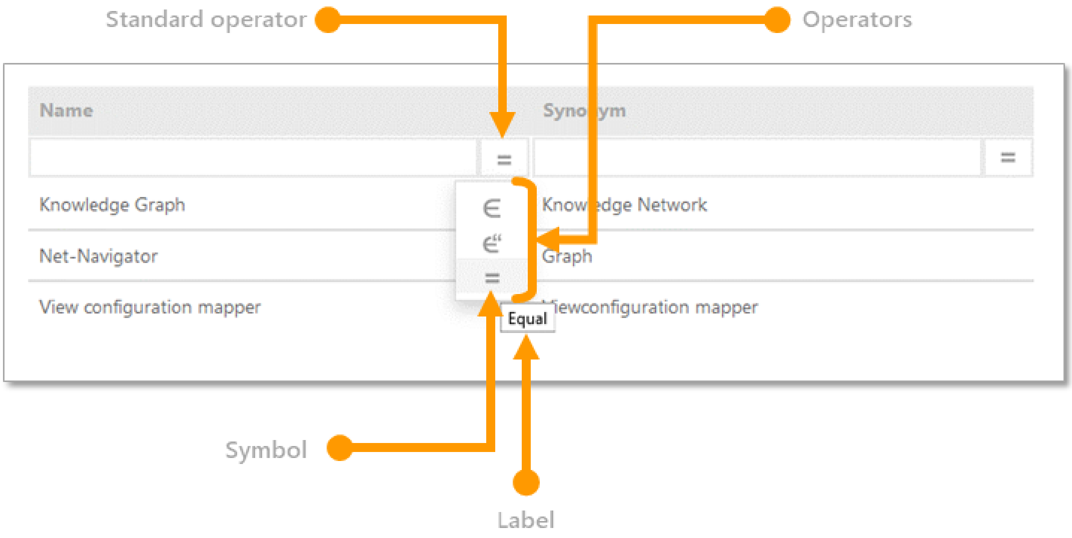
For example, the difference between "contains phrase" and "contains string" is as follows:

- " **contains phrase** ": When entering several words (= phrase) into the filter, only content with the same word order will be found
- " **contains string** ": When entering several words into the column filter, content matching an arbitrary combination of the entered words will be found

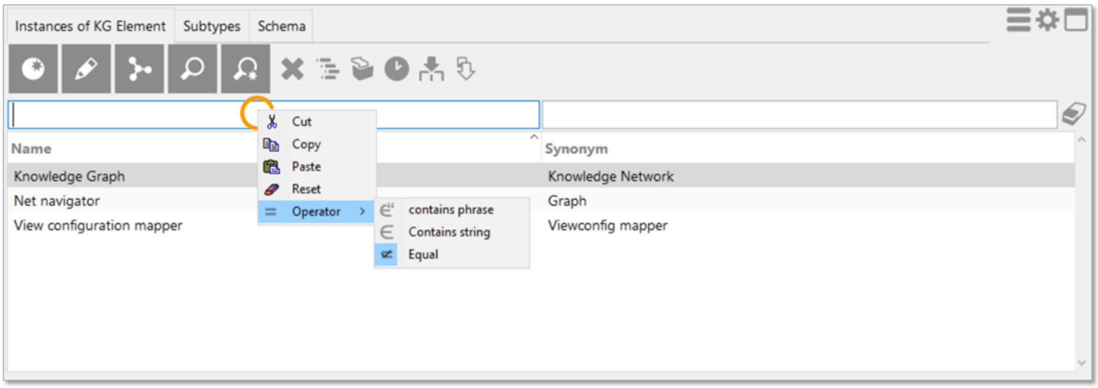


This allows to use different filtering behaviors when filtering large tables to narrow down the search results to specific content.

For all filter operators, a dropdown provides a selection of all operators defined for the respective column:



If the table is used within the Knowledge Builder, a context menu is provided additionally for selecting or removing effective operators:



Name	Synonym
Knowledge Graph	Knowledge Network
Net navigator	Graph
View configuration mapper	Viewconfig mapper

Creating new column operators

New column operators can be created as follows:

Precondition: the respective column element needs to have defined its property to be shown.

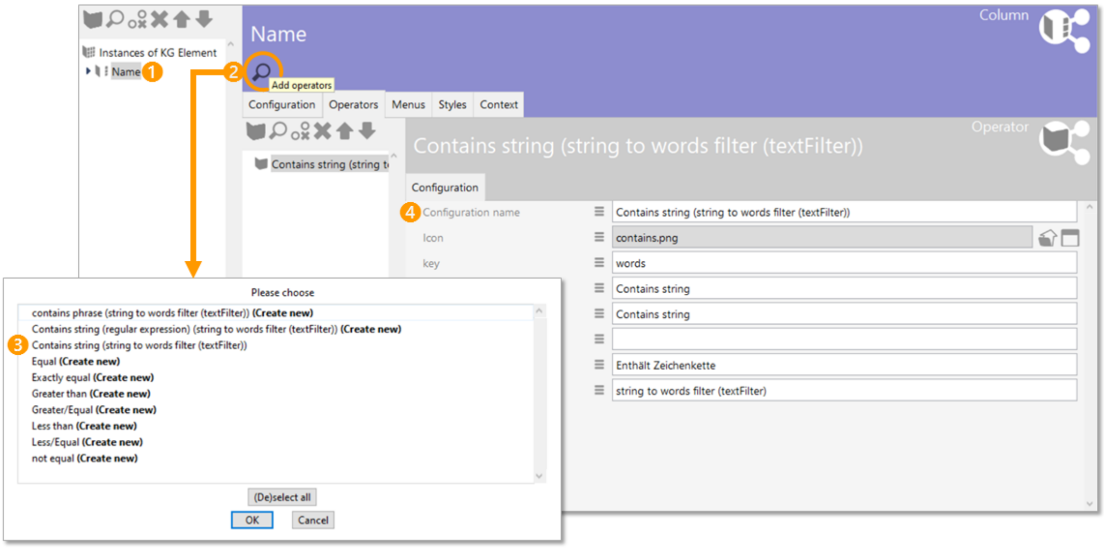
NOTE Since the application of operators depends on the value type of the property to be filtered for and on the indices, the preset operators are only available if the property of the column element has been defined. If string operators are needed, a correctly configured index including index filter is required.

❶ After having specified the property of the column element, select the column itself again.

❷ Click onto the search button: a selection of operator templates will be shown, each applicable on the value type of the property. Operator templates shown with the appendix "Create new" indicate that they are not used until now (no instance has been created from the template).

❸ Select the needed kind of operator.

❹ The "Operator" tab shows the newly created and assigned operator. Each operator listed here will be available for the column filter. Operators can be reused for other table columns.



Contains string (string to words filter (textFilter))

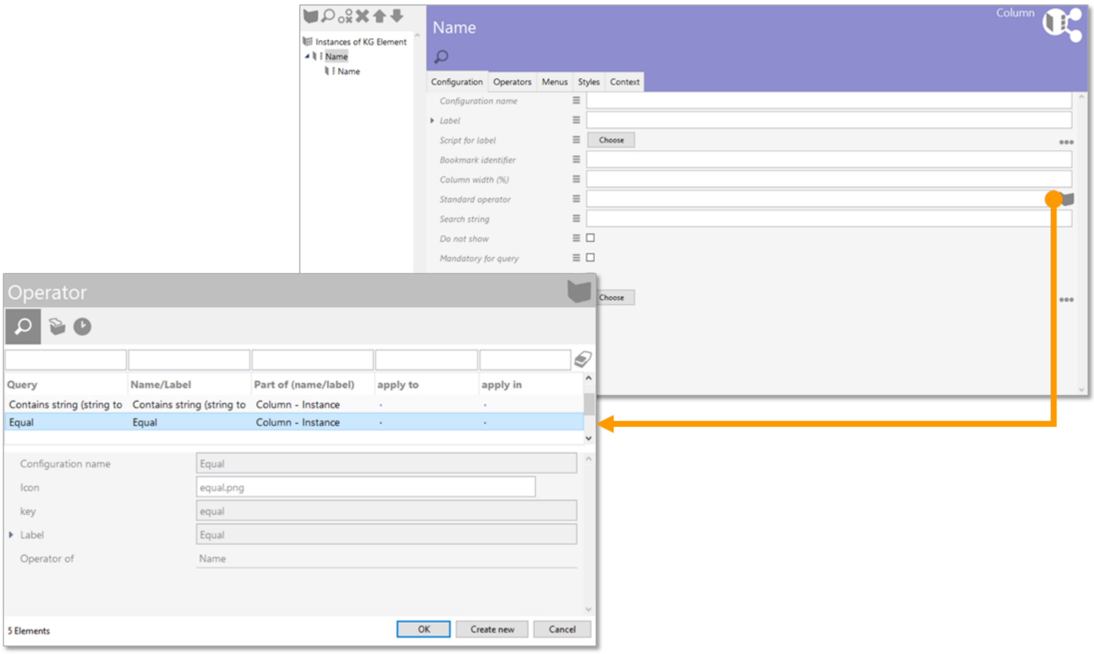
Configuration

Configuration name	Contains string (string to words filter (textFilter))
Icon	contains.png
key	words
	Contains string
	Contains string
	Enthält Zeichenkette
	string to words filter (textFilter)

Please choose

- contains phrase (string to words filter (textFilter)) **(Create new)**
- Contains string (regular expression) (string to words filter (textFilter)) **(Create new)**
- Contains string (string to words filter (textFilter))
- Equal **(Create new)**
- Exactly equal **(Create new)**
- Greater than **(Create new)**
- Greater/Equal **(Create new)**
- Less than **(Create new)**
- Less/Equal **(Create new)**
- not equal **(Create new)**

❺ For the default operator, switch to the "Configuration" tab and select one of the operators:



Query	Name/Label	Part of (name/label)	apply to	apply in
Contains string (string to	Contains string (string to	Column - Instance	·	·
Equal	Equal	Column - Instance	·	·

Configuration name	Equal
Icon	equal.png
key	equal
Label	Equal
Operator of	Name

5 Elements

NOTE Within the Knowledge Builder, the standard operator will not be shown in the respective column filter, but it is active when no other operator has been selected in the context menu.

Operators also can be defined by yourself. For the operator, following properties can be specified:

Property	Description	Value type
Configuration name	The configuration name is used for identification and reuse of the configuration element.	String
Icon	The icon which will be shown in the filter and its dropdown selection. **NOTE** Without further plugins, vector images like *.svg cannot be used for configuration elements within the Knowledge Builder.	Blob
key	The operator key for the operator. See table below.	String
Label	Text for the tooltip which will be shown at the symbol in case of mouse-over.	String
modifier	Name of the index filter.	String

Operator keys

Operator name	Description	Short term
containsPhrase	Contains phrase	
covers	contains	
distance	Distance	
equal	Equal	==
equalBy	Corresponds to	
equalCardinality	Equal cardinality	
equalGeo	Equal (geo)	
equalMaxCardinality	Cardinality smaller than or equal to	
equalMinCardinality	Cardinality greater than or equal to	
equalPresentTime	now (present)	
equalsTopicOneWay	filter with	
fulltext	Contains string	
greater	Greater than	>
greaterOrEqual	Greater/equal	>=
greaterOverlaps	Overlaps from above	
greaterPresentTime	after now (future)	

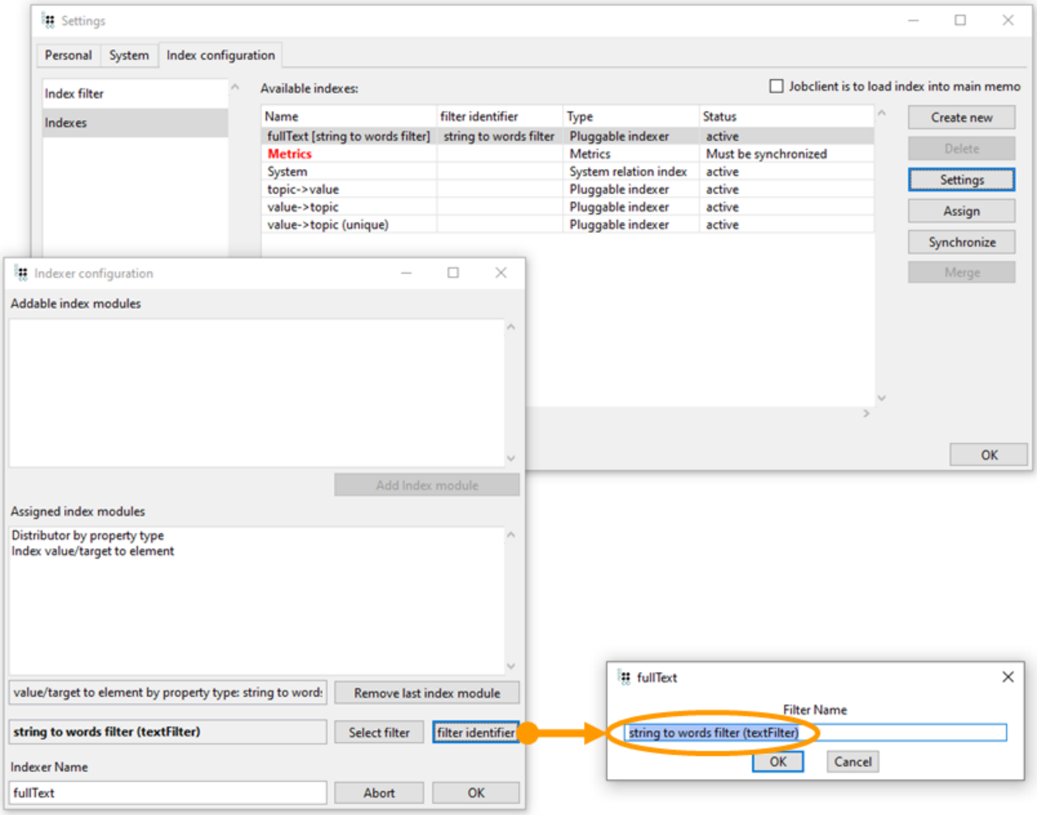
Operator name	Description	Short term
isCoveredBy	is contained in	
less	Less than	<
lessOrEqual	Less/equal	⍰
lessOverlaps	Overlaps from below	
lessPresentTime	before now (past)	
notEqual	Not equal	!=
overlaps	overlaps	
range	Between	
regexEqual	Regular expression	
regexFulltext	Contains string (regular expression)	
unmodifiedEqual	Exactly identical	
words	Contains string	

Modifiers

For using operators like "Contains phrase", the respective operator key like "containsPhrase" requires a modifier which depends on an index filter.

Index filters are used within an index. The index configuration is done in the global settings of the Knowledge Builder: Settings > Index configuration.

Within the configuration of the index, the name of the assigned index filter can be specified and copied for using as modifier:



Available indexes:

☐ Jobclient is to load index into main memo

Name	filter identifier	Type	Status
fullText [string to words filter]	string to words filter	Pluggable indexer	active
Metrics		Metrics	Must be synchronized
System		System relation index	active
topic->value		Pluggable indexer	active
value->topic		Pluggable indexer	active
value->topic (unique)		Pluggable indexer	active

Addable index modules

Assigned index modules

Distributor by property type
Index value/target to element

value/target to element by property type: string to word:

string to words filter (textFilter)

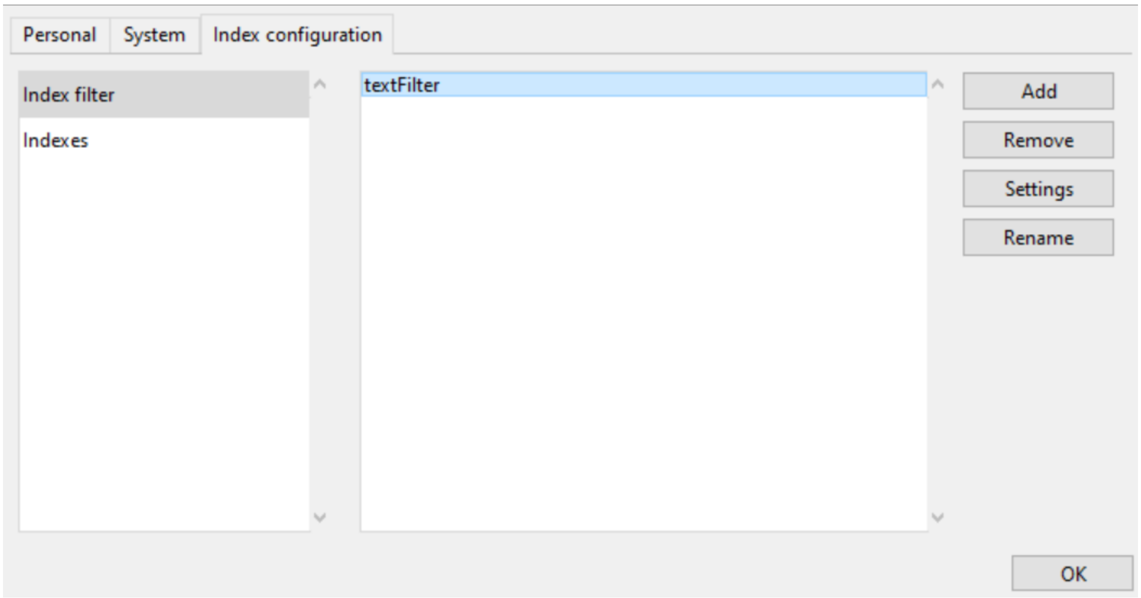
Indexer Name

fullText

Filter Name

string to words filter (textFilter)

New index filters are defined within the main settings of the Knowledge Builder: Settings > Index configuration > Index Filter



textFilter

1.3.4.8.3. Column element

A *column element* is used to assign the content that a table column is supposed to show, and how that should take place. You can either specify properties, such as attributes and relations, that are defined by the semantic objects, or you can use structured query modules or script modules.

Setting options

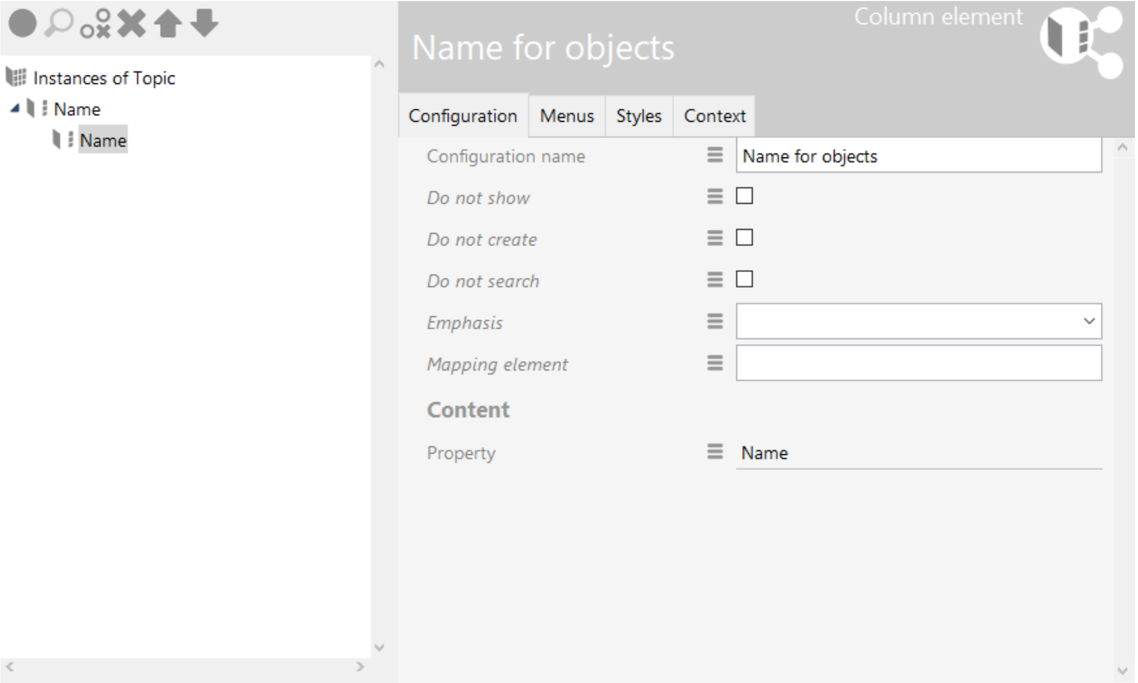
Name	Value
Configuration	
Configuration name	The configuration name is used for identification and reuse of the configuration.
Do not show	Use this Boolean attribute to control whether the values of the selected property should be displayed. By default all properties are displayed.
Do not create	This attribute controls whether this property is supposed to be created when a new object is created if the relevant input field in the column contains a value. By default new properties are created.
Do not search	Here you can specify that the configured property is not transferred to the search. This means that this property is not used to search for the entered search values. **NOTE** If all column elements in a column are set to "Do not search", this has the same effect as "Do not show"!
Emphasis	Here you can provide formatting specifications for the display of values; currently, the only available option is *underline*.
Relation target view	Currently only the *Drop down* alternative is available. If you select it, the possible values that can be entered for this column for filtering in this table are compiled from the possible relation targets as per the schema into a drop-down list, so that a possible value can be specified quickly. This is recommended for manageable amounts of potential relation targets. **NOTE** This parameter is only available if a Relation type property was selected.
Content	Properties in this group determine the content of the table cell. Most of the following options are mutually exclusive, indicated by a *.
Property*	Link to the property type that is to be displayed.
Structured query element*	A structured query can be used to determine the property.

Name	Value
Script*	A script can be used which returns the cell values to be displayed (a property, hit, element or primitive value).
Mapping element*	.
Show name*	Show the name of the row element, independently of which attribute type is defined as name.
Quality*	For the web frontend, this option displays a bar showing the hit quality incl. percentage value.
Show size	Instead of the properties that are determined by any of the above methods, show only the number of properties.
Use structuring relation	Modifies all of the above determination methods such that they refer to the structuring relation that is defined for this embedded table instead of the relation target (for structuring relations see the section on general table configuration).

It is possible to define multiple column elements for a column configuration. This makes sense, for example, if multiple attributes are to be considered in the search, for example the Name and Synonym attributes, but only one of them is to be displayed.

Example

The *Name* attribute is configured in the first column element of the *Name* column configuration.



Name for objects

Configuration name: Name for objects

Do not show ☐

Do not create ☐

Do not search ☐

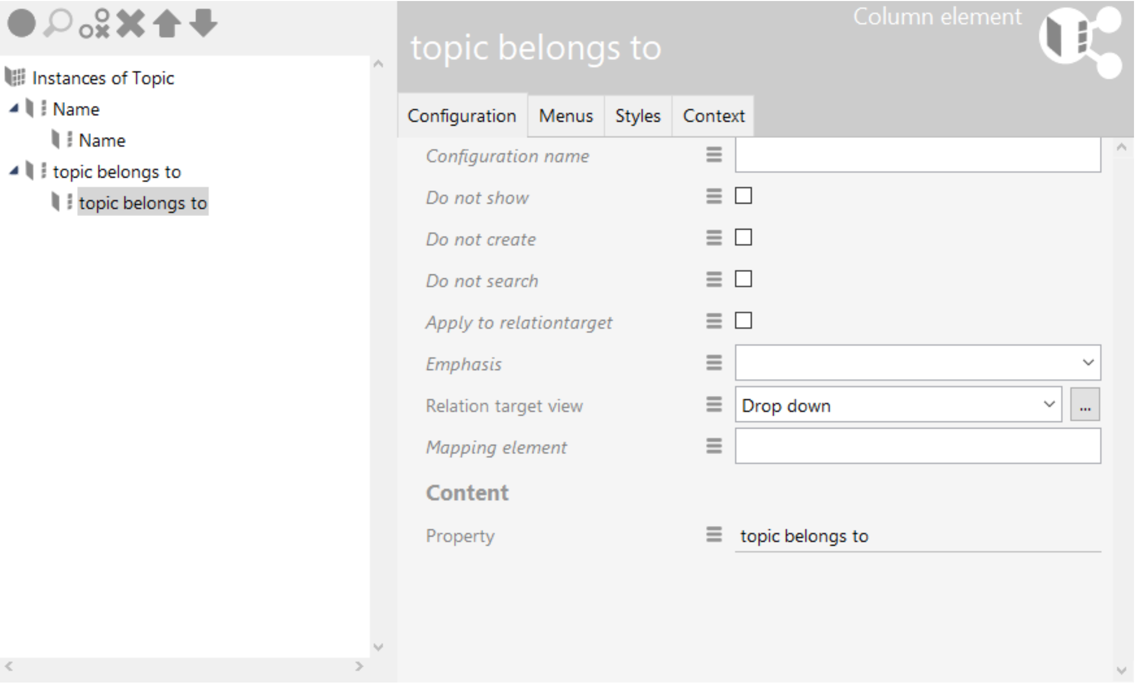
Emphasis

Mapping element

Content

Property: Name

The *Topic belongs to* relation is configured in the column element of the second column.



Instances of Topic
- Name
  - Name
- topic belongs to
  - topic belongs to

Column element

topic belongs to

Configuration name

Do not show □

Do not create □

Do not search □

Apply to relationtarget □

Emphasis

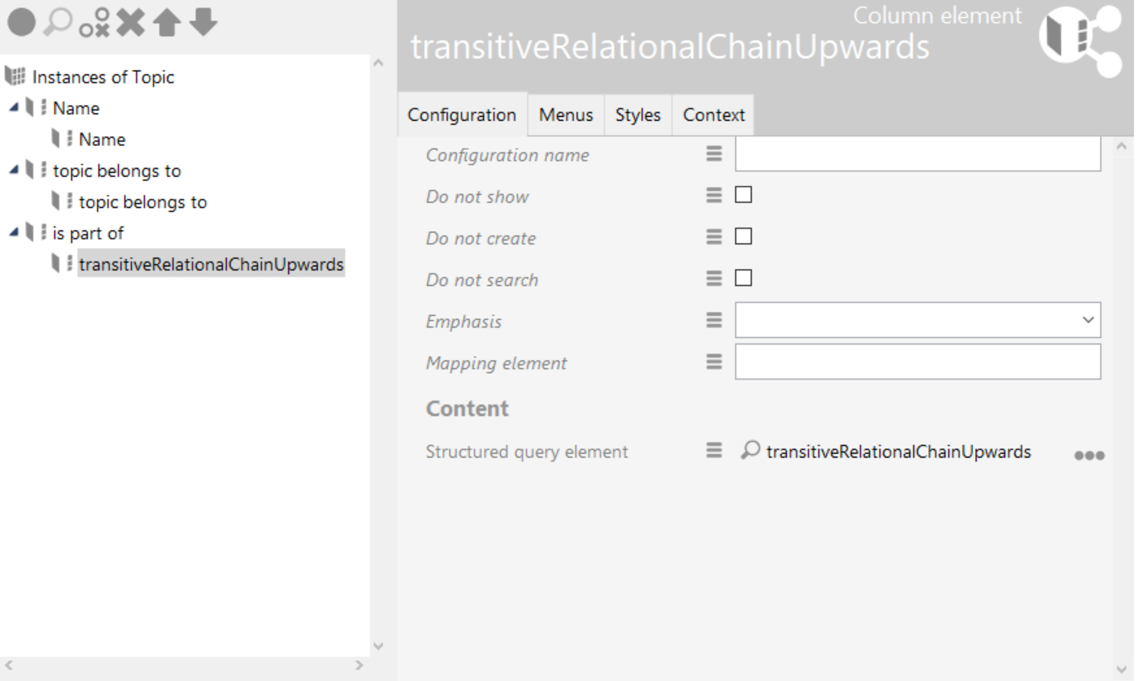
Relation target view: Drop down

Mapping element

Content

Property: topic belongs to

The *transitiveRelationalChainUpwards* structured query module is configured in the column element of the third column.



Instances of Topic
- Name
  - Name
- topic belongs to
  - topic belongs to
- is part of
  - transitiveRelationalChainUpwards

Column element

transitiveRelationalChainUpwards

Configuration name

Do not show □

Do not create □

Do not search □

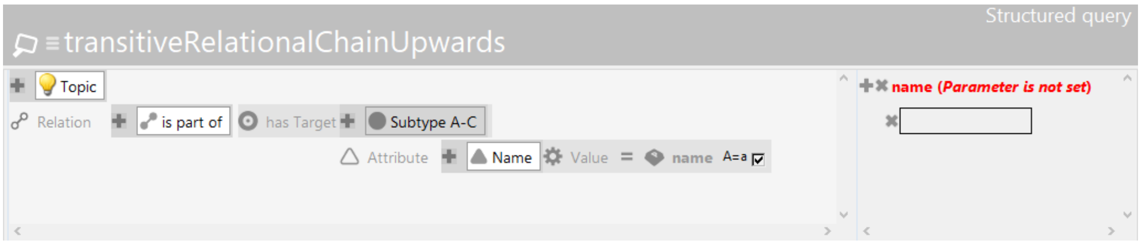
Emphasis

Mapping element

Content

Structured query element: transitiveRelationalChainUpwards

Related structured query:



Structured query

transitiveRelationalChainUpwards

Topic

Relation is part of has Target Subtype A-C

Attribute Name Value = name A=a

name (Parameter is not set)

To make it possible to adopt values from the input field of the column, the structured query must have configured parameters. Multiple parameters can be applied, all of which are assigned the same value when the structured query is evaluated.

NOTE

This is different from other cases in which the structured query is used. Normally the results are determined by the initial object (in this case "Topic"). In this case, the results are determined by the objects or properties to which the parameter is attached (in this case the name attribute).

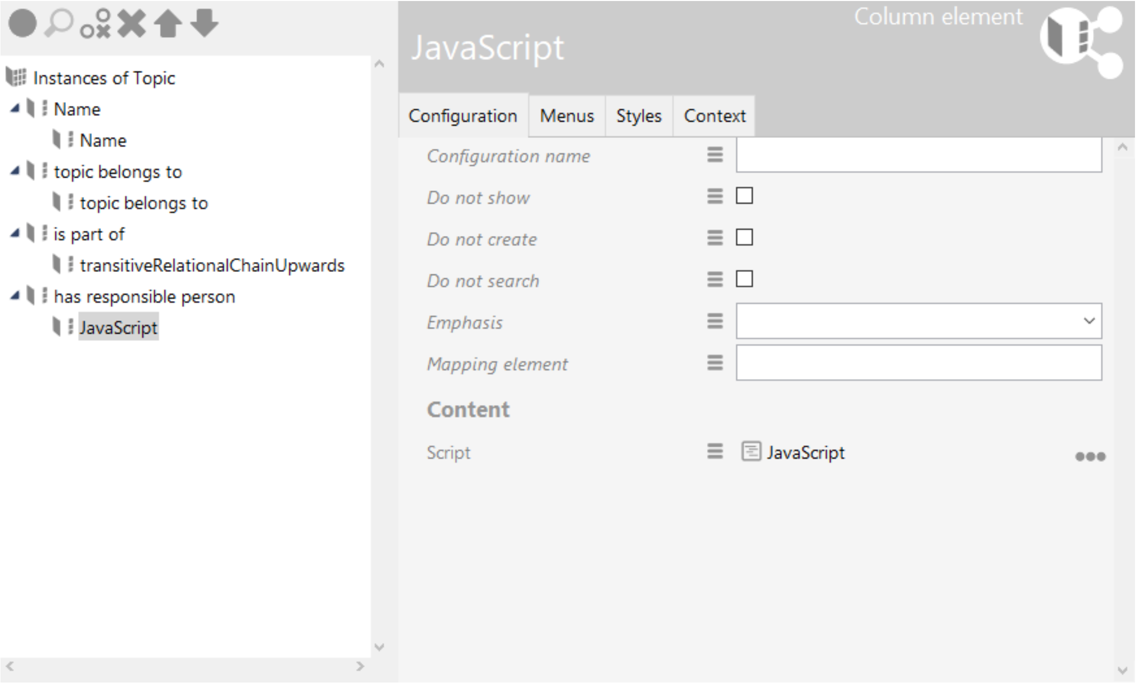
Unless further changes are made, the value displayed in the column is the value of the attribute used for filtering. If the displayed value does not result from the attribute used for filtering, there are two options:

- The "*renderTarget*" identifier can be attached to relation targets. Objects marked in this way are displayed in the table as the column value. “renderTarget” also has the effect that, during output via the JavaScript API, the properties relating to display are included in the output as a link.
- The identifier "*renderProperty*" can be attached to attributes. Properties marked in this way are displayed in the table column as the column value.

If the search module is not used for filtering, the element to be displayed must be determined by means of a manually defined parameter or by means of predefined parameters like renderTarget/renderProperty!

The structured queries that can be included in the module of the column element can be selected from a list of structured queries that have already been registered, but it is also possible to create new structured queries for exactly this module, which includes the allocation of a registration key. The *Do not create* property does not affect columns that have been assigned a structured query module.

A script module is mapped to the fourth column



Configuration name

Do not show ☐

Do not create ☐

Do not search ☐

Emphasis

Mapping element

Script: JavaScript

The aim is to display the persons responsible for the objects to which the topic listed in the table is linked by means of *Topic belongs to* . As with the structured query, it is possible to select the assigned script from a list of registered scripts or to create (and register) a new one in the dialog. The script editor opens when you click the script module name.

```
/*
 * Returns matching elements for column search value "objectListArgument"
 * Note: "elements" may be undefined if no partial query result is
available.
 * Return undefined if the script cannot provide any partial result
itself.
 */
function filter(elements, queryParameters, objectListArgument) {
  return elements;
}

// Returns cell values rendered as topics for the given element
// For cell values rendered as Hits, use renderHits() instead
function renderElements(element, queryParameters) {
  var result = new Array()
  var firstTargets = element.relationTargets("isTopicOf")
  if ( firstTargets.length == 0 )
     return result ;
  else {
     for (var i = 0; i < firstTargets.length; i++) {
         var secondTargets = firstTargets[i].relationTargets(
"hasResponsiblePerson")
```

```
         for (var j = 0; j < secondTargets.length; j++)
             result.push(secondTargets[j].name())
         }
     }
  return result.join(', ');
}
```

In this case the language of the script module is JavaScript. Two parts have to be maintained here: the upper part is used to filter all elements in the table on the basis of the *objectListArgument* value entered in the column, while the second part specifies how the value to be output for an element is calculated. This first part has not been described as yet. A code pattern is added to both parts during creation, and it can be built upon during creation.

If KScript was selected as the language in the script module for controlling the output of a column, the selected (registered) script must provide a return value for the column for every object that forms a row.

As KScript is in principle designed for only one output, the following convention has been reached for filtering:

If the selected script contains a function named *objectListScriptResults* and a declared parameter, this function is called with the argument of the corresponding search input in order to return the set of matching objects. The function is called as the initial object on the root term or the former hit list - depending on the best way to resolve the search. To make this version truly efficient, it is recommended to evaluate the search inputs accordingly and use the result to call a registered structured query in order to forward its result to the object list.

1.3.4.9. Query

The user can use the view configuration element “Query” to configure query options for the Knowledge Graph. The query can either be a predefined query with parameters, or be a search field input screen for the user.

The “Query” can be selected as a sub-configuration of an *alternative* or a *layout*. Any type of query is obligatory here, the results of which are displayed. Searches for user inputs can also be configured; instead of the configuration element “Query” (object configuration), the configuration element “Search field element” is used for the view configuration. Examples of the panel configuration for the web front-end can be found in chapter 3 “ViewConfiguration Mapper”.

When a search is to be configured for the web frontend containing facets, then the functional chain should be observed in the case of panel influencing: Query *or* Search field element -→ Facet -→ Search result.

Setting options

Name	Value
Configuration name	The configuration name is used for identification and reuse of the configuration.
Label	A label is only used if this configuration is embedded in another configuration, e.g. *Alternative* .
Script for label	The label can, alternatively, be determined using a script.
Bookmark identifier	The bookmark identifier is used to represent a query parameter in forms of an expression within the web frontend URL. It can be used for query views and table column filters and synchronizes parameter value and URL in both directions.
Table	A table configuration is specified here which is used to display the search results.
Script for table configuration	The table can also be determined using a script.
Query	A search can be selected here that is executed as soon as the configuration element is displayed. The semantic object, for which the view configuration is displayed, can be used as an accessed element in the query.
Script for visibility	A script can be used to control whether the configuration element should be displayed.

Setting options for a query

The following parameters are maintained as meta properties for a *query* .

Name	Value
Parameter name	Specifies a parameter name that is to be used in the query.

Setting options for a parameter name

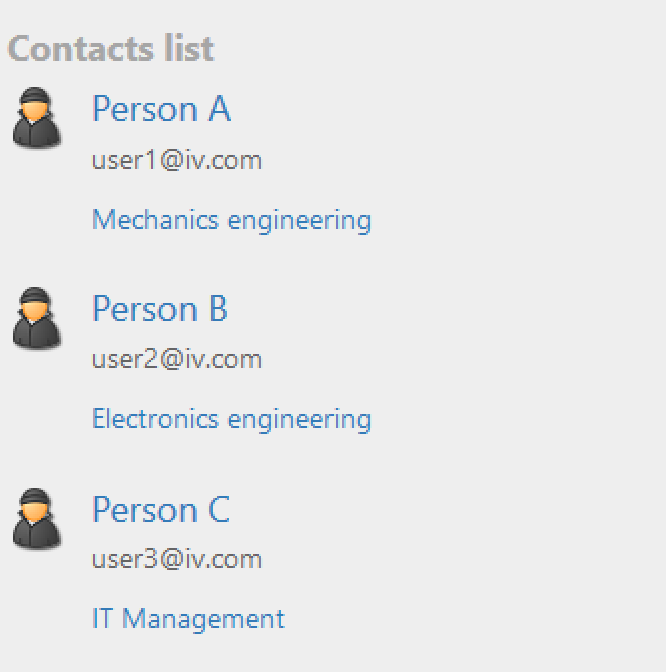
The following parameters are maintained as meta properties for a *parameter name* :

Name	Value
Script for value determination	The script with the function *parameterValue* is used for determining the search value for the specified parameter name.
Script for parsed value	

Name	Value
Value determination	Specifies the value determination path. • **Script:** The value is determined from the script and must not be overwritten by the user. • **Script, overwritable by user input:** The script determines the value. The user may overwrite it. • **User input** : No script evaluation. User input only.
Value disposition	.
Type	xsd-type
Label	During output to JSON, this value ends up in *label.*
Bookmark identifier	.
Tooltip	.
Query for proposed values	.
Script for proposed values	.
Sort Order	.

Display in an application

Query results are output in a table by default.



Contacts list

Person A

user1@iv.com

Mechanics engineering

Person B

user2@iv.com

Electronics engineering

Person C

user3@iv.com

IT Management

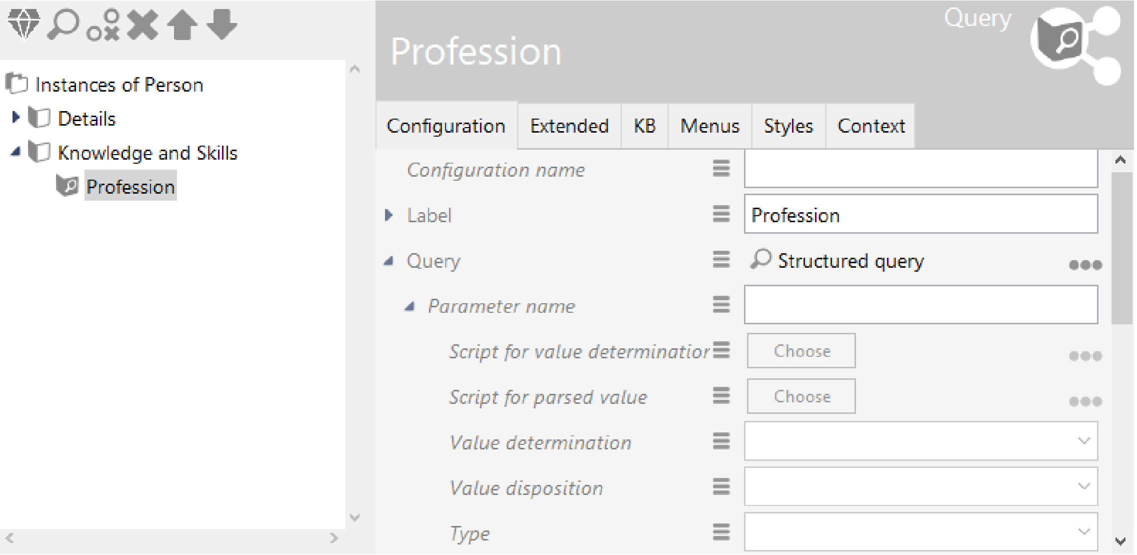
In this example, query results are output in the web frontend as a table view “mediaList” render mode style. The “mediaList” render mode converts the typical table view into a sizable list with an *icon and link to the objects. Additional properties of the object can be specified by means of further column elements (in this case, the email address as an attribute and the profession as a relation target of persons).*

Instead of using the individual configuration element "Query" for the Web-Frontend, searches can be split into the separate configurations "Query" and "Search result view".

Display in Knowledge Builder

The results of any query are always shown in an object list in the Knowledge Builder.

Example:



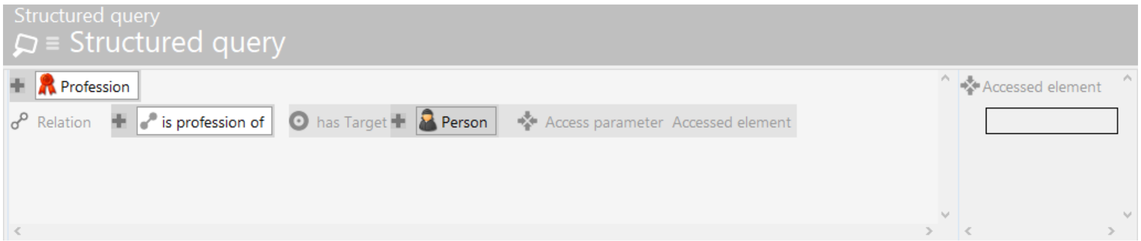
Instances of Person
Details
Knowledge and Skills
Profession

Query
Profession

Configuration | Extended | KB | Menus | Styles | Context

Configuration name
Label: Profession
Query: Structured query
Parameter name
Script for value determination: Choose
Script for parsed value: Choose
Value determination
Value disposition
Type

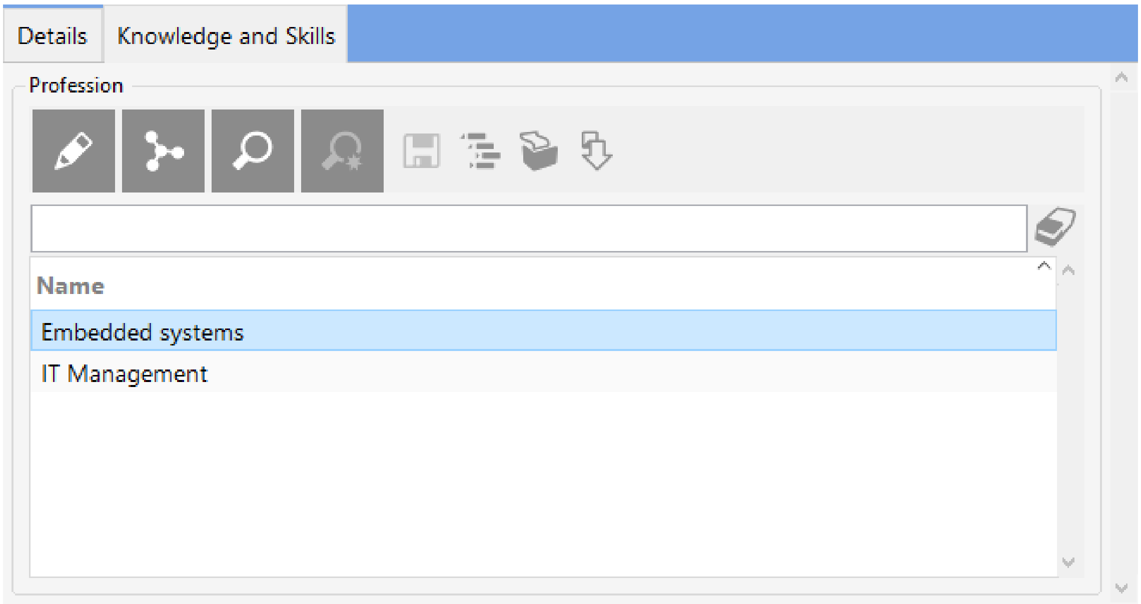
The "Details" and "Knowledge and Skills" tabs are defined in the view configuration. "Profession" is a configuration element of the type "Search". An existing query can be selected or a new one be created directly, under "Query".



Structured query
Structured query

Profession
Relation: is profession of — has Target: Person — Access parameter Accessed element
Accessed element

Definition of the search



Embedded systems

IT Management

The result of the query is displayed in the "Knowledge and Skills" tab in the Knowledge Builder for objects of the type "Person".

1.3.4.10. Graph

The contents of the Knowledge Graph are plotted in a graph with their objects and connections (*see chapter Knowledge Builder > Basics > Graph editor*).

Setting options

Name	Value
Configuration name	The configuration name is used for identification and reuse of the configuration.
Label	A label is only output if this configuration is embedded in another configuration, e.g. *Alternative* .
Script for label	A script that returns the label.
Graph configuration	A graph configuration object is defined here.
Height/width	This defines the width and height of the configuration element, either as a percentage or exact to the pixel.
Hide legend	This defines whether the legend for the node types is to be displayed.
Initial topics query	Query which determines the semantic elements which are displayed initially when the graph is displayed.
Initial topics script	Script which determines the semantic elements which are displayed initially when the graph is displayed.
Script for visibility	The visibility of the configuration element can be defined in a script referenced here.

1.3.4.10.1. Graph configuration

The graph configuration only allows specific types and relations to be displayed in the graph. This prevents unwanted types and relations from appearing in the graph. The graph configuration can also be queried using JavaScript functions. It is, for example, used in the Net-Navigator.

Node category elements are subordinate to a graph configuration.

Setting options

Name	Value
Configuration name	The configuration name is used for identification and reuse of the configuration.
Label	A label is only used if this configuration is embedded in another configuration, e.g. an additional *alternative* .
Maximum node distance	Integer value which determines the maximum displayable amount of nodes across their links; thus determining the longest possible graph path distance.
Maximum node age	Integer value for maximum amount of steps after which the first nodes are faded out when links are expanded.

1.3.4.10.2. Node category

Node categories are subordinated to graph configurations.

They are assigned subordinate link elements.

Setting options

Name	Value
Configuration	
Configuration name	The configuration name is used for identification and reuse of the configuration.
Label	Label which is used for the legend of the nodes in the web frontend. This option has no effect within the Knowledge Builder.
Script for label	Script that returns an element name or a string for the label instead of using the label attribute.
Adapt to specific type	When this option is enabled, only the subtypes will be displayed as legend instead of the overall supertype.
Hide abstract types	This option prevents abstract types from being displayed in the legend.

Name	Value
Show in legend	This option is for the net navigator in the web frontend only: • If needed: The legend for the node at top of the graph is only shown when the node is existent within the graph. • Always: The legend is shown disregarding the nodes being shown in the graph. • Never: The legend is never shown, even if the respective node is shown in the graph.
Icon	Icon which is displayed for the node category in the graph exclusively. When no icon is specified, the (inherited) icon of the respective semantic element of the Knowledge Graph is shown. When no icon is specified at all, types are shown in forms of colored rings and objects are shown in forms of colored and filled circles.
Script for icon	Script which returns the icon for a node category instead using the icon attribute. Return value is a blob attribute or a value of the type $k.Blob.
Expand extensions initially	When this option is enabled, extensions are expanded initially when the core object is displayed in the graph.
Color	Color assigned to the nodes of this category. This affects the coloring of the node circles and of the legend.
Script for color	Script which returns the color assigned to the nodes of this category instead of using the color attribute. Return value is a hexadecimal color value.
Category	
Menus	.
Nodes	
Menus	When displaying the graph in the web frontend (net navigator), actions can be added for nodes as follows: • Node satellite menu buttons for expanding, hiding and pinning the node. • Action being executed when clicking onto the node itself. For further information, see the respective chapter *View Configuration Mapper > Viewconfig elements > Graph configuration*.
Context	

Name	Value
Apply to	Determines for which instances or types the node category is applied. One node category can be assigned to several different instances or several different types.

1.3.4.10.3. Link

Links are subordinate to a node category. They represent the edges of the graph, thus the relations of the Knowledge Graph.

Setting options

Name	Value
Configuration name	The configuration name is used for identification and reuse of the configuration.
Label	The label attribute is not effective for links. Use the script for label instead.
Script for label	Returns an element name or a string for labeling the link.
Color	Determines the color of the link.
Query for link	Query which determines the target element of the link, based on the origin which is the superordinate node element.
Relation for link	Relation which is used for the link. The definition range of the used relation type needs to comprise the type od the related node element.
Script for link	A script referenced here can be used to define the link. Return value is a relation at the semantic element of the node.
Initially expanded	If this option is enabled, the link will be expanded automatically as soon as the node element is initially displayed.
Preferentially expanded	If a node element has several links which are set to expand initially, this option can be enabled for prioritizing one of the links.

1.3.4.11. Text

This configuration element outputs a simple text. This is either configured fixed or determined via a script.

Setting options

Name	Value
Configuration name	The configuration name is used for identification and reuse of the configuration.
Label	A label is only output if this configuration is embedded in another configuration, e.g. *Alternative* .
Script for label	A script that returns the label instead of using the label attribute.
Text	Text that is to be output.
Script for text	A script that returns the text to be displayed.
Script for visibility	A script that returns a Boolean value for whether the view is to be displayed or not.

1.3.4.12. Image

Static graphics can be integrated with the aid of this configuration element.

Name	Value
Configuration name	The configuration name is used for identification and reuse of the configuration.
Label	A label is only output if this configuration is embedded in another configuration, e.g. *Alternative* .
Script for label	Alternatively, this can be used to determine the label using a script.
Image	The image file that is to be output.
Script for image	Alternatively, the graphics can be returned using a script. Not applicable within the Knowledge Builder.
Height/width	Scales the image file to the dimensions specified.
Script for visibility	A script is used to determine whether the graphics are to be displayed.

1.3.4.13. Script generated view/HTML

Script-generated view

A view created using a script saved in the Knowledge Graph. This is written in JavaScript and can use a custom template (a Ractive.js “partial”). This allows complex views to be created, which extend beyond the functionalities of the standard view configuration.

Setting options

Name	Value
Configuration name	The configuration name is used for identification and reuse of the configuration.
Label	A label is only used if this configuration is embedded in another configuration, e.g. *Alternative* .
Script for label	Script for determining the label.
Script	Script for generating the view.
viewType	Name of the partial.
Script for visibility	Script for determining the visibility. Return value is a Boolean value.

Script-generated HTML

This view configuration shows an HTML fragment that is generated using a script stored in the Knowledge Graph. In it, the JavaScript API of i-views is used to access semantic elements and their properties and an XML writer object generates an HTML structure and fills it with data.

Setting options

Name	Value
Configuration name	The configuration name is used for identification and reuse of the configuration.
Label	A label is only used if this configuration is embedded in another configuration, e.g. *Alternative* .
Script for label	Script for determining the label.
Script	Script for generating HTML output.
Script for visibility	Script for determining the visibility.

Example of a script that generates simple HTML output:

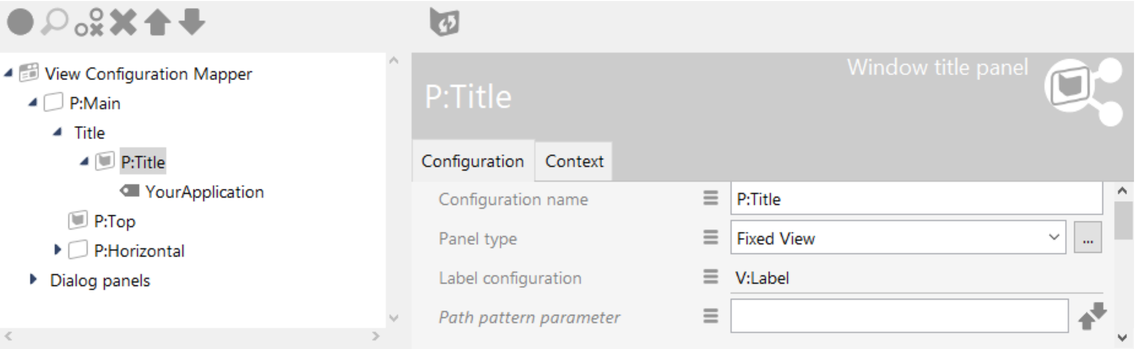
```
function render(element, document) {
  var writer = document.xmlWriter();
  writer.startElement("div");
    writer.startElement("h2");
      writer.cdata(element.name());
    writer.endElement();
  writer.endElement();
}
```

Output:

```
<div>
  <h2>Hermann</h2>
</div>
```

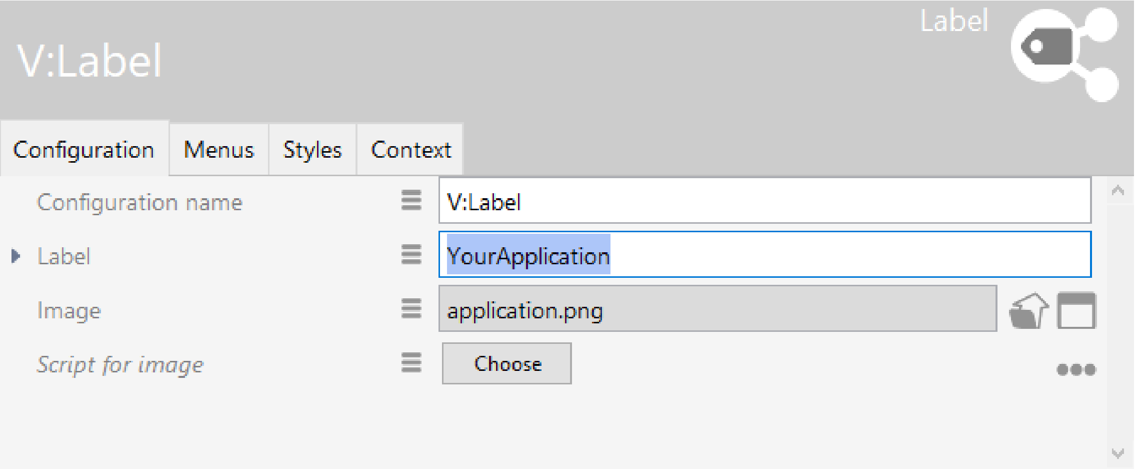
1.3.4.14. Label

The label configuration allows, for example, the labeling of a website or the labeling of a dialog panel. The label configurations are managed in the category “Subordinate configuration” in the Knowledge Builder. For example, labels are used in the window title panel; this requires creation of a new object underneath “Label configuration:”



Configuration name: P:Title
Panel type: Fixed View
Label configuration: V:Label
Path pattern parameter:

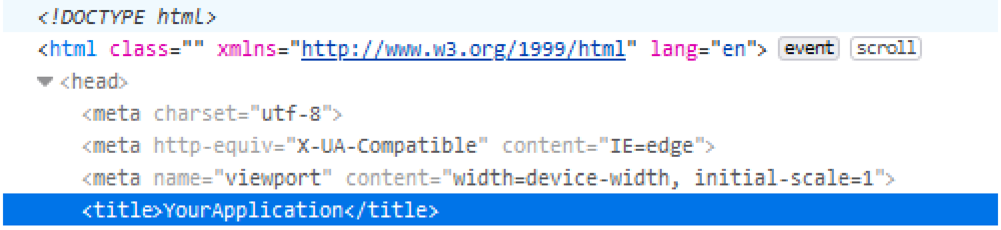
The entries can then be made under “Label” and “Image”:



Configuration name: V:Label
Label: YourApplication
Image: application.png
Script for image

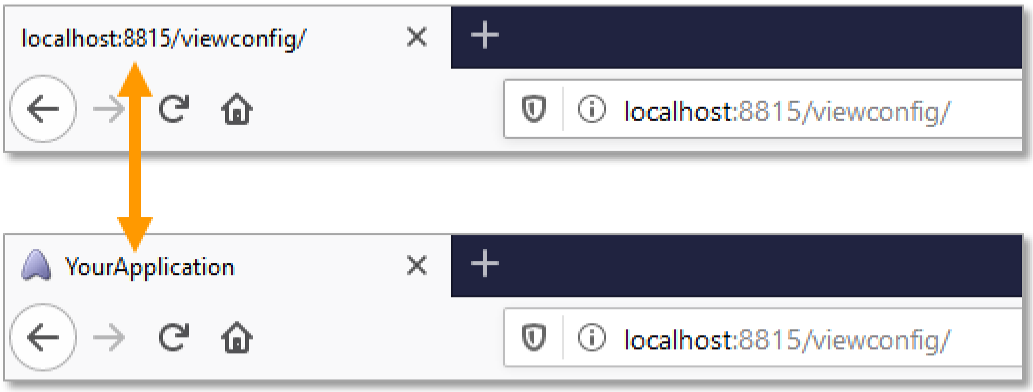
NOTE The view configuration element (“Label”) is titled “Label - Object” by default in the Knowledge Builder. If a string is entered under “Label”, then this appears as the element name of the view configuration element. When a configuration name (“Label”) is assigned, this appears as the element name.

If the label view is applied to the main window panel of the ViewConfiguration Mapper, the label content will be displayed in forms of the <title> element in the <head> section:



```
<!DOCTYPE html>
<html class="" xmlns="http://www.w3.org/1999/html" lang="en"> event scroll
▼ <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>YourApplication</title>
```

A comparison shows the different states of the website without a label (title = website path) or with a label (title = label):


localhost:8815/viewconfig/
localhost:8815/viewconfig/
YourApplication
localhost:8815/viewconfig/

1.3.5. Knowledge Builder configuration

The view configurations described here exclusively relate to Knowledge Builder. Additional view configurations that affect Knowledge Builder are also described at other points in chapter 7 but can then also relate to the output in JSON.

1.3.5.1. Folder structure

The left part of the main window in Knowledge Builder is used for navigating through the Knowledge Graph. To do so, a hierarchical folder structure is displayed there. This can be split into several main areas that are then displayed as bars. If you click on such a bar, the folder structure underneath it is expanded. This then enables you to access the contents (elements, queries, import/export mappings etc.). The contents are listed on the right side where they can be edited.

1.3.5.1.1. Default folder structure

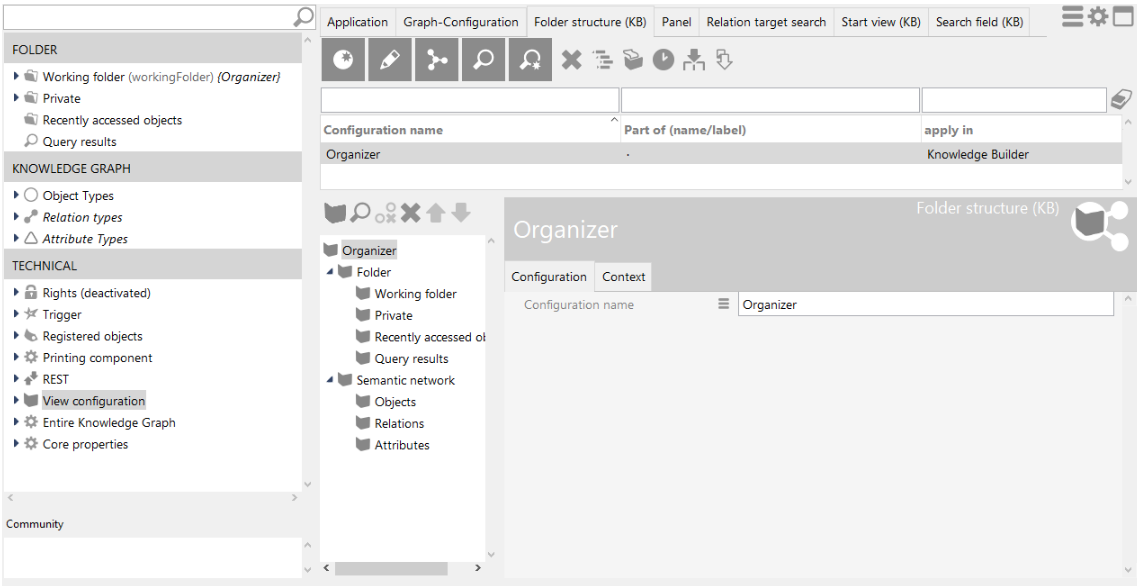
The configuration of the standard folder structure provides folders, making it possible to navigate the Knowledge Graph and store contents there. Three main areas are available to administrators.

The upper main area is **“FOLDER”** and provides folders for creating further folders and for managing content. These are the working folder, the private folder, the “Most recently used objects” folder and the “Query result” folder.

The second main area “ **KNOWLEDGE GRAPH”** makes it possible to navigate to the elements via the hierarchy of the types. The elements to be reached here are types, objects, attributes and relations. The area contains three folders:

- Object types for the hierarchy of object types and their concrete objects
- Relation types for the hierarchy of the relations
- Attribute types for the hierarchy of the attributes

The third main area is **"TECHNICAL"** ; it enables administrators to make changes, settings and configurations of all kinds in the Knowledge Graph. These include, among others, registered objects, the rights system and triggers.



Configuration name	Part of (name/label)	apply in
Organizer	·	Knowledge Builder

Configuration name: Organizer

The configuration of this standard folder structure can be viewed, modified and adapted to the users' needs in the Technical area >> View configuration.

NOTE Administrators always see the standard folder structure. If you configure a view configuration for folders, these are displayed only to non-administrators. If an administrator wishes to see the configured view of the folder structure, this can be set in the personal settings for the Knowledge Builder: Under "Settings" > "Personal" > "View configuration", select the "Configured" option.

1.3.5.1.2. Configuration of the folder structure

The folder structure is configured in the technical area under *View configuration* >> *Object types* >> *Knowledge Builder configuration* >> *Folder structure*. The admin is granted quick access to the configurations by selecting the *View configuration* node in the technical branch, and then selecting the *Organizer* object in the *Folder structure* tab in the pane to the right.

Folder structure elements are linked to each other as a hierarchy in the configuration. The root node of this hierarchy is an object of the *folder structure* type. It initially contains a folder structure called *Organizer*. All sub-nodes and their sub-nodes are of the *folder structure element*s type. The hierarchy in the configuration shows the hierarchy shown in the main window directly. The direct sub-nodes of the root node are shown as bars in the main window, resulting in a visual distinction between the various folder hierarchies.

Label is a parameter that all configuration types have in common. A node that is described by a configuration is labeled with this value. The content displayed in the right part of the main window when a node is selected depends on the parameters of the folder structure element. To do so, a type must be assigned to the parameter **Folder type**, for which a range of types is available. These folder types and their additional parameters are listed in the following table.

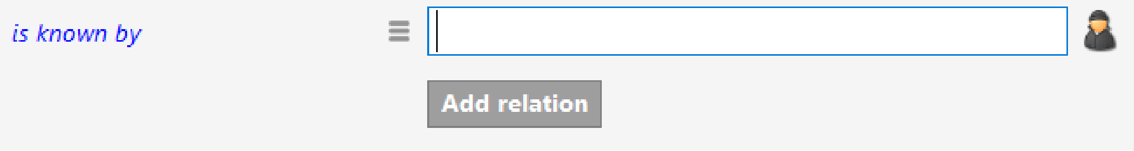
Folder type (obligatory)	**Parameter**	**Description**
Attribute types	Type	The attribute type specified, and all it subtypes, are displayed in a hierarchy-based tree.
Private folder	-	Display of the folder that only the actual user may view, and which is different to each user.
Relation types	Type	The attribute type specified, and all it subtypes, are displayed in a hierarchy-based tree.
Organizing folder	Organizing folder	Any *organizing folder* can be added here.
Query result folder	-	Each user has a query result folder of their own in which the user’s most recent query results are saved.
Type-based folder structure	“Without inheritance” view, type	The specified *type* and its subtypes are listed in a table. If the parameter "*Without inheritance view*" is set, then only the specified type is displayed. **NOTE** In order to manage which table configurations are used on the right-hand side, the *apply in* relation found there must be linked to this *folder structure element*.
Virtual folder	-	A folder that is used for structuring the folders.
Last objects used	-	Each user has a folder of their own in which the last objects used are saved for quicker access.

Only the configuration type *Virtual folder* can contain additional sub-configurations, and it is the only one for which sub-configurations make sense.

NOTE In the case of the folder type “Attribute types,” “Relation types” and “Type-based folder structure,” the parameter “Type” is used for specifying the attribute type, relation type or object type, and its subtypes, should be displayed in the folder.

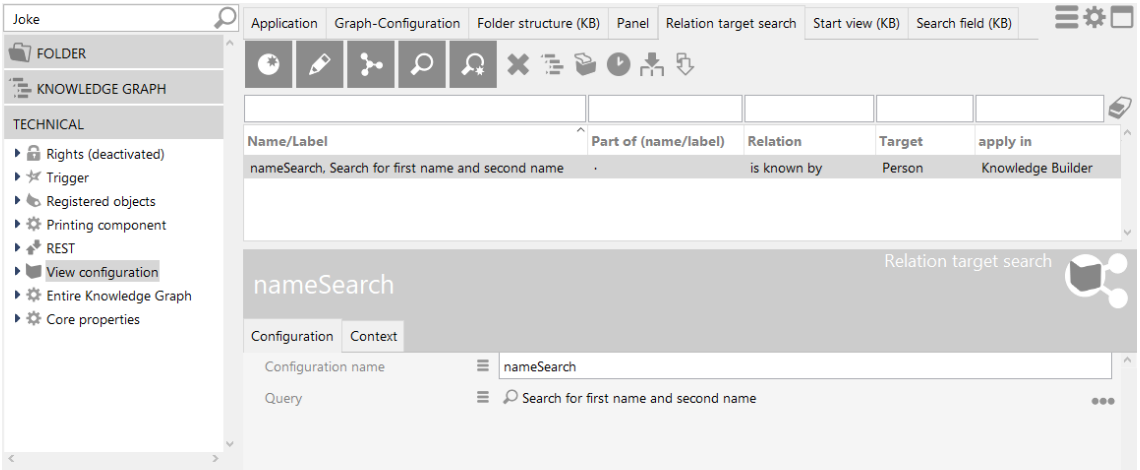
1.3.5.2. Relation target search

The configuration of relation targets makes it possible to influence the strategy used to search for possible relation targets.



If a Knowledge Graph does not include a search for relation targets, entering “Egon” always results in a search for objects named “Egon” (i.e. the respective defined name attribute is used). This response can be modified by specifying a previously defined query. Ordinary queries rather than structured queries are usually used for this purpose.

For example, to search for persons, you could define a query that searches both the first name and the last name. If you then search for a target of a relation whose target domain is person, the first names and last names of persons are searched for the entry “Egon.” A modified search for relation targets also makes sense if you want to search for objects and object synonyms at the same time, so that e.g. the “Architecture” object is also found if a user enters “the art of construction.”



Name/Label	Part of (name/label)	Relation	Target	apply in
nameSearch, Search for first name and second name	·	is known by	Person	Knowledge Builder

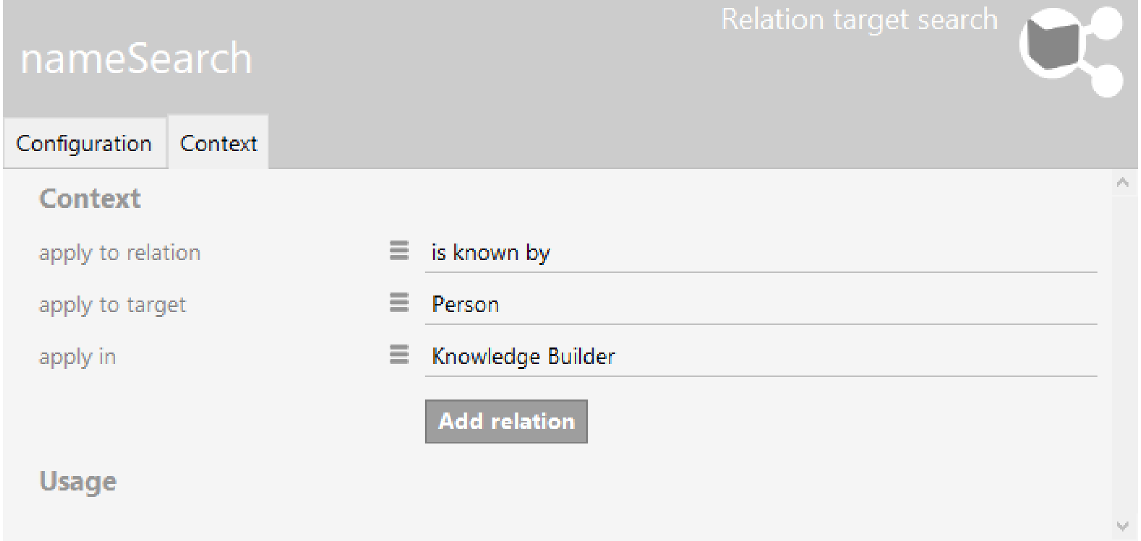
nameSearch

Configuration name: nameSearch

Query: Search for first name and second name

Relation target search configured to search for persons

As with all configurations, the context must be specified in which the relation target search is to be used. To do this, the relation to which the relation target search is to be applied must be entered for “apply to relation.”



Context

apply to relation: is known by

apply to target: Person

apply in: Knowledge Builder

Usage

The properties “apply to target” and “apply in” can be used as well in any combination as required.

1.3.5.3. Home view

You can use the configuration *Start view (KB)* (available as a tab in the view configuration area) to define which background image and which actions are supposed to be displayed on the start screen in Knowledge Builder on the right side. The display can be highlighted by means of de-selection (clicking on the selecting in the left navigation tree).

Setting options

Name	Value
Background	An image
Color value for font of an action	Depending on the image selected, a different color must be selected for labeling the actions in order to make the text readable.

In addition to this, actions can be defined. Refer to the Action chapter. An action type can also be specified. The following entries are available in this case:

Action type	Action
Manual (specialized web link)	Web manual is opened in the browser
Home page (specialized web link)	The home page is opened in the browser.
Support email (specialized web link)	A window opens for a new email to the email address of the Support department.
Web link	Freely definable web link
<no action type>	Execute configured action (using a script)

A web link must be configured completely; otherwise it will not be displayed.

However, this is not necessary for the three action types (specialized web links) displayed above. They use default values if a property is missing. It is possible to override the default values.

Possible configuration for a web link

Name	Value
Label	Display name after the icon
Symbol	Icon that is displayed in front of the label
URL	URL that is to be opened

1.3.5.4. Search field

The quick search field can be found in the upper left corner of the main window. This field provides quick access to queries. These are provided by the administrator or can also be added by the user. All queries that are used here may only expect a search string or no search input.

No search input makes sense for queries like this, the result of which changes from time to time. Executing a search like this in the quick search field then shows the current result without the need to look up the corresponding query in a folder, for example, every time. For example, there could also be a search query that displays all songs that the active user has already listened to.

1.3.5.4.1. Search field configuration for administrators

The “Search field” configuration defines which queries are made available by the administrator in the quick search field of the Knowledge Builder.

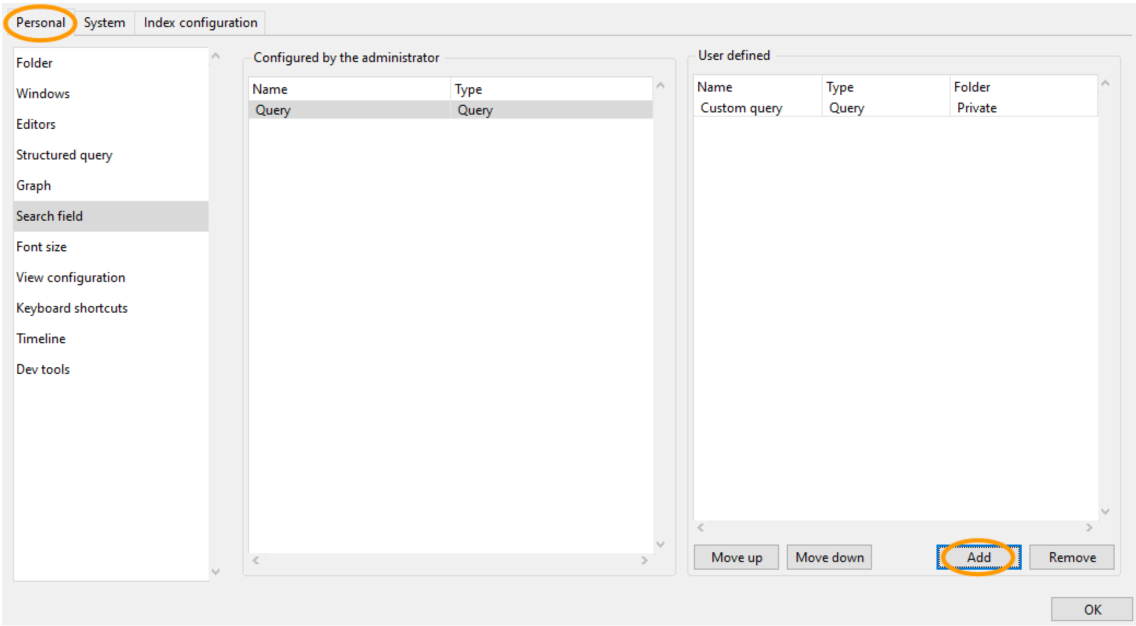
Newly created Knowledge Graphs feature a search field configuration that is the same for all users. The administrator can expand this search field configuration to make other queries accessible to all users. Moreover, each user can add further queries to their quick search field, which are then only visible to this particular user.

A search field configuration is comprised of “Quick search elements” that must contain a reference to a query and can optionally be given a label. The order of the quick search elements is determined by the order of the menu entries at the quick search field.

1.3.5.4.2. Search field configuration for users

The user can add queries by dragging an existing query to the quick search field.

Adding can also take place via the *Settings* . The *Search field* item is available on the *Personal* tab. On the right, in the *User-defined* section, the *Add* and *Remove* operations are available as well as an option for changing the order.



Name	Type
Query	Query

Name	Type	Folder
Custom query	Query	Private

1.3.6. Style

The view configuration is responsible for the structural formatting of elements of the Knowledge Graph for the display. If purely visual properties or information without context is also be specified, a “Style” element is used.

There are a number of Style elements that are already defined in i-views. The following section explains what these elements are and how these style elements are created in Knowledge Builder so that they can then be linked to individual elements of the view configuration of an application or Knowledge Builder.

In the view configuration, you first have to select the element with which one or more style elements are to be linked. Almost every view configuration type has a “Styles” tab. There, you can either define a new style element ★ or link an existing style element 🔍. If a new style element is defined, this must first be given a configuration name. You can then configure it on the right side of the editor.

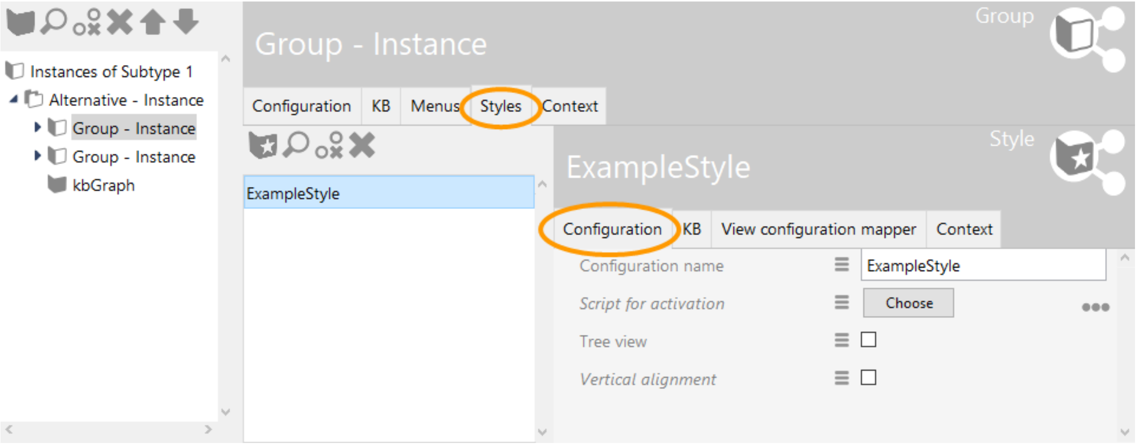
A style element can be filled with any number of style properties. The style properties are always distributed across several tabs, which are described in the sections below.

NOTE

Not all properties of a style make sense for all configurations. The tables of the following sections therefore contain a column called “Configuration type” which shows which view configuration type is supported by the respective property. The effect is described in the last column.

1.3.6.1. Style properties in applications and in the Knowledge Builder

This chapter describes the “Configuration” tab of a style element, which contains the style properties used in both the Knowledge Builder and the view configuration mapper.



Configuration name ExampleStyle

Script for activation Choose

Tree view □

Vertical alignment □

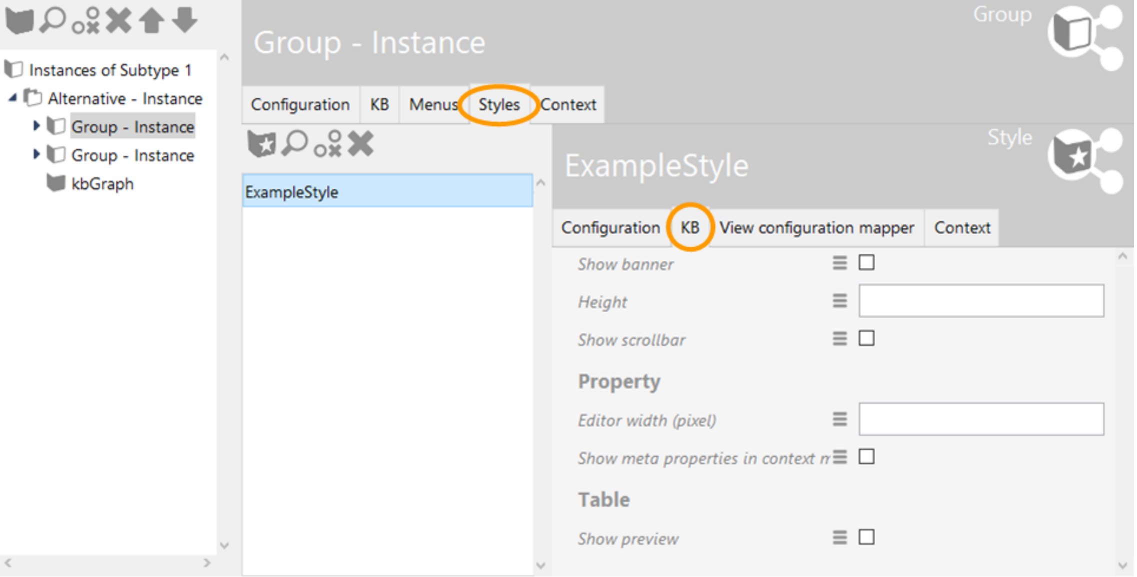
Style property	Configuration type	Effect
Configuration name	All	The configuration name is used for identification and reuse of the configuration.
Script for activation	All	The style can be activated in dependence on the active element by means of a script.
Tree view	-	*Due to the deprecation of groups, this option is no longer available.*
Vertical alignment	-	*Due to the deprecation of groups, this option is no longer available. Use layouts with vertical orientation instead.*

1.3.6.2. Style properties in applications

The “ViewConfiguration Mapper” tab is only displayed when the component “ViewConfiguration Mapper” has been installed. The style properties available for this component are included in the chapter Style of the ViewConfiguration Mapper (chapter 3).

1.3.6.3. Style properties in the Knowledge Builder

This chapter describes the “KB” tab of a style element, which contains the style properties used only in the Knowledge Builder.



Show banner ☐

Height

Show scrollbar ☐

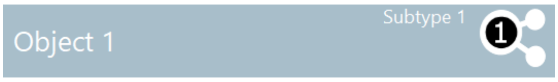
Property

Editor width (pixel)

Show meta properties in context m ☐

Table

Show preview ☐

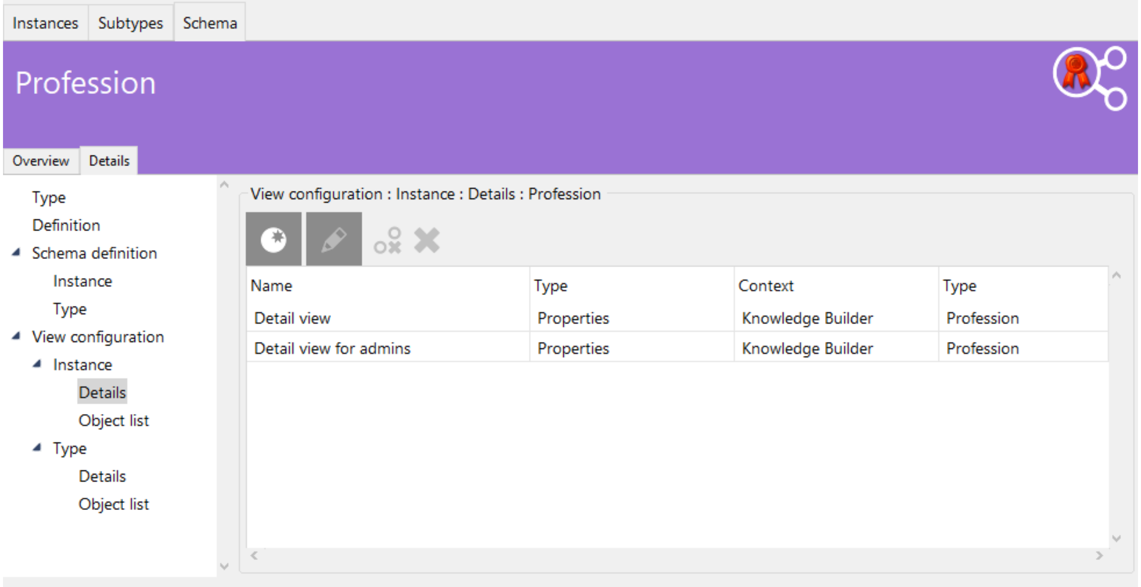
Style property	Configuration type	Effect
Configuration name	All	The configuration name is used for identification and reuse of the configuration.
Show banner	Object configuration	Display banner, including object name and type name as well as a buttons for the context menu for editing. When created a new configuration, the default value is *false* . 
Height	Property	Height in lines for string attributes (not: "Text" view).
Show scrollbar	Object configuration	If enabled, a scrollbar is shown if the respective view is too large for being displayed in full size within the given display area. This option is useful for grouping view elements containing more than one configuration, such as "Properties" or "Layouts".
Property		
Editor width (pixel)	Property	Width in pixels of a property
Show meta properties in context menu	(Meta-) property (properties)	Meta properties are shown in the context menu of the property. You can thus show either individual meta properties or all meta properties in a meta properties configuration. **NOTE** the *Add meta properties* menu option remains unchanged.

Style property	Configuration type	Effect
Table		
Show preview	Table	Controls whether an editor is shown underneath the table.

1.3.7. Detector system for determining the view configuration

View configurations can be linked to conditions using the detector system. The detector system determines when which configuration should be displayed. The way the detector system functions and the interplay with view configurations are explained in the following using an example.

Several displays can be created for objects of an object type using the settings in the view configuration. They can be linked to conditions using the detector system – for example, to a specific user. For the example described here, two views were configured for the objects of any type using the view configuration.



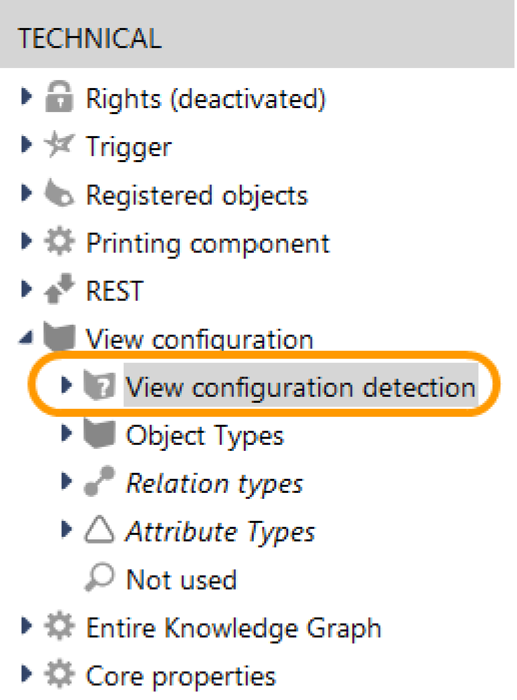
View configuration : Instance : Details : Profession

Name	Type	Context	Type
Detail view	Properties	Knowledge Builder	Profession
Detail view for admins	Properties	Knowledge Builder	Profession

Users who are administrators of the professions list which they wish to access should see the “Detail view for admins”. All users who are not administrators of the professions list which they wish to access should see the “Detail view”. The conditions that determine how the views are used are defined in the detector system.

Creation of a view configuration determination

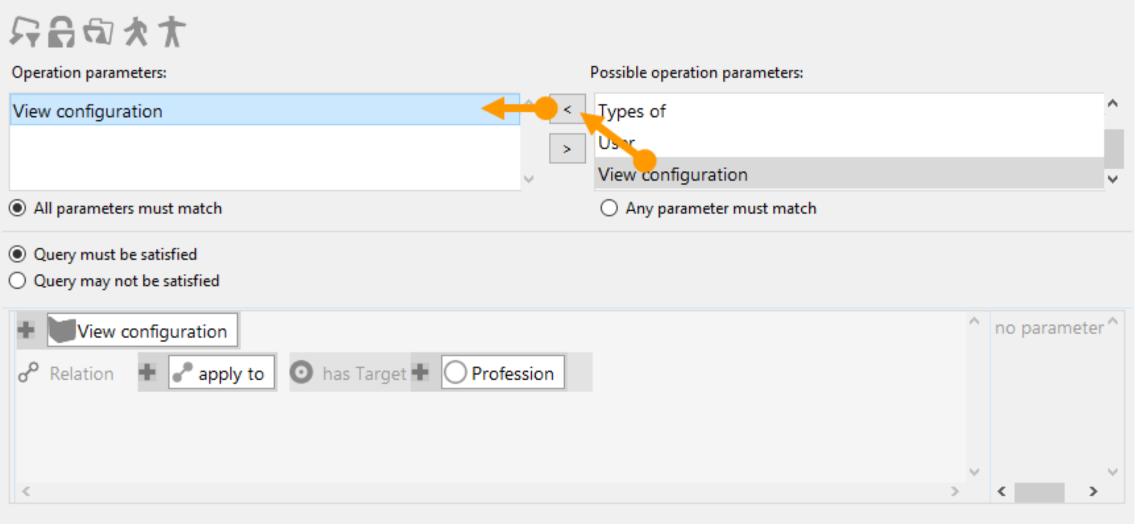
The detector system is located in the folder hierarchy on the left in the “TECHNICAL” section, and has been designated as “View configuration detection” under “View configuration”.



TECHNICAL

- Rights (deactivated)
- Trigger
- Registered objects
- Printing component
- REST
- View configuration
  - View configuration detection
  - Object Types
  - *Relation types*
  - *Attribute Types*
  - Not used
- Entire Knowledge Graph
- Core properties

By creating a new query filter (see the “Query filter” chapter) in the first step, the starting point must be defined. This means that you have to define to what other things the following settings are supposed to apply. In this example, our starting point is therefore a view configuration (in this case: “Detail view for admins”), for which a condition is created at the same time. “View configuration” must be selected from the list and be entered as the operation parameter. The query filter then looks as follows:



Operation parameters:

View configuration

● All parameters must match

Possible operation parameters:

Types of
User
View configuration

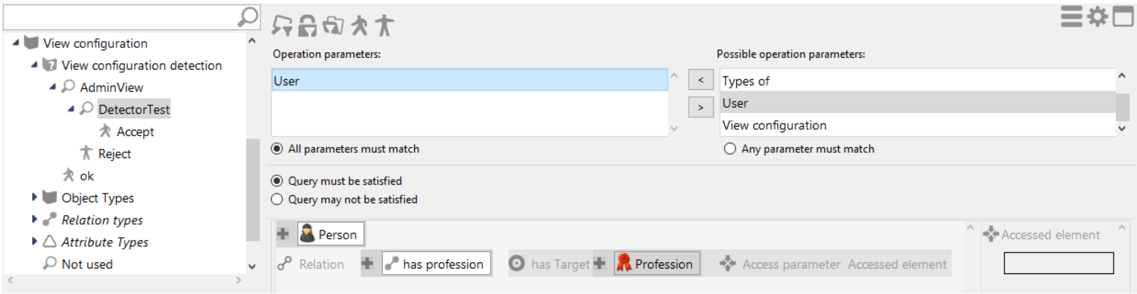
○ Any parameter must match

● Query must be satisfied
○ Query may not be satisfied

View configuration
Relation — apply to — has Target — Profession

no parameter

A new query filter must now be created under the query filter that is searching for the view configuration “Detail view for admins” and which describes the condition for this view configuration: the view configuration “Detail view for admins” should only be visible to users who have the profession that they are currently viewing. The second query filter therefore checks whether the active user is a person of the same profession. By clicking on , the set of search results is then permitted to view the configuration “Detail view for admins”. The following diagram shows the query filter for users who are persons of the same profession that they are currently viewing and the folder hierarchy that was created so far on the left-hand side.



- View configuration
  - View configuration detection
    - AdminView
      - DetectorTest
        - Accept
      - Reject
    - ok
  - Object Types
  - Relation types
  - Attribute Types
  - Not used

Operation parameters: User

Possible operation parameters: Types of, User, View configuration

● All parameters must match ○ Any parameter must match

● Query must be satisfied
○ Query may not be satisfied

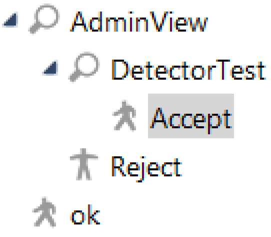
Person
Relation has profession has Target Profession Access parameter Accessed element

Accessed element

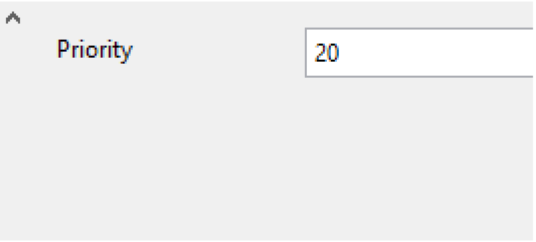
The view configuration “Detail view” is automatically used for those users who are not person of the same profession that they are currently viewing.

Weighting of the configurations in the detector system

The configurations in the detector system “View configuration detection” are weighted from top to bottom in the application. This means that access settings made closer to the top have a higher weighting that those further down. In order to bypass this default setting, the authorizations or denials can be given priorities.



- AdminView
  - DetectorTest
    - Accept
  - Reject
- ok



Priority: 20

Priority 1 is the highest priority. If the condition instructions overlap, then the authorization or denial conditions with the highest priority is implemented. If no specifications have been made for priorities, or if all priority numbers have the same value, then the previous conditions are implemented in the detector tree.

1.4. JavaScript API

1.4.1. Introduction

The JavaScript API is a server-side API for accessing a semantic Knowledge Graph. The API is used in triggers, REST services, view configuration etc.

By means of the API, the Knowledge Graph can be accessed with read operations (processing queries, querying properties etc.) and modifying operations (creating objects, changing attributes etc.).

The Knowledge Builder provides an integrated editor for editing, executing and debugging JavaScript code. The editor is available when accessing the respective code snippet. Registered JavaScript code can be accessed via TECHNICAL > Registered objects > Scripts. New JavaScript can be created where needed (REST interface configuration, view configuration) or in the working/private folder of the Knowledge Builder.

NOTE When commenting out references to queries or other elements of the Knowledge Graph, the reference of the previously referenced element to the JavaScript will not be listed anymore when invoking the "References" list for the element.

1.4.1.1. API reference

The API reference is available here:

https://documentation.i-views.com/6.0/javascript-api/index.html

1.4.1.2. The namespace $k

Most objects are defined in the namespace $k. The namespace object itself has a few useful functions, e.g.

```
$k.rootType()
```

which returns the root type of the Knowledge Graph, or

```
$k.user()
```

which returns the current user.

1.4.1.3. Registry

Another important object is the Registry object $k.Registry. It allows to access objects by their registered key (folder elements) / internal name (types).

Examples:

```
$k.Registry.type("Article")
```

returns the type with the internal name "Article".

```
$k.Registry.query("rest.articles")
```

returns the query with the registered key "rest.articles".

The Registry object is a singleton, similar to JavaScript’s Math object.

1.4.1.4. Working with semantic elements

Semantic elements are usually retrieved from the registry or by a query.

```
// Get the person type by its internal name
const personType = $k.Registry.type("Person");

// Perform the query named "articles",
// with the query parameter "tag" set to "Sailing"
const sailingArticles = $k.Registry.query("articles").findElements({tag:
"Sailing"});
```

The properties of an element can be accessed by specifying the internal name of the property type.

```
// Get the value of the attribute "familyName"
const familyName = person.attributeValue("familyName");
// Get the target of the relation "bornIn"
const birthplace = person.relationTarget("bornIn");
```

A shortcut to access the value of the name attribute is the function name()

```
const name = birthplace.name();
```

If an attribute is translated, the desired language can be specified, either as 2-letter or 3-letter ISO 639 language code or as locale with language and territory. The current language of the environment is used if no language is specified.

```
const englishTitle = book.attributeValue("title", "eng");
const swedishTitle = book.attributeValue("title", "sv_SE");
const currentTitle = book.attributeValue("title");
```

1.4.1.5. Transactions

Transactions are required to create, modify or delete elements. If transactions are controlled by the script, a block can be wrapped in a transaction:

```
$k.transaction(() =>
    $k.Registry.type("Article").createInstance()
);
```

It is possible to configure if the script controls transactions or if the entire script should be run in a transaction. The only exception are trigger scripts, which are always run as part of a writing transaction.

A transaction may be rejected due to concurrency conflicts. An optional function can be passed to $k.transaction() that is evaluated in such cases:

```
$k.transaction(
    () => $k.Registry.type("Article").createInstance(),
    () => throw "The transaction was rejected"
);
```

Transactions, like the ones described above, may not be nested. There are, however, cases in which nesting is unavoidable; for example, because a script function is called both by functions that are already encapsulated in a transaction and functions for which this does not apply. A so-called “optimistic transaction” can be used in this case. This construction uses the external transaction if there is one, or it starts a new transaction.

```
$k.optimisticTransaction(() =>
    $k.Registry.type("Article").createInstance()
);
```

Constructions like this should be avoided, because a transaction represents a practical operational unit which is executed in whole or not at all. Either what is embedded makes sense and is complete in itself, or is not.

NOTE

A troubleshooting function in the event of failure of the optimistic transaction is not available. If an external transaction exists, its troubleshooting function is executed in the event of failure.

1.4.1.6. Modify elements

1.4.1.6.1. Create elements

```
// Create a new instance
const person = $k.Registry.type("Person").createInstance();

// Create a new type
const blogType = $k.Registry.type("CommunicationChannel").createSubtype();
blogType.setName("Blog");
```

1.4.1.6.2. Add and change attributes

Attribute values can be set with setAttributeValue(), which implies that a single attribute is either already present or created. Existing attribute values are overwritten. An exception is thrown when more than one attribute of a type is present.

```
person.setAttributeValue("familyName", "Sinatra");
person.setAttributeValue("firstName", "Frank");
// Overwrite the value "Frank" with "Francis"
person.setAttributeValue("firstName", "Francis");
```

createAttribute() allows to create more than one attribute of a type.

```
// Create two attributes
person.createAttribute("nickName", "Ol' Blue Eyes");
person.createAttribute("nickName", "The Voice");
```

The attribute values are represented by different object types depending on the type of attribute, some of which are native to JavaScript while others belong to the $k namespace:

Type of attribute	Object type
Choice	$k.Choice
Boolean	boolean
File	$k.Blob
Date	$k.Date
Date and time	$k.DateTime
Color value	string (hex value)
Flexible time	$k.FlexDateTime

Type of attribute	Object type
Float	number
Integer	number
Geo position	$k.GeoPosition
Group	(no value)
Internet shortcut	string
Interval	$k.Interval
String	string
Time	$k.Time

1.4.1.6.3. Add relations

A relation between two elements can be created with createRelation():

```
const places = $k.Registry.query("places").findElements({name: "Hoboken
"});
if (places.length == 1)
    person.createRelation("bornIn", places[0]);
```

1.4.1.6.4. Delete elements

Any element can be deleted with the remove() function:

```
person.remove();
```

This also deletes all properties of the element.

1.4.2. Examples

1.4.2.1. Queries

Using the API, one can execute registered queries. Queries are represented by objects of the class $k.Query while for structured queries there is the subclass $k.StructuredQuery.

Search for elements: Perform the query "articles" with parameter tag = "Soccer"

```
const articles = $k.Registry.query("articles").findElements({ tag: "
Soccer" });
for (let a of articles) {
  $k.out.print(articles[a].name() + "\n")
}
```

Return hits: Perform the query "mainSearch" with the search string "Baseball"

```
const hits = $k.Registry.query("mainSearch").findHits("Baseball");
for (let hit of hits) {
  $k.out.print(`${hit.element().name()} (${Math.round(hit.quality() *
100)}%)\n`)
}
```

A hit wraps an element and adds a quality value (between 0 and 1) and additional metadata.

Convert query results to JSON:

```
const elements = $k.Registry.query("articles").findElements({ tag:
"Snooker" })
const json = elements.map(element => ({
  name: element.name(),
  id: element.idString(),
  type: element.type().name()
}))
$k.out.print(JSON.stringify(json, undefined, "\t"))
```

1.4.2.2. Runtime generated queries

The JavaScript API also makes it possible to generate queries dynamically. Here are several examples from a Knowledge Graph on films:

1.4.2.2.1. Search for films by year + name

```
const query = new $k.StructuredQuery("imdb_film")
query.addAttributeValue("imdb_film_year", "year")
query.addAttributeValue("name", "name")
query.findElements({ year: "1958", name: "Vert*" })
```

The domain is transferred to the constructor. In case of internal names, the search automatically looks for objects of this type. The setDomains() function offers more options.

1.4.2.2.2. Year + number of directors >= 3

```
const query = new $k.StructuredQuery("imdb_film")
query.addAttributeValue("imdb_film_year", "year")
query.addCardinality("imdb_film_regisseur", 3, ">=")
query.findElements({year: "1958"})
```

1.4.2.2.3. Year + name of director

```
const query = new $k.StructuredQuery("imdb_film")
query.addAttributeValue("imdb_film_year", "year", ">=")
const directorQuery = query.addRelationTarget("imdb_film_regisseur"
).targetQuery()
directorQuery.addAttributeValue("name", "director")
query.findElements({ year: "1950", director: "Hitchcock, Alfred" })
```

1.4.2.2.4. Alternatives (OR conditions)

```
const query = new $k.StructuredQuery("imdb_film")
query.addAttributeValue("imdb_film_year", "year")
const alternatives = query.addAlternativeGroup()
alternatives.addAlternative().addAttributeValue("name", "name")
alternatives.addAlternative().addAttributeValue("imdb_film_alternativeTite
l", "name")
query.findElements({ year: "1958", name: "Vert*" })
```

1.4.2.2.5. Operators

Operator name	Short form	Description
containsPhrase		Contains phrase
covers		Contains
distance		Distance
equal	==	Equal
equalBy		Corresponds to
equalCardinality		Equal cardinality
equalGeo		Equal (geo)
equalMaxCardinality		Cardinality smaller than or equal to
equalMinCardinality		Cardinality greater than or equal to
equalPresentTime		Now (present)
equalselementOneWay		Filter with
fulltext		Contains string

Operator name	Short form	Description
greater	>	Greater than
greaterOrEqual	>=	Greater/equal
greaterOverlaps		Overlaps from above
greaterPresentTime		After now (future)
isCoveredBy		Is contained in
less	<	Less than
lessOrEqual	⍰	Less/equal
lessOverlaps		Overlaps from below
lessPresentTime		Before now (past)
notEqual	!=	Not equal
overlaps		Overlaps
range		Between
regexEqual		Regular expression
regexFulltext		Contains string (regular expression)
unmodifiedEqual		Exactly identical
words		Contains string

1.4.2.3. Creating and changing elements

1.4.2.3.1. Creating a person

```
// Get the person type by its internal name
const personType = $k.Registry.type("Person");
// Create a new instance
const person = personType.createInstance();
// Set attribute values
person.setAttributeValue("familyName", "Norris");
person.setAttributeValue("firstName", "Chuck");
```

1.4.2.3.2. Setting the full name of a person

```
const familyName = person.attributeValue("familyName");
const firstName = person.attributeValue("firstName");
if (familyName && firstName) {
  const fullName = familyName + ", " + firstName;
  person.setAttributeValue("fullName", fullName);
}
```

1.4.2.3.3. Setting the value of an attribute

```
// Boolean attribute
element.setAttributeValue("hasKeycard", true);

// Choice attribute
// - internal name
element.setAttributeValue("status", "confirmed");
// - choice object
const choiceRange = $k.Registry.attributeType("status").valueRange();
const choice = choiceRange.choiceInternalNamed("confirmed");
element.setAttributeValue("status", choice);

// Color attribute
element.setAttributeValue("hairColor", "723F10");

// Date / Time / DateAndTime attribute
element.setAttributeValue("dateOfBirth", new Date(1984, 5, 4));
element.setAttributeValue("lastModification", new Date());
element.setAttributeValue("teatime", new Date(0, 0, 0, 15, 30, 0));

// FlexTime attribute
// - $k.FlexTime (allows imprecise values)
element.setAttributeValue("start", new $k.FlexTime(1984, 6));
// - Date (missing values are set to default values)
element.setAttributeValue("start", new Date(1984, 5, 3));

// Number (integer / float) attribute
element.setAttributeValue("weight", 73);

// Interval
element.setAttributeValue("interval", new $k.Interval(2, 4));

// String attribute
// - untranslated
element.setAttributeValue("familyName", "Norris");
// - translated (language is an ISO 639-1 or 639-2b code)
element.setAttributeValue("welcomeMessage", "Welcome", "en");
element.setAttributeValue("welcomeMessage", "Bienvenue", "fre");
```

1.4.2.3.4. Creating a new attribute

```
person.createAttribute("nickName", "Ground Chuck");
```

1.4.2.3.5. Creating a new relation

```
const places = $k.Registry.query("places").findElements({name: "Oklahoma
"});
if (places.length == 1)
    person.createRelation("bornIn", places[0]);
```

1.4.2.3.6. Deleting an element, including its properties

```
person.remove()
```

1.4.2.3.7. Converting a string to an attribute value

The ValueRange of an attribute type knows the valid values of the attribute and can parse a string. It throws an exception if the string is not valid.

```
const statusRange = $k.Registry.type("status").valueRange();
const statusConfirmed = statusRange.parse("Confirmed", "eng");
```

1.4.2.3.8. Setting change metadata

```
element.setAttributeValue("lastChangeDate", new $k.Date());
const userInstance = $k.user().instance();
// Ensure that a single relation to the user instance exists
if (element.relationTarget("lastChangedBy") !== userInstance) {
  const relations = element.relations("lastChangedBy");
  for (let relation of relations)
    relation.remove();
  element.createRelation("lastChangedBy", userInstance);
}
```

1.4.2.4. Date and time

Wenn man ein JavaScript-Date als Attributwert setzt, wird der Wert in der lokalen Zeitzone gespeichert. Die Attribute selbst speichern keine Zeitzone, nur Datum/Uhrzeit.

```
const task = $k.Registry.type('Task').createInstance()
task.setAttributeValue('dateOfCreation', new Date())
```

Wenn man dieses Script zum Zeitpunkt 20.6.2023 12:58 MEZ ausführt, wird "20.6.2023 12:58" als Attributwert gesetzt.

Um einen Attributwert unabhängig von der lokalen Zeitzone zu speichern, kann man die $k.DateTime-API verwenden. Dieses hat eine mit Date verwandte API, kann aber zusätzlich mit toUTC() die Zeitzone des Werts wandeln:

```
const task = $k.Registry.type('Task').createInstance()
task.setAttributeValue('dateOfCreation', new $k.DateTime().toUTC())
```

Dieses Script setzt zum selben Zeitpunkt "20.6.2023 10:58" als Attributwert.

Da das Attribut keine Zeitzone speichert, ist die Darstellung bei Clients unabhängig von der lokalen Zeitzone.

Für eine Darstellung in der lokalen Zeitzone kann $k.DateTime mit `toUTCDate()` den in UTC gespeicherten Attributwert in eine Date in der lokalen Zeitzone umwandeln.

```
task.attributeValue('dateOfCreation').toUTCDate()
```

NOTE

- `toUTC()` ist nicht beim ECMAScript-Date definiert, nur bei $k.DateTime und $k.Time
- `toUTCDate()` ist leider leicht mit `toUTC()` zu verwechseln.
- `toUTCDate()` liefert ein ECMAScript-Date, `toUTC()` ein $k.DateTime / $k.Time-Objekt

Wenn man zu einem Datum/Uhrzeit-Wert nur das Datum oder nur die Uhrzeit ausgeben möchte, kann man dazu $k.Date und $k.Time verwenden:

```
// Anlegezeitpunkt in lokaler Uhrzeit darstellen
new $k.Time(task.attributeValue('dateOfCreation').toUTCDate())
```

NOTE

Im Gegensatz zur ECMAScript-API ist $k.Date nur das Datum ohne Uhrzeit. $k.DateTime hat Datum + Uhrzeit.

1.4.2.5. Sessions

To provide a semantic element or a specific value for a later accessed view, session variables can be used.

The assignment of the session variable is done by:

```
$k.Session.current().setVariable('nameOfVariable', elementOrValue)
```

Reading out the session variable works like this:

```
$k.Session.current().getVariable('nameOfVariable')
```

1.4.2.6. REST

A REST script must define a respond() function that receives the HTTP request, the parsed request parameters and an empty HTTP response. The script then fills header fields and the contents of the response.

```
function respond(request, parameters, response) {
  response.setText("REST example");
}
```

1.4.2.6.1. Restlet that returns a blob

```
function respond(request, parameters, response) {
  const name = parameters["name"];
  if (name) {
    const images = $k.Registry.query("rest.image").findElements({"name":
name});
    if (images.length == 1) {
      // Set the contents and content type (if known) from the image blob.
      response.setContents(images[0].value());
      // Show the image instead of asking to download the file
      response.setContentDisposition("inline");
    } else {
      response.setCode($k.HttpResponse.BAD_REQUEST);
      response.setText(images.length + " images found");
    }
  } else {
    response.setCode($k.HttpResponse.BAD_REQUEST);
    response.setText("Name not specified");
  }
}
```

1.4.2.6.2. Restlet that creates an instance with an uploaded blob

```
function respond(request, parameters, response) {
  const formData = request.formData();
  const name = formData.name;
  const picture = formData.picture;
  if (name && picture) {
    const city = $k.Registry.type("City").createInstance();
    city.setAttributeValue("image", picture);
    city.setName(name);
    response.setText("Created city " + name);
  } else {
    response.setCode($k.HttpResponse.BAD_REQUEST);
    response.setText("Parameters missing");
  }
}
```

1.4.2.7. XML

1.4.2.7.1. Transforms query results into XML elements

```
function respond(request, parameters, response) {
  const name = parameters["name"];
  if (name) {
    // Find points of interest
    const elements = $k.Registry.query("rest.poi").findElements({name:
name});
    // Write XML
    const document = new $k.TextDocument();
    const writer = document.xmlWriter();
    writer.startElement("result");
    for (let element of elements) {
      writer.startElement("poi");
      writer.attribute("name", element.name());
      writer.endElement();
    }
    writer.endElement();
    response.setContents(document);
    response.setContentType("application/xml");
  } else {
    response.setCode($k.HttpResponse.BAD_REQUEST);
    response.setContents("Name not specified");
  }
}
```

XML output

```
<result>
  <poi name="Plaza Mayor"/>
  <poi name="Plaza de la Villa"/>
  <poi name="Puerta de Europa"/>
</result>
```

1.4.2.7.2. Using qualified names

```
const document = new $k.TextDocument();
const writer = $k.out.xmlWriter();
writer.setPrefix("k", "http://www.i-views.de/kinfinity");
writer.startElement("root", "k");
writer.attribute("hidden", "true", "k");
writer.startElement("child","k").endElement();
writer.endElement();
```

XML output

```
<k:root xmlns:k="http://www.i-views.de/kinfinity" k:hidden="true">
  <k:child/>
</k:root>
```

1.4.2.7.3. Defining a default namespace

```
const document = new $k.TextDocument();
const writer = $k.out.xmlWriter();
writer.startElement("root");
writer.defaultNamespace("http://www.i-views.de/kinfinity");
writer.startElement("child").endElement();
writer.endElement();
```

XML output

```
<root xmlns="http://www.i-views.de/kinfinity">
```

```
  <child/>
</root>
```

1.4.2.8. HTTP client

A script can also be used to send HTTP requests.

1.4.2.8.1. Loading a picture via HTTP and store it as a blob

```
const http = new $k.HttpConnection();
const imageUrl = "http://upload.wikimedia.org/wikipedia/commons/e/e7/2007-07-06_GreatBriain_Portree.jpg";
const imageResponse = http.request(new $k.HttpRequest(imageUrl));
if (imageResponse && imageResponse.code() == $K.HttpResponse.OK ) {
  const portree = $k.Registry.type("City").createInstance();
  portree.setAttributeValue("image", imageResponse);
  portree.setName("Portree");
}
```

1.4.2.8.2. Updating the weather report of all cities

```
const instances = $k.Registry.type("City").instances();
const http = new $k.HttpConnection();
for (let instance of instances) {
  const city = instance;
  const weatherUrl = "http://api.openweathermap.org/data/2.5/weather";
  const weatherRequest = new $k.HttpRequest(weatherUrl);
  weatherRequest.setQueryData({q: city.name()});
  try {
    const weatherResponse = http.request(weatherRequest);
    if (weatherResponse.code() == $k.HttpResponse.OK) {
      const json = JSON.parse(weatherResponse.text());
      const weather = json.weather[0].description;
      city.setAttributeValue("weather", weather);
    }
  } catch (e) {
  }
}
```

1.4.2.8.3. Basic authentication

In the following example, a username and password is extracted from an encrypted string. These strings can be created using the admin tool, navigating to System configuration > Access authorisation and using the "Encrypt name/password" button. They are only valid for the Knowledge Graph for which they were created.

```
const http = new $k.HttpConnection()
const account =
'GH1Z4FXWrCdEoiDSlCVMZJQ6QaBZ4rfAcJdDliUHn8ep00ZKmUR+f8nvAFEObB1pjrQId0Bn9
rjmaSZJtz4X6RSAGONfHRxlWG62V3itUPeHzqs7DE90/jG+cv/rVKNrxcdFGRja6cjnHOTK4LG
jZiuUV313GsC1EDr8GEctfeo='
http.authenticateFromEncrypedAccount(account)
const request = new $k.HttpRequest('http://example.org/restricted')
const response = http.request(request)
```

1.4.2.8.4. Sending JSON object as query data via POST request

```
const http = new $k.HttpConnection();
const objectToPost = [{foo: 'bar'}, 'baz'];
const destinationURL = 'http://upload-via-post.domain.com';
const postRequest = new $k.HttpRequest(destinationURL, 'POST');
postRequest.setText(JSON.stringify(objectToPost))
postRequest.setHeaderField('Content-Type', 'application/json');
const response = http.request(postRequest);
```

1.4.2.8.5. Sending a blob via PUT request

```
const blob = $k.Registry.elementAtValue('isbn', '978-0544003415'
).attributeValue('coverPicture')
const http = new $k.HttpConnection()
const request = new $k.HttpRequest('http://mybookservice/cover/978-
0544003415', 'PUT')
request.setContents(blob)
const response = http.request(request)
```

The content type is taken automatically from the blob.

1.4.2.8.6. Sending two blobs via POST request as multipart/form-data

```
const book = $k.Registry.elementAtValue('isbn', '978-0544003415')
const pdfBlob = book.attributeValue('pdf')
const previewBlob = book.attributeValue('preview')
const http = new $k.HttpConnection()
const request = new $k.HttpRequest('http://mybookservice/ebooks/978-
0544003415', 'POST')
```

```
request.setContentType('multipart/form-data')
const pdfPart = new $k.NetEntity()
pdfPart.setContentDisposition('form-data; name="ebook"')
pdfPart.setContents(pdfBlob)
request.attach(pdfPart)
const previewPart = new $k.NetEntity()
previewPart.setContentDisposition('form-data; name="preview"')
previewPart.setContents(previewBlob)
request.attach(previewPart)
const response = http.request(request)
```

The filename of the form data is taken from the blob. If another filename is needed, it can be set using setFilename(string).

1.4.2.8.7. Sending an URL encoded form via POST request

```
const data = { name: 'Gandalf', occupation: 'Wizard' }
const http = new $k.HttpConnection()
const request = new $k.HttpRequest('http://mybookservice/user', 'POST')
request.setFormData(data)
const response = http.request(request)
```

The data is send with Content-Type application/x-www-form-urlencoded.

1.4.2.8.8. Restricting outgoing domains

To prevent scripts from sending requests to arbitrary hosts, a whitelist can be defined in the configuration file of the application.

```
[script]
allowedOutgoingDomains=*.i-views.de,*.intelligent-
views.com,ivinternal:8080
```

The comma separated strings are compared to the domain part of the URL. "*" can be used as wildcard character. Optionally, a port can also be specified. If domain or port do not match, executing the request throws an URIError. If no specific port is defined, every port is accepted.

1.4.2.9. Sending mails

Mails can be sent with the MailMessage object. To do so, an SMTP server must be configured in the Knowledge Graph (Settings → System → SMTP).

```
const mail = new $k.MailMessage();
```

```
mail.setSubject("Hello from " + $k.volume());
mail.setText("This is a test mail");
mail.setSender("kinfinity@example.org");
mail.setReceiver("developers@example.org");
mail.setUserName("kinf");
mail.send();
```

The user account “kinf” is used for authentication. The password is saved in the SMTP settings. === Sending a mail via $k.SmtpConnection

In the following example, a username and password is extracted from an encrypted string. These strings can be created using the admin tool, navigating to System configuration > Access authorisation and using the "Encrypt name/password" button. They are only valid for the Knowledge Graph for which they were created.

```
const mail = new $k.MailMessage()
mail.setSubject('Hello from ' + $k.volume())
mail.setText('This is a test mail')
mail.setSender('kinfinity@example.org')
mail.setReceiver('developers@example.org')
const smtp = new $k.SmtpConnection()
smtp.setHost('mailgateway.local', 22)
smtp.authenticateFromEncrypedAccount('Qi3Eky7itkf2NckwgcKemiZvNGGoXcbo43O2
/nZ5RvoRvv7AukUM0LIVUw1WJ+uMDgzxw7JA5gtYyLgNg7fHaC4wJCQIgnIfXVPSW6u391NmUq
nZkcuc0nl4u2nPymAmcqzoUDJRSHrMVy1qEsbxXtfhsJzh7e4EDKIAeJ75BxE=')
smtp.send(mail)
```

1.4.2.9.1. Sending a mail with attachment

```
const mail = new $k.MailMessage()
mail.setSubject('Daily report')
mail.setText('Here is the daily report')
const attachment = new $k.NetEntity()
attachment.setContentType('text/html')
attachment.setText('<html><body><h1>Daily report</h1>No problems
found</body</html>')
attachment.setContents(report)
mail.attach(attachment)
mail.setSender('kinfinity@example.org')
mail.setReceiver('developers@example.org')
mail.setUserName('kinf')
mail.send()
```

1.4.2.10. Data source mappings

Per API kann man registrierte Abbildungen von Datenquellen ausführen. Die Abbildungen werden durch Objekte der Klasse $k. Mapping repräsentiert. Abbildungen zur Laufzeit zu generieren ist derzeit nicht möglich.

Einen Export mit einer registrierten Abbildung mit dem Registierungsschlüssel "products"durchführen:

```
const mapping = $k.Registry.mapping("products")
mapping.runExport()
```

Bei dateibasierten Datenquellen verwendet die API standardmäßig die konfigurierten Ein-/Ausgabedateien. Alternativ kann von/in eine $k.NetEntity im-/exportiert werden:

```
const mapping = $k.Registry.mapping("products")
const productsEntity = new $k.NetEntity()
mapping.setParameter("netEntity", productsEntity)
mapping.runExport()
```

Dadurch können die Inhalte per HTTP oder E-Mail transportiert werden. Derzeit werden die Inhalte der NetEntity im Hauptspeicher abgelegt, für große Datenmengen ist diese Methode deshalb nicht geeignet.

1.4.2.11. ZIP files

Zip files can be read and written. Both the zip file and the contained files are represented by $k.NetEntity objects, but it is also possible to add $k.Blob objects as zip content.

1.4.2.11.1. Return a zip file as response to a REST request

```
function respond(request, parameters, response) {
  const zip = new $k.Zip('avatars.zip')
  $k.Registry.type('account').allInstances().forEach(account =>
    zip.addEntry(account.attributeValue('avatar'))
  )
  response.setContents(zip)
}
```

1.4.2.11.2. Read out contents of a zip file in a POST request body

The constructor is called with a $k.NetEntity object.

```
function respond(request, parameters, response) {
  if (request.contentType() !== 'application/zip') {
    response.setCodeBadRequest().setText('Zip expected')
    return
  }
  const zip = new $k.Zip(request)
  zip.filenames().forEach(filename => {
    const entityInZip = zip.entry(filename)
    const account = $k.Registry.type('upload').createInstance()
    account.setAttributeValue('file', entityInZip)
  })
}
```

1.4.2.12. Mustache templates

The following restlet function renders a document using the Mustache template library. It expects the following schema of a template document:

- a string attribute (internal name "template.id") to identify a template
- a document blob (internal name "template.file") containing the template, e.g. an HTML document
- a relation to a media type(internal name "template.contentType")

A query ("rest.articles") returns the elements that should be rendered. The Mustache library is registered as "mustache.js".

```
function respond(request, parameters, response) {
  // Include Mustache library
  $k.module("mustache.js");

  // Get template
  const templateId = parameters["templateId"];
  const templateelement = $k.Registry.elementAtValue("template.id",
templateId);
  const templateText = templateelement.attributeValue("template.file"
).text("utf-8");

  // Find elements
  const elements = $k.Registry.query("rest.articles").findElements
(parameters);

  // Prepare template parameters
  const elementsData = elements.map(element => ({
    name: element.name(),
    id: element.idNumber(),
    type: element.type().name()
  }))
  const templateParameters = {
    elements: elementsData
  };

  // Render with Mustache
  const output = Mustache.render(templateText, templateParameters);

  // Return the rendered document
  response.setText(output);
  response.setContentType(templateelement.relationTarget(
"template.contentType").name());
}
```

1.4.2.13. Java native interface

Java can be accessed via JNI (Java Native Interface).

WARNING

JNI is an experimental feature and has several restrictions:

- JNI cannot be used in triggers
- It is not possible to define classes (e.g. for callbacks)
- Generics are not supported
- JNI allows accessing system resources (files etc.), so take care when using JNI in REST services
- JNI has to be enabled and configured in the configuration file of each application. The classpath cannot be changed during runtime.

1.4.2.13.1. Configuration

```
[JNI]
classPath=tika\tika-app-1.5.jar
libraryPath=C:\Program Files\Java\jre7\bin\server\jvm.dll
```

1.4.2.13.2. Basic example

A list of classes is imported using the function `$jni.use()`. For each class, a function object of the same name is created and can be instantiated with new. All static properties are transferred to the function object. The java class namespace can optionally be omitted.

```
// Import the StringBuilder class, without namespace
$jni.use(["java.lang.StringBuilder"], false);
// Create a new instance
const builder = new StringBuilder();
// Javascript primitives and Strings are automatically converted
builder.append("Welcome to ");
builder.append($k.volume());
// toJS() converts Java objects to Javascript objects
$k.out.print(builder.toString().toJS());
```

1.4.2.13.3. Text/metadata extraction with Apache Tika

```
$jni.use([
    "java.io.ByteArrayInputStream",
    "java.io.BufferedInputStream",
    "java.io.StringWriter",
    "org.apache.tika.parser.AutoDetectParser",
    "org.apache.tika.metadata.Metadata",
    "org.apache.tika.parser.ParseContext",
    "org.apache.tika.sax.BodyContentHandler"
  ], false);
// Get a blob
const blob = $k.Registry.elementAtValue("uuid", "f36db9ef-35b1-48c1-9f23-
1e10288fddf6").attributeValue("ebook");
// Blobs have to be explicitely converted to Java byte arrays
const bufferedInputStream = new BufferedInputStream(new
ByteArrayInputStream($jni.toJava(blob)));
// Parse the blob
try {
  const parser = new AutoDetectParser();
  const writer = new StringWriter();
  const metaData = new Metadata();
  parser.parse(bufferedInputStream, new BodyContentHandler(writer),
metaData, new ParseContext());
  const string = writer.toString().toJS();
  // Print extracted metadata
  const metaNames = metaData.names().toJS().sort((a, b) => a.
localeCompare(b));
  for (let name of metaNames)
    $k.out.print(`${name} = ${metaData.get(name)}`).cr();
  // Print extracted text (first 100 chars)
  $k.out.cr().cr().print(`${string.substring(1, 100)} [...]\n\n(${string
.length} chars)`);
} catch (e)  {
  $k.out.print("Extraction failed: " + e.toString());
} finally {
  bufferedInputStream.close();
}
```

1.4.2.14. Parsing XML

The experimental DOMParser API offers a subset of the web API functionality for parsing XML content.

1.4.2.14.1. Read XML as DOM

```
const xml = '<rootNode><node1>Some text</node1><node2>More
text</node2></rootNode>'
const dom = new $dom.DOMParser().parseFromString(xml)
$k.out.print(dom.firstChild.children[0].nodeName)
```

1.4.2.14.2. Address nodes using XPath

```
const xml = '<rootNode><node1>Some text</node1><node2>More
text</node2></rootNode>'
const dom = new $dom.DOMParser().parseFromString(xml)
$k.out.print(dom.evaluate('//node2').stringValue)
```

1.4.3. Modules

1.4.3.1. Define modules

A module is defined with the define() function. The argument is either a module object or a function that returns an module object. A module should contain only a single definition.

Example: Define a module with a function jsonify())

```
$k.define({
    /*
     * Create a JSON object array for the elements
     */
    jsonify: function(elements) {
        return elements.map(element => {
            name: element.name(),
            id: element.idString(),
            type: element.type().name()
        });
    }
});
```

define() allows to specify dependencies from other modules. The following script defines a module that uses another module ("rest.common").

```
$k.define(["rest.common"], function(common) {
    return {
        stringify: function(elements) {
            return JSON.stringify(common.jsonify(elements), undefined, "
\t")
        }
    }
});
```

1.4.3.2. Use modules

A module can be used either with require() or module().

require() expects an array of module names and a callback function. The arguments of the callback function are the module objects. `require()` returns the return value of the callback function.

```
const elements = $k.Registry.query("rest.poi").findElements({name: "
Madrid"});
const json = $k.require(["rest.common"], function(common) {
     return common.jsonify(elements);
});
$k.out.print(JSON.stringify(json, undefined, "\t"));
```

module() expects the name of a module and returns the module object.

```
const json = $k.module("rest.common").jsonify(topics);
$k.out.print(JSON.stringify(json, undefined, "\t"));
```

module() can also be used to include scripts that doe not define a module at all. The script is evaluated and all declared functions are instantiated. These functions can then be called.

1.4.3.3. AMD

To integrate JavaScript libraries that support the AMD standard, you first have to globally define `require()` and `define()`.

```
this.define = $k.define;
this.define.amd = {};
this.require = $k.require;
```

If a library defines a module with a certain ID and you want to register this library under a different name, you can map the module IDs to registry IDs.

```
$k.mapModule("underscore", "lib.underscore");
```

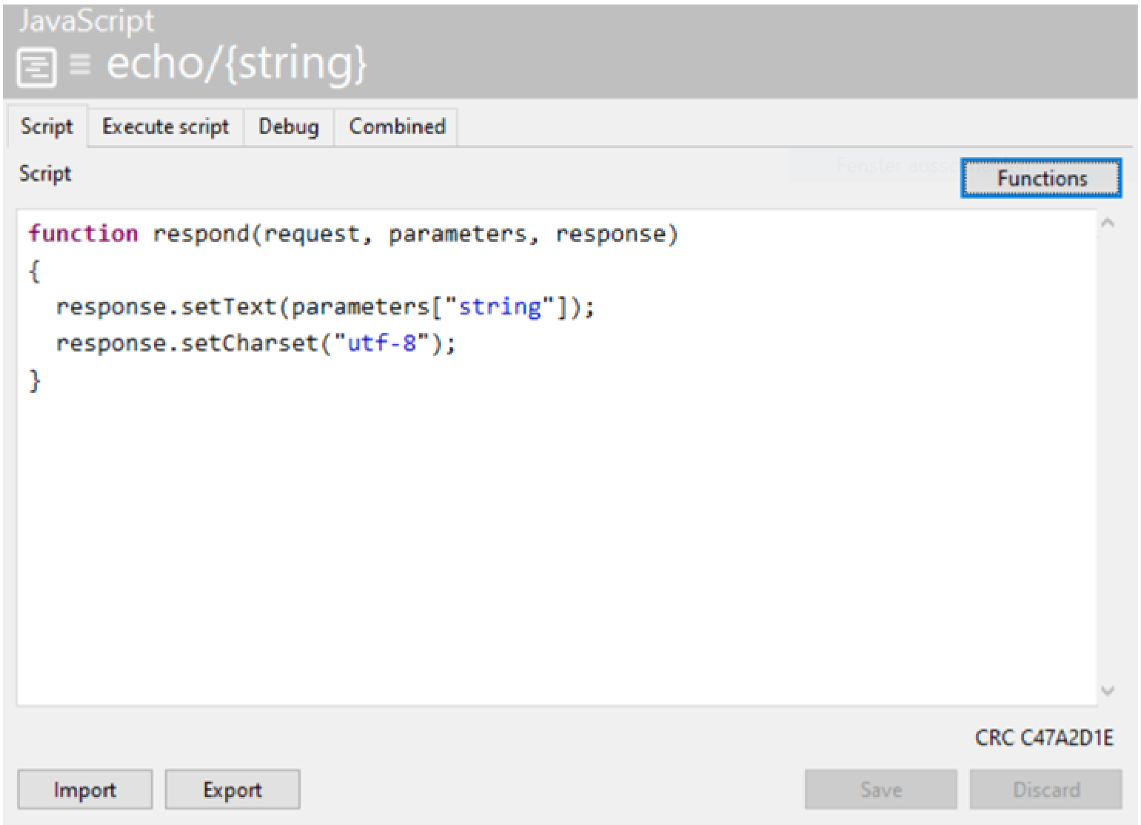
You can now register underscore.js as "lib.underscore" and use the "underscore" module defined there.

1.4.4. Editor and debugger

The editor itself provides the four tab-separated sections:

1.4.4.1. Script

Functionalities: Importing, editing and exporting scripts



```
function respond(request, parameters, response)
{
  response.setText(parameters["string"]);
  response.setCharset("utf-8");
}
```

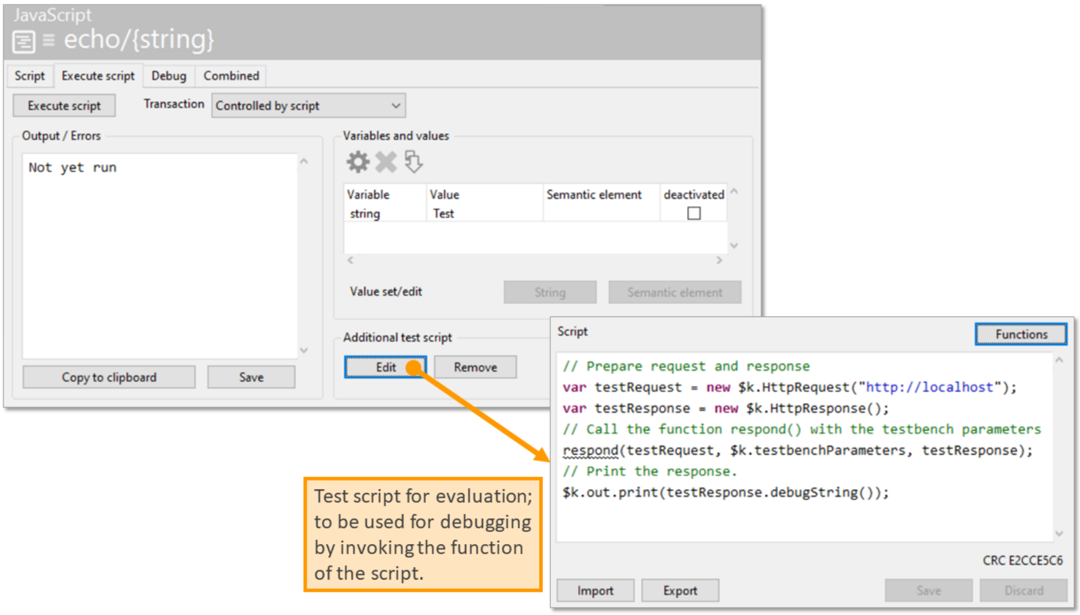
Function	Description
Import/Export	Allows importing/exporting of *.js files from/to the file system of the PC.
Functions	Lists all used and named function calls within the code. When selecting a function out of the list, the editor jumps to the line where the function call is located.
Save	Saves the changes made to the code (shortcut: Ctrl + S).
Discard	Discards all changes since the last time of saving.

1.4.4.2. Execute script

Functionalities: Executing script, displaying output, implementing test script for debugging.

The wrapper script shown in the following image is an example for testing a restlet in the Knowledge Builder. The test script can be defined in the script editor on the "Execute script" tab as "Additional test script".

Breakpoints can be set on the "Debug" tab.



Transaction: Controlled by script

```
Not yet run
```

Variable	Value	Semantic element	deactivated
string	Test		□

```
// Prepare request and response
var testRequest = new $k.HttpRequest("http://localhost");
var testResponse = new $k.HttpResponse();
// Call the function respond() with the testbench parameters
respond(testRequest, $k.testbenchParameters, testResponse);
// Print the response.
$k.out.print(testResponse.debugString());
```

CRC E2CCE5C6

Test script for evaluation; to be used for debugging by invoking the function of the script.

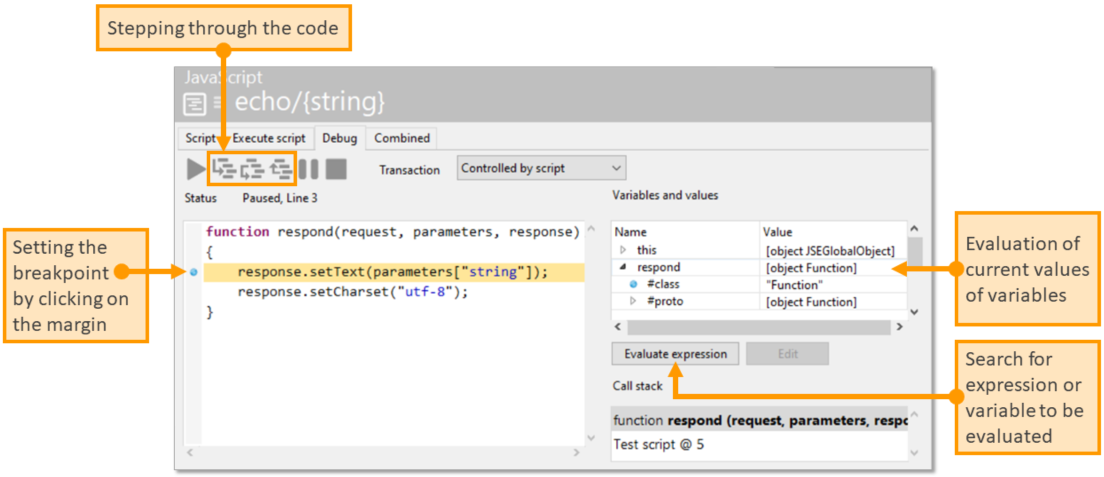
Function	Description
Execute script	Execute the script in one cycle, without interruption.

Function	Description
Transaction	Writing actions (e. g. creating or deleting objects) from within the script require a transaction. When the script is executed within an action of the web frontend, it will automatically be surrounded by a transaction. If the script is executed or debugged without being initiated by the web frontend, a transaction needs to encapsulate the script by using one of the following options: • **Controlled by script:** In this case, the script needs to contain code which encapsules actions within a transaction. For creating a transaction, see the i-views JavaScript API reference. • **Read only:** Allows executing/debugging of the script as long as no writing actions are being executed on the graph. • **Read and write:** Allows reading and writing of graph structures, without explicit transaction control in the script code.
Copy to clipboard	Copies the output to the clipboard.
Save	Stores the output to the filesystem.

NOTE The configuration for custom variables and the additional test script has been removed in version 5.7

1.4.4.3. Debug

Functionalities: Setting of breakpoints, stepping through code, evaluating expressions


Stepping through the code
JavaScript
echo/{string}
Script
Execute script
Debug
Combined
Setting the breakpoint by clicking on the margin
Evaluate expression
Edit
Evaluation of current values of variables
Search for expression or variable to be evaluated

Transaction: Controlled by script

Status: Paused, Line 3

```
function respond(request, parameters, response)
{
    response.setText(parameters["string"]);
    response.setCharset("utf-8");
}
```

Variables and values

Name	Value
this	[object JSEGlobalObject]
respond	[object Function]
#class	"Function"
#proto	[object Function]

Call stack

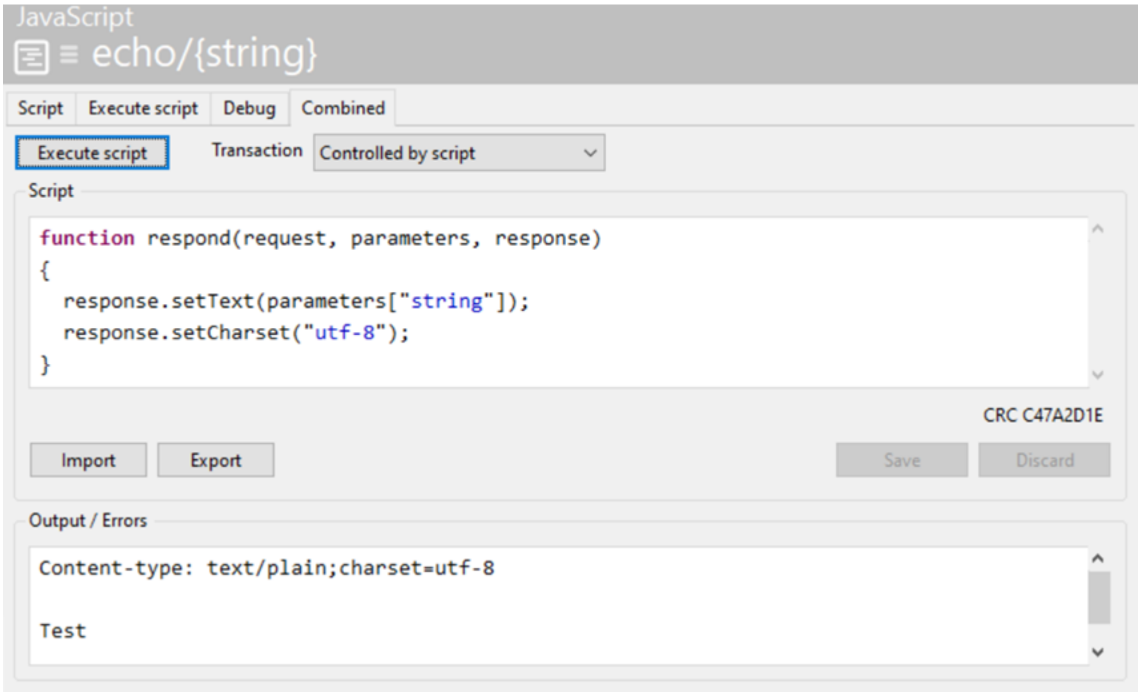
function **respond (request, parameters, respc**

Test script @ 5

Function	Description
Start/Resume (F4)	This action starts executing the script, if no breakpoints are set or it starts debugging the script (step-by-step) if at least one breakpoint has been set before. Caution: When a breakpoint is set at a code line which only includes a comment, the breakpoint will be ignored.
Single step (F5)	Executes the next logical step.
Single step (entire block) (F6)	Executes the current block completely.
Return from context (F7)	Executes the referenced code and returns to the originally invoked code.
Suspend (F9)	Suspends (pauses) executing the code. When debugging, the debugger goes on to the next breakpoint nevertheless.
Terminate (F10)	Terminates executing or debugging the script.
Evaluate expression	Serves for evaluating the value of a variable after the debugger has reached the next breakpoint.
Edit	If a variable is selected that points to an element of the knowledge graph, this button opens the editor window on the element.

1.4.4.4. Combined

Functionalities: Combines Script execution and output into one view



```
function respond(request, parameters, response)
{
  response.setText(parameters["string"]);
  response.setCharset("utf-8");
}
```

```
Content-type: text/plain;charset=utf-8

Test
```

1.4.5. API extensions

1.4.5.1. Additional functions

The API can be extended by adding functions to the prototypes. The following example extends schema prototype objects to print schema information.

```
// Print the schema of the instances and subtypes of a type
$k.Type.prototype.printSchema = function () {
  this.typesDomain().printSchema("Type schema of \"" + this.name() + "\"
");
  this.instancesDomain().printSchema("Instance schema of \"" + this.name()
+ "\"");
  this.subtypes().forEach(subtype => subtype.printSchema());
}

// Print information about a property type
$k.PropertyType.prototype.logPropertySchema = function () {
  $k.out.print("\t" + this.name() + "\n");
}

// Attribute types print their type
$k.AttributeType.prototype.logPropertySchema = function () {
  $k.out.print("\t" + this.name() + " (Attribute of type " + this
.valueRange().type() + ")\n");
}

// Relation types print their target domains
$k.RelationType.prototype.logPropertySchema = function () {
  $k.out.print("\t" + this.name());
  const inverse = this.inverseRelationType();
  if (inverse) {
    const inverseDomains = inverse.domains();
    if (inverseDomains.length > 0) {
      $k.out.print(" (Relation to ");
      let separate = false;
      inverseDomains.forEach(function (inverseDomain) {
        if (separate)
          $k.out.print(", ");
        else
          separate = true;
        $k.out.print("\"" + inverseDomain.type().name() + "\"");
      });
      $k.out.print(")");
    }
  }
  $k.out.cr();
}

// Print all properties defined for a domain
$k.Domain.prototype.printSchema = function (label) {
  const definedProperties = this.definedProperties();
  if (definedProperties.length > 0) {
    $k.out.print(label + "\n");
    definedProperties.sort((p1, p2) => p1.name().localeCompare(p2.
name()));
    definedProperties.forEach(propertyType => propertyType
.logPropertySchema());
  }
}

// Print the entire schema
$k.rootType().printSchema();
```

1.4.5.2. Define your own prototypes

The prototype of a semantic element is usually one of the built-in prototypes (Instance, Relation etc.). It is possible to assign custom prototypes to instances of specific types with the function mapInstances(internalName, prototype).

Example: A basket prototype

```
// Define a Basket prototype with a function totalPrice()
function Basket() { }

Basket.prototype.totalPrice = function() {
  return this.relationTargets("contains").reduce(
    (sum, item) => sum + item.attributeValue("price"),
    0
  );
}

// Set the prototype of instances of the basket type
$k.mapInstances("Basket", Basket);

// Print the total price of all baskets
const baskets =  $k.Registry.type("Basket").instances();
for (let basket of baskets)
  $k.out.print(basket.totalPrice() + "\n");
```

For using within other scripts, the module needs to be loaded first:

```
$k.module('myBasketSkript');
const basket = $k.Registry().elementWithID('ID_123');
$k.out.print(basket.totalPrice() + "\n");
```

1.5. REST services

The REST interface can be used for read and write access to the Knowledge Graph. To do so, *resources* and *services* need to be defined. Resources describe the interface behavior when accessing a specific path. The behavior of a resource is controlled using scripts. In addition, predefined resources may also be used. Services bundle resources of a common path together.

Access takes place via HTTP requests that are structured according to the pattern

```
https://<hostname>:<port>/[<service-path>||<service-id>]/<resource-path-and-parameter>
```

1.5.1. Configuration

The REST component must be added in the Knowledge Graph. It defines the necessary schema, which is found in the "Technical" area → "REST" in the Knowledge Builder.

The REST interface is usually provided by the bridge service. This responds to HTTP prompts using the REST configuration in the Knowledge Graph. The interface is already included in the tryout version of the Knowledge Builder, and no bridge service is required.

Changes to the configuration in the Knowledge Graph do not automatically affect interfaces that are already running. This only happens when the menu item "Administrator → Update REST interface" is executed in the main menu of the Knowledge Builder.

The bridge service requires a suitable configuration file (bridge.ini). The name of the server (host), the Knowledge Graph (volume) and the REST service ID need to be specified there. The line with "services" can be omitted entirely to automatically activate the resources of all existing service objects.

```
[Default]
host=localhost
loglevel=10

[KHTTPRestBridge]
volume=demo
port=8086
services=core,extra
```

1.5.2. Services

Services combine several resources. Resources may be contained in several services.

The service editor in the Knowledge Builder shows the resources in its structure view. A new resource can be created using “Link new” and is added to the service. A resource that has already been defined can be added to the service using “Link existing”.

1.5.3. Resources

Resources describe the response in the event of an HTTP request to the interface. There are the following types of resources:

Resource	Description
Script resource	Resources that can be defined by scripts.
Built-in resource	Predefined resource with a response that is defined by the system. These resources are created by the component.
Static file resource	Serves files from the file system.

A resource has the following configurable properties:

Property	Description

Path pattern	Defines the URL of the resource relative to the address of the service. The path can be parameterized by adding parameters in curly brackets:
	`albums/{genre}`
	Several parameters can be specified. Each parameter must, however, be a part completely separated by `/`:
	`albums/{genre}-{year}`
	is not valid,
	`albums/{genre}/{year}`
	is valid

Part of service	Services that use this resources
Description	Description for documentation purposes
Requires authentication	Authentication is required for access to the resource

1.5.3.1. Methods

A resource is linked to one or more *methods* . This defines the response as well as the supported input and output types (content types). The methods and types of the HTTP request are used to select a suitably configured method.

In the structure view, methods are displayed as subelements of resources and can be created/deleted there.

Property	Description
HTTP method	Supported HTTP methods (GET, POST, PUT, DELETE). Multiple entries are possible.
Input media type	Only POST/PUT: expected content type of the content of the request.
Output media type	Content type of the response. If the request specifies an expected content type via `Accept`, the output media type must match this.
Script	Registered script for the definition of the response (only relevant for script resources)
Transaction	Transaction control (only relevant for script resources)

Transaction control is relevant for write accesses to the Knowledge Graph because these are only possible within a transaction.

Transaction control	Description
Automatic	For GET read access only; for POST/PUT/DELETE the script is executed in a transaction. This is the default setting.
Controlled by script	No transaction; the script must control this itself.
Read	Read access only; the script cannot start a transaction.
Write	The script is executed in a transaction.

1.5.3.2. Script resource

A script is used to define the response to an HTTP query for a method of a script resource. For this purpose, the respond function (request, parameters, response) that must be defined in the script is called from the interface.

Argument	Type	Description
request	$k.HttpRequest	Request (URL, header etc.)
parameters	object	Parameter extracted from the request
response	$k.HttpResponse	Response

The function then fills out the header and content of the response. There is no return value.

If a type has been defined for a parameter (e.g. xsd:integer), then the converted value is transferred. If not, a string is transferred. Parameters that can occur more than once by definition are always transferred as an array.

If an output content type was defined for the response in the method, this is set automatically. Alternatively, it is also possible to define the content type in the script.

The following script searches for albums and converts them into JSON objects. The parameters of the resource are transferred to the query as search parameters.

```
function respond(request, parameters, response) {
  const albums = $k.Registry.query("albums").findElements(parameters);
  const albumData = albums.map(album => ({
    name: album.name(),
    id: album.idString()
  }));
  response.setText(JSON.stringify(albumData, undefined, "\t"));
  response.setContentType("application/json");
}
```

This script could be used, for example, in the resource

```
albums/{genre}/{year}
```

and use the search parameters "genre" and "year" as the search conditions in the "albums" query.

1.5.3.3. Built-in resources

Built-in resources are predefined resources with a response specified by the system. Each predefined response can be assigned using an assigned value of the string attribute *Rest resource ID*.

Rest resource ID	Method	Description
BlobResource	GET	Returns the binary content of an existing blob attribute. The blob attribute is identified using the query parameter `blobLocator`. Optionally, the parameter `allowRedirect` can be used to define that blobs may not be obtained directly by the blob service (fixed value: false).

Rest resource ID	Method	Description
BlobResource	POST, PUT	Changes the binary content of a blob attribute. The blob attribute is identified using the query parameter `blobLocator`. Depending on the type of the `blobLocator`, a new attribute is created or an existing one changed.
EditorConfigResource	GET, POST, PUT	Output and import of an XML representation of a semantic element.
ObjectListResource	GET	Returns a table of instances or subtypes of the specified type. The set of objects can optionally be filtered, sorted or be defined directly.
ObjectListPrintTemplateResource	GET	Returns a table of instances or subtypes in printed form. The print template must be specified.
ObjectListPrintTemplate ResourceWithFilename	GET	Returns a table of instances or subtypes in printed form. The print template must be specified. The parameter (filename) is not evaluated, and is only used to improve its use in the browser.
TopicIconResource	GET	Returns the icon or image of the specified semantic element.

Version 4.1 or higher of i-views allows a JavaScript (`rest.preprocessScript`) to be attached to the resource. The function it contains (`preprocessParameters(parameters, request)`) can provide the parameters. Any blobLocator (or the associated blob attribute) still missing can, for example, be determined from the parameters transferred, which would otherwise require an additional script resource call.

1.5.3.3.1. BlobResource

This built-in resource allows contents of file attributes to be loaded and saved.

Download

The `GET` method can be used to download the binary content of an existing file attribute. The file attribute is then identified by means of the query parameter `blobLocator`.

Upload

In the case of an upload, the parameter `blobLocator` either identifies an existing file attribute or a potential file attribute (i.e. new one to be created). The syntax for a potential attribute has the following form: `PP~ID1_115537458~ID36518_344319903`, whereby the first ID represents the semantic element and the second ID the attribute prototype.

The binary data can optionally be transmitted as a multipart or single part. In the case of multipart, several files can potentially be uploaded at the same time, which, of course, only makes sense when each file is written to a newly created file attribute. In any case, the file name must be set for every file transmitted.

The optional parameter `binaryKey` defines the form key used to transmit the binary data in multipart.

If the optional Boolean parameter `uploadOnly` is set to `true`, then the binary data are uploaded only, and are not written into the file attribute. This mode is used in interplay with the ViewConfiguration Mapper. The JSON value is returned in this case (fileName, fileSize, binaryContainerId), which can be written into the attributes using the mapper in a second step. The content type of the returned JSON value is normally `application/json`, however can be set to another value using the parameter `overrideContentType` should the browser (e.g. IE) encounter problems doing so.

1.5.3.3.2. Topic icon

The following path can be used to load the image file to a given topic. If an individual does not have an image file of their own, the image file of the type is used, which is, in turn, inheritable. The optional parameter `size` can be used to select the image file with the size that is most suitable, providing several image sizes are saved in the Knowledge Graph.

```
http://{server:port}/baseService/topicIcon/{topicID}?size=10
```

1.5.3.3.3. Object list

The following path can be used to request an object list in the JSON format:

```
http://{server:port}/baseService/{conceptLocator}/objectList
```

The object list type is referenced via the `conceptLocator` parameter, which is followed by the format for topic references in the remaining URL (see link).

Alternatively, the `conceptLocator` can also reference the single prototype (individual or type) of the type to be used.

The optional `name` parameter determines the object list to be used for the output.

Filter

The optional and multi-value query parameter `filter` can be used to filter the object list. A filter can take two different forms:

1. <column name/column no.> ~ <operator> ~ <value>
2. <column name/column no.> ~ <value>

The available operators are: equal, notEqual, greater, less, greaterOrEqual, lessOrEqual, equalCardinality, containsPhrase, covers, isCoveredBy, distance, fulltext, equalGeo, equalPresentTime, greaterOverlaps, greaterPresentTime, lessOverlaps, lessPresentTime, equalMaxCardinality, equalMinCardinality, overlaps, unmodifiedEqual.

Sorting

The optional and multi-value query parameter `sort` can be used to sort the object list. The order of sorting parameters determines the sorting priority. Sorting can be specified in two forms:

1. <column name>
2. {-}<column no.>

If you prefix a minus sign in variant 2, sorting is performed in descending order, otherwise it is in ascending order.

Setting the starting set of the list

The optional `elements` query parameter can be used to transmit a comma-separated list of topic references to be used as list elements.

As the list of elements can be very long, the request can also be sent as POST and the parameters can be transferred as form parameters.

Inheritance

Inheritance can be suppressed via the optional query parameter `disableInheritance`. The parameter only makes sense if no `elementsPath` is set.

JSON output format (example)

```
{
  "rows": [
    {
      "topicID": "ID123_987654321",
      "row": [
        "MM",
        "Mustermann",
        "Max",
        "111",
        "m.mustermann@email.net",
        "10",
        "6",
        "2000-01-01",
        "project A, project B"
      ]
    },
    {
      "topicID": "ID987_123456789",
      "row": [
        "MF",
        "Musterfrau",
        "Maxine",
        "222",
        "m.musterfrau@email.net",
        "10",
        "8",
        "2000-01-01",
        "project X, project Y, project Z"
      ]
    }
  ],
  "columnDescriptions": [
    {
      "label": "Login",
      "type": "string",
      "columnId": "1"
    },
    {
      "label": "Last name",
      "type": "string",
      "columnId": "2"
    },
    {
      "label": "First name",
      "type": "string",
      "columnId": "3"
    },
    {
      "label": "Telephone extension",
      "type": "string",
      "columnId": "4"
    },
    {
      "label": "email",
      "type": "string",
      "columnId": "5"
    },
    {
      "label": "Availability",
      "type": "number",
      "columnId": "6"
    },
    {
      "label": "Expenditure",
      "type": "string",
      "columnId": "7"
    },
    {
      "label": "created on",
      "type": "dateTime",
      "columnId": "8"
    },
    {
      "label": "Project",
      "type": "string",
      "columnId": "9"
    }
  ]
}
```

1.5.3.3.4. Object list print template

The following path can be used to fill an object list in a ‘print template for list’ and download the result:

```
http://{server:port}/baseService/{conceptLocator}/objectList/printTemplate
/
```

```
{templateLocator}/{filename}
```

The service functions exactly the same way as retrieving an object list, however, as an additional parameter, features a reference to the individual of the type *print template for list* in the Knowledge Graph.

`templateLocator` must have one of the formats described under *General*

The optional path parameter `filename` is not evaluated, and is used to improve browser performance.

The header field `Accept` is used to control the output format into which conversion occurs. If there is no header field, or the value is `*/*`, no conversion occurs. Accept with multiple values is not supported and will result in an error message.

The optional query parameter `targetMimeType` is used to overwrite the value of the `Accept` header field. This is necessary when the user would like to call the request from a browser, and has no influence on the header fields.

1.5.3.3.5. Print topic

The following path can be used to fill out a topic in a print list template and download the result:

```
http://{server:port}/baseService/{topicLocator}/printTemplate/
```

```
{templateLocator}/{filename}
```

`templateLocator` must have one of the formats described under *General*

The optional path parameter `filename` is not evaluated, and is used to improve browser performance.

The header field `Accept` is used to control the output format into which conversion occurs. If there is no header field, or the value is `*/*`, no conversion occurs. Accept with multiple values is not supported and will result in an error message.

The optional query parameter `targetMimeType` is used to overwrite the value of the `Accept` header field. This is necessary when the user would like to call the request from a browser, and has no influence on the header fields.

1.5.3.3.6. Document format conversion

You can use the following path to convert a document to another format (e.g. odt in pdf):

```
http://{server:port}/baseService/jodconverter/service
```

The service maps the JOD converter (see http://sourceforge.net/projects/jodconverter/) and is used for downward compatibility for installations that used to be operated with the JOD converter.

For the service to work OpenOffice/LibreOffice (version 4.0 or above) must be installed and the configuration file "bridge.ini" must have an entry that refers to the "soffice" file.

```
[file-format-conversion]
sofficePath="C:\Program Files (x86)\LibreOffice 4.0\program\soffice.exe"
```

1.5.3.4. Static File Resource

Delivers files from the file system.

With this type of resource, you merely use *Path pattern* to specify the directory under which the files are delivered. The directory is addressed relative to the content directory of the REST bridge.

Example:

Enter an `icons` directory with the file `bullet.png` . The path pattern of the resource is `icons`, the corresponding service has the *Service ID* `test`. The file `bullet.png` is thus accessed via:

```
http://localhost:{bridge-port}/test/icons/bullet.png
```

1.5.3.5. Resource parameters

The *parameters* for the resource can be defined below methods. This is not absolutely essential, does, however, have a number of advantages:

- The parameters can be checked and converted by using type specifications (e.g. in numbers or objects)
- Documentation for customers

The following parameter properties can be configured:

Property	Value
Style	Type of parameter • path (part of the path of the URL) • query (query parameter of the URL) • header (HTTP header)
Type	Data type of the parameter. Parameters have been validated and converted when passed to the script.
Repeating	Parameters may occur multiple times. When this is activated, an array of values is always passed to the script, even if there is only one parameter value in the request.
Required	Parameter must be provided.
Fixed value	Default value when no parameter was provided.

1.5.4. Authentication

Each resource can be assigned to an authentication object to restrict the access and to bind the request to a user object.

Authentications are defined as objects of type "Authentication" in the "REST" area. Resources are assigned to an authentication object with the relation "Authentication" of the resource.

The kind of authentication is defined by the attribute "Authentication type" of the authentication object. Some configuration values are only available for specific authentication types.

Authentication name

Arbitratry name to distinguish configurations

Cache duration

The user lookup is cached for this amount (in seconds). See Login configuration.

Trusted login

Credentials (e.g. the password) are not validated when enabled

1.5.4.1. Authentication schemes

No authentication

As the name says, no authentication is performed and thus no user is activated by default. If a REST resource using this authentication defines a fallback user instance, then the request is bound to this object.

Basic authentication

Authenticate a user with the Basic scheme defined by RFC 7617. Requires identiying a user object with a matching password. See Login configuration.

Bearer authentication

Uses JSON Web Tokens defined by RFC 7519 to authenticate requests.

Users are identified by the subject claim. The value of this claim is the element ID of the user object, unless a "Token subject attribute" is specified in the authentication configuration that specifies the value of an attribute of the user objects. The Login configuration is not used here.

By default, the token can be passed as header, cookie or query paramerer. This can be customized by setting the attributes *Allow authorization header*, *Allow cookie* and *Allow query parameter*.

The "Allow cookie" attribute can be set to define additional cookie parameters, which can be specified as a meta attribute. The vaue is added to the *Set-Cookie* header in the response.

The *Cookie/parameter name* attribute defines the name of the cookie / query parameter.

Token expiry interval and *Token renew interval* define the lifespan of new tokens.

Negotiate

Uses Windows Negotiate defined by RFC 4559 to authentication requests. This authentication type is only supported on Windows platforms.

It is not recommended to use this authentication scheme. Negotiate authenticates connections, not requests, which is not suitable for load balancing.

Scripted

This authentication type uses a script with four custom definable functions to handle user authentication. This is useful to outsource authentication to third party central authentication providers like OpenID Connect, or to implement custom authentication schemes.

NOTE

While the all script functions have sensible default implementations that prevent any unauthorized access, it is easily possible to expose parts of the REST service due to careless implementations.

Each function receives the request object, an object with the request's query parameters and a response object as input. The main authentication function that is evoked with each request is called **authenticate**:

```
function authenticate(request, parameters, response) {
    const encryptedToken = request.cookies().access_token
    if (encryptedToken) {
        const token = $k.JWT.parse(encryptedToken)
        try {
            token.verify()
            return $k.Registry.elementWithID(token.payload().sub)
        } catch (e) {}
    }
    response.setCode(302)
    response.setHeaderField('Location',
'http://auth.example.com?return_url=' + request.url())
}
```

In this example, the authentication routine expects an encrypted JWT token called **access_token** that is sent as cookie with the request. If the token is present and successfully verified, a user object can be derived from the token's payload. The request is then executed under this user. Otherwise, a redirect (HTTP 302) to an external authentication provider is triggered. It is implicitly assumed that the authentication provider will set the access_token cookie after successful authentication, such that the next request to this endpoint will succeed.

Note that if the authenticate function does not return a user instance or null, the authentication is considered failed, and the actual request handler is not evoked. The function may return null to indicate a successful authentication from which no user can be derived. The request handler is then evoked without an active user.

The functions **login**, **logout** and **renew** can be implemented for more sophisticated authentication schemes that support user logout etc. They are evoked when the respective built-in request (accessToken/login, accessToken/logout, accessToken/renew) is called. These are also used to handle the authentication flow of the View Configuration Mapper and therefore need to be implemented to ensure smooth interaction.

1.5.4.2. Login configuration

The Basic authentication looks up user objects with a login search. The login search can be defined by creating a query with the registry key *login*. If the query is a structural query, then the identity is passed to each defined query parameter.

If no login search is configured, then the user object is determined by searching for the value of the attribute *login* of the user objects. This attribute is defined when the user type is defined in the access rights settings.

The type of the user objects is defined by the access rights settings. The found objects must match the specified type, otherwise the authentication fails.

If no or more than one user object is found for the identity, then the authentication fails.

The hash value of the provided password is checked against the hash value stored in the corresponding attribute of the user object. The authentication fails if they do not match. The password check is skipped if *Trusted login* is activated in the authentication configuration.

The request is then bound to the user, e.g. access rights are checked for this user.

1.5.5. CORS

In the case of OPTIONS requests, the REST interface responds by default with

```
Access-Control-Allow-Origin: *
Access-Control-Allow-Headers: Origin, X-Requested-With,Content-Type,
Accept
```

These headers can be configured in the configuration file (bridge.ini):

```
[KHTTPRestBridge]
accessControlAllowOrigin=http://*.i-views.de
accessControlAllowHeaders=Origin, X-Requested-With,Content-Type, Accept
```

1.5.6. OpenAPI documentation

i-views offers the possibility to generate OpenAPI-3.0 documentation for configured services. For this purpose, service configurations and resource configurations can be enriched with documentation data.

1.5.6.1. Configuration

1.5.6.1.1. Service

Property	Description	Mapping to OpenAPI 3.0
Service Description	Free text description of the service; supports GitHub Flavored Markdown	`info.description`
Service Version	Version specification which is interpreted as Semantic Version.	`info.version`
Service ID		`info.title`
OpenAPI components	Script which generates reusable OpenAPI-3.0 components in forms of a JSON object.	`components`

1.5.6.1.2. Resource

Property	Description	Mapping to OpenAPI 3.0
Resource Description	Free text description of the resource, supports GitHub Flavored Markdown	`paths.{path}.description`

1.5.6.1.3. Method

The mappings to the OpenAPI elements are specified relatively to paths.{path}.{method}

Property	Description	Mapping to OpenAPI 3.0
Method description	Free text description of the resource; supports GitHub Flavored Markdown	`.description`

Property	Description	Mapping to OpenAPI 3.0
Request Body	See section *Request Body*	`.requestBody`
Response	See section *Response*	`.responses.{code}`

1.5.6.1.4. Parameter

The mappings to the OpenAPI elements are specified relatively to paths.{path}.{method}.parameters.{index}

Property	Description	Mapping to OpenAPI 3.0
Parameter Description	Free text description of the parameter; supports GitHub Flavored Markdown	`.description`
Parameter name		`.name`

Repeating	In case of path parameters, this option MUST NOT be enabled.	`.explode: true`
		`.schema: {"type": "array"}`

Required	In case of path parameters, this option MUST be enabled.	`.required`
Style		`.in`
Type		`.schema`

1.5.6.1.5. Request Body

The mappings to the OpenAPI elements are specified relatively to paths.{path}.{method}.requestBody

Property	Description	Mapping to OpenAPI 3.0
Request Body Description	Free text description of the body; supports GitHub Flavored Markdown	`.description`
Required		`.required`
Media type	Replaces the *Input media type* which could be stored at the method regarding i-views 5.3. i-views 5.4 supports description of several possible request formats. See section *Media Type* .	`.content.{mediaType}`

1.5.6.1.6. Response

For a valid OpenAPI documentation, a response needs to be documented for each request. The specified mappings relate to their response object, respectively.

Property	Description	Mapping to OpenAPI 3.0
Response Code	HTTP status code of the response	Key
Response Description	Free text description of the response; supports GitHub Flavored Markdown	`.description`
Media type	Replaces the *Output media type* which could be stored at the method regarding i-views 5.3. i-views 5.4 supports description of several response formats. See section *Media Type* .	`.content.{mediaType}`

1.5.6.1.7. Media Type

From OpenAPI 3.0 and on, the support for several input formats and output formats by a request can be documented by specifying several media types.

Property	Description	Mapping to OpenAPI 3.0
Media Type Name	The MIME string which defines the Media Type.	Key
OpenAPI schema	Script which generates a JSON schema object, which in turn describes the format of the structure with this Media Type.	`.schema`

1.5.6.1.8. JSON schema definitions

For creating JSON schema for further descriptions of input and output, scripts can be defined at different locations. The scripting supports a subset of the JSON schema standard which can be seen in the OpenAPI specification.

Example script for OpenAPI *components* :

```
function openAPIComponents() {
    return {
        "schemas": {
            "Example": {
                "properties": {
                    "id": { "type": "integer" },
                    "name": { "type": "string" }
                }
            }
        }
    }
}
```

Exmaple script for *OpenAPI schema* with reference to the definition above:

```
function swaggerJSONSchema() {
    return {
        "$ref": "#/components/schemas/Example"
    }
}
```

1.5.6.2. Generating the API documentation

1.5.6.2.1. Manual generation within the KB

For generating a .json file manually by means of the OpenAPI documentation within the Knowledge Builder, a button *Export as OpenAPI 3.0* is provided above the list of the services.

1.5.6.2.2. CLI

The same export also is provided by means of the command line interface:

```
bridge.exe -exportBuiltInRequestAPI {filename} {serviceID}
```

1.5.6.2.3. As REST API endpoint

In i-views 5.4, a built-in resource called *APIResource* is available, which provides the API documentation. It can be added to the respective service by means of the button and it is available at /api or at the configured path accordingly.

1.6. Reports and printing

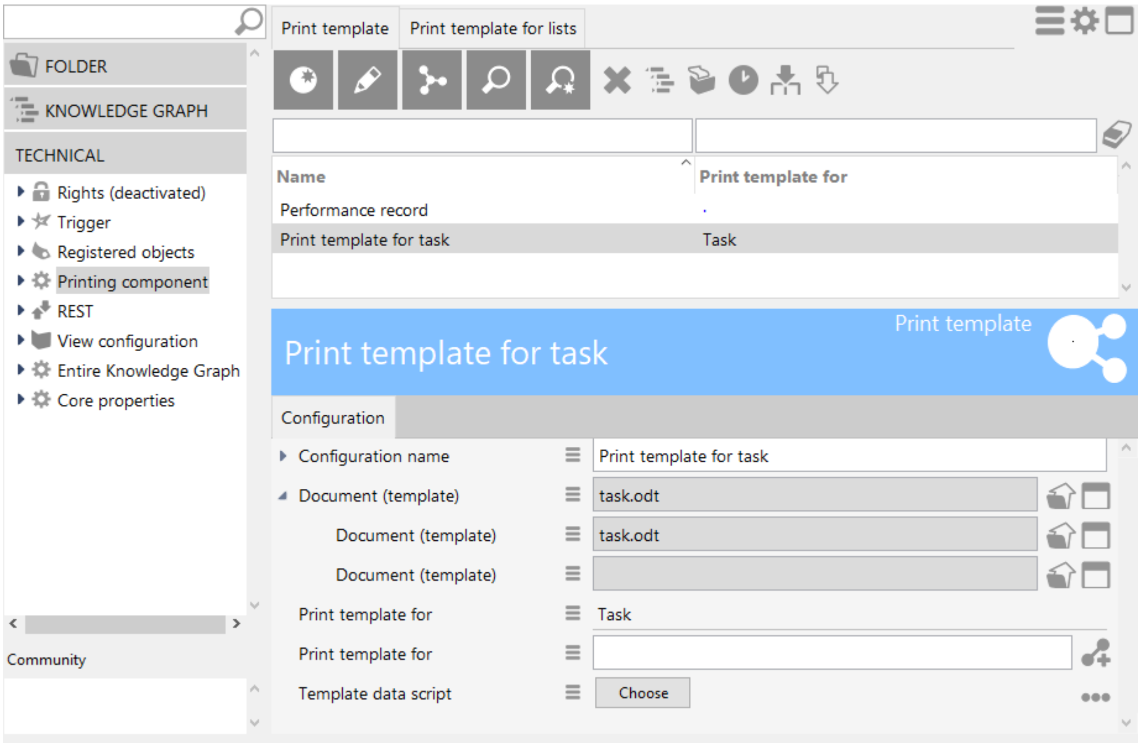
You can use the printing component to use document templates (ODT/DOCX/XLSX/RTF files) with KPath expressions on objects or object lists and then use them to generate an adapted output file, which can be either printed or stored.

The adding of the printing component via the Admin tool creates configuration schemas for objects ("print template") and lists ("print template for lists") in the Knowledge Graph. The existence of this component is prerequisite for the print function being available in Knowledge Builder or via the REST interface.

1.6.1. Create print templates

In Knowledge Builder, print templates are created in the "Technical → Printing component" area. Each print template object contains a print template document (ODT, DOCX, RTF) and a relation that specifies to which objects the print template is to be applied.

The following example shows an ODT print template for objects of the "Task" type.



Name	Print template for
Performance record	
Print template for task	Task

Print template for task
Print template

Configuration

Configuration name	Print template for task
Document (template)	task.odt
Document (template)	task.odt
Document (template)	
Print template for	Task
Print template for	
Template data script	Choose

The following chapters explain how print template documents are created.

1.6.1.1. Create RTF templates

The RTF template files can contain evaluable KPath expressions with the key words **KPATH_EXPAND** and **KPATH_ROWS** as well as calls for registered KScripts with the key words **KSCRIPT_EXPAND** and **KSCRIPT_ROWS** . The path expressions or the name of the script to be called are always placed between angle brackets and after the key word, separated by a space.

KPATH_EXPAND

The KPath expression after this key word should return a single semantic object or a simple value (date, string etc.). In the evaluation the original expression is replaced by the result. The formatting of the expression is retained, and breaks in the value are converted into line breaks.

- *Example:* ** The template is:

```
Sender:
<KPATH_EXPAND @$address$/rawValue()>
```

After the evaluation the output file says:

```
Sender:
intelligent views gmbhJulius-Reiber-Str. 1764293 Darmstadt
```

KSCRIPT_EXPAND

As an alternative to the path expression, KSCRIPT_EXPAND can be used to call a registered KScript. The output of this script (script elements with <output>) is transferred to the document. Scripts are registered in the Knowledge Builder in the folder TECHNICAL/Registered objects/Scripts

- *Example:* ** The template is:

```
<KSCRIPT_EXPAND aScriptWithOutput1to9>
```

After the evaluation the output file says:

```
123.456.789
```

KPATH_ROWS

This expression must be in a table. The KPath expression after this key word must return a list of semantic objects. During evaluation the table row of the KPATH_ROWS expression is evaluated once for each result of the KPath expression. This allows tables to be completed dynamically. By the way, it does not matter which column contains the KPATH_ROWS expression.

KSCRIPT_ROWS

In case of KSCRIPT_ROWS the objects for the table rows are determined via a registered KScript. The name of the registered script is specified directly after KSCRIPT_ROWS. The script must be of the KScript type and return the objects for output.

Example: The template is:

Column1	Column2
<KSCRIPT_ROWS allPersons><KPATH_EXPAND @$lastname$>	<KPATH_EXPAND @$firstname$>

After the evaluation the output file says:

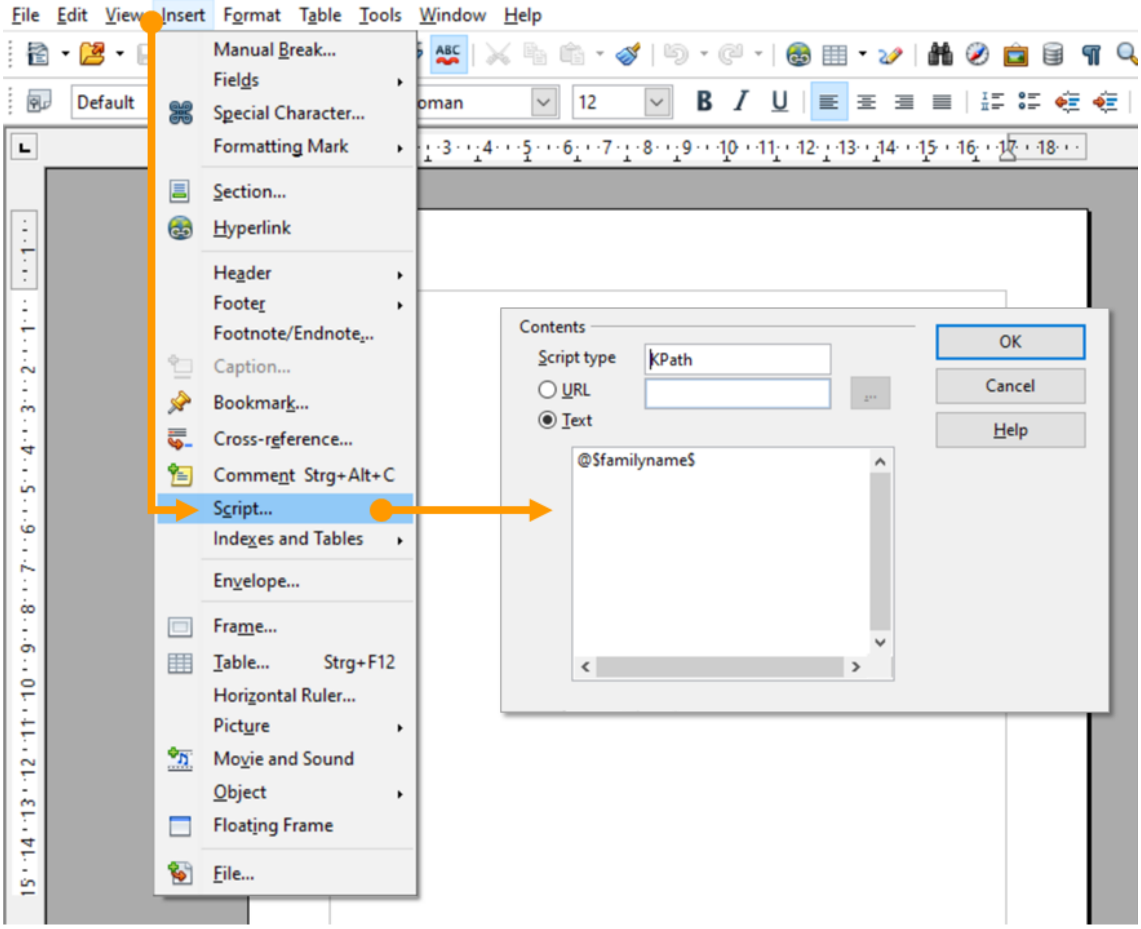
Column1	Column2
Meier	Peter
Schulze	Helmut

1.6.1.2. Create ODT documents (OpenOffice)

Printing using the ODT format (Open Document Text, open standard) has many advantages compared to the RTF format:

- The embedded script instructions are not part of the text, and are instead filed in special script elements. This ensures that the formatting is not destroyed by lengthy scripts.
- The ODT format supports a large set of format instructions (comparable with MS Word) that RTF cannot process.
- As a format, RTF does not have a uniform standard (MS Word can, for example, “do more” than the standard).
- Editing of the RTF templates is highly fragile. MS Word, above all, tends to ‘supplement’ the templates with control elements (for example, the cursor position current during the most recent editing), preventing the scripts from being reliably identified.

ODT templates can be created using OpenOffice or LibreOffice. They are created the same way as RTF templates are created, with the only difference being that the path/script instructions are saved in script elements, as the following diagram shows.



Script type: KPath

○ URL

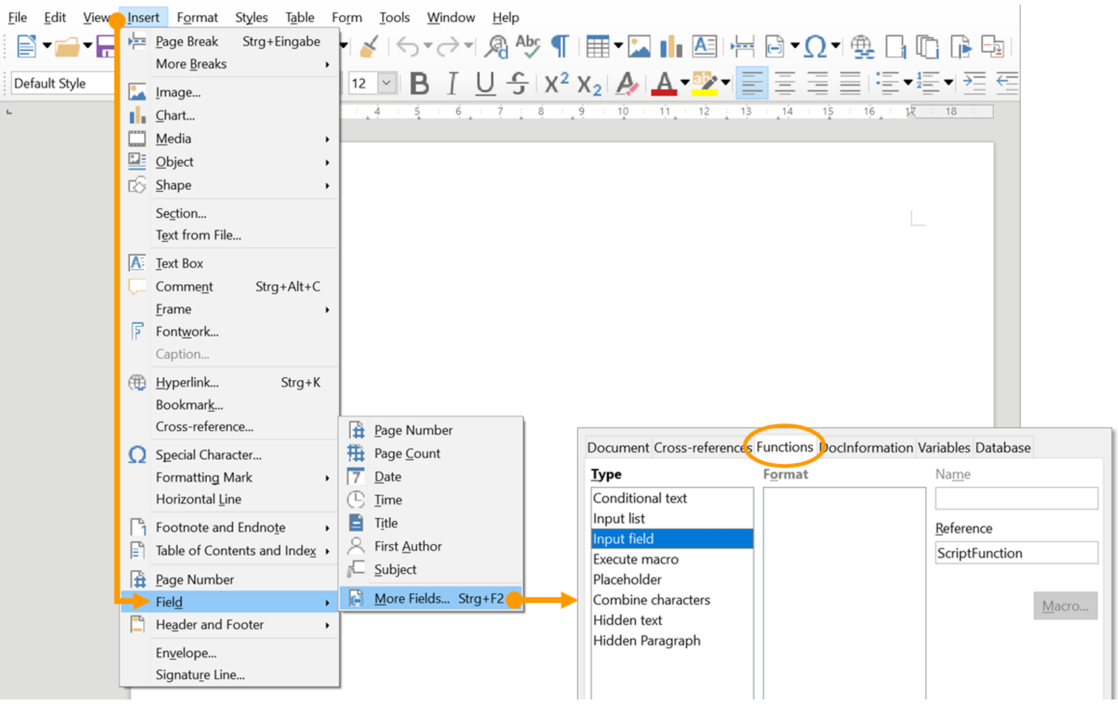
● Text

@$familyname$

The script field can no longer be integrated in LibreOffice 5. As an alternative to this, the “Input field” can be used:

Insert > Field command > Other field commands (alternative keyboard shortcut Ctrl+F2)

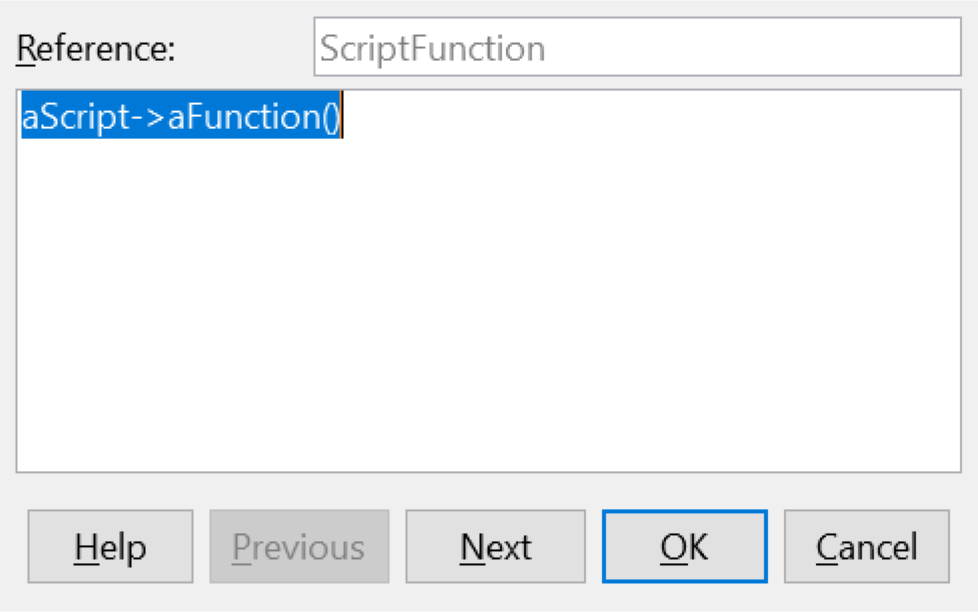
The input field is found there on the “Functions” tab.



Type: Input field

Reference: ScriptFunction

“Note” is equivalent to the previous “Script type”; after clicking on insert, another window opens in which the script can be entered.



Reference: ScriptFunction

```
aScript->aFunction()
```

Available script types

There are the following script types:

- **KPath** : analogous to **KPATH_EXPAND**
- **KScript** : analogous to **KSCRIPT_EXPAND**
- **KPathRows** : analogous to **KPATH_ROWS**

- **KPathImage** : for embedding images
- **ScriptFunction** : Calls a function of a registered script. A string with the following format is expected as text:

```
ScriptID->Functionname()
```

The function call is automatically expanded by two arguments: the semantic element and the variables determined by the environment

An example of a script that was called:

```
function headerLabel(element, variables)
{
   return element.name().toLocaleUpperCase();
}
```

- **ScriptRowsFunction** : Analogous to ScriptFunction. Table rows are generated for the returned objects, analogous to KPathRows.
- **ScriptImageFunction:** for adding bitmap images
- **ScriptSVGImageFunction:** for adding SVG drawings * **DataPath:** The “script for generating JSON contents” must be set on the print template. The corresponding key can now be used to access the values of the JSON object.

Example of generating the JSON object:

```
function templateData(element)
{
    return {
            name: element.name(),
            idNumber: element.idNumber(),
            someData: { idString: element.idString() }
    }
}
```

To access the value idString, for example,

```
someData.idString
```

must be set as text. * **DataRowsPath:** In table rows or sections (Libre Office only), DataRowsPath can be used to transform an array of objects in the templateData JSON to a table or sequence of sections in the printed document. Each object in the array is transformed into a new row with identical formatting as the row the DataRowsPath element is placed in. This allows having lists of variable length in the printed document. DataPath and DataConditionPath elements in the same table row or section as a DataRowsPath element are interpreted relative to the path of the DataRowsPath element.

```
function templateData(element) {
    return {
        rowData: [
            { name: "Element 1", someValue: 123 },
            { name: "Element 2" }
        ]
    }
}
```

- **DataConditionPath:** Like DataRowsPath elements, DataConditionPath can be placed in table rows or sections. Unlike DataRowsPath elements, DataConditionPath can reference anything in the templateData JSON, not only arrays of objects. When the referenced property in the templateData JSON is a JavaScript falsy value (false, undefined, null, 0 or an empty String) or an empty Array, the table row or section the DataConditionPath element is placed in is removed from the printed document.

File attributes or URLs can be used for embedding images. When URLs are used, an attempt is made to load an image from the address specified.

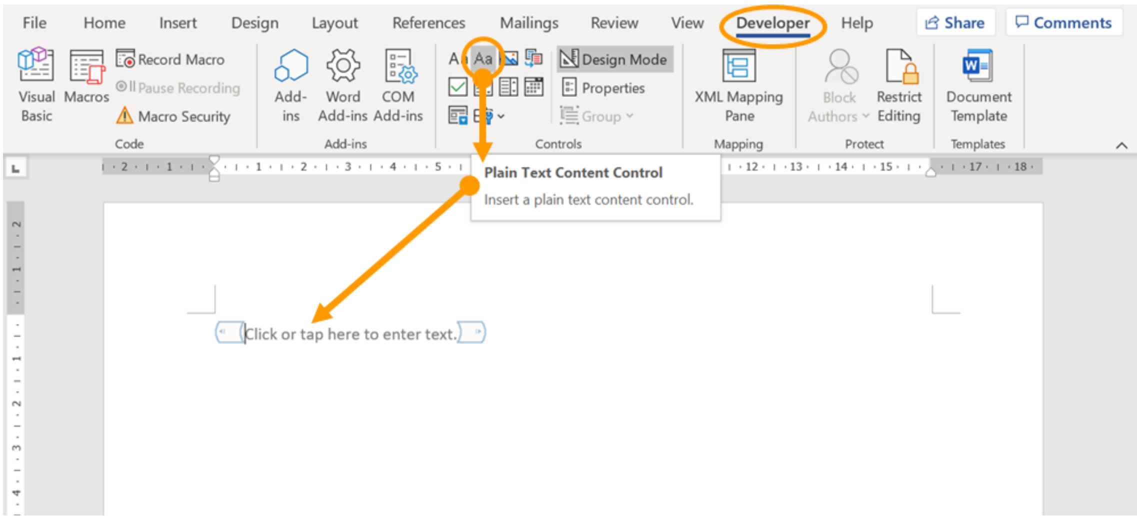
Embedded images are always sourced in their original size (at 96d dpi). If another size should appear in the printout, a frame with the required dimensions (absolute dimensions in cm must be used!) must be built around the script element. The resulting embedded image is then fit into the frame so that the frame dimension is not exceeded while retaining the image aspect ratios.

1.6.1.3. Create DOCX documents (Micrsoft Word)

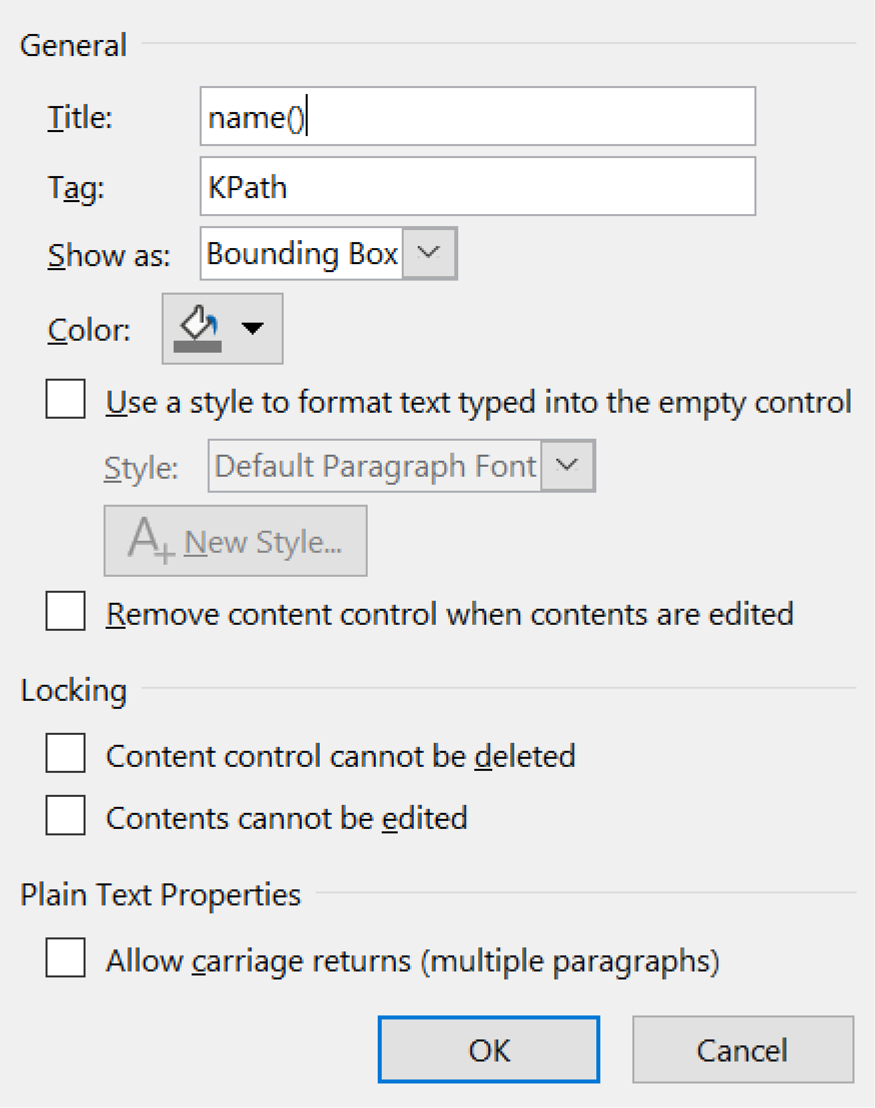
DOCX templates can be created using Microsoft Word 2007 or higher.

They are created the same way as RTF templates are created, with the only difference being that the path/script instructions are saved in text content control elements.

To insert the control elements, it is first necessary to activate the developer tools in Word. To do so, go to the Office menu, open the **Word options** , go to the **Popular commands** category and activate the option **Show Developer tab in the ribbon** . Now go to the **Developer tools** tab and activate **Design mode** .



To add KScript/KPath expressions, insert a **Text-only content control element** . The text of the control element is replaced by the calculated text. Go to the properties of the control element (via the context menu on the closing bracket) and specify the KScript or KPath under **Title** . If you leave the title empty, the text of the control element will be used instead. Enter the script type under **Tag** . The available script types are all the types available in ODT, with the exception of KPathImage **.**



General

Title: name()

Tag: KPath

Show as: Bounding Box

Color:

□ Use a style to format text typed into the empty control

Style: Default Paragraph Font

□ Remove content control when contents are edited

Locking

□ Content control cannot be deleted

□ Contents cannot be edited

Plain Text Properties

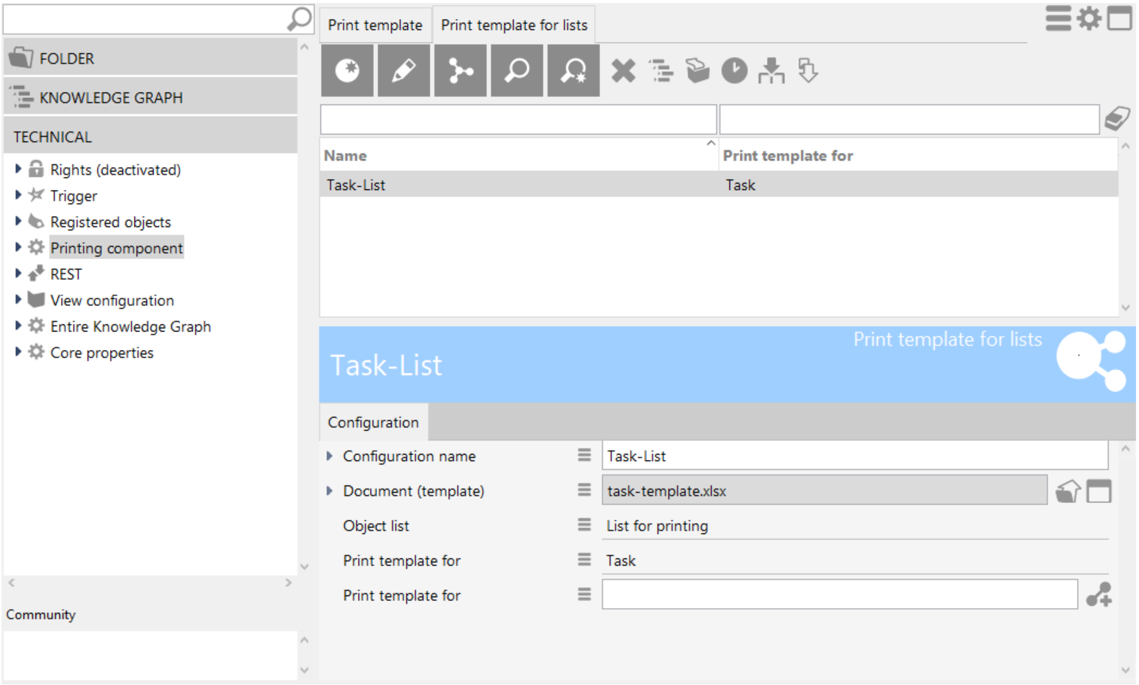
□ Allow carriage returns (multiple paragraphs)

1.6.2. Create print templates for lists

Print templates for lists are saved in the “TECHNOLOGY/Print components” area in the Knowledge Builder. Each “Print template for lists” object contains a print template document (XLSX) and a relation that specifies to which objects the print template is to be applied. Optionally, an object list can be specified that should be used for generating the output. This allows the format of the list that the user sees on the screen, and the format of the list that was output, to be different.

When the attribute “Document (print template)” was not created, then when a document is generated, an Excel file is generated that contains one spreadsheet with the data in the object list and the column headings from the object list configuration, i.e. an Excel file does not necessarily have to be specified as the print template.

The following example shows a print template for lists with objects of the “Task” type.



Name	Print template for
Task-List	Task

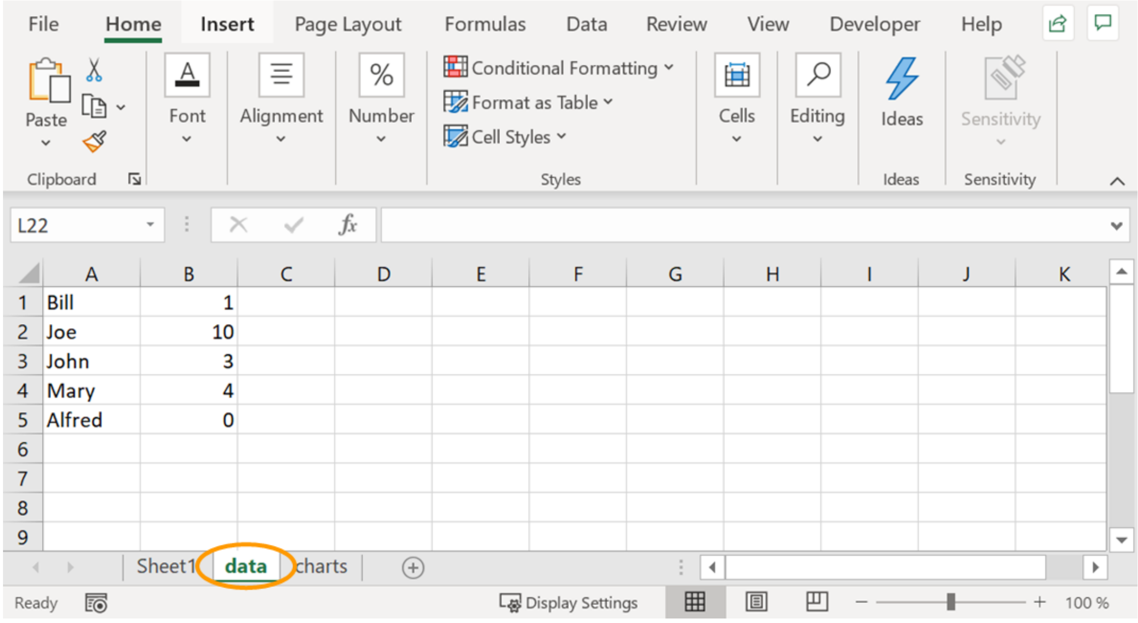
Task-List

Configuration name	Task-List
Document (template)	task-template.xlsx
Object list	List for printing
Print template for	Task
Print template for	

XLSX templates can be created using Microsoft Excel 2007 or higher. These templates only function with object lists.

Creating the Excel file

A standard Excel file is used as a template, and must include an additional spreadsheet called “data”. This spreadsheet is subsequently filled with the object list data, and this without headings and beginning with cell A1.



	A	B
1	Bill	1
2	Joe	10
3	John	3
4	Mary	4
5	Alfred	0

The other spreadsheets can reference data from the “data” sheet in formulas. i-views ensures that all formulas are calculated again as soon as the completed Excel file is next opened using Excel.

1.6.3. Document format conversion with OpenOffice/LibreOffice

The output format of the print operation corresponds to the template used. If you would like to receive a different output format, you have to set up a converter.

To do so, you need an installation of LibreOffice or OpenOffice Version 4.0 or above on the computer that is to perform the conversion. This is usually located in the same place as the bridge or Job-Client that also executes the print operation.

In the configuration file (bridge.ini, jobclient.ini, etc.) you also have to specify the path to the "soffice” program which is part of the LibreOffice/OpenOffice installation and located in the "program" subdirectory there. This must be specified as an absolute path; relative paths (..\LibreOffice\etc.) are not possible here.

```
[file-format-conversion]
sofficePath="C:\Program Files (x86)\LibreOffice 4.0\program\soffice.exe"
```

Conversion service

If you do not want to keep a LibreOffice/OpenOffice installation on all workstations or server installations from which formats are to be converted, an appropriately converted REST bridge can perform the conversion.

To do so, the .ini file of the REST bridge must have the following format:

```
[Default]
host=localhost

[KHTTPRestBridge]
port=3040
volume=cardAdmin
services=jodService

[file-format-conversion]
sofficePath="C:\Program Files (x86)\LibreOffice 4.0\program\soffice.exe"
```

In the Admin tool, you enter the address at which the conversion service can be reached under system configuration/components/conversion service.

Example:

```
http://localhost:3040/jodService/jodconverter/service
```

Document formats

To ensure output formats are available, appropriately configured objects of the “Converter document format” type must be available in the Knowledge Graph.

The important thing is that not all formats can be converted into all formats. The most important ones are:

Name	Extension	Mime type
Portable Document Format	pdf	application/pdf
OpenDocument Text	odt	application/vnd.oasis.opendocument.text
Microsoft Word	doc	application/msword

1.7. Development support

1.7.1. Dev tools

Different tools are available to facilitate development.

- K-Infinity plug-in: Offers support or JetBrains's products This includes the synchronization of source files, KJavascript and KPath support.

1.7.2. Dev service

The Knowledge-Builder provides the option of allowing access from external applications. This allows, for example, synchronization with development environments or specific elements of an application to be opened from the browser.

The Dev service must be started in the Knowledge-Builder for this. To do so, start by opening the *Settings* and in the *Personal* tab, going to *Dev tools* . A port can now be specified here at which the service should be able to be reached. The service can be started and stopped manually using the buttons next to it. If the "Automatic start" checkbox is activated, then the service is automatically started with the Knowledge-Builder.

If the Knowledge-Builder features an ini file (the default name is "kn.ini"), then it can save the settings permanently. The settings can also be entered manually in the ini file:

```
[DevService]
autostart=true
port=3050
```

1.8. KB plugins and components

1.8.1. Units component

The units plugin serves for adequate display of values of scale units - consisting of the numeric value and its appended unit symbol. For different decimal prefixes, multiple entries with relative factors can be defined for one and the same scale unit type. The output of the values and their units takes effect within the Knowledge Builder and the web frontend via the view configuration mapper. For example, an export can use the information to convert the values of an attribute into another unit required by a target system (unit conversion).

Since 5.4, the units engine is an integrated component of the Knowledge Graph. After installation by means of the Admin Tool, the units component is available within the Knowledge Graph in the section "TECHNICAL" > "Scale units". It contains the required object types "Kind of Quantity" and "Scale Unit".

Examples:

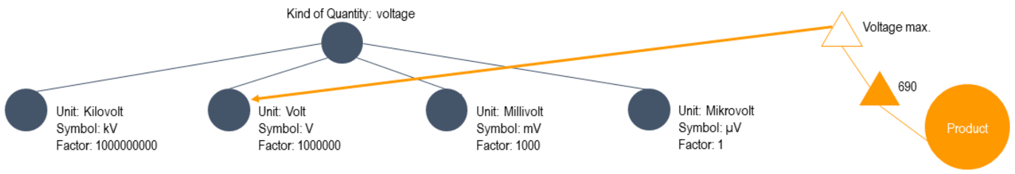
- Kind of Quantity: length, voltage, temperature
- Scale unit: meter, inch, millivolts, Kelvin

A kind of quantity is related to the respective scale unit via the relation "measured in", whereas a scale unit can be related to one kind of quantity only. Via the relation "Base unit of", a scale unit is assigned to an attribute type. Attributes of this attribute type then are entered and displayed in forms of the related scale unit.

The units component "ETIM" comes with standard units of the ETIM classification. These units can be supplemented with customized units.

Configuration:

- " **Unit symbol** ": string which is going to be appended to the value. For a value of "1", the "Unit symbol (singular)" attribute is used instead.
    - **Example:** "2.5 cm", "4 minutes", "1 minute".
- **"Factor":** factor of the value in relation to the base value. The scale unit with factor "1" represents the base value in which the attribute value is going to be stored.
    - **Example:** "Unit (distance)" has the scale unit "mm" with factor "1", the scale unit "cm" has factor "10" and the scale unit "m" has the factor "1000". The 'raw value' stored in a related attribute therefore represents the value in millimeters, "mm".
- **"Fraction":** fraction of the value in relation to the base unit. By using factor and fraction, a higher accuracy is achieved when converting units.



NOTE When displaying attribute values by means of virtual property scripts, value() will return the attribute value itself whereas valueString() will return the value and its unit according to the units plugin.

For more information about the units plugin, please contact empolis intelligent views: support@i-views.com.

1.8.2. Custom components

Custom components are bundles of semantic elements, queries, scripts and other elements. These can be transferred to other Knowledge Graphs. Common usage scenarios:

- Define a component that acts as the base for specialized components
- Develop components and transfer them to integration and production systems

A component is an object that consists of

- a name
- a version
- an URI that is used as a base for RDF-URIs
- a string prefix that is used as a base for configuration names
- optional rules that define which objects are part of the component

To simplify and shorten this chapter everything that can be assigned to a component will be called an element. This includes:

- Object-concepts
- Objects
- Relation-concepts
- Attribute-concepts
- REST elements
- ViewConfig elements
- Data sources
- Mappings of data sources

- Queries
- Topic collections
- Folders
- Scripts
- Triggers
- Accessright-paramerters

WARNING The actual property-objects themselves are not to be assigned to a component. They will be transfered along with their object(s).

NOTE When exporting folders or topic collections, they will only know their elements (including subfolders), but not what folder they are a subfolder of. Therefore, after an import, these folders can’t be found in the folders section but instead under **TECHNICAL** → **Registered objects** → **Folders/Topic collections**.
This also means that only the top folder of a hierarchy needs to be reassigned to the folders section after import since it still knows its subfolders.

If folders or topic collections contain elements which are neither part of the component nor exist in the target graph, they can still be exported but will produce warnings on import since their elements can’t be found.

NOTE When exporting a property that uses an index, the index itself will also be exported. If the target graph does not have an index with the same name and configuration, it will be created. Otherwise, the imported elements will be assigned to the existing one. If the import results in multiple indexes with the same configuration, they can be merged in the KB options via **Index configuration** → **Indexes**.

1.8.2.1. Configuration

Add the software component ***Custom components*** in the Admin Tool. This component adds the required schema. Components can then be managed at: **Technical** → **Custom components**

All defined components are listed here and new components can be created.

NOTE There are additional hidden columns for this table showing the ***Assignment-style*** and the ***Handling of surplus elements***, that can be enabled via the **Choose columns** option in the table settings. The table settings need to be enabled in the KB settings under **Personal** → **Editors** → ***Show table column settings***. The table settings can then be found in the top right of the table after reloading it.

A component is an object that consists of:

Configuration value	Description
Name	The name of the component.
Description	A short text that should describe the use or content of the componet.
Prefix	A short string that can be used to identify elements of this component.
Base URI	An URI that can be used to identify elements of this component. **NOTE** See the chapter Choosing prefix and base URI for more information on valid base URIs.
Select elements based on prefix/URI/Relation	Boolean value. If true, elements of this component are identified by using its base-URI and prefix.
Include dependencies	Boolean value. If true, when exporting this component, all elements, that this component is dependent on, will be exported aswell, regardless of wether they are assigned to this component or not.
Handling of surplus elements	Choose how elements that are part of the component in the target Graph but aren’t part of the exported file will be handled when importing this component into the target Graph. • **Keep:** Keep the surplus elements as part of the component • **Put in the bin:** Remove the surplus elements from the component and save them in the bin section of the cutom components • **Execute script:** Send them to a java-script which has to be specified before the export • **Delete:** Delete the surplus elements from the graph **NOTE** When importing a component the settings and script of the imported component will be used regardless of what options were chosen for the same component in the target-graph.
Script to process surplus elements	The java-script which will be called with the surplus elements on import if the ***Execute script*** option was selected for the handling of surplus elements.
Keep additional translations	Boolean value. If true, don’t overwrite any additional translations that are configured for attributetypes in the target-graph when importing this component.

Configuration value	Description
Read only	Boolean value. If true, it prevents the user from editing anything related to this component except the component object itself. Nothing can be added or removed from the component and its elements cannot be edited. It is however still possible to draw relations to and from elements of the component. It is also still possible, although not recommended, to manually add the base-URI or prefix to an unrelated topic making it a part of the component. This only allows you to add the element in one way though since after that it is *write protected*.
Deactivate 'Read only' after import	Boolean value. If true, the ***Read only*** attribute of this component will be set to false after being imported.
Attribute for identification during transfer	Declares an userdefined attribute to be used as an identifier for elements on transfer. **NOTE** The attribute needs to have an uniqueness index and should be a string-attribute.
Assignment-style	Defines how elements are assigned to this component: • **Relation:** A one-way-relation is drawn from the element to the component. • **Relation (Adjust internal name):** A one-way-relation is drawn from the element to the component. Additionally, the internal name of the element will be adjusted with the prefix where possible. • **Prefix/Base-URI:** The configured prefix and base-URI are used to mark the name, internal name and RDF-URI of the element.
Requires component	Declare other components as necessary for this one to work.
Overwrites component	Boolean value. If true, an element that is part of this component and of one that is necessary for it, it will only belong to this component.
Version	The version of the component consisting of major version, minor version and patch.

There are some more options to select elements to be part of a component. For more information see the chapter Additional selection and configuration of specific elements.

1.8.2.2. A minimal example

Open the Custom components area and create a new component topic. It will ask for a name, which can be freely chosen and changed at any time and another three mandatory values:

- **Prefix** : A string prefix that is used to select objects that have a configuration name starting with this string. It will also be used to suggest a configuration name when creating new elements. Use 'accounting' for example.
- **Base URI** : A URI that will be used as base for creating RDF URIs. It should end with the prefix, e.g. 'http://example.org/accounting'.
- **Handling of surplus elements:** This specifies what happens when a component is imported and the same component is already part of the graph but has elements which are not in the import.

Now that the component is set up you can start assigning elements to it. To do so navigate to any element you want to assign and open its context-menu. There you should see the **Custom components** submenu which contains three ways to assign the element:

1. **Assign element** directly assigns the element to the chosen component.
2. You can also choose **Open** assignment-tool to get an overview of what elements are connected to this one and then assign any element you deem necessary for the component.
3. Additionally, there is the option **Assign to x aswell**, which assigns the element to the last component that an element was assigned to.

When looking at the assigned element you should see its assignment noted on the right side of its banner region as well as the updated RDF-URI and internal name or registry key depending on what kind of element you assigned.

1.8.2.3. Choosing prefix and base URI

Although there are no technical restrictions when specifying a prefix or a base URI, there are a few rules that make it easier to handle custom components:

- Only use alphanumerical characters and periods.
- Do not add a period at the end of the prefix as it is automatically used as a separator.
- Do not add a '#' at the end of the base-URI as it is automatically used as a separator.

WARNING	In version 5.4 the '#' still needs to be manually added to the base URI.

- Do not use generic names that can be mistaken for built-in components, e.g. 'viewconfig' or 'rest'.
- Use a prefix that makes it obvious which component it belongs to.
- Use the prefix as the last part of the base URI, as shown in the minimal example in the preceding chapter.

NOTE	An URI (unique resource identifier) is usually used to precisely locate a resource on the internet and is thus a web address. Since our base-URI is a namespace using that concept, it should also start with **http://** or **https://** followed by a domain representing your company or project. After that should come another '/' and the prefix of the component. This would lead to something like:

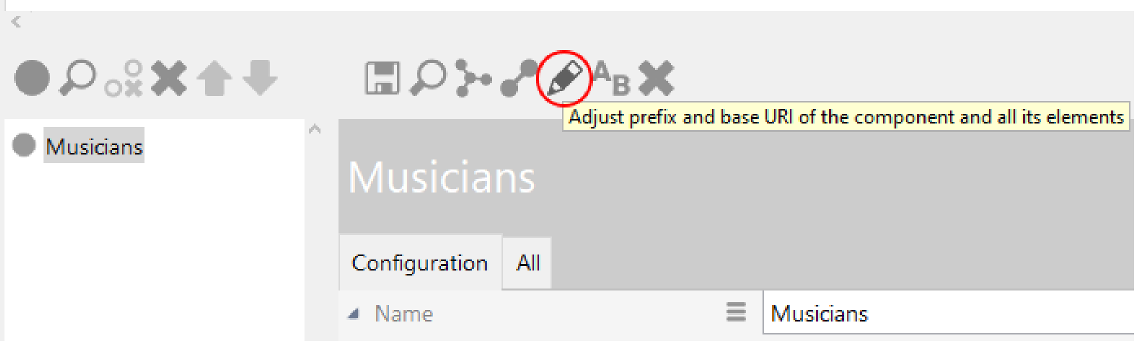
https://example.org/accounting with the project being to provide examples and the component for accounting elements.

Since we are just using the URI to identify elements and their assigned component in the graph, it does not need to actually lead to anything when looked up in a web browser.

1.8.2.4. Changing prefix and base URI

Manually changing the prefix or base URI does not carry these changes to the selected elements thus deselecting them.

You can change the prefix or base URI and update the selected elements with the new values by pressing the edit icon in the detail-view of the component. In the custom components section select the component and in the bottom left the specific object you want to update. Then click the edit icon above the banner section of the component.


Adjust prefix and base URI of the component and all its elements
Musicians
Musicians
Configuration
All
Name
Musicians

In the dialog you can change existing values and new ones but not remove existing ones. Changed or new values will immediately be used to update the elements that are selected by the edited object.

When changing the component-object itself only the elements that are selected exclusively by said object will be affected. The elements that are also selected by sub-objects will not be updated as the sub-objects override the component selection because of their additional options.

WARNING

If the prefix and base URI of the component and its sub-objects are the same all objects will select all the elements and it will be impossible to automatically separate them again.

NOTE

Should a problem occur during the overwriting process, the elements that were already overwritten will keep their new values while the component still has the old ones. While this means that those elements are temporarily not part of the component it also makes it easy to just restart the overwriting after fixing the problem to adjust the rest of the elements.

1.8.2.5. Assignment of Elements

How an element is assigned to a component depends on the assignment-style chosen for the component.

If the chosen style is ***Relation***, semantic elements will be assigned by drawing a one-way-relation from the element to the component.

There is also a variation of this style that will adjust internal names additionally to the relation in order to prevent confusion because of no or wrong prefixes.

If the chosen style is ***Prefix/Base-URI***, the assignment of an element is signaled by some of its identifing attributes:

- RDF-URI starts with the base-URI specified by the component
- Internal name starts with the prefix specified by the component
- Registry key starts with the prefix specified by the component
- Name starts with the prefix specified by the component

The name is only used by custom components for specific elements that follow naming conventions like for example view-config elements.

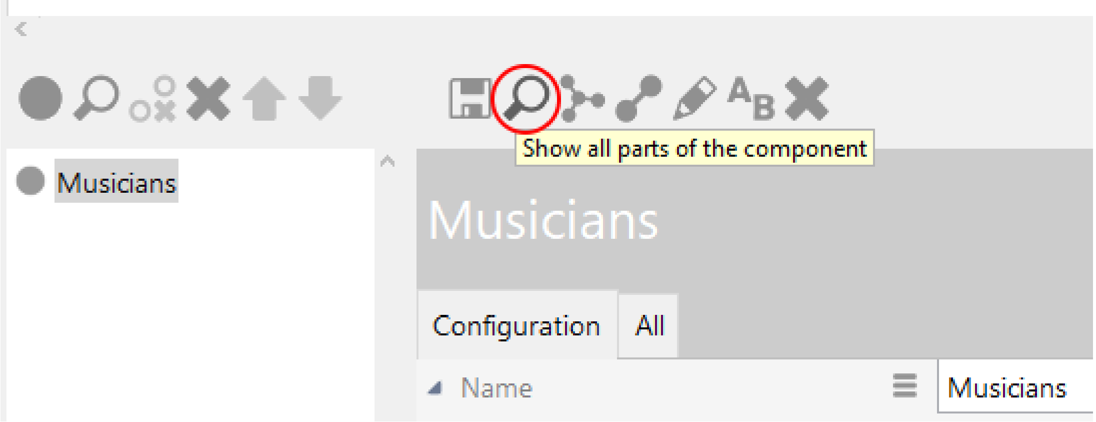
NOTE

Only one of these attributes is needed to recognize the assignment, but a RDF-URI is necessary to export elements. If no RDF-URI is set before export it will be created automatically by combining the base-URI of the component and the name of the element.

Usually these attributes are edited automatically when using the provided tools to assign or unassign elements, but they can also be adjusted manually.

The name of the component that an element belongs to is shown on the right side of its banner region. This can be deactivated in the Knowledge-Builder settings via the menu **Editors**.

It’s also possible to view which elements are part of a component by clicking the right magnifying glass in the detail view of the component.



1.8.2.5.1. Assigning elements

For most elements, the UI displays a list of available components in form of a drop-down field upon creation. When selecting a component, the prefix of that component is used to fill in the name of the new element. The UI also tries to automatically select a component based on the context. For example, when creating a subtype of a type that is part of a component, then that component will be selected.

Under certain circumstances however, some elements are assigned automatically:

- When creating a new mapping using an existing data-source, that is unassigned and unused, assign the data-source to the same component as the mapping.
- When registering the unregistered data-source used by an assigned mapping, assign the data-source to the same component.
- When linking a previously unused and unassigned data-source to an assigned mapping, assign the data-source to the same component.
- When adding an extension to a topic, assign the extension to the same component that the topic is assigned to.

The UI does not force providing an internal name or registry key for new elements. When an element is created without a registry key and a component is selected, the registry key will be created by combining the prefix of the component and the name of the element. Internal names are not created from scratch and only existing ones get adjusted by adding the prefix. This can be adjusted in the settings.

After creation, single and multiple elements can easily be assigned to a component via the context menu options in the **custom components** submenu: **Assign element**, **Assign to x as well**, with x being the last assigned component, or via the assignment-tool.

NOTE The exact behavior when assigning elements can be adjusted as described in the chapter System settings.

The context menu additionally allows you to open the current component of single elements, to avoid having to navigate to the custom components section, and to open the assignment-tool which is also used to assign elements but provides a lot more options and insights than the regular assign dialog.

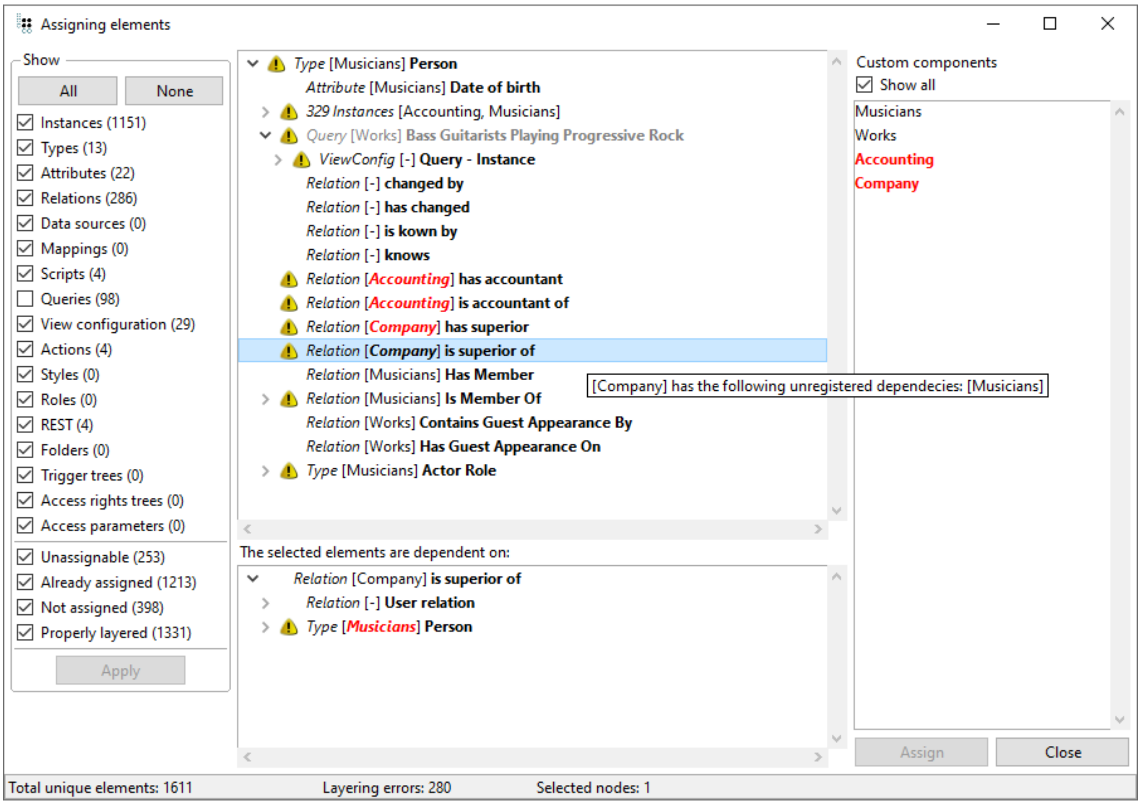
> **NOTE**
>
> When assigning a relation concept to a component both of its directions will be assigned. The custom component logic always treats the two halfs of a relation concept as one element.
> When assigning an element whose RDF-URI does not match the base URI of the component, it will get an RDF-URI-Alias using the base URI.

Additionally, there are two options which are only available for ViewConfig elements that can be used to replace an element of the component with another external element and prevent future updates from overwriting the replacement.

> **WARNING**
>
> This feature should only be used as a last resort, as it is potentially very prone to errors because of the many possible interactions with replaced elements.

1.8.2.5.2. The assignment-tool

The assignment-tool is used to get a good overview of how the elements you want to assign are connected and what layering issues they have.



Assigning elements

Show

All | None

☑ Instances (1151)
☑ Types (13)
☑ Attributes (22)
☑ Relations (286)
☑ Data sources (0)
☑ Mappings (0)
☑ Scripts (4)
☐ Queries (98)
☑ View configuration (29)
☑ Actions (4)
☑ Styles (0)
☑ Roles (0)
☑ REST (4)
☑ Folders (0)
☑ Trigger trees (0)
☑ Access rights trees (0)
☑ Access parameters (0)

☑ Unassignable (253)
☑ Already assigned (1213)
☑ Not assigned (398)
☑ Properly layered (1331)

Apply

Type [Musicians] **Person**
Attribute [Musicians] **Date of birth**
329 Instances [Accounting, Musicians]
Query [Works] **Bass Guitarists Playing Progressive Rock**
ViewConfig [-] **Query - Instance**
Relation [-] **changed by**
Relation [-] **has changed**
Relation [-] **is kown by**
Relation [-] **knows**
Relation [***Accounting***] **has accountant**
Relation [***Accounting***] **is accountant of**
Relation [***Company***] **has superior**
Relation [***Company***] **is superior of**
Relation [Musicians] **Has Member**
Relation [Musicians] **Is Member Of**
Relation [Works] **Contains Guest Appearance By**
Relation [Works] **Has Guest Appearance On**
Type [Musicians] **Actor Role**

[Company] has the following unregistered dependecies: [Musicians]

The selected elements are dependent on:

Relation [Company] **is superior of**
Relation [-] **User relation**
Type [***Musicians***] **Person**

Custom components
☑ Show all

Musicians
Works
Accounting
Company

Assign | Close

Total unique elements: 1611 Layering errors: 280 Selected nodes: 1

The left side provides filters for the top tree view to remove specific types of elements. The second section lets you filter elements in regard to their assignment status. Every filter shows how many unique elements of that category are in the top tree view. When toggling off a filter the corresponding elements will be hidden, except if there are other elements below them that shouldn’t be hidden. In that case the node will just be greyed out as seen in the picture above. In the upper section of the filters, which is about the types of elements, there are buttons to toggle every filter of that section on or off. If the main view contains a lot of elements, the filters will not be applied autoamtically. Instead, there will be an ***Apply*** button that applies all changed filters at once, to reduce waiting times when wanting to toggle multiple filters.

The top middle has a tree view of the element(s) that the tool was opened on with all the elements that somehow depend on that element. This does not apply to ViewConfig-elements as it makes more sense also show things like used scripts instaed of just dependents.

Each node shows what type of element it represents, what custom component that element is assigned to and the name of the element. If any element produces layering issues due to its assigned component or because it is assigned to multiple components at once, the component(s) will be highlighted in bold, red letters and a warning symbol is displayed in front of the node. When hovering a node with a layering issue a tooltip will appear describing the cause of the layering issue. Elements with sub-nodes that have layering issues will also have the same icon to indicate a problem further into the tree.

NOTE These tooltips are only meant to explain the current situation that causes the layering issue, not as a prompt to add nonsensical dependencies between components to remove these layering issues.

For better visibility instances are bundled under a special node that shows the number of instances below it and up to 5 components that some of the different instances are assigned to. If the setting the use instances is disabled these nodes will not exist as no instances that are only there through their concepts will be displyed.

The bottom tree view contains all elements that the currently selected element(s) from the top tree view are dependent on. If a selected element from above has a layering issue the cause will be highlighted here.

Example: In the picture above you can see that the selected relation 'is superior of' has a layering issues because it belongs to the 'Company' component, but is dependent on the type 'Person' which is assigned to the 'Musicians' component, as you can quickly see in the list of dependencies. This means that the 'Company' component is dependent on the 'Musicians' component which should not be the case which is why it is not configured and thus causes a layering issue. As suggested on the right side, the only components that this relation can currently be assigned to without issues are the Musicians and Works components.

On the right side is the custom component selection. By default, it shows all components that the currently seleted element(s) in the top tree view can be assigned to without causing layering issues. To see all components ***Show all*** can be selected. The additional components that will then be displayed are highlighted in red. Doubleclicking a component open a new window to edit the component.

At the bottom you can see how many unique elements the top tree view contains, how many layering issues these have and the number of currently selected elements in the top tree view. Unique elements means, that if the top tree view shows two types that can both have the same relation, that relation will be shown below both types but will only count as one unique element.

The context menu of elements in the tree-views have the options to open a new assignment-tool window with the selected elements as the roots and to edit a single selected element by spawning a new window of that element. Additionally, there is an option to expand and select all sub-nodes of the selected nodes.

NOTE

In case of looping dependencies all sub-elements will be selected exactly once even if not all nodes can be expanded. So after using this option on the root node(s) there will always be one selected node for every unique element.

Once at least one element of the top tree view and a component are selected the ***Assign*** button in the bottom right can be pressed which opens a list of all elements that will be assigned to confirm the assignment. After confimation the assigning process starts and in the end all elements that couldn’t be assigned along with the reason for the failure will be displayed.

In both lists you can double-click an Element to open and edit it.

NOTE

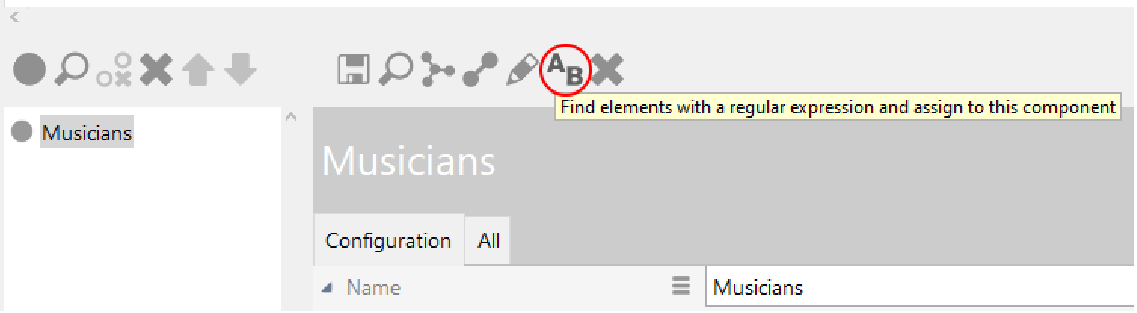
Having an assignment-tool window open while assigning an element will automatically update that element if present. This does not work if the element is assigned by manually editing its attributes.

When changing components while having an assignment-tool open, you can press F5 to refresh the tool. This will update the list of components, their dependencies and all layering errors. Holding down the 'Ctrl' key while pressing F5, will also reload the assignments of all elements in the tool.

If you just want a quick overview for the direct dependencies of an element, refer to Showing dependencies in the graph editor.

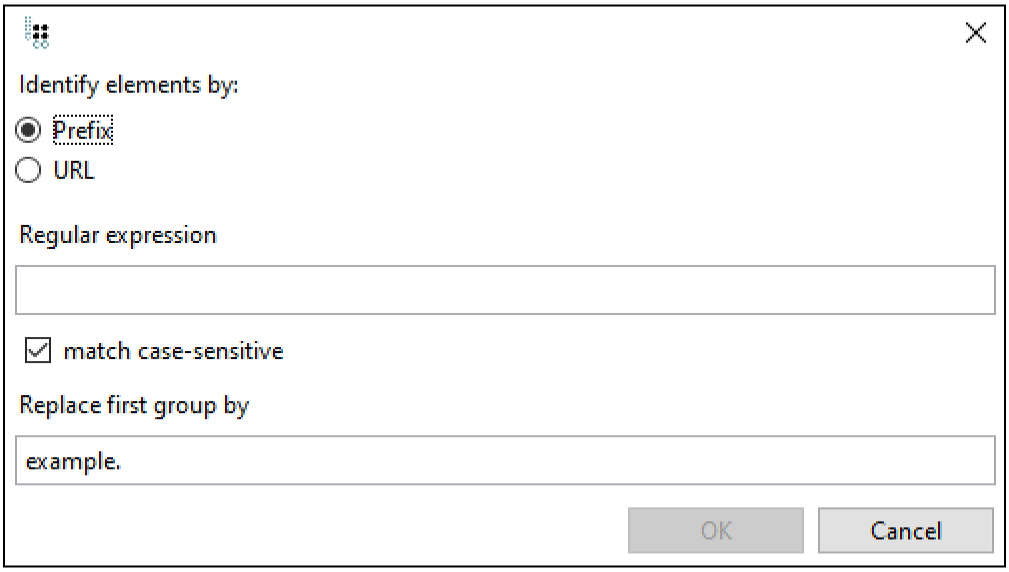
1.8.2.5.3. Assign with regex search

Another way to assign elements is to define a regular expression and assign all matching elements to the selected component. This feature can be found under **TECHNICAL** → **Custom components** by navigating to the desired component.


Musicians
Find elements with a regular expression and assign to this component
Musicians
Configuration
All
Name
Musicians

The first thing to choose here is wether to check elements by all attributes that are used for the prefix or by their RDF-URI. Next you define the regular expression used to search for elements to assign. This regular expression needs to contain at least one group part which will then be replaced by what is written in the second textfield. By default, this is the prefix of the component with a dot.

NOTE If the group defined in the expression is not at the beginning of the found value it will not be assigned to the component as the prefix/base-URI will not be at the start of the value either.


Identify elements by:

● Prefix

○ URL

Regular expression

☑ match case-sensitive

Replace first group by

example.

Example: Using the regular expression ^(oldPrefix\.).* will match everything that starts with 'oldPrefix.'.

1.8.2.5.4. Component suggestions

The list of available components only shows components that won’t cause layering issues for the element and thus follows some rules to ensure that:

- Write-protected components are never suggested.
- Only components that are dependent on all components of the most direct elements needed for this element to work (e.g. the type of an object; a relation used by a query) that have a

component will be suggested.

- Only components that are depended on by all components of the most direct elements that need this element to work (e.g. an object of a type; a query using a relation) that have a component will be suggested.
- If no restrictions are found, all non-write-protected components are suggested.

1.8.2.5.5. Unassigning elements

To unassign an element from its current component you can open the context-menu of the element and select **custom components → Remove element**.

Unassigning an element will delete its RDF URI, deregister the registry key and remove the prefix from the normal and internal Name. Additionally, the element will be added to a bin folder that is located under **TECHNICAL → Custom Components → Removed elements**.

NOTE	When un- or re-assigning an element, it keeps its name and its internal name without the prefix or with a new prefix. If an element is renamed and the internal name and URI need to use the new name as well, they need to be either adjusted manually or deleted before reassigning the element, otherwise they still use the old name.

1.8.2.5.6. The bin folder

The bin-folder is located under **TECHNICAL → Custom components → Removed elements**. It has two subfolders for semantic elements and registered objects.

Elements are added to their respective bin-folder in these two situations:

1. They get unassigned from their component
2. They are surplus elements after an import and the imported component has the surplus handling option set to ***Put in the bin***

There are three ways for elements to be removed from their bin-folder:

1. The element gets assigned to a component
2. The element gets restored via right-clicking it or its bin-folder and selecting the restore option
3. The element gets removed via right-clicking it or its bin-folder and selecting a removal option that applies to it

Restoring an element will set its internal name and RDF-URI or its registry key to the value they had when the element was put in its bin folder.

It is possible to see the references to the element before it was removed from its component via its context-menu.

NOTE	If the component that the element was unassigned from has changed its prefix or

base-URI before the element is restored, the changes will not apply to the restored element which will still have the exact values from when it was unassigned.

WARNING

Danger of loss of data!
Unlike schema elements such as object types or property types, registered objects will be deregistered on removing their assignment. Deregistration leads to deletion if the object is not located in at least one folder.

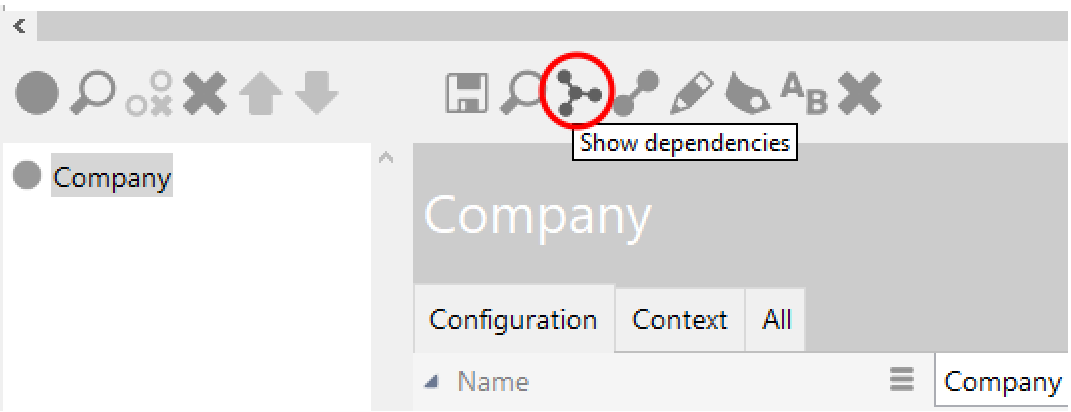
Thus, removing registered objects from the bin folder will most often delete them.

Accidentally deleted elements cannot be restored. In that case the elements must be recreated.
References to the registry key of deleted objects will not work anymore and must be fixed manually.

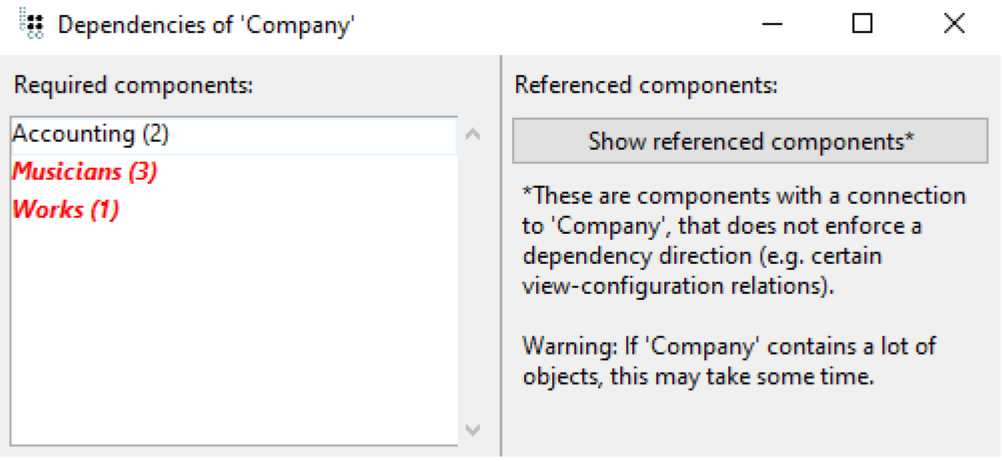
→ To prevent accidental deletion, make sure that such objects are registered or that they at least are located in some different folder.

1.8.2.5.7. Finding layering issues

Apart from looking at the assignment-tool there is another way to quickly find all layering issues of a component.



After choosing a component under **TECHNICAL → Custom components** and clicking the **Show dependencies** button on the left, you get a list of all components that the analyzed one is dependent on.



Dependencies of 'Company'

Required components:

Accounting (2)
Musicians (3)
Works (1)

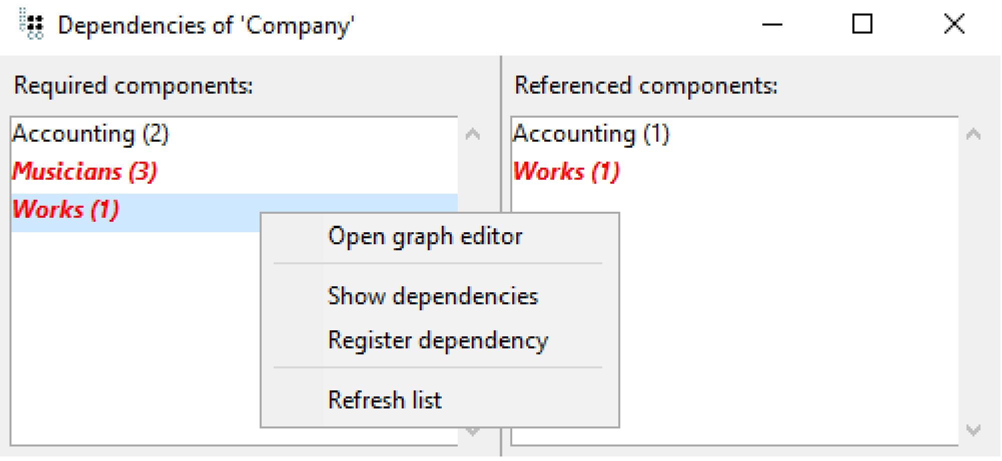
Referenced components:

*These are components with a connection to 'Company', that does not enforce a dependency direction (e.g. certain view-configuration relations).

Warning: If 'Company' contains a lot of objects, this may take some time.

Every component shows the number of dependencies to itself in brackets.
Highlighted components are not declared as a dependency, but still have elements that the analyzed component needs, thus causing a layering issue.

When clicking **Show referenced components*** on the right, another list will be shown. This list contains components with a connection to the analyzed one, that does not enforce a specific dependency direction.



Dependencies of 'Company'

Required components:

Accounting (2)
Musicians (3)
Works (1)

Referenced components:

Accounting (1)
Works (1)

NOTE If the component to analyze contains a lot of objects, the analysis of the right list may take some time.

Both lists have the same context-menu with the following options:

Open graph editor

Opens the graph editor on all elements connecting the selected component with the analyzed one. This also happens when double-clicking a component. This will not close the window.

Show dependencies

Opens this window for the selected component.

Register dependency

Draws the *Requires component* relation from the analyzed component to the selected one.

Refresh list

Refreshes the list by analyzing the component again. This only analyzes what is necessary for the selected list. Presing F5 will refresh both lists, unless the right one is not shown, in which case only the left one is refreshed.

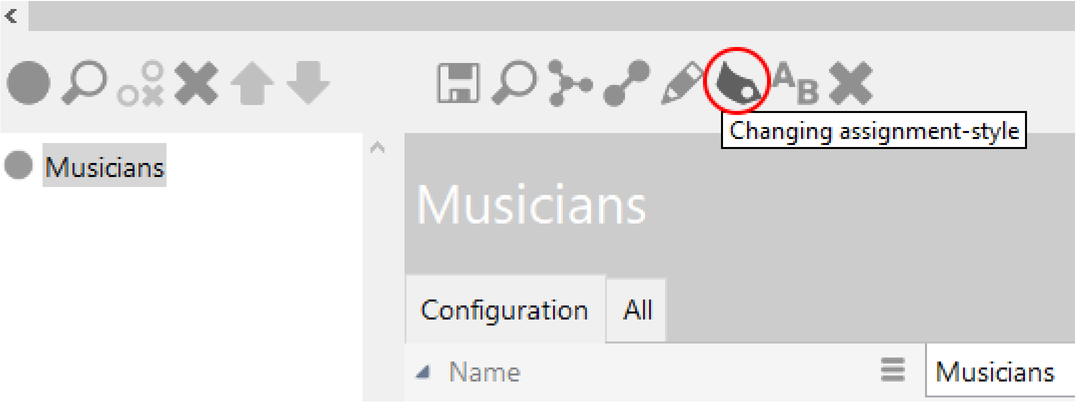
NOTE Refreshing a list takes as long as it did when opening them the first time.

Pressing the F5 key will reload all shown lists.

1.8.2.5.8. Changing the assignment-style

There are two ways to change the assignment-style of a component:

1. You can just directly change the configuration of the component. From that point on all elements will use the chosen style for assignment. Previously assigned elements however are not changed by this.
2. You can use the tool acccessed by clicking the **Changing assignment-style** button at the component, as shown below. In the dialog that opens, you can choose which assignment-style the component should use from now on. This method will also adjust all assigned elements of the component to use the new assignment style. Additionally, the dialog offers the option to remove all previously used means of assignment (the assignment-relation and RDF-URI-aliases).



1.8.2.6. Showing dependencies in the graph editor

Using the context-menu of an element under **custom components** → ***Show dependencies*** you can open the graph editor with the selected element and all of its direct depedencies. This graph editor will use a special configuration for custom components.

Nodes will not be connected via relations, but instead by dependencies. The type of connection depends on the direction of the dependency.

Furthermore, every node has a button at its top left, which adds its direct dependencies.

The legend is changed as well and does not show the concepts of elements, but which component they are assigned to. Component objects themselves have a separate category to avoid confusion. Every node has the same color as its category. These colors will be assigned automatically, but can be defined individually by setting the color attribute of a component. The colors of the categories for unassigned elements and component objects themselves can be configured in the Personal settings. There you can also configure, wether nodes should use the icons defined for their assigned component, where applicable.

New nodes can be added to the graph editor as per usual, even registered objects.
You can also create bookmarks, which save the registered objects. Looking at these bookmarks in the KB will not show the registered objects, though.

NOTE	Graph editors using this configuration cannot be opened for cooperative use.

1.8.2.7. Access Rights and Trigger Definitions

Component-specific rules for access rights or triggers can be defined directly for the respective component. The two attributes ***Access rights*** and ***Trigger*** at the component definition object are used for this purpose.

Rules defined here are automatically inserted into the rights or trigger decision tree. Custom component-specific rules are inserted after the system component-specific rules and before the general user-defined rules. Within the custom component-specific rules, the ordering is determined by the component dependency.

NOTE	To assign existing rights and trigger definitions to a component, you can simply use drag & drop to move them under a new Konfiguration, that is part of a component.

WARNING	Make sure that all registered objects referenced by component-specific rules are either assigned directly to the same component or to one of the components in the dependency chain.

1.8.2.8. Additional selection and configuration of specific elements

By default, the prefix and base URI define which objects are selected when determining the contents of the component. This default can be turned off by deselecting the option ***Select elements based on prefix/URI***.

NOTE	Using an **assign** function after disabling the selection via prefix and URI will still add the prefix and base URI to the element but without recognizing it as part of the component.

Even after disabling that it is still possible to add elements of the component and even give them special configurations. This can be achieved by using the left part of the detail-view of a component to add the following sub-objects:

- Selection of semantic elements
- Selection of registered objects

These objects have options to select and configure elements separately from the component. The configurations of these objects only apply to elements that are selected by the respective objects.

Apart from those configurations there are no differences between elements selected by the component itself or by these sub-objects. They are all part of the component and are presented and exported as one set.

If an element is assigned via the option in its context menu, the prefix and base URI of the component itself and not the ones configured in these sub-objects are used.

1.8.2.8.1. Selection of semantic elements

This sub-object can select elements using a prefix, base URI or query to be part of the component. Using a query will select the elements without changing them (e.g. no specific URI or internal Name needed).

Configuration value	Description
Include dependencies	Boolean value. If true, required elements of selected elements are selected as well. This is limited to certain built-in dependencies, e.g. selecting a ViewConfig table also selects its table columns.
Select all instances of component types	Boolean value. If true, all elements of selected types are selected, too.
Prefix	Optional prefix; used to select elements for this object. If not set, the prefix of the component itself will be used.
Select by internal name	Boolean value. If true, elements are selected if their internal name match the prefix. Overrides the choice made in the component object itself.
Base URI	Optional URI used to select elements for this object. If not set, the base URI of the component itself will be used.
Select by RDF URI	Boolean value. If true, elements are selected if their RDF-URI match the base URI. Overrides the choice made in the component object itself.
Query for semantic elements	A query that defines which additional semantic elements should be selected.

If the selection via internal name or RDF-URI is enabled without specifying a prefix or base-URI respectively, they will use the prefix and base-URI provided by the component-object itself. Doing so means that the additional options from these sub-objects will apply to all elements selected by the component-object itself since they will also be selected by this sub-object.

1.8.2.8.2. Selection of registered objects

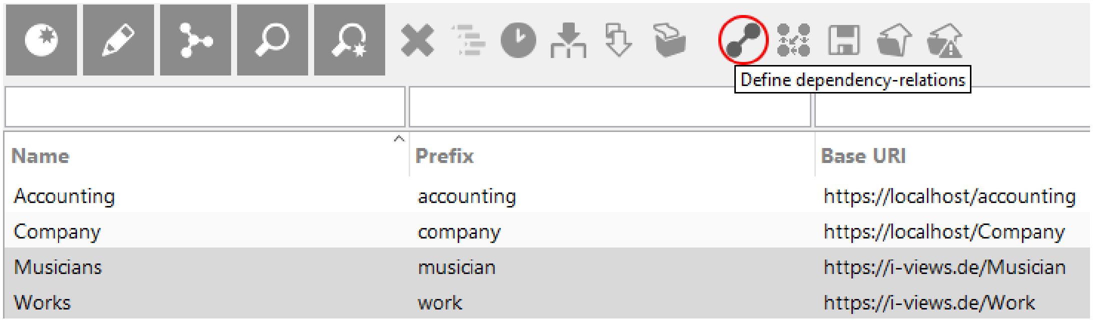
This sub-object can select elements using a prefix and restrict in which registries will be searched.

Configuration value	Description
Include dependencies	Boolean value. If true, elements that are required by selected elements are selected as well. **Examples:** Scripts referencing queries; queries containing query macros
Prefix	Optional prefix used to select elements for this object. If not set, the prefix of the component itself will be used.
Covers registry	Selects which registry types these configurations apply for. If none are selected, they apply for all registry elements that are part of the component.

If no prefix is specified, then the prefix of the component-object itself will be used. Doing so means that the additional options from these sub-objects will apply to all elements selected by the component-object itself since they will also be selected by the sub-object.

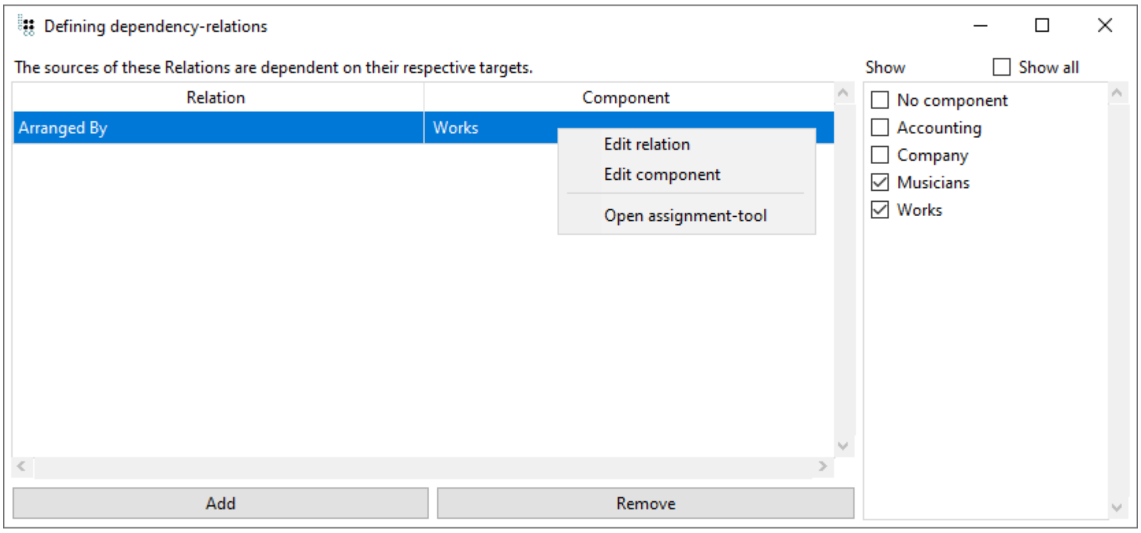
1.8.2.9. Adding custom dependencies of elements

Using the dependency-definement-tool you can designate any relation to be a dependency-relation. These relations will tell the custom components that their sources are dependent on their respective targets and will be used to, for example, build the tree of the assignment-tool or find possible components for elements.



Name	Prefix	Base URI
Accounting	accounting	https://localhost/accounting
Company	company	https://localhost/Company
Musicians	musician	https://i-views.de/Musician
Works	work	https://i-views.de/Work

The tool can be opened by clicking the relation icon of a component under **TECHNICAL** → **Custom components**.



The sources of these Relations are dependent on their respective targets.

Relation	Component
Arranged By	Works

Show □ Show all

□ No component
□ Accounting
□ Company
☑ Musicians
☑ Works

On the left is a table of all relations that have been designated as a dependecy-relation along with the component they are assigned to. The table is sorted alphabetically by components and then the name of the relations.

If the table contains a relation and its inverse relation their names will be highlighted in bold, red letters. Additionally, if a relation is assigned to more than one component, those components will also be highlighted in bold, red letters.

The context-menu of table-rows allows you to open new windows to edit the relation or the component as well as open an assignment-tool on the relation. Double-clicking will also open a new window to edit the relation.

To the right are filters for every component in the graph as well as one for relations that are not yet assigned. There is also a checkbox at the top to show every relation regardless of the filters. At the start the only active filter will be the one of the component that was selected when opening the tool.

At the bottom are two buttons to add new relations as dependency-relations or remove some of the current ones. When adding a new relation the corresponding filter will automatically be activated.

1.8.2.10. Settings

To get to the custom component settings click on the cog icon in the top right of the KB.

There are two types of settings:

- Settings regarding the appearance of custom components in the **Personal** tab
- Settings regarding assignment of custom components in the **System** tab

1.8.2.10.1. Personal settings

These settings are used to customize how custom components are shown to a user. They are tied to that user and have no influence on other users.

Show assignment in banner-region

If true, each element will have its assigned component shown on the right side of its banner region.

Show column with assignment in tables

If true, most default tables in the KB will have an additional column containing the components assigned to the elements in the table.

NOTE	Regardless of the enablement of this setting, the custom component column can still be configured for individual tables under **Personal** → **Editors** by enabling the ***Show table column settings*** option and then clicking the menu to the top right of the chosen table. There you can choose which columns should be displayed. These settings override the settings to display the custom component column.

Use instances for component calculation

If true, every time the recommended components of an element are computed concepts of instances will take their instances into consideration. Additionally, instances will be shown the assigment-tool.

WARNING	This is disabled by default because having to go through a lot of instances can have a huge impact on the performance, especially when opening the assignment-tool, and usually does not change the result much.

Show dialog to override handling of surplus elements

If true, before every import a dialog will ask if you want to handle surplus elements differently than configured at the component that is about to be imported. This override only applies to this specific import and is not saved.

Settings for the component view of the graph editor

These settings apply to the custom component configuration of the graph editor:

Color of unassigned elements

Here you can define the color used for the legend category of elements without assignment.

Color of custom components

Here you can cefine the color used for the legend category of custom component objects themselves.

Use the icon of the component for its elements

If true, the icons of components will be used for its assigned nodes where applicable.

1.8.2.10.2. System settings

These settings are used to customize the behavior of custom components when assigning elements. These changes are system-wide and thus effect other users.

Add the prefix to configuration-names on assignment

If true, configuration-names of view-configuration elements will have the prefix added when assigned to a component.

Create missing configuration-names on assignment

If true, the assigned component will try to create a configuration name for its newly assigned view-configuration element if it didn’t have one before.

Create mising internal names on assignment

If true, the assigned component will try to create an internal name for its newly assigned topic if it didn’t have one before.

When assigning a mapping also assign the used data-source

If true, data-sources will also be assigned when the mapping that uses them is assigned to a component.

Only when the data-source is unassigned

This is a restriction for the prior setting. If true, data-sources will only be assigned with the mapping if they are not already assigned to another component.

When assigning a data-source use the registry key of the sole mapping using it as fallback

If true, a data-source that only has one mapping using it and no way to create a registry key other than its private id, will copy the registry key of said mapping.

When assigning an object also assign all of its extensions

If true, assigning an object will automatically assign all of its extension-objects to the same component.

Not if the extensions are assigned to a different component than their core-object

This is a restriction for the prior setting. If true, the extension-objects will only be assigned with the core-object if they are not already assigned to another component.

1.8.2.11. Transfer

When transfering components from one graph to another, all instances of types that exist and can be identified in both graphs will be synced as follows:

- Attributes will always be overwritten by the import.
- For relations the behavior depends on the component of the relation target.

Component of relation target	Overwritten on import
The imported component	Yes
A component, that the imported one is dependent on	Yes
A component, that is dependent on the imported one	No
A component, that is independent of the imported one	No
No component	No

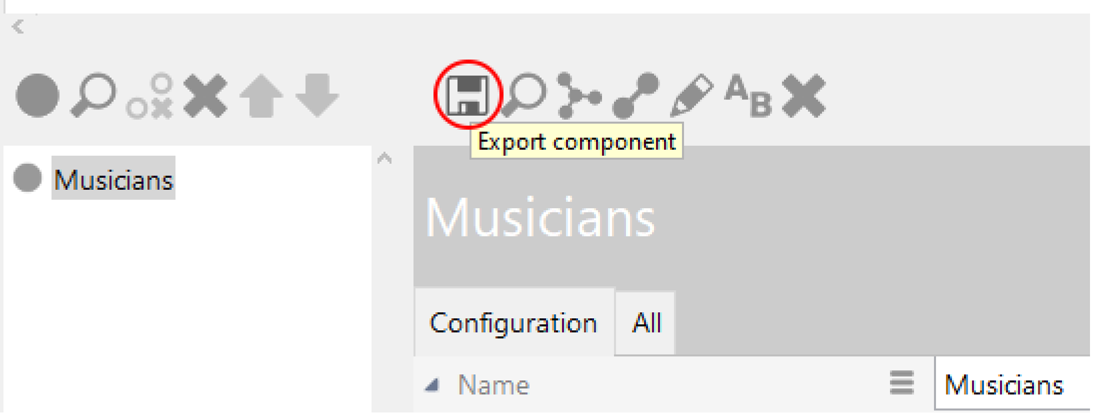
NOTE

If a concept of the component is a subconcept, its superconcept needs to be either in the same component or in a component that is configured as a dependency of the compenent of the subconcept. Otherwise, the subconcept will turn into an independent concept after importing the component. The only exception to this is if the superconcept is part of a software component, as it is currently impossible to configure the dependency of a custom component of a software component.

Components can also be exported and imported via the batch tool as described in the chapter Commandline commands.

1.8.2.11.1. Export

A properly configured component can be exported at any time by opening the component object in the Knowledge-Builder and pressing the button **Export component** in the detail editor.



By navigating to **Technical** → **Custom components** you can also use the **Export component** button atop the table. This will export all selected compnents into one file.

NOTE

When exporting a component all semantic elements will be given a RDF-URI if they don’t have one already. If the component has an ID-attribute defined, it also tries to generate missing IDs for all semantic elements.

When clicking export you can choose between to options:

1. **All:** Export the whole component includeing all assigned elements

2. **Definition:** Export only the component itself and its sub-objects without any assigned elements

> **NOTE**
>
> When choosing to only update the definition of a component, the ***Handling of surplus elements*** attribute will have no effect, as it would apply to all elements of the component.

Before a component can be exported some conditions need to be fulfilled:

- The component needs a name.
- The component needs a prefix.
- The component needs a proper base-URI.
- The graph needs a proper base-URL (Can be set in the KB options under **System** → **RDF**).
- The component needs to know how to handle surplus elements.
- The component cannot be part of a dependency loop.
- If the component has an ID-attribute defined, it needs to have a uniquness index, an internal name and a RDF-URI.

Additionally there are some conditions which are not mandatory but very helpful:

- If the component has an ID-attribute defined, it should be a string attribute, as otherwise missing IDs will not be generated on export.
- If the component has an ID-attribute defined, the attribute should be assigned to the component.

> **NOTE**
>
> Using the ***Select with dependencies*** button, above the table of all components, you can see all configured dependencies of the currently selected components.

1.8.2.11.2. Import

An exported component can be imported via the Admin Tool or via the Knowledge Builder.

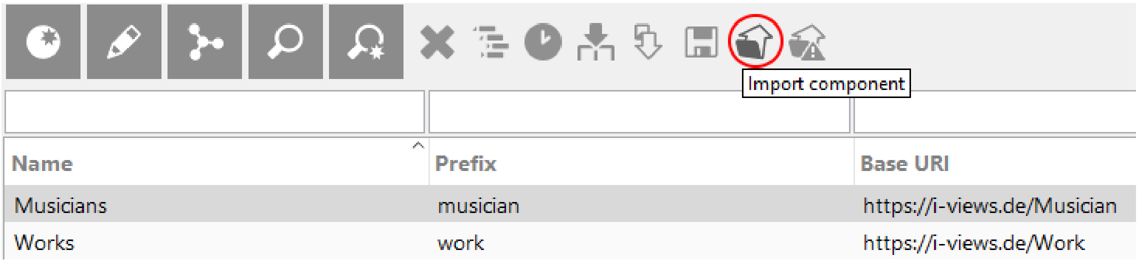
Import via Admin Tool:

1. Open **System configuration** → **Components**.
2. Press **Add model component** and choose ***Import custom component***.
3. Select the exported file or specify a file URL.
4. Choose wether to use or override the *handling of surplus elements* configuration of the component for this import.
5. **Result:** The imported component appears in the list of components. Note that the import also adds the required software component ***Custom components*** in case it hasn't already been added.

The component object can be opened in the Knowledge-Builder under **Technical** → **Custom components**

Import via Knowledge Builder:

NOTE This requires the software component ***Custom components*** to be added to the system beforehand.



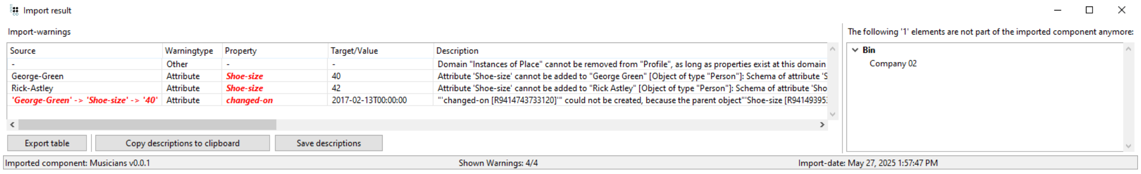
Name	Prefix	Base URI
Musicians	musician	https://i-views.de/Musician
Works	work	https://i-views.de/Work

1. Navigate to **TECHNICAL** → **Custom components**.
2. Press the **Import component** button which is the rightmost icon at the top.
3. Select the exported file directly or specify a file URL.
4. Choose wether to use or override the handling of surplus elements of the component for this import.

Warnings after the import:

If the import had no problems and there were no surplus elements, a **Close** button will appear below the loading bars. It will complete the import.
Otherwise, a new window will open. On its left side will be a table containing all warnings that occured during the import. The right side will show all surplus elements and how they were handled.
If one of these lists is empty, it will not be shown.

These warnings can include things like syncing indexes or that some domains could not be removed because they are still in use in the target-graph. Mostly it will be warnings about Elements not being found though. These can usually be fixed by looking up the missing elements in the source graph and assigning them to the desired component.



Source	Warningtype	Property	Target/Value	Description
-	Other	-	-	Domain "Instances of Place" cannot be removed from "Profile", as long as properties exist at this domain
George-Green	Attribute	*Shoe-size*	40	Attribute 'Shoe-size' cannot be added to "George Green" [Object of type "Person"]: Schema of attribute 'S
Rick-Astley	Attribute	*Shoe-size*	42	Attribute 'Shoe-size' cannot be added to "Rick Astley" [Object of type "Person"]: Schema of attribute 'Sho
'George-Green' -> 'Shoe-size' -> '40'	Attribute	*changed-on*	2017-02-13T00:00:00	"'changed-on [R9414743733120]'" could not be created, because the parent object"'Shoe-size [R94149395

The following '1' elements are not part of the imported component anymore:

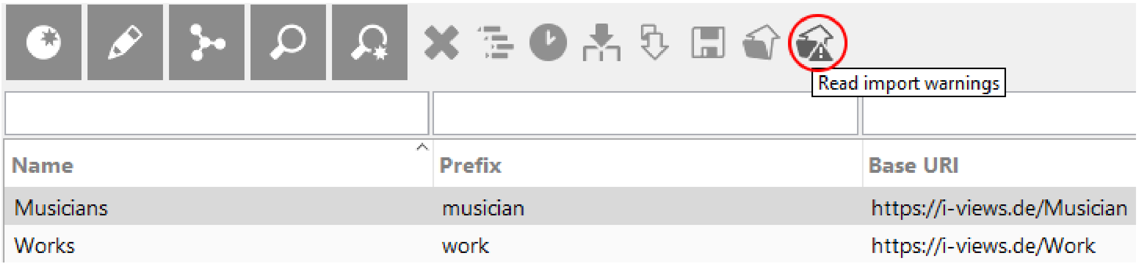
- **Bin**
  - Company 02

Imported component: Musicians v0.0.1 Shown Warnings: 4/4 Import-date: May 27, 2025 1:57:47 PM

In this table all elements are displayed using their RDF-name as they might not exist in the target graph and just use the import identifiers. The highlighted elements could not be found in the target graph.

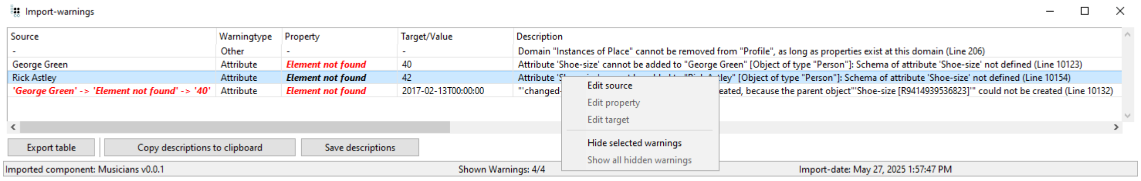
To help fix these problems, the table of warnings can be exported via the button in the bottom left.

It can then be read from the source graph to show the same table.



Name	Prefix	Base URI
Musicians	musician	https://i-views.de/Musician
Works	work	https://i-views.de/Work

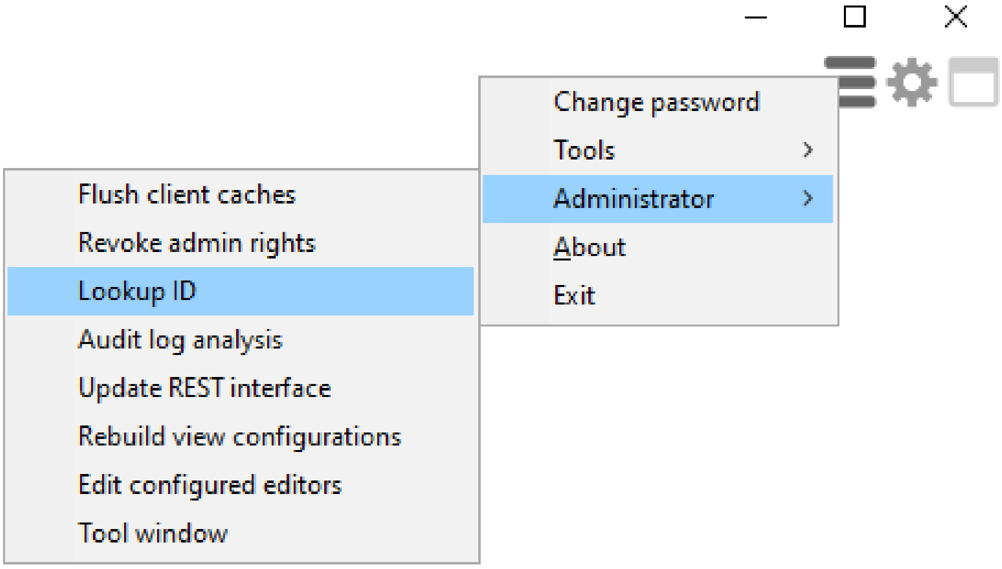
From there you can fix the problems directly at the source by editing the elements via right-clicking them. In this table the elements will also be displayed by their actual names unless they were deleted before opening the table, in which case it will say **Element not found**. You can also hide rows that are already fixed using their context-menu.



Source	Warningtype	Property	Target/Value	Description
-	Other	-	-	Domain "Instances of Place" cannot be removed from "Profile", as long as properties exist at this domain (Line 206)
George Green	Attribute	*Element not found*	40	Attribute 'Shoe-size' cannot be added to "George Green" [Object of type "Person"]: Schema of attribute 'Shoe-size' not defined (Line 10123)
Rick Astley	Attribute	***Element not found***	42	Attribute 'S ley" [Object of type "Person"]: Schema of attribute 'Shoe-size' not defined (Line 10154)
'George Green' -> 'Element not found' -> '40'	Attribute	*Element not found*	2017-02-13T00:00:00	"'changed- eated, because the parent object"'Shoe-size [R9414939536823]'" could not be created (Line 10132)

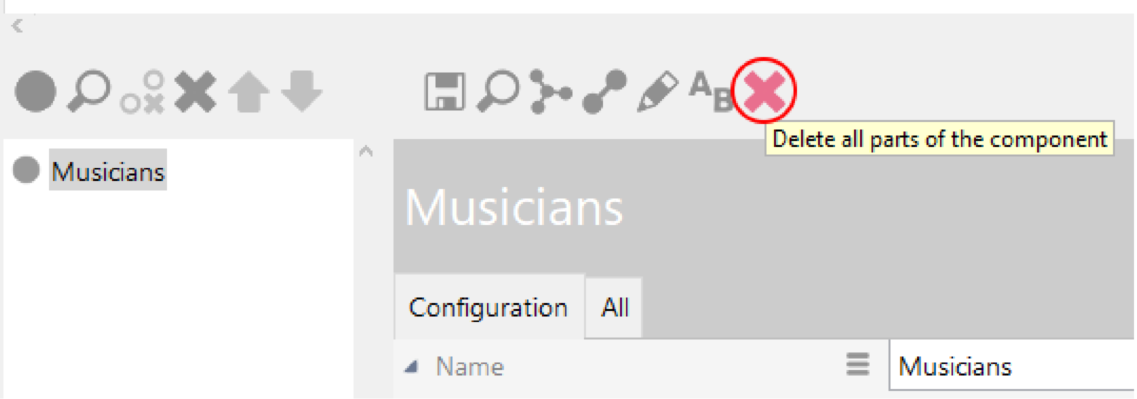
Imported component: Musicians v0.0.1   Shown Warnings: 4/4   Import-date: May 27, 2025 1:57:47 PM

In case there are no Elements to be edited in the table you might be able to find the ID of relevant elements in the description on the right. These IDs can be looked up in the KB by clicking on the menu in the top right selecting **Administrator** → ***Lookup ID***.



1.8.2.11.3. Removing an imported component

An imported component can be removed from within the Knowledge Builder by selecting the component under **TECHNICAL** → **Custom components** and clicking the **Delete** button on the right side of the detail view of the component.



When deleting a component you can choose wether to only delete the component and its sub-objects for additional selection or the entire component including all assigned elements (semantic elements, queries, etc.).

NOTE Any additional property-concepts which are defined exclusively for elements that will be removed will also be deleted as they cannot exist without a defined domain.

WARNING If you try to remove a component like other objects via its context-menu or the buttons above the table it will only remove the component-object and its sub-objects without the assigned elements.

1.8.2.12. Commandline commands

The Batch-tool provides the following commands for custom components:

Command	Parameter	Value	Optional
ImportCustomComponent	file	filename of the component to import	No
	surplusHandling	*keep*, *bin* or *delete* to override the handling of surplus elements of the component to import	Yes
ExportCustomComponent	file	filename for the exported component	No
	uri	the base-URI of the component to export	No
	includeDependencies	Boolean. If true, all components, that the exported one is dependent on, will be exported as well.	Yes

1.9. External Index

In contrast to internal indexing, external indexing involves transferring data from the knowledge graph to a third-party system so that its features can be used in the search. The tools for mapping data sources are used to transfer the data. Triggers are used to update the external data, and specialized implementations are available for the third-party system for the search functionality.

1.9.1. Application Areas

- Realization of functions (aggregation, linguistics, path algorithms, etc.), which are not offered by i-views.
- Acceleration of the search, result display and faceting (especially for large data volumes)
- Decoupling in the architecture of the application (e.g. UI directly on external index)
- "Overhang" data - i.e. there are more objects in the external index than are known to the K.Graph
- Coupling/data exchange with other systems

2. Admin Tool

You can use the Admin tool to create new Knowledge Graphs, manage all Knowledge Graphs of a mediator and configure individual Knowledge Graphs.

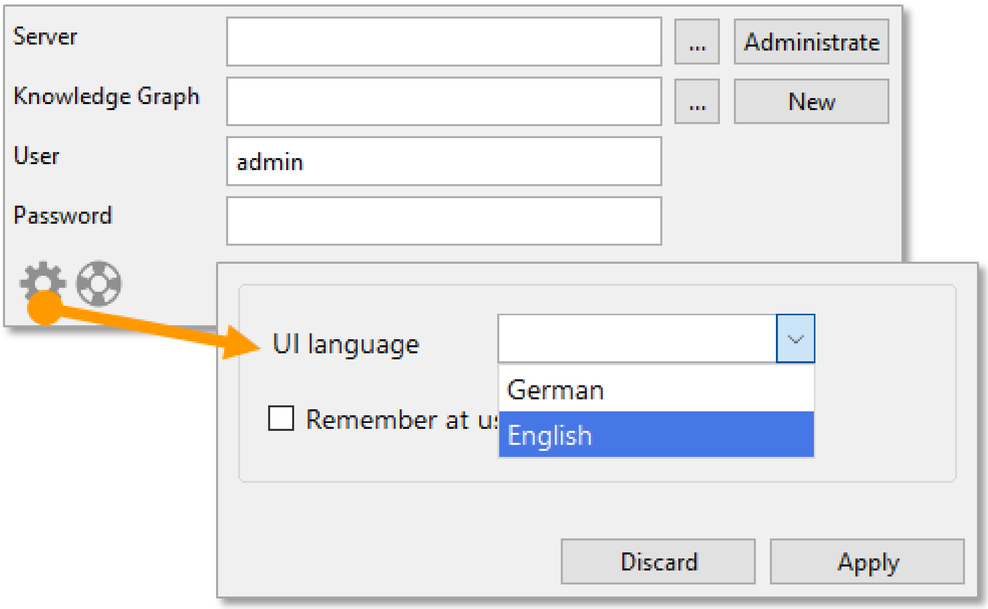
The Admin tool is invoked by default with the following:

- Windows: `admin.exe`
- Mac OS: `admin`
- Linux: `visual admin.im`

It is safe to rename the files.

2.1. Admin tool configuration

Like the Knowledge Builder, the admin tool can be startet with English or German user interface (UI). The preset UI language ist German. To start the admin tool with English UI, a configuration needs to be done using the selection dialog, an ini file or a command line argument. The language selection dialog is available via the start dialog:



Server
Knowledge Graph
User
admin
Password
UI language
□ Remember at u
German
English

NOTE If a new Knowledge Graph is created using the admin tool, the system attributes and system relations are created in the same language as the admin tool has been started with.

Besides setting the UI language of the admin tool using the selection dialog, setting the UI language of the (initial) default UI language can be set using the ini file or a command line argument.

The content of the ini file "admin.ini" for starting the admin tool with English UI is as follows:

```
[Default]
language=eng
```

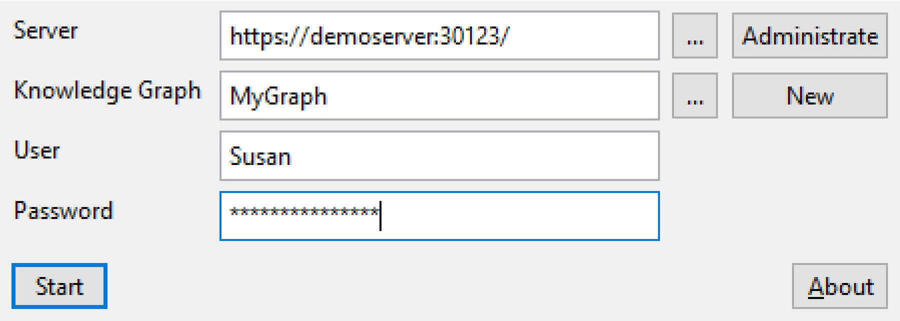
The command line argument to switch the language is:

```
... -language eng
```

Please note that without further configuration, the ini file needs to be located in the same directory as the admin tool itself to take effect.

2.2. Launch window

After the Admin tool has started, the **Start window** appears.



Server https://demoserver:30123/

Knowledge Graph MyGraph

User Susan

Password ***************

2.2.1. Server

The URL of the server is entered in the free text field **Server**. Valid URLs use one of the protocols `cnp://,cnps://,http://` or `https://` followed by `[<hostname or IP address>[:<port number>]]`. If no protocol is specified, the protocol `cnp://` is used. If not port is given, the standard port for the protocol is selected. This format corresponds to the interface setting on the mediator.

If the mediator that is used to administrate the Knowledge Graphs is running on the same computer as the Admin tool, it can also be addressed using the computer name `localhost`.

If the field remains blank, then the Knowledge Graphs are accessed which are in the direct subfolder `volumes` relative to the position of the Admin tool. No mediator is required for this type of access.

Entries entered once in the free text field are saved. The **...** button allows them to be selected from a list in a separate window.

The **Administrate** button is used to access the **server administration**, for a subsequent dialog requires the servers password.

2.2.2. Knowledge Graph

The Knowledge Graph that is to be administrated is specified in the free text field **Knowledge Graph** .

Entries entered once in the free text field are saved. The **...** button allows them to be selected from a list in a separate window. To display all Knowledge Graphs, the user may be prompted to enter the server password.

2.2.3. Administrate, New and Start

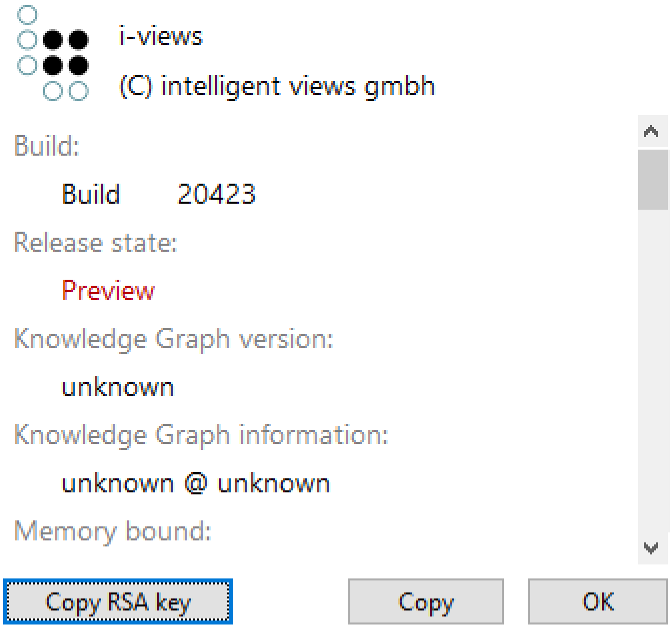
Administrate is used to access the **server administration**, for which authentication using the server password is required.

New forwards to Knowledge Graph generation.

Start forwards to the individual graph administration. The entries **user name** and **password** are used for this for logging in with an administrator account.

2.2.4. About

You can use the **About** button to retrieve version-specific information in a separate window via the Admin tool.



i-views

(C) intelligent views gmbh

Build:

Build 20423

Release state:

Preview

Knowledge Graph version:

unknown

Knowledge Graph information:

unknown @ unknown

Memory bound:

Specifically, you can retrieve:

- the version number of the Admin tool (*Build*),
- the publication status of the Admin tool (*Release state*),
- the maximum system memory in bytes that can be used by the Admin tool (*Memory bound*),
- the amount of memory used at which the process will start reclaiming memory (*GC threshold*),
- the version number and the digital finger print of the execution environment used by the Admin tool (*VM version*),
- the active HTTP proxy configuration (*HTTP Proxy*),
- the external libraries present and their versions (*External Libraries*),
- the language setting active in the operating system (*Locale*),
- the fonts used in the Admin tool (*Fonts*),
- the software components including version numbers present in the software (*Software*

components) and

- the core packages including version number used in the Admin tool (*Packages*).

Information on the *License*, *Knowledge Graph version* and *Knowledge Graph information* is not decisive here.

The information is shown in an invisible text field, which has a context menu that can be activated by right-clicking:

- **Copy** copies the selected text area to the clipboard of the operating system.
- **Find** allows a string to be input in a separate window, and its next occurrence in accordance with the read direction in relation to the position of the cursor set by clicking the mouse. The query is case-sensitive.
- **Find Again** searches for the selected text area and finds its next occurrence in according to the read direction.
- **Select All** selects all the text. Alternatively, the mouse pointer can be used to mark any text segment.

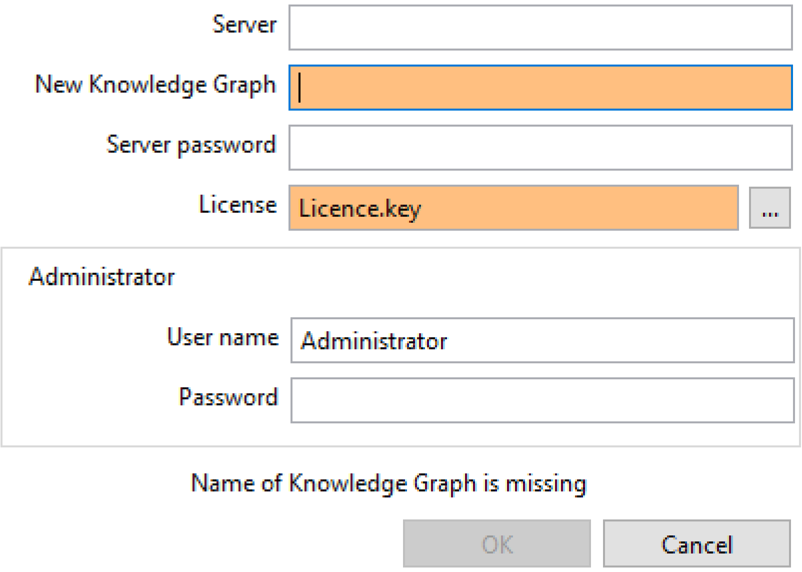
The **Copy** button at the bottom copies all information to the clipboard of the operating system.

The **Copy RSA key** button copies the unique key for this build of the Admin tool to the clipboard of the operating system. This key can be used in the configuration of a mediator to restrict the tool build versions allowed to access the mediator.

The **OK** button enables you to return to the start window.

2.3. Create a new Knowledge Graph

A new Knowledge Graph is created via a separate **Knowledge Graph creation window** . It can be reached via the **New** button on the **start screen** . Any inputs in the **Knowledge Graph** free text field of the start screen is ignored.



Server

New Knowledge Graph

Server password

License: Licence.key

Administrator

User name: Administrator

Password

Name of Knowledge Graph is missing

2.3.1. Server

The name or the IP address of the computer on which the mediator is running is specified in the free text field **Server**. The value is copied from the start screen. If you use the server field in this dialog the same conditions as in the start screen apply.

If the field remains blank, the Knowledge Graph is generated in the `volumes` subfolder relative to the location of the Admin tool.

2.3.2. New Knowledge Graph

The name of the Knowledge Graph is specified in the free text field **New Knowledge Graph**. The characters allowed for this purpose are specified by the file system of the operating system on which the Knowledge Graph is to be stored. To ensure that the data can also be stored in different file systems, consider the following best practices:

- 64 characters maximum
- No blank spaces at the start or end
- Characters used: Upper and lower case ASCII letter characters, numbers, space and `-+.` `Permitted are also all Latin letters, !@#$%&'()+-.[]^`` ` ``{}~Œœ and ASCII characters 160-255 but not advised
- The following character sequences are not allowed: AUX, CON, NUL, PRN as well as COM0-

COM9 and LPT0-LPT9

A name must be specified.

The name can subsequently be changed only during copy processes of the Knowledge Graph or by changing file and directory names. If you make a change, keep in mind that the name of the Knowledge Graph might be referenced in initialization files and that the license might have been adapted to the name.

2.3.3. Server password

The mediator server supports authentication via a password. If a password has been set for the mediator it will be used to create the new Knowledge Graph. The server password must be entered in the **Server password** free text field. If no password has been assigned, the field must remain empty.

2.3.4. License

A Knowledge Graph must have a valid license so that Knowledge Builder and other software components (with the exception of the Admin tool) can access it. Use the **...** button to access the file system of the operating system in order to load a license key (file name: `<License name>.key`).

2.3.5. User name

The name of the first user registered in the Knowledge Graph is specified in the **User name** free text field. The type and quantity of permitted characters is not restricted. The Administrator default setting is simply a suggestion. This field must not remain empty.

The name can be changed later on in the Admin tool or the Knowledge Builder. The user created in this way automatically is granted administrative rights.

2.3.6. Password (user)

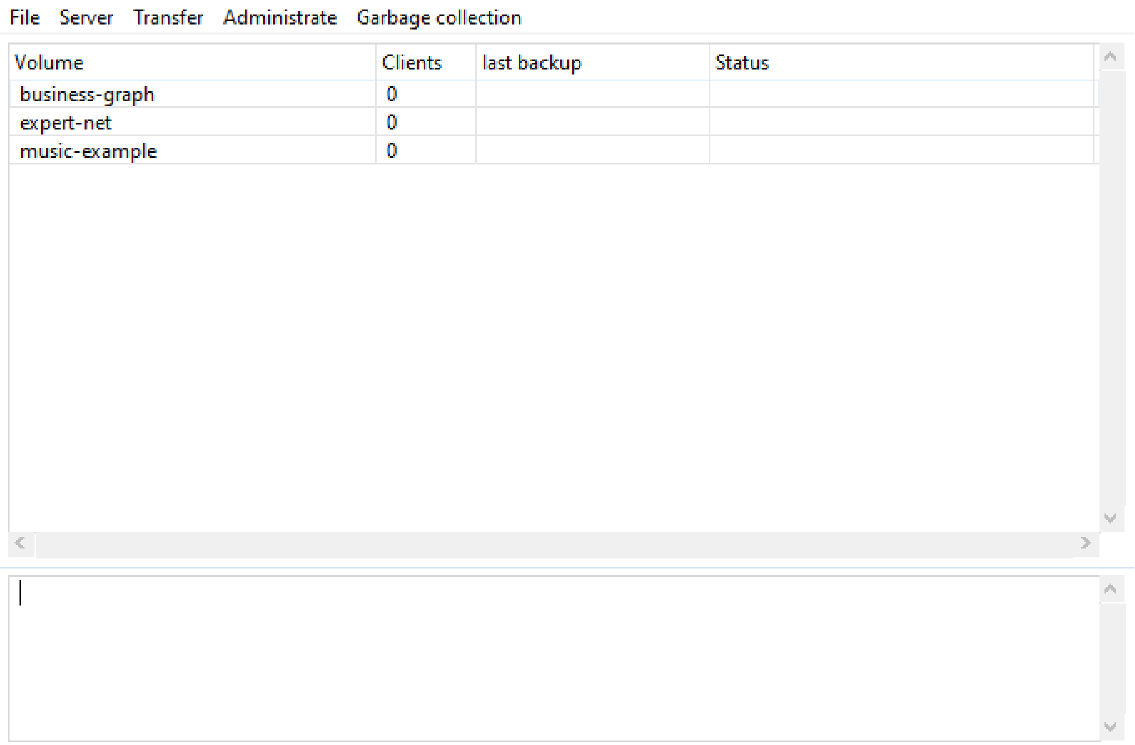
In the **Password** free text field, you should enter a password for the first user registered in the Knowledge Graph. This password will be required later on when this user attempts to log in to the Knowledge Builder or the Admin tool for the new Knowledge Graph.

2.3.7. Ok and Cancel

The **OK** button creates the Knowledge Graph, factoring in the data entered. The **Cancel** button cancels the process. In both cases, the system returns to the **start screen**.

2.4. Server administration

The overall Knowledge Graph administration allows the administration of all Knowledge Graphs of a mediator, or the local subfolder *volumes* respectively. It can be reached via the **Administrate** button on the **start screen**. A corresponding entry in the **Server** field is necessary for this. Any entries in the **Knowledge Graph** field of the **start screen** are ignored. If the Knowledge Graphs to be administrated are addressed using a mediator, the correct mediator password must also be specified in a separate window.



Volume	Clients	last backup	Status
business-graph	0		
expert-net	0		
music-example	0		

The **overall graph administration window** is comprised of a **graph overview** in the form of a table, a **message area** and a **menu line** at the top.

2.4.1. Graph overview

The **graph overview** in the form of a table provides details about

- the name (*Volume*)
- the number of clients currently connected (*Clients*),
- the date and time of the last backup (*last backup*) and
- the last status message (*Status*) of the respective Knowledge Graph.

The individual columns can be sorted by clicking on the head of the column.

The data is only updated when triggering operations, and therefore is not always up-to-date. A manual update can be forced at any time using the menu item **Server** -→ **Refresh Knowledge Graph**

list .

2.4.2. Message area

The **Message area** outputs all status reports for all Knowledge Graphs. Status reports are created when activities are triggered in the Admin tool. They are not saved when the Admin tool is closed. To prevent this, they can be exported via the menu option **File** -→ **Write administration log** . The **Message area** can be edited, but changes are ignored during export.

2.4.3. Menu line

The **menu line** consists of the following menu tabs:

File

Save administration log

saves all entries in the message window in a text file (default file name: *admin.log*). You can freely choose the name and storage location in a saving dialog. This operation requires the Admin tool to be connected to a mediator.

Reset session password

Log off

Closes server administration and opens the log-in window again.

Exit

Closes server administration

Server

Refresh Knowledge Graph list

Reloads the data collected in the **graph overview** in the **overall graph administration window**.

Re-import ini file

Triggers the server to reload its ini file. Note, that not all options can be updated during operation. The server outputs a message about updated options.

Download log

Downloads a copy of the mediator log file if present on the server (default file name: *mediator.log*) to the local machine. You can freely choose the name and storage location of the file in a saving dialog. The mediator log file keeps a log of all the mediators activities from its first commissioning.

Server connections

Shows the client id and individual IP address of all software clients currently connected to the mediator in the **message field**. The output contains a total count and is grouped by Knowledge Graph. The client id is generated sequentially by the mediator and assigned (reusing free ids) whenever a new software component registers.

Transfer

Download Knowledge Graph

Creates a copy of the Knowledge Graph selected in the **graph overview** and saves it locally in the *volumes* subfolder that is located relative to the location of the Admin tool. A new name can be assigned to this copy in a separately appearing dialog.

Copy Knowledge Graph

Creates a copy of the selected Knowledge Graph and saves it on the same server as the original Knowledge Graph. A new name must be assigned to this copy in a separate dialog.

Upload Knowledge Graph

Uploads a local Knowledge Graph to the server. In a separate dialog, a local graph can be selected from a list, which is filled from the *volumes* directory relative to the Admin tools directory location. A new name can be assigned to this copy and it must differ from all graph names already present on the server.

Replace Knowledge Graph

Exchanges the contents of a selected Knowledge Graph with the contents of a locally present graph. In the process, the uploaded copy is given the name of the Knowledge Graph it has replaced. The local Knowledge Graph, which must be stored in the *volumes* subfolder that is relative to the position of the Admin tool, is selected in a separate selection window.

As a result of the copy processes initiated by transfer operations, the block allocation of the clusters and blobs within the Knowledge Graph copies is redefined, and their space consumption is optimized in the process. The resulting compression effect is identical to the one achieved by the operation **Manage** → **Compress volume**.

With the exception of the **Copy volume** operation, all these operations require the Admin tool to be connected to a mediator.

Administrate

Open Admin tool

Opens the Admin tool on the selected Knowledge Graph. Since this administration interface used authentication on the server level, no further authentication is required. In a separate window a list of accounts with administrative access in the selected Knowledge Graph is presented. Here the account to use when opening the Admin tool in "per graph" mode can be selected.
This makes it possible to access the user management of the volume if the administrator password has been lost.

Create backup

Creates a backup of the Knowledge Graph selected in the **Knowledge Graph overview** and saves it in the *backup* folder on the server. Every backup is a full copy of the original Knowledge Graph.

When the backup is initiated, a separate window asks whether the user wants to wait until the copy process is complete. If applicable, further use of the Admin tool is blocked until this time. Otherwise the copy process starts in the background, and there is no message regarding the process or completion of the copy process.

Restore backup

Offers a list of available Knowledge Graphs with backups present on the server. After picking a knowledge graph a second dialog offers the list of timestamps for backups of that graph. When selecting a timestamp, a new name must be assigned for the backup to be restored to.

Delete backup

Deletes a selected backup. To select this backup, two separate selection windows must be navigated: in the first, the Knowledge Graph must be selected; in the second, the backup timestamp must be selected from a list sorted by creation date.

The block assignment of clusters and blobs within the original Knowledge Graph is not modified when a Knowledge Graph copy is created. The copy process initiated by the backup operations therefore creates no compression effect.

Delete Knowledge Graph

Deletes the Knowledge Graph selected in the **Knowledge Graph overview**.

Compress Knowledge Graph

Remove free blocks in the files comprising the selected Knowledge Graph. Free blocks originate from changed data freed by the garbage collection.

The copying processes for clusters and blobs first move all unused blocks to the file end and then release them in the file system of the operating system.

Update volume version

Updates the version of the internal file system of the Knowledge Graph selected in the **Knowledge Graph overview**. If the Knowledge Graph is addressed via a mediator, the version it contains is used; otherwise, the version included in the Admin tool is used. The update should be performed, when upgrading from previous i-views versions.

Garbage collection

Garbage collection is a procedure that deletes objects that are no longer referenced (according to a programming terminology reading) from the Knowledge Graph. It thereby minimizes reduces the storage usage of the Knowledge Graph. Use of the garbage collection requires that the Knowledge Graph that is to be cleaned up is accessed via a mediator.

Start

Initiates a new garbage collection run for the Knowledge Graph selected in the **Knowledge Graph overview** or continues a garbage collection paused for it. No confirmation is sent when the process is completed. You can determine its progress via the **Status** menu option.

Pause

Halts the execution of the active garbage collection for the Knowledge Graph selected in the **Knowledge Graph overview**.

Stop

Terminates the execution of the active garbage collection for the Knowledge Graph selected in the **Knowledge Graph overview**.

Status

Fetches the status of the current garbage collection process for the Knowledge Graph selected in the **Knowledge Graph overview** and displays it in the status column of the **Knowledge Graph overview** and in the **message field**. If garbage collection is active, feedback on its progress is provided in percent.

2.5. Individual Knowledge Graph administration

Individual Knowledge Graph administration allows you to manage an individual Knowledge Graph. It can be reached via the **Start** button on the start screen. This requires the corresponding entries in the fields **Server**, **Knowledge Graph**, and authentication field combination, by default **Authentication** set to `Name and password`, **User** and **Password** of the start screen.

2.5.1. User authentication

To access the **Knowledge Graph administration window** a user account with administrative rights needs to be authenticated.

If you no longer have access to the Knowledge Graph, you can access the Knowlede Graph through authentication on the server by logging on to the **Server administration**.

Valid authentication methods for logging in to a specific Knowledge Graph depend on the configuration of the mediator and the graph. These details need to be provided to the users of the Admin tool.

Possible authentication methods are:

Name and password

The fields **User** and **Password** are used to authenticate against accounts defined in the Knowledge Graph

JSON Web Token

(*experimental*) A valid JWT needs to be pasted into the **Password** field. The token issuer needs to include the allowed operations as claims in the token. The **User** field contents are ignored.

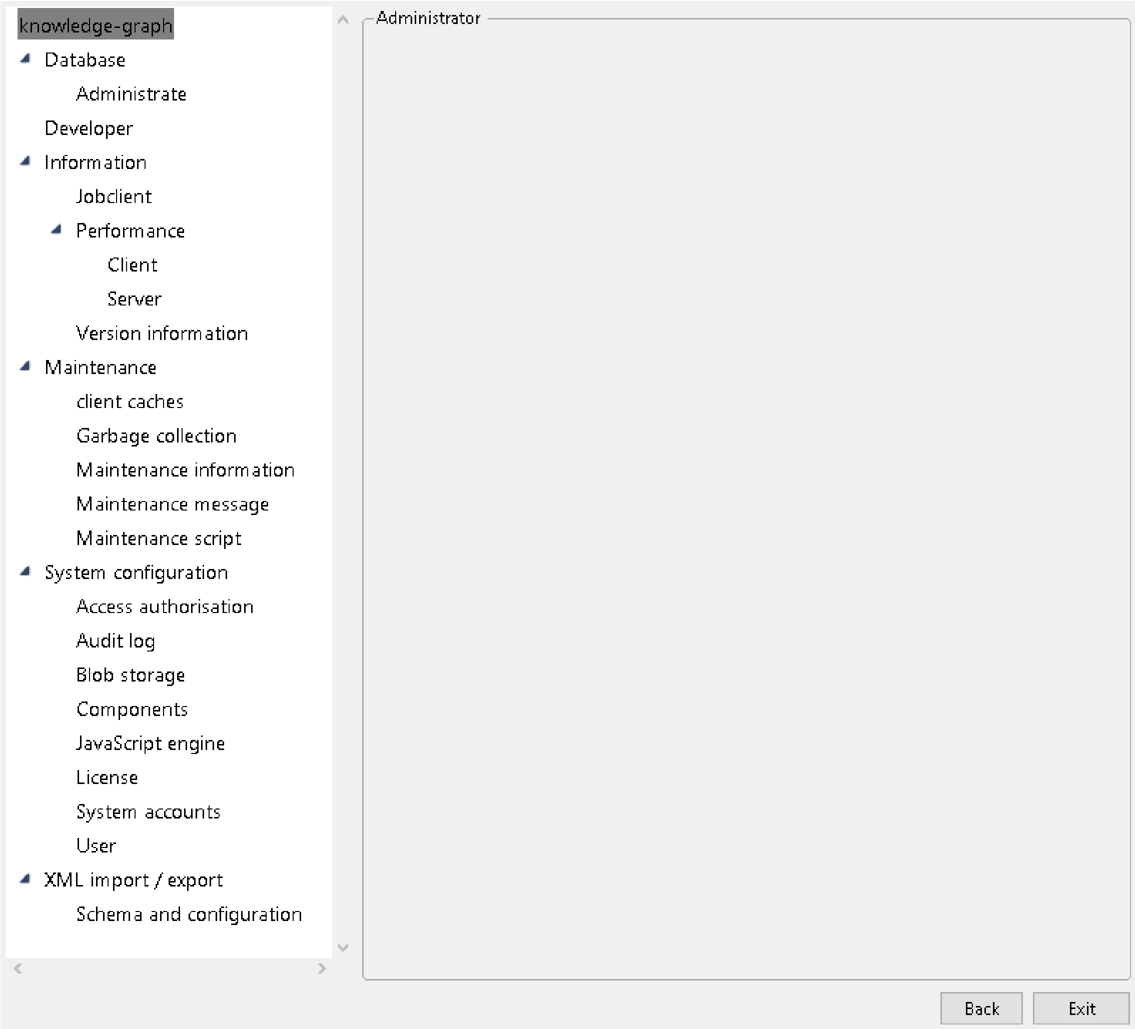
OAuth

To log in the user is redirected to the configured OAuth authentication provider service in the browser. After successful authentication and if the users account is authorized to log in to the installation, the Admin tool opens.

Windows negotiate

Use the Windows negotiate API to authenticate the user with the Knowledge Graph.

2.5.2. Individual Knowledge Graph administration window

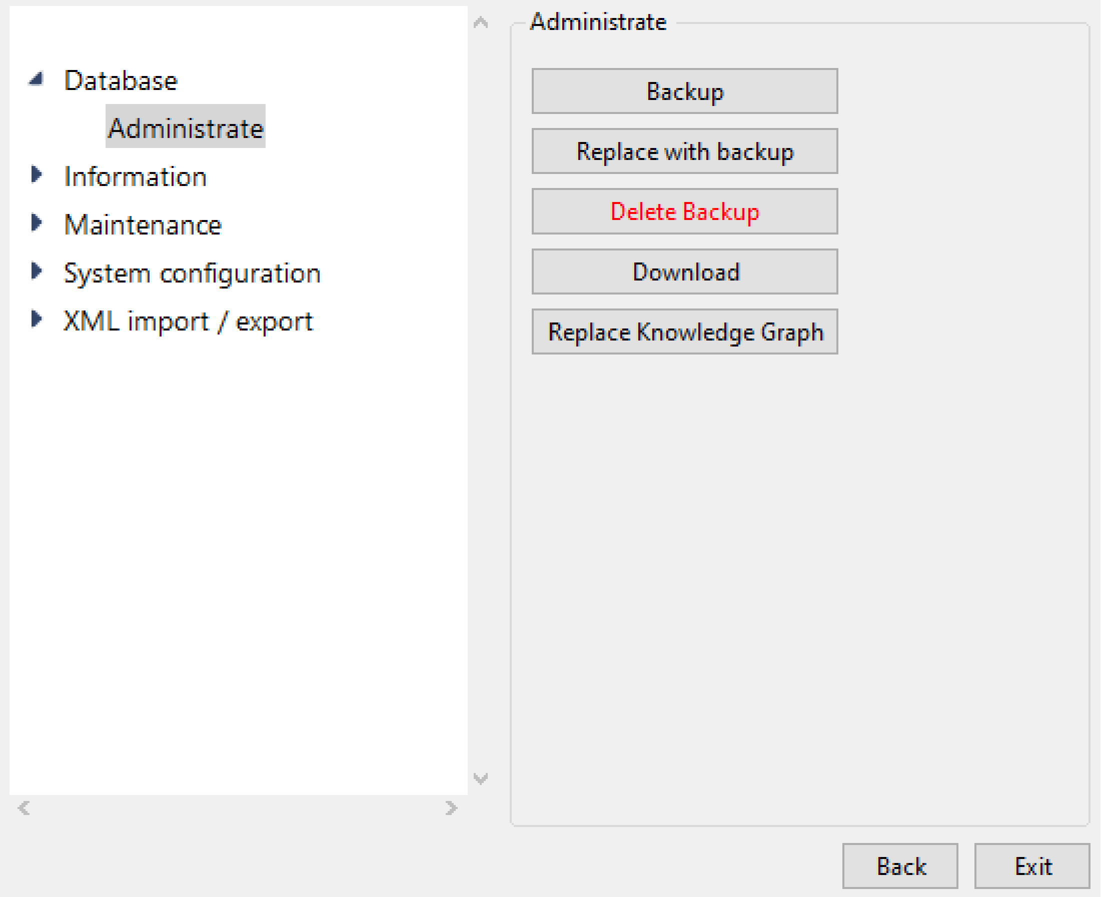


The **Knowledge Graph administration window** has a menu list with a multilevel structure on the left, and an operation window on the right. The content of the operation window depends on the menu option selected in the menu list.

The **Back** button returns you to the start window.

The **Exit** button closes the Admin tool.

If the Knowledge Graph to be administrated is addressed without a mediator, other clients cannot access the Knowledge Graph via the Knowledge Builder or another instance of the Admin tool for as long as the **Knowledge Graph administration window** is open.



2.5.2.1. Database

2.5.2.1.1. Administrate

Backup

Creates a backup of the Knowledge Graph on the server. Every backup is a full copy of the original Knowledge Graph.
Before the backup is created, a separate window asks whether the user wants to wait until the copy process is complete. If applicable, further use of the Admin tool is blocked until this time. Otherwise the copy process starts in the background, and there is no message regarding the process or completion of the copy process.

The block assignment of clusters and blobs within the original Knowledge Graph is not modified when a Knowledge Graph copy is created. The copy process initiated by the backup operations therefore creates no compression effect.

Replace with backup

Allows to replace the current Knowledge Graph with one of its backups created earlier on. A dialog with available backups offers the selection of a specific backup set. If performed all connected clients (including this Admin tool) will shortly lose the connection and reconnect again.

Delete backup

Deletes an individual backup of this Knowledge Graph from the list of available backups.

Download

Downloads a copy of the Knowledge Graph and saves it locally to the *volumes* subfolder that is located relative to the location of the Admin tool. A new name can be assigned to this copy in a separately appearing free text field.

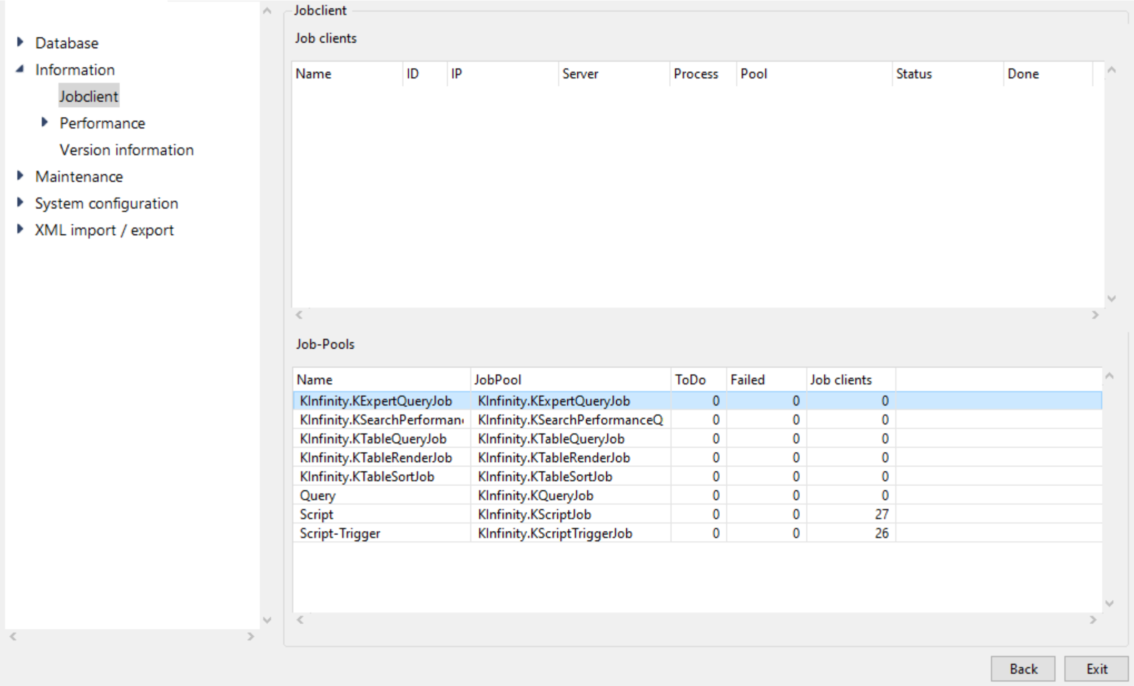
Upload volume

Transfers a locally stored Knowledge Graph and replaces the current Knowledge Graph with this Knowledge Graph (afterwards you are logged off automatically).

2.5.2.2. Information

2.5.2.2.1. Jobclient

In order to relieve the workload on the Knowledge Builder for specific, processor-intensive processes such as indexing, querying Knowledge Graphs and executing scripts, these processes can be performed by Job-Clients. Some are optionally executed by Job-Clients while others are exclusively performed them. The use of jobs is usually configured in the graph, but the user interface of the Knowledge Builder offers the execution as job in several locations. To perform these jobs at least one Job-Client must be configured and running for the intended kind of job. The Admin tool largely functions as an observer in this case. Jobs not completed appear in the Knowledge Builder under the entry *Tasks* in the *Technology* category. In order to use the Admin tool to manage Job-Clients, the Admin tool must be connected to a mediator.



Jobclient

Job clients

Name	ID	IP	Server	Process	Pool	Status	Done

Job-Pools

Name	JobPool	ToDo	Failed	Job clients
KInfinity.KExpertQueryJob	KInfinity.KExpertQueryJob	0	0	0
KInfinity.KSearchPerforman	KInfinity.KSearchPerformanceQ	0	0	0
KInfinity.KTableQueryJob	KInfinity.KTableQueryJob	0	0	0
KInfinity.KTableRenderJob	KInfinity.KTableRenderJob	0	0	0
KInfinity.KTableSortJob	KInfinity.KTableSortJob	0	0	0
Query	KInfinity.KQueryJob	0	0	0
Script	KInfinity.KScriptJob	0	0	27
Script-Trigger	KInfinity.KScriptTriggerJob	0	0	26

The **Job-Clients overview** table on the top right shows the following for each job-client that is currently running:

- its name in the format [Job-Client-Name]@[Mediator-Name] (*name*),
- its client id (*ID*),
- its IP address relative to the mediator (*IP*),
- the name of the mediator connected to it (*Server*),
- the process number assigned by the operating system (*process*),
- the job types assigned to it (*Pool*),
- its work status (*Status*) and
- the number of jobs it has completed (*Done*).

The Client id assigned by the mediator and a separate number is assigned with each new client connection. The Job-Client name and the job types assigned to the Job-Client are defined in the initialization file for the respective Job-Client (default file name: *jobclient.ini*) under the key *name* or the key *jobPools* respectively. Each job type of a Job-Client is shown in a row of its own in the Job-Client overview, so that a Job-Client regularly takes up several rows.

The individual columns of the **Job-Clients overview** can be sorted by clicking on the head of the column. Right-clicking a row also opens a context menu:

- Display information:: displays all data listed in the selected row, with the exception of the job type and the completed number of jobs, in a new window. Added are
  - the date and time of the last time the Job-Client was started (*startUpTime*),
  - the maximum working memory capacity available for use by it in bytes (*max Memory*),
  - the name of its log file (*logFileName*) and
  - its specific name, under which it can be forced to shut down (a concatenation of the string “jobclient” and the Job-Client number) (*shutDownString*).
- The data there can be copied to the clipboard of the operating system (**Copy to clipboard** button) or be exported to any location as a text file that can be given any name using a saving dialog (**Save** button).
- The operation triggered using the menu item **Display information** can, alternatively, be performed by double-clicking a row in the Job-Clients overview.
- **Remove Job-Client** ends the Job-Client selected in the **Job-Clients overview** .
- **Remove all Job-Clients** ends all Job-Clients listed in the **Job-Clients overview** .

The **job pools overview** on the bottom right in form of a table lists all job types that are assigned to at least one Job-Client in the **Job-Clients overview** . For each job type it displays,

- its name (*name*),
- its technical name used in the Job-Client’s initialization file (*JobPool*),
- the number of uncompleted jobs of this job type (*ToDo*),
- the number of failed jobs of this job type (*Failed*) and

- the number of Job-Clients available to execute the type of job (*Job-Clients*).

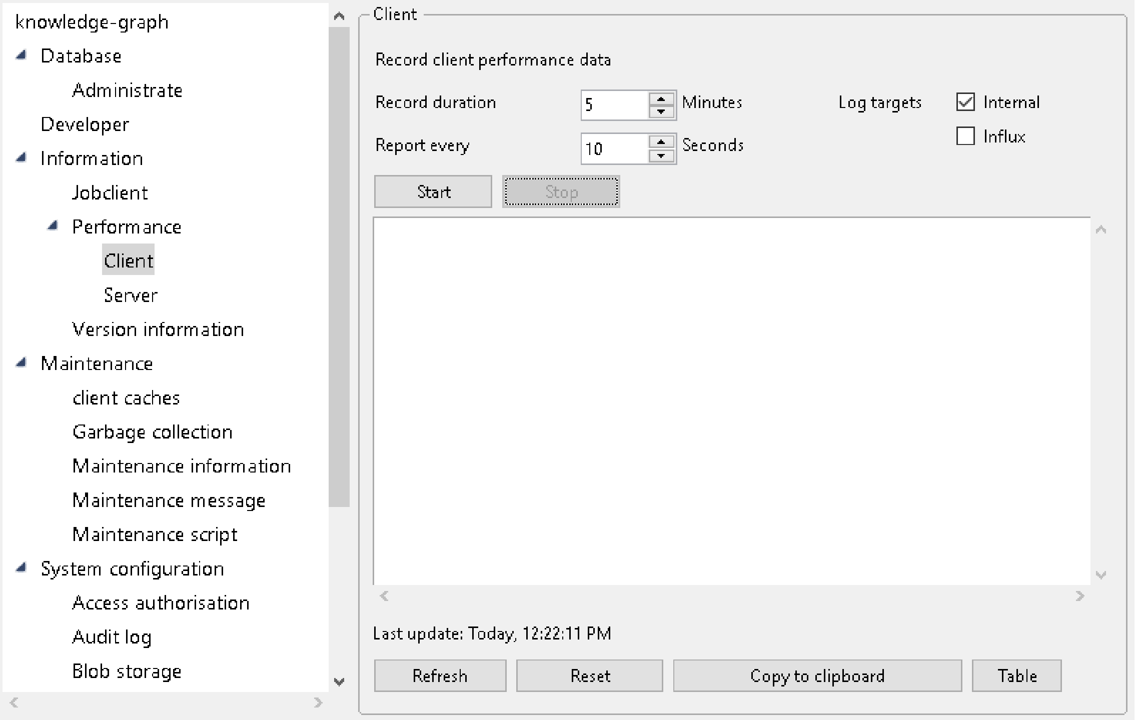
The individual columns of the **job pools overview** can be sorted by clicking on the head of the column. Right-clicking a Job-Client also opens a context menu:

- **Empty job pool** deletes all uncompleted and failed jobs of the job type selected in the **job pools overview** . This operation is only possible when no Job-Client is running.
- **Configure error messages to ignore** allows specific error messages to be blocked when executing jobs of the job type selected in the **job pools overview**.
    - If an error message is blocked this way, the job related to the error is not factored in when determining the number of failed jobs in the **job pools overview**. This operation is only possible when there are already jobs of the job type selected in the **job pools overview** waiting to be processed, or that were already processed.
    - The error messages to be blocked are administrated in a separate window:
        - All error messages to be blocked are listed in the alphabetically sorted **error message list**. An error message is blocked when its output text matches a text in the **error message list**.
        - "**+**" allows input of an error message to be blocked using a separate window. The error message appears in the **error message list**.
        - "**...**" allows the error message selected in the **error message list** to be changed.
        - "**-**" deletes the error message selected in the **error message list** .

2.5.2.2.2. Performance

Client

Allows to activate the collection of performance data for clients.



Record client performance data

Record duration 5 Minutes

Report every 10 Seconds

Log targets ☑ Internal ☐ Influx

Last update: Today, 12:22:11 PM

Record duration configures the number of minutes to keep the performance data collection active. The preset is 5 minutes.

Report every configures the duration in seconds between performance data snapshots. The preset is every 10 seconds.

Log targets offers the logging to the internal log collector and/or a configured Influx time series database.

Start starts the client performance data collection with the given parameters. It will stop after the configured duration or when klicking

Stop stops a running client performance analysis.

The key performance indicators are shown in a nested list in the **key performance indicator overview**. Clicking on the triangle symbols to the right of the categories allows listed subitems to be expanded and collapsed. Alternatively, this can be performed using a context menu, which can be accessed by right-clicking a list item:

- **Expand** opens all directly listed subitems in the list item selected.
- **Expand fully** opens all directly and indirectly listed subitems in the list item selected.
- **Contract fully** collapses all listed subitems in the list item selected.

Double-clicking on a list item allows all key performance indicators stored below it to be shown at a glance in a separate window. There, they can be copied to the clipboard of the operating system (**Copy to clipboard** button) or exported to a text file (**Save** button).

Below the collected statistics there are for buttons available:

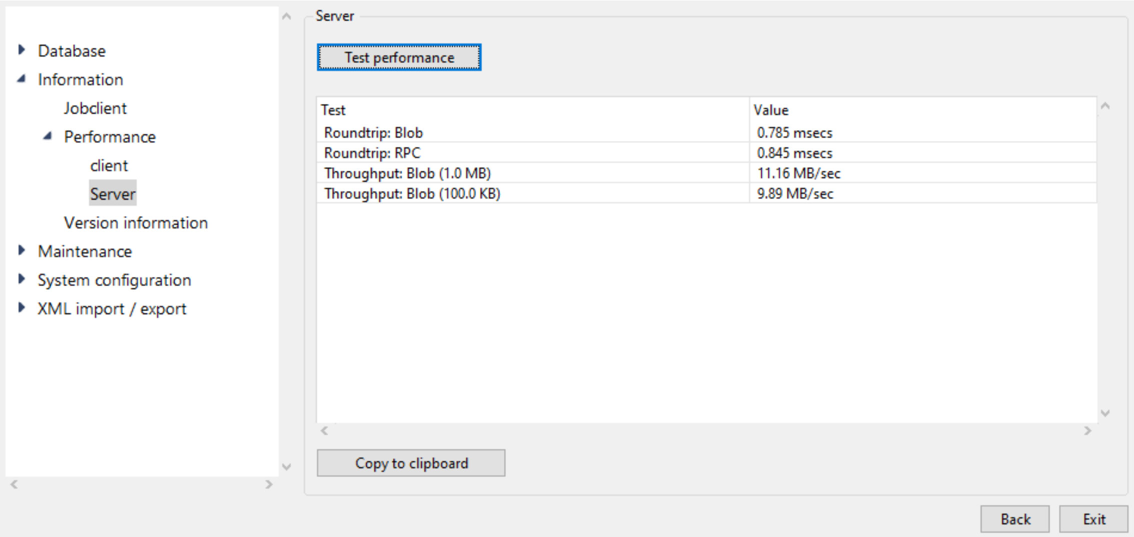
Refresh updats the collectected performance indicators shown in the **key performance indicator overview**.

Reset deletes the collected performance indicators shown in the **key performance indicator overview**.

Copy to clipboard copies the key performance indicators shown in the **key performance indicator overview** to the clipboard of the operating system.

Table opens a dialog with a tabular display of the collected performance data.

Server



Test	Value
Roundtrip: Blob	0.785 msecs
Roundtrip: RPC	0.845 msecs
Throughput: Blob (1.0 MB)	11.16 MB/sec
Throughput: Blob (100.0 KB)	9.89 MB/sec

Test performance starts a process to evaluate the performance of the mediator connection. This sends several requests to the mediator and monitors timing and throughput of the connection. Measurements are taken of

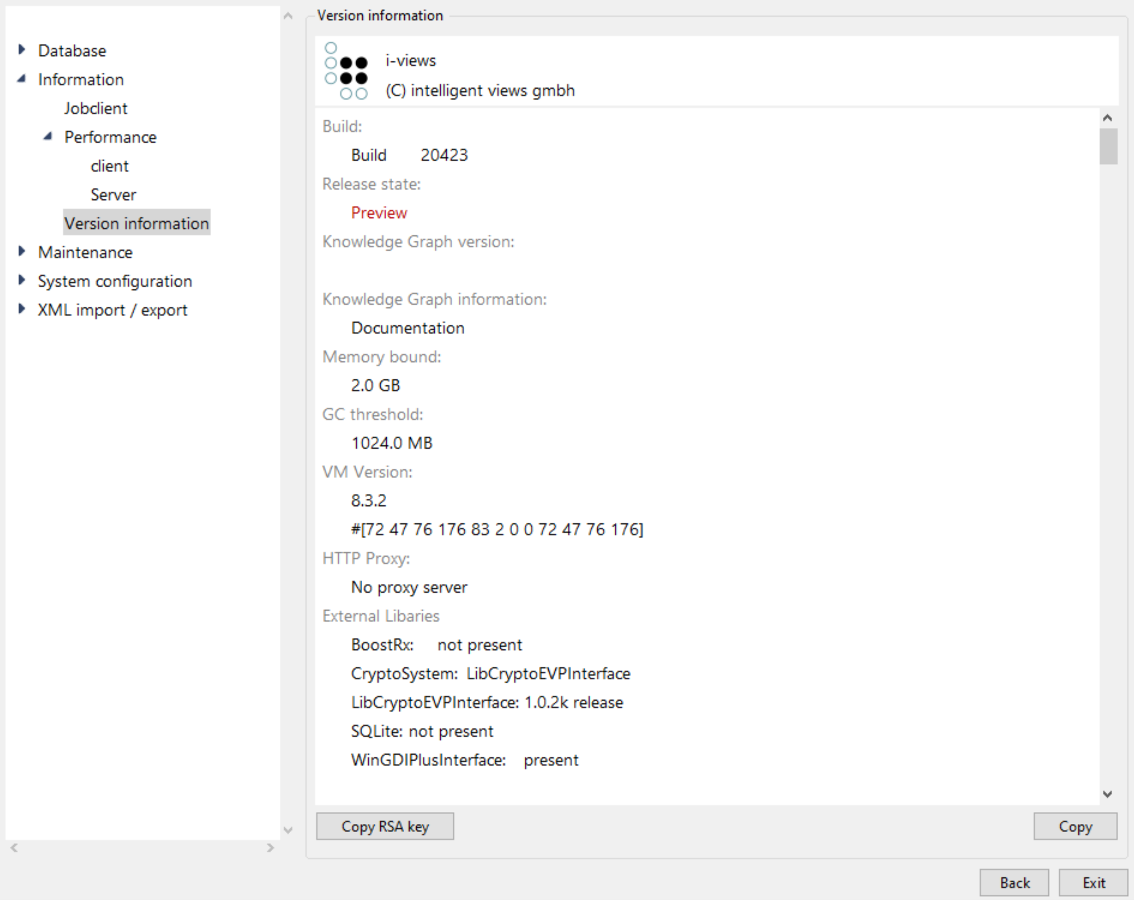
- the duration until a small file is transmitted (*Roundtrip: Blob*)
- the result of a mediator support call request (*Roundtrip: RPC*)
- the average transmission rate when sending several 1 MB files (*Throughput: Blob (1.0 MB)*) and
- the average transmission rate when sending several 100 KB files (*Throughput: Blob (100.0 KB)*).

The test results are shown to the **results list** after the test is finished.

Copy to clipboard copies the test results in the **results list** to the clipboard of the operating system as plain text.

2.5.2.2.3. Version information

This menu item can be used to retrieve version-specific information for the Knowledge Graph and Admin tool.



Build:
Build 20423
Release state:
Preview
Knowledge Graph version:

Knowledge Graph information:
Documentation
Memory bound:
2.0 GB
GC threshold:
1024.0 MB
VM Version:
8.3.2
#[72 47 76 176 83 2 0 0 72 47 76 176]
HTTP Proxy:
No proxy server
External Libaries
BoostRx: not present
CryptoSystem: LibCryptoEVPInterface
LibCryptoEVPInterface: 1.0.2k release
SQLite: not present
WinGDIPlusInterface: present

Specifically, you can retrieve:

- the version number of the Admin tool (*Build*),
- the publication status of the Admin tool (*Release state*),
- the license information present in the graph (*License*),
- the version number of the Knowledge Graph (*Knowledge Graph version*),
- the name of the Knowledge Graph and the mediator used (*Knowledge Graph information*),
- the maximum system memory in bytes that can be used by the Admin tool (*Memory bound*),
- the amount of memory used at which the process will start reclaiming memory (*GC threshold*),
- the version number and the digital finger print of the execution environment used by the Admin tool (*VM version*),
- the active HTTP proxy configuration (*HTTP Proxy*),
- the external libraries present and their versions (*External Libraries*),

- the language setting active in the operating system (*Locale*),
- the fonts used in the Admin tool (*Fonts*),
- the software components including version numbers present in the software (*Software components*) and
- the core packages including version number used in the Admin tool (*Packages*).

The information is output in a text area, which has a context menu that can be activated by right-clicking:

- **Copy** copies the selected text area to the clipboard of the operating system.
- **Find Again** searches for the selected text area and finds its next occurrence in according to the read direction.
- **Find** allows a string to be input in a separate window, and its next occurrence in accordance with the read direction in relation to the position of the cursor set by clicking the mouse. The query is case-sensitive.
- **Select All** selects all the text. Alternatively, the mouse pointer can be used to mark any text segment.

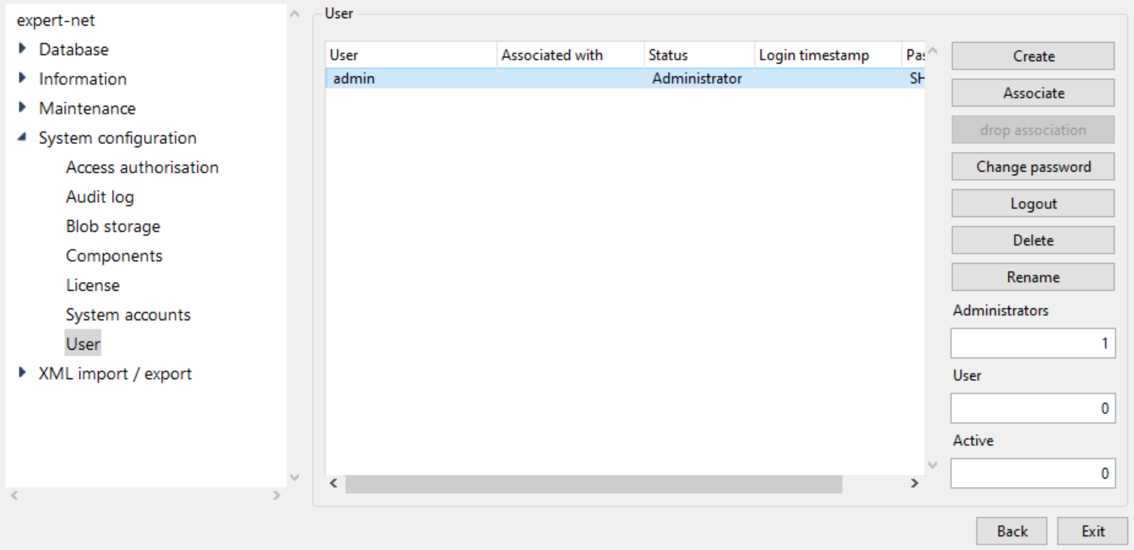
The **Copy** button at the bottom copies all information to the clipboard of the operating system.

The **Copy RSA key** button copies the unique key for this build of the Admin tool to the clipboard of the operating system. This key can be used in the configuration of a mediator to restrict the tool build versions allowed to access the mediator.

2.5.2.3. System configuration

2.5.2.3.1. User

The user administration show the list of configured user accounts in the Knowledge Graph. It can also link these accounts relevant for the login to the Knowledge Builder and Admin tool to user objects in the graph.



User	Associated with	Status	Login timestamp	Pa:
admin		Administrator		SH

Administrators 1

User 0

Active 0

The **user overview** in the form of a table shows, information for the accounts registered in the Knowledge Graph. The fields are:

- the user name (*User*),
- the object of the user-generated subgraph the user is optionally linked to (*Associated with*),
- which status the user currently has (*Status*),
- the date and time of the last login of this account using the Knowledge Builder (*Login timestamp*) if the user is still logged in.

The *Status* provides information about whether a user has administrator rights, whether a user with administrator rights does not have a password and whether a user is logged into the Knowledge Graph using the Knowledge Builder. Names of users with administrator rights without a password are marked in red.

Create creates a new user. User name (obligatory) and password (optional) are defined in a separate window. The type and quantity of permitted characters is not restricted.

Associate associates the account with a user object from the graph model. This association is necessary for Knowledge Graph with enabled access rights managment and accounts, that do not have administrative access.

Drop association removes the selected users association with a Knowledge Graph object.

Change password changes the password of the user selected in the **user overview** . The new password is entered two times in two windows that appear consecutively.

Logout logs out the user selected in the **user overview** from the Knowledge Graph following a security confirmation. This works only for users currently logged in with the Knowledge Builder.

Delete deletes the account object selected in the **user overview** following a security confirmation. At least one user with administrator rights must remain.

Rename allows a new user name to be assigned for the user selected in the **user overview** by means of a free text field in a separate window. If the free text field remains blank, no renaming occurs.

Message opens a text input window to send a message to the user selected in the **user overview**. The message is buffered in the graph and appears to the user in the Knowledge Builder as soon as the user logs in to the Knowledge Graph. The user cannot reply to this message sent from the Admin tool.

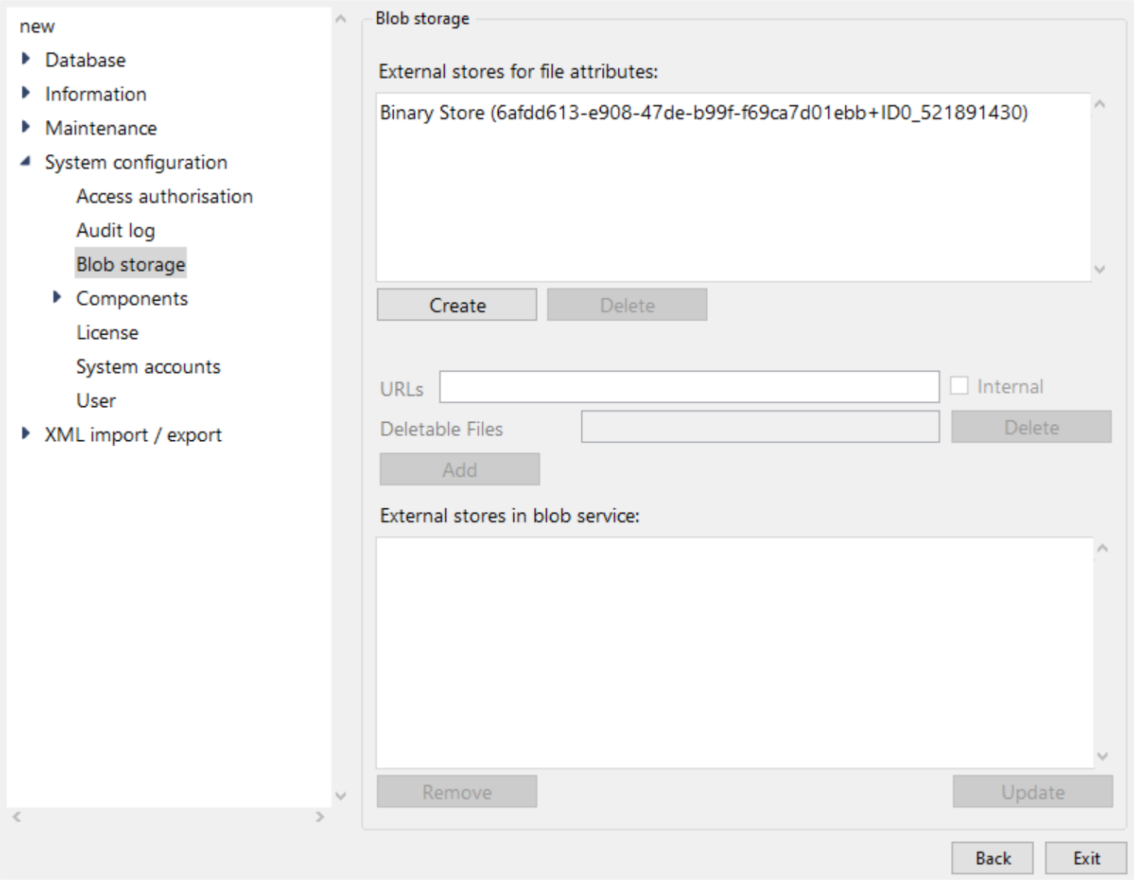
Administrator assigns administrative rights to the account selected in the **user overview**, or removes the flag. A user must have a password to obtain administrator rights. At least one user in the graph must have administrator rights.

Password change is a flag, that when set forces the user to change the password after successful login in the Knowledge Builder. This is usually used when a new account is created by an administrative to to hand out an initial password.

The following three fields **Administrators**, **Users** and **Active** indicate the number of user account with adminitrative rights, the number of all user accounts and the number of active user accounts.

2.5.2.3.2. Blob storage

Attribute values of attributes with the attribute value type *file* (called blobs) can also be stored in a blob store outside the Knowledge Graph. The advantage of this is that they can be managed independently of the Knowledge Graph and can thus be managed in a different system environment. To store blobs in a blob store, the blob store must be set up and connected to a configured blob service (a software service).



External stores for file attributes:

Binary Store (6afdd613-e908-47de-b99f-f69ca7d01ebb+ID0_521891430)

URLs

Deletable Files

External stores in blob service:

Create generates a new blob store. Using the name format *[Knowledge Graph ID]+[blob store ID]* , the **blob store overview** appears in the text field above it.

Delete deletes the blob store selected in the **blob store overview** .

The numeric field **Deletable files** shows the number of blobs no longer required in the **blob store overview** of the selected blob store. Blobs are no longer required when their respective attributes have been deleted from the Knowledge Graph or if the connection between blob service and blob store has been removed using the Admin tool.

Delete deletes all blobs that are no longer required in the blob store selected in the **blob store overview** .

You can identify a blob service in the free text field **URLs** . This is done by entering the network address of the initialization file of the corresponding blob service (default file name: *blobservice.ini*) stored under the *interfaces* key including the prefix *http* . If the blob service is supposed to be addressed via several network addresses, these can be entered in comma-separated form.

Alternatively, the blob service integrated in the mediator can also be addressed. In the initialization file of the mediator (default file name: *mediator.ini*), the value *true* must be set under the key *startBlobService* and the free text field **URLs** must be left blank. The **internal** checkbox to the right of the free text field **URLs** indicates whether the integrated blob service or an external blob service is addressed. The blob service integrated into the mediator is not configured via the mediator initialization file but via a separate initialization file (default file name: *blobservice.ini*).

Add connects the blob store selected in the **blob store overview** to the blob service identified via the free text field **URLs** . To do so, the blob service must be active. If linking is successful, the blob store using the name format *[Knowledge Graph ID]+[blob store ID]* appears in the text field below, the **overview of registered blob stores**.

Update updates the **overview of registered blob stores**. To do this, a blob store must be selected in the **blob store overview**.

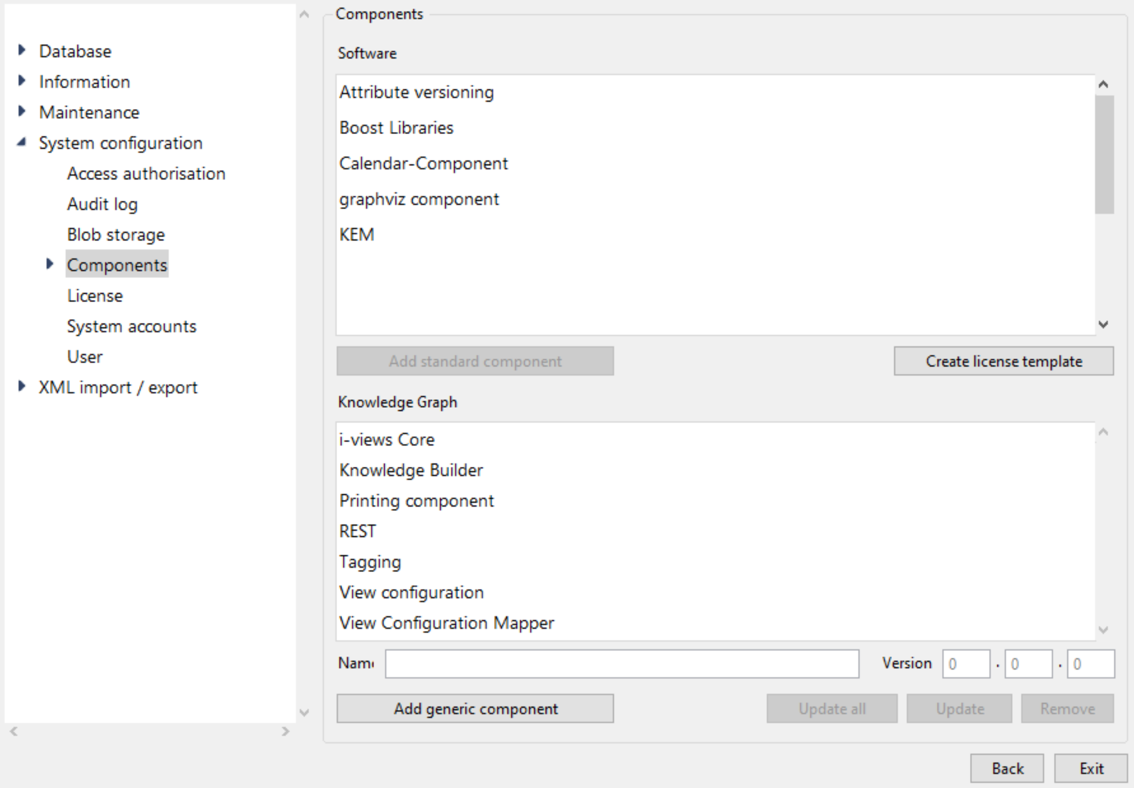
Remove interrupts the connection of the blob store selected in the **overview of registered blob stores** to the blob service and removes the blob store from the overview. In doing so, all blobs stored in the blob store irrevocably lose their internal references to the respective attributes in the Knowledge Graph and can no longer be retrieved in the Knowledge Graph. To ensure removal is successful, the blob store selected in the **overview of registered blob stores** must also be selected in the **complete blob store overview**.

All blobs stored via a blob service are stored in a subfolder called *blobs* that is located relative to the position of the blob service. The internal assignment of every blob to its blob store and its Knowledge Graph is established using an SQLite database.

2.5.2.3.3. Components

Knowledge Graphs consist of Knowledge Graph components. In addition to the basic functions, they basically provide the Knowledge Graph with additional interfaces and user interfaces for user data that can be displayed in the browser (web front-ends).

Publication status components (*Release States*), of which there are three variants (*Preview* , *Release Candidate* , *Release*) are a special subgroup of Knowledge Graph components. If such a component is installed in the Knowledge Graph, only software components with suitable publication statuses are able to access the Knowledge Graph.



The **Software list** provides an alphabetical list of all Knowledge Graph components supplied with the Admin tool and their respective version numbers. If they need a separate license, there is also a note as to whether this is included in the current license of the Knowledge Graph. Publication status components do not have a version number.

If you right-click on a Knowledge Graph component, a context menu appears. The menu item **Add standard component** available there has the same functions as the button of the same name.

Add standard component installs the Knowledge Graph component selected in the **software list** in the Knowledge Graph. A separate window informs of the installation status. Some Knowledge Graph components require other Knowledge Graph components installed in the Knowledge Graph. Most installed Knowledge Graph components (except for publication status components) appear as separate entries in the *Technical* category in Knowledge Builder. Only one publication status component can be installed at a time.

Write license template generates a template whose content is to be completed for the component license configuration file to be used to generate the license key, and stores it at a location of your choice via a saving dialog (default file name: *[Knowledge Graph].componentLicenseTemplate.ini*). Irrespective of the configuration of the Knowledge Graph just administered, configuration placeholders are specified for the components *KEM* , *i-views core* and *Knowledge Builder* . The version number of the respective Knowledge Graph component supplied in the Admin tool is pre-entered in every configuration placeholder.

The **Knowledge Graph list** alphabetically lists all Knowledge Graph components installed in the Knowledge Graph with their respective version numbers. An installed Knowledge Graph component for which a newer version is provided in the Admin tool is highlighted in red. The optional *Knowledge Builder* component is pre-installed in a new Knowledge Graph by default.

The text fields **Name** and **Version** show the name and the three-digit version number of the installed Knowledge Graph component selected in the **Knowledge Graph list**.

Add generic component adds a generic model component or a generic software component to the **Knowledge Graph list**. The component type is selected in a separate window. Generic components allow bundling of project-specifically created Knowledge Graph extensions and simplify their installation (removal) and version monitoring via the Admin tool. The name and version number of a generic Knowledge Graph component installed in the Knowledge Graph can be freely assigned in the corresponding text fields.

Update (the name changes to **Renew**, if it can be deactivated) updates the installed Knowledge Graph component selected in the **Knowledge Graph** to the version supplied in the Admin tool. If the language of the currently running Admin tool differs from the language of the Admin tool with which the Knowledge Graph component was originally installed in the Knowledge Graph, identifiers of all elements and element types of this Knowledge Graph component are also updated. Depending on the Knowledge Graph component, the update of the old identifiers either adds new identifiers in the language of the Admin tool that is currently running (the respective applicable language version is then displayed depending on the language setting in Knowledge Builder) or replaces the old identifiers with new identifiers.

Remove removes the installed Knowledge Graph component selected in the **Knowledge Graph list**. If Knowledge Graph components in the installed status in Knowledge Builder have an entry in the *Technical* category, they leave their own subgraph after they have been removed, which has to be removed manually. Knowledge Graph components can only be removed if no other Knowledge Graph components that depend on the Knowledge Graph component to be removed are installed. The two Knowledge Graph components *i-views Core* and *View Configuration* offer basic functions and cannot be removed.

Boost libraries 1.18.0

This configuration menu appears only if the *boost libraries* Knowledge Graph component is installed.

With the exception of the blob service and the mediator, all the software components can interpret JavaScript. In order to improve the scope and speed of interpretation of regular expressions embedded in JavaScript, it is possible to transfer their interpretation to the Boost.Regex library. Under Windows and Linux, the library (file name in Windows: *boost_regex.dll* , file name in Linux: *libboost_regex.so*) must be in the same directory as the transferred software component. In Mac OS the library is integrated in the file of the transferring software component.

The *boost libraries* Knowledge Graph component makes it possible to ensure that access to the Boost.Regex library is possible.

If the **Boost libraries required for all incl. Admins** option is selected, all software components apart from the Admin tool can only access the Knowledge Graph if they can access the Boost.Regex library.

If the **Boost libraries required for all apart from Admins** option is selected, all software components apart from the Admin tool can only access the Knowledge Graph if they can access the Boost.Regex library. The only ones excepted from this access lock are users with administrator rights who access the Knowledge Graph via the Knowledge Builder.

If the **Boost libraries not required, logging only** option is selected, each software component enters a corresponding warning in its respective log file, if available, if it cannot access the Boost.Regex library during start-up. Access to the Knowledge Graph remains possible regardless.

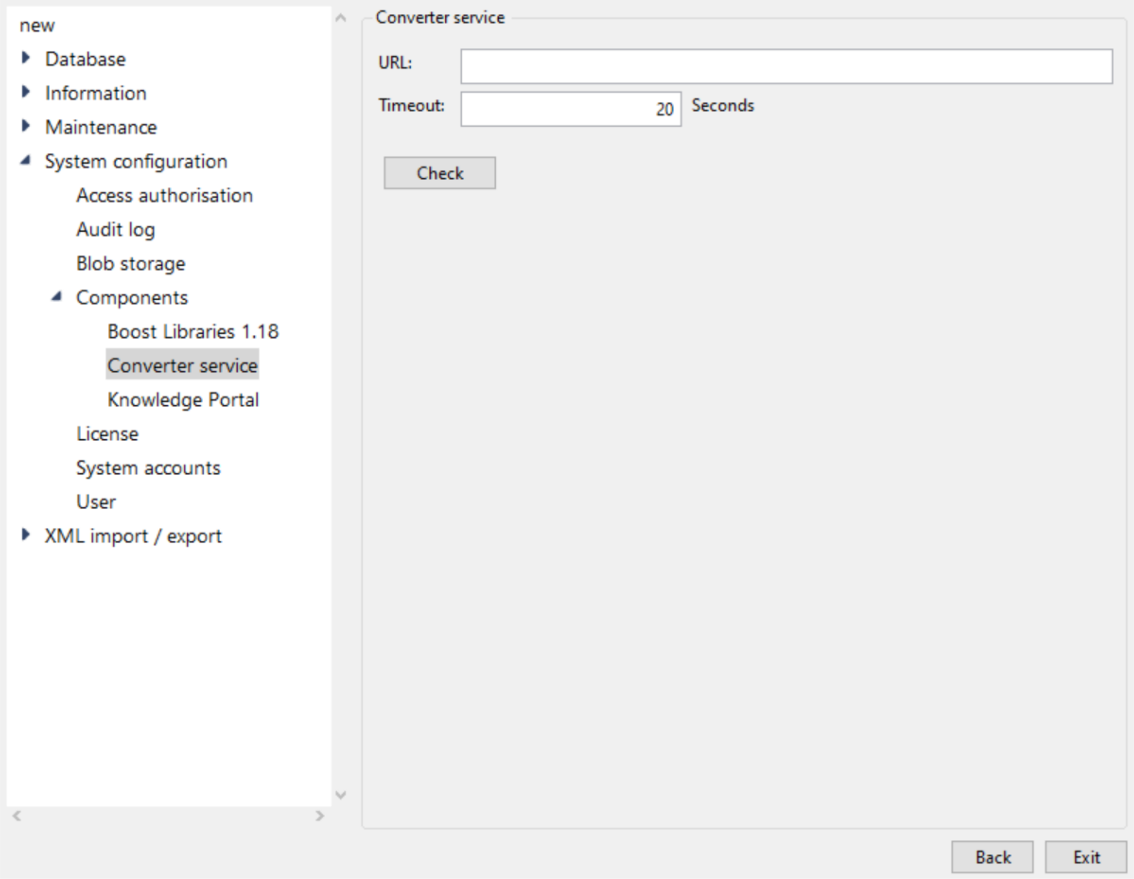
Converter service

This configuration menu appears only if the *print component* Knowledge Graph component is installed.

The *print component* allows selected Knowledge Graph elements to be integrated into an electronic document that can be saved. To do so, a document template in the formats ODT, DOCX or RTF must be imported into the Knowledge Graph using the Knowledge Builder and be linked to the Knowledge Graph element to be integrated into a document. This layout of this document template is created in an external Office program. You can use KScript and KPath to define placeholders to be filled out by elements of the Knowledge Graph.

The conversion service is a function of the print component. If the context menu item **Print** is used to generate a document in the Knowledge Builder, then along with the original format of the imported document template, diverse other output formats can be selected into which the document template can be converted. To ensure this conversion functions, a suitably configured bridge (a software service) must be started and be linked to the print component, and a version of LibreOffice or OpenOffice must be installed.

The bridge is suitably configured using its initialization file (default file name: *bridge.ini*). The value *jodService* must be added in the section *[KHTTPRestBridge]* under the key *services* . Moreover, a new section *[file-format-conversion]* must be created and be stored there using the key value pair *sofficePath="[File path]/soffice.exe"* with a correct path name for the location of the LibreOffice or OpenOffice start file.



URL:

Timeout: 20 Seconds

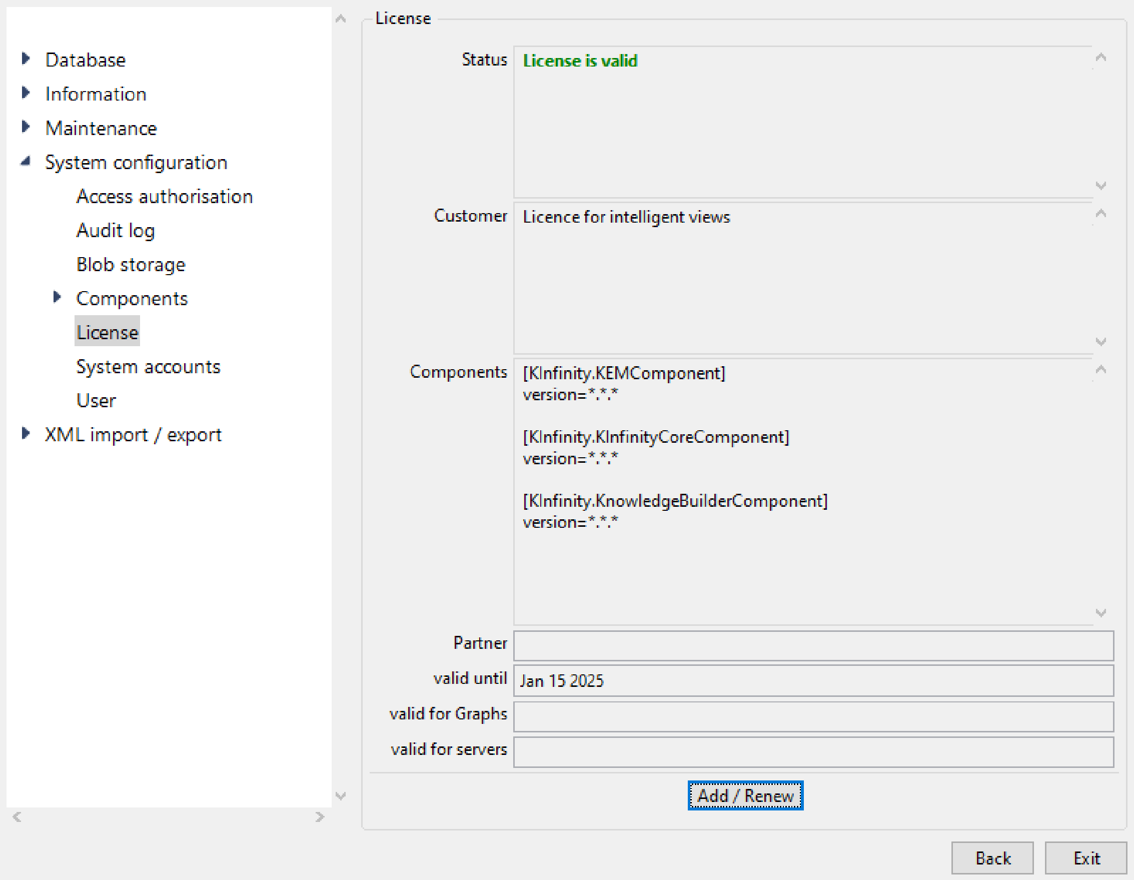
The bridge is linked to the print component using the free text field **URL**. The network address of the bridge is entered there in the format *http://[Bridge-IP-Number]:[Bridge-Port]/jodService/jodconverter/service* . The path section * /jodService/jodconverter/service* has historical reasons and activates the predefined jodService.

Check starts a test process. The test process uses REST to send a test document to the bridge defined using the network address and expects that a properly converted test document is returned. The test result is output in a separate window.

The free text field **Timeout** is used to define how many seconds to wait for the return of the converted test document before generating an error message. The preset is 20 seconds.

2.5.2.3.4. Licence

A Knowledge Graph must have a valid license so that Knowledge Builder and other software components (with the exception of the Admin tool) can work with it.



Status: License is valid

Customer: Licence for intelligent views

Components:

```
[KInfinity.KEMComponent]
version=*.*.*

[KInfinity.KInfinityCoreComponent]
version=*.*.*

[KInfinity.KnowledgeBuilderComponent]
version=*.*.*
```

Partner:

valid until: Jan 15 2025

valid for Graphs:

valid for servers:

The **Status** field specifies whether the license is currently valid or invalid. If it is invalid, a reason is also stated. Reasons for an invalid license can be exceedance of the validity date or maximum number of allowed registered users.

The **Customer** field describes the client for whom the license was issued. In addition to the name, address and department may also be listed.

The **Components** field displays the content of the component license configuration file *[Knowledge Graph].componentLicenseTemplate.ini* used to generate the license key. This specifies

- The licensed versions of individual components (*version*),
- The maximum number of registered users with administrator rights (*maxAdminUsers*) and
- The maximum number of registered users without administrator rights (*maxUsers*)

The **Partner** field contains the name of the partner via which the license is forwarded.

The **Valid to** field contains the date on which the license expires.

The **Valid for Knowledge Graphs** field contains a list of names of all Knowledge Graphs to which the license is restricted. This can be entered using a regular expression.

The **Valid for servers** field contains a list of all IP addresses and port numbers that can be used to reach a mediator connected to the Knowledge Graph.

The fields **Partner** , **Valid to** , **Valid for Knowledge Graphs** and **Valid for servers** can be left blank.

All fields have a context menu that can be activated by right-clicking.

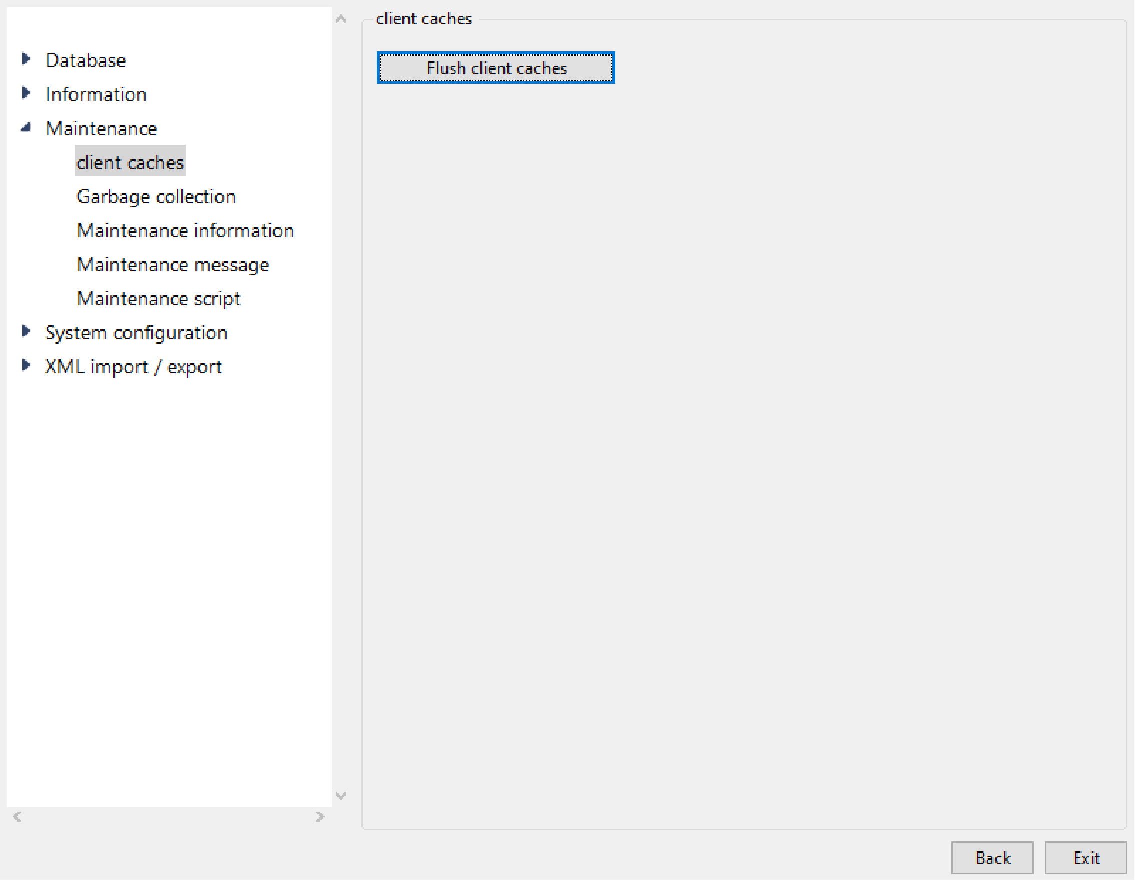
- **Select All** selects all the text. Alternatively, the mouse pointer can be used to mark any text segment.
- **Copy** copies the selected text area to the clipboard of the operating system.
- **Find Again** searches for the selected text area and finds its next occurrence in according to the read direction.
- **Find** allows a string to be input in a separate window, and its next occurrence in accordance with the read direction in relation to the position of the cursor set by clicking the mouse. The query is case-sensitive.

Add / Renew makes it possible to load a new license key (file name: *[License name].key*) via the file system of the operating system.

2.5.2.4. Maintenance

2.5.2.4.1. Client caches

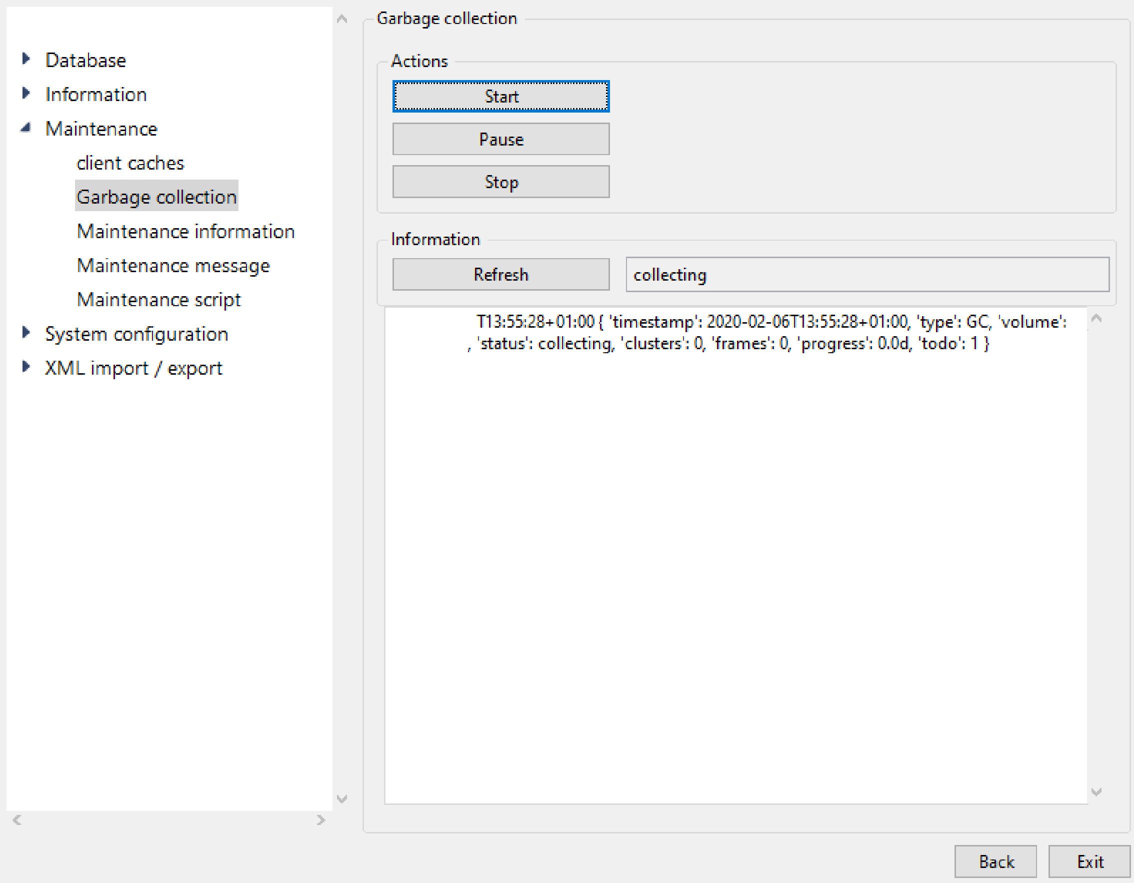
To improve performance, software components accessing the Knowledge Graph often fall back on their own buffers (cache). These buffer the schema and configuration data of the Knowledge Graph so they can access them more quickly if they need to use them later on.



Reset client caches deletes these buffered data. This makes sense if they are obsolete due to changes to the schema or the configuration. This operation requires that the Knowledge Graph is activated via a mediator.

2.5.2.4.2. Garbage Collection

Garbage collection is a procedure that deletes objects that are no longer referenced (according to a programming terminology reading) from the Knowledge Graph and thereby minimizes the memory usage of the Knowledge Graph. Use of the garbage collection requires that the Knowledge Graph that is to be cleaned up is activated via a mediator.



```
T13:55:28+01:00 { 'timestamp': 2020-02-06T13:55:28+01:00, 'type': GC, 'volume':
, 'status': collecting, 'clusters': 0, 'frames': 0, 'progress': 0.0d, 'todo': 1 }
```

Start launches a new garbage collection for the Knowledge Graph or continues a paused garbage collection. No confirmation is sent when the process is completed. You can determine its progress via the **Refresh** menu option.

Pause interrupts the execution of the active garbage collection for the Knowledge Graph.

Stop cancels the execution of the active garbage collection for the Knowledge Graph.

Refresh writes the current state of the garbage collection for the Knowledge Graph to the neighboring text field. If garbage collection is active, feedback on its progress is provided in percent.

2.5.2.4.3. Maintenance

Perform maintenance now checks

- the license (*license*)
- indexes (*indexes*),
- registered objects (*the registry*),
- rights (*access rights*),
- triggers (*trigger*) and
- installed Knowledge Graph components (*active components*)

for faults. Over the course of the check, the statistics for property frequencies per object (metrics) that can be viewed using the Knowledge Builder are updated.

Any faults found are collected in a **fault overview** in the form of a table. For each fault,

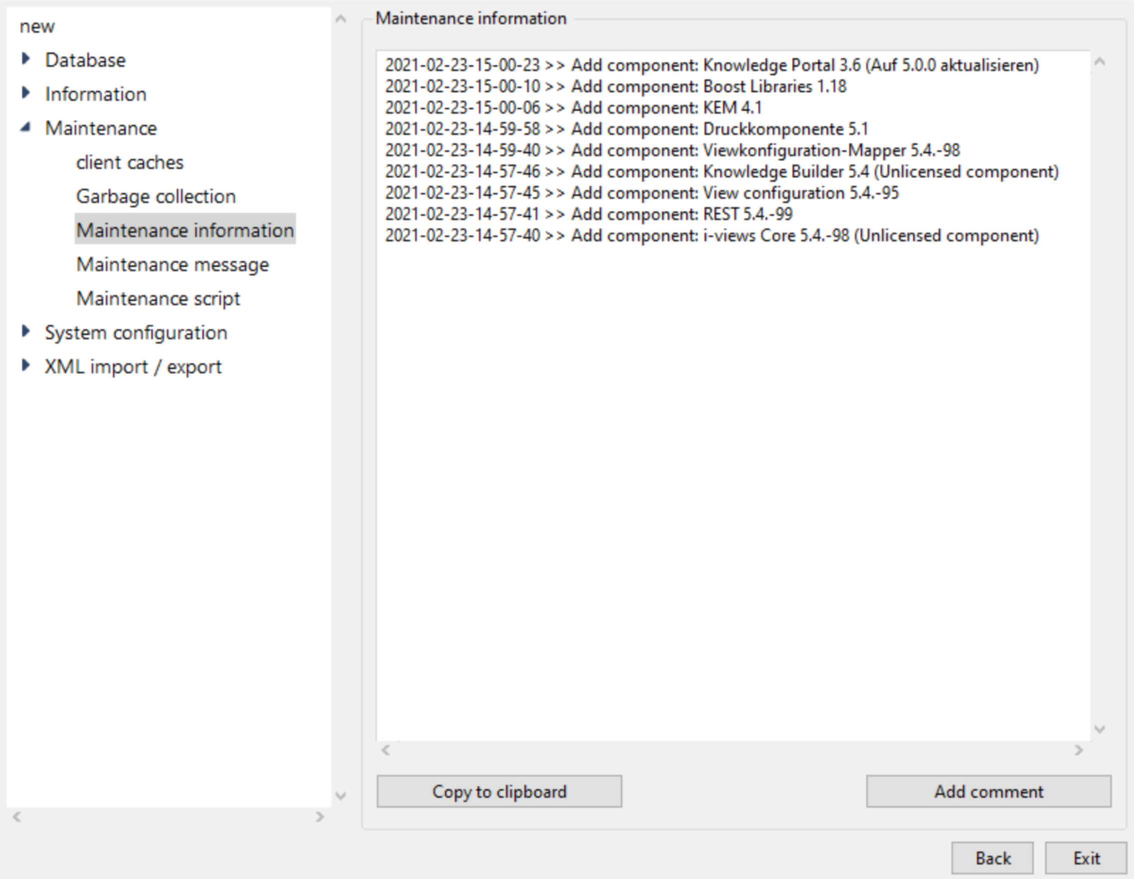
- a short description, if relevant including the cluster ID and the frame ID (format *cluster ID/frame ID*) of the faulty object (in the terminology interpreted by the program) (*notification*),
- the superordinate semantic element affected by the fault (*object*),
- its type (*type*),
- the severity of the fault (*priority*) and
- the first point in time at which it was identified in the form of a date (*date*)

are output. The individual columns of the table can be sorted by clicking on the head of the column.

Details displays all data listed in the **fault overview** of the selected fault in a new window. The time of the first point in time at which it was identified and date and time of the last time it was identified are added. The data there can be copied to the clipboard of the operating system (**Copy to clipboard** button) or be exported to any location as a text file that can be given any name using a saving dialog (**Save** button). The operation triggered using the **Details** button can, alternatively, be performed by double-clicking a fault in the fault overview.

Remove deletes the fault selected in the **fault overview**. This does not effect the first point in time at which the fault was identified.

2.5.2.4.4. Maintenance information



```
2021-02-23-15-00-23 >> Add component: Knowledge Portal 3.6 (Auf 5.0.0 aktualisieren)
2021-02-23-15-00-10 >> Add component: Boost Libraries 1.18
2021-02-23-15-00-06 >> Add component: KEM 4.1
2021-02-23-14-59-58 >> Add component: Druckkomponente 5.1
2021-02-23-14-59-40 >> Add component: Viewkonfiguration-Mapper 5.4.-98
2021-02-23-14-57-46 >> Add component: Knowledge Builder 5.4 (Unlicensed component)
2021-02-23-14-57-45 >> Add component: View configuration 5.4.-95
2021-02-23-14-57-41 >> Add component: REST 5.4.-99
2021-02-23-14-57-40 >> Add component: i-views Core 5.4.-98 (Unlicensed component)
```

This menu option can be used to call up a chronologically ordered **maintenance history** of all essential administration processes in the Knowledge Graph since its creation. It contains backup and transfer processes, component installations and updates, and the execution of maintenance scripts and garbage collection, each with the time and date.

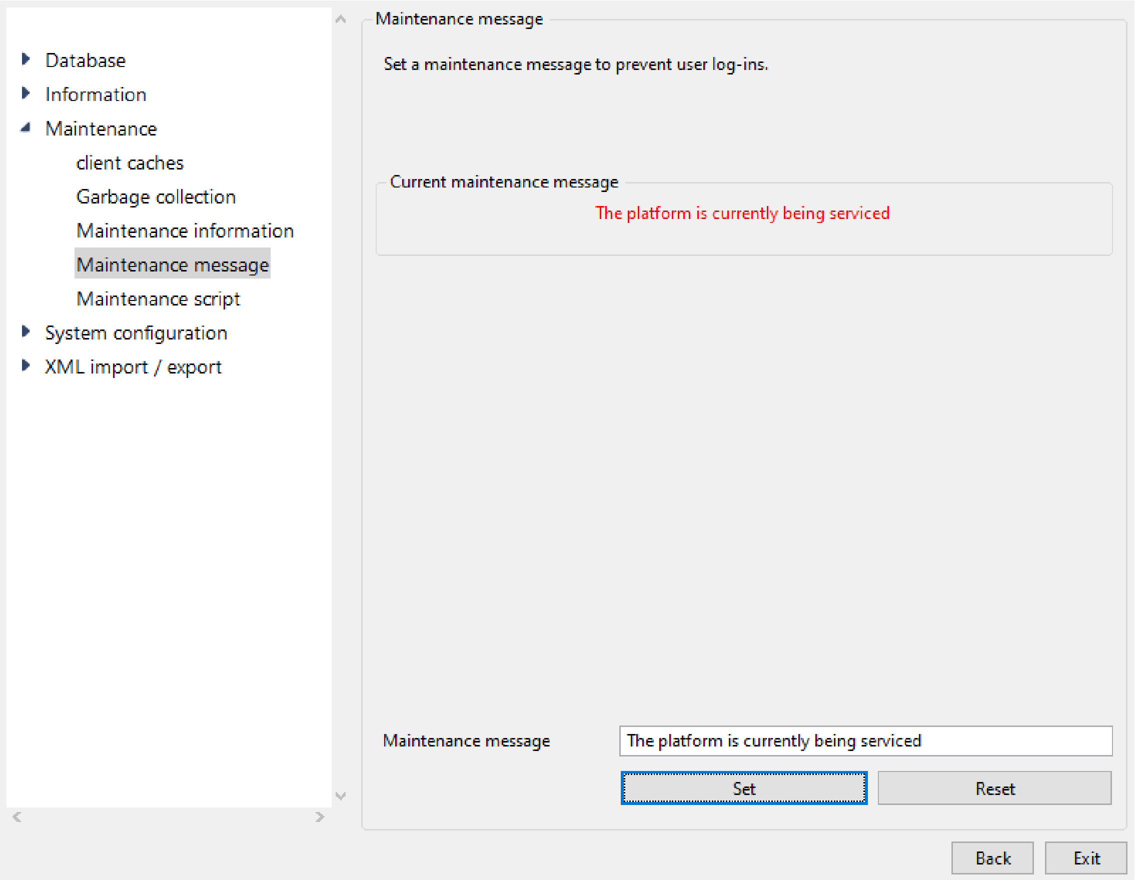
The **maintenance history** has a context menu that can be activated by right-clicking:

- **Select All** selects all the text. Alternatively, the mouse pointer can be used to mark any text segment.
- **Copy** copies the selected text area to the clipboard of the operating system.
- **Find Again** searches for the selected text area and finds its next occurrence in according to the read direction.
- **Find** allows a string to be input in a separate window, and its next occurrence in accordance with the read direction in relation to the position of the cursor set by clicking the mouse. The query is case-sensitive.

Copy to clipboard copies the entire **maintenance history** to the clipboard of the operating system.

Add comment allows a note to be entered via a free text field in a separate window. It is given a timestamp and added to the **maintenance history**. Notes added to the **maintenance history** cannot be deleted.

2.5.2.4.5. Maintenance message



The platform is currently being serviced

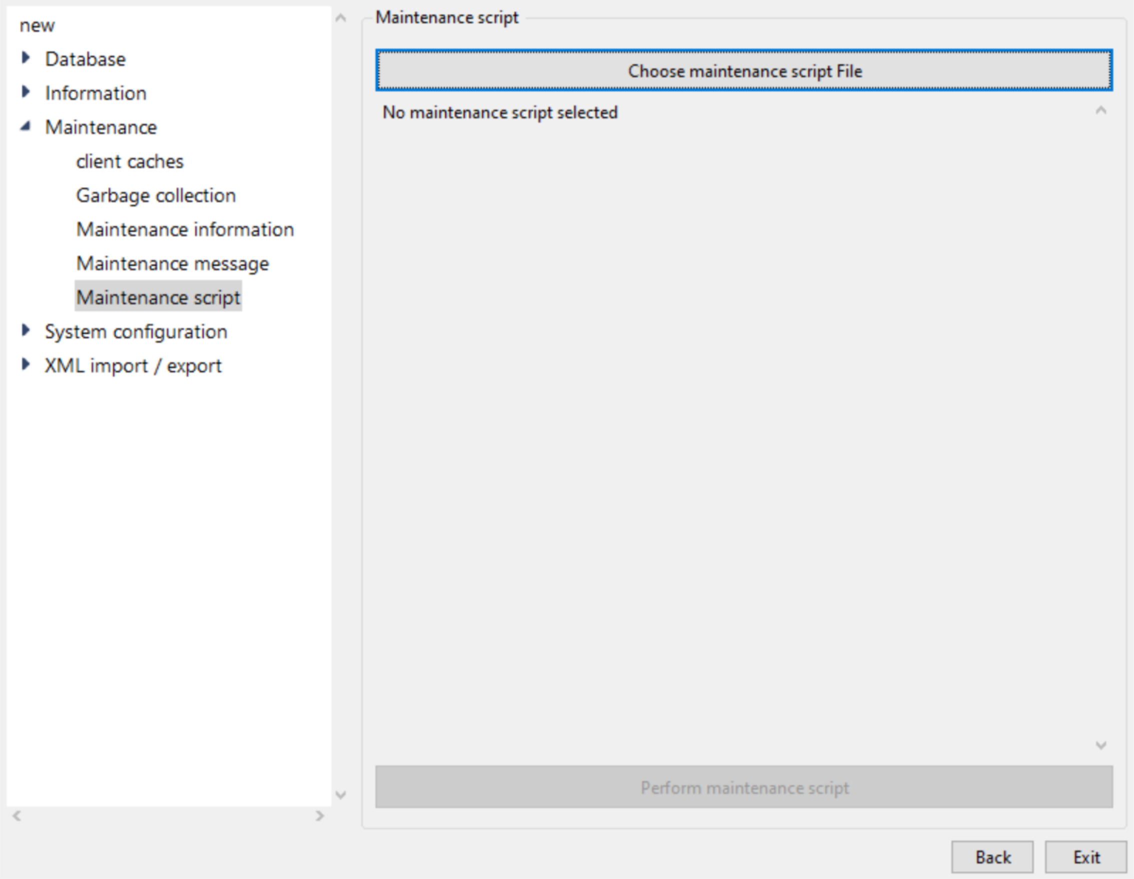
Maintenance message: The platform is currently being serviced

The **Set** button activates a maintenance block that prevents all users from accessing the Knowledge Graph via the Knowledge Builder. To do this, a maintenance notification must be written.

The maintenance notification is written in the free text field **Maintenance notification**. It is displayed as an error message shown to all users who try to access the Knowledge Graph via the Knowledge Builder when the maintenance block is active.

The **Reset** button removes the previously set maintenance block and deletes the maintenance notification.

2.5.2.4.6. Maintenance script



Select maintenance script can be used to access the file system of the operating system in order to load a maintenance script (file name: *[Maintenance script].kss*). Maintenance scripts are produced on a case-specific basis in the programming language Smalltalk and permit operations that cannot be implemented using the predefined functions of the Admin tool or using the KEM or JS interfaces.

If the maintenance script has a description, this description is output in an invisible text field under the **Select maintenance script** button after the maintenance script has been loaded. This text field has a context menu that can be activated by right-clicking:

- **Select All** selects all the text. Alternatively, the mouse pointer can be used to mark any text segment.
- **Copy** copies the selected text area to the clipboard of the operating system.
- **Find Again** searches for the selected text area and finds its next occurrence in according to the read direction.
- **Find** allows a string to be input in a separate window, and its next occurrence in accordance with the read direction in relation to the position of the cursor set by clicking the mouse. The query is case-sensitive.

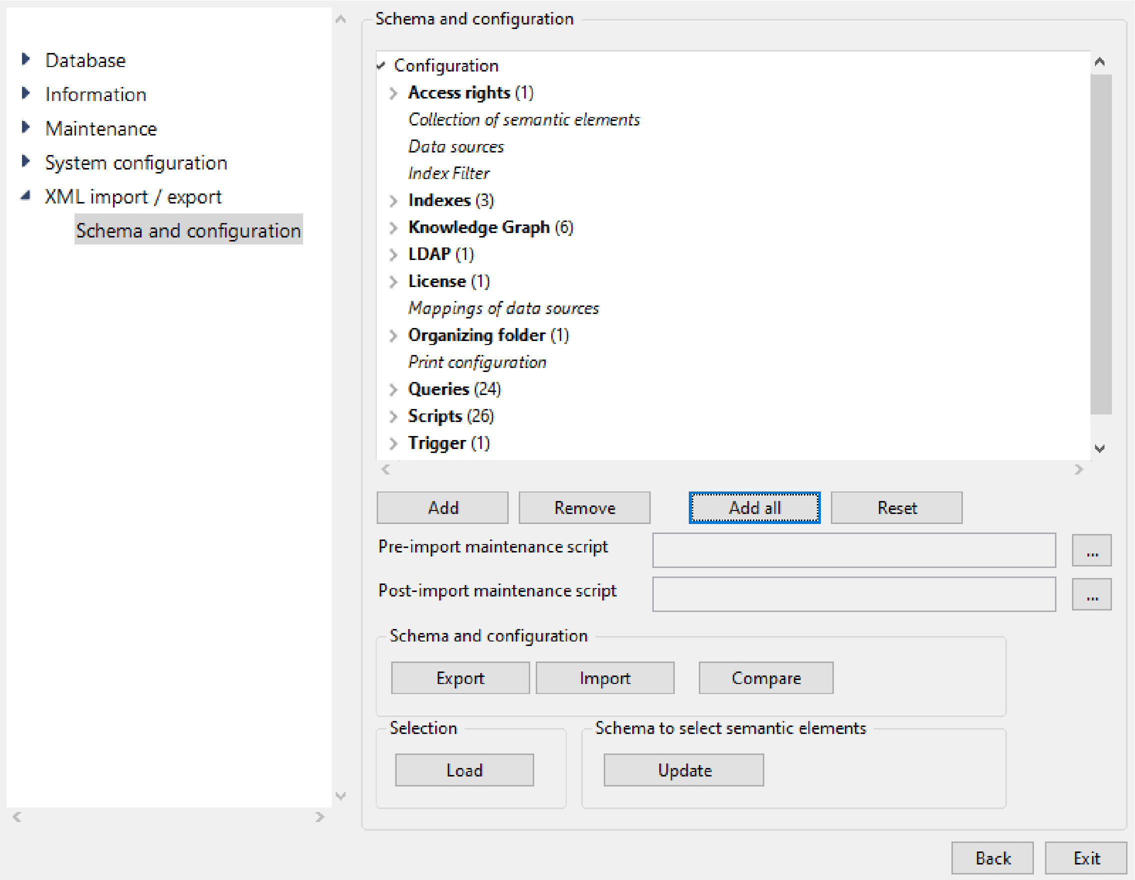
Execute maintenance script starts the maintenance script. A separate window tells you when the maintenance script was executed and, depending on the script, offers additional execution information or permits script-specific execution options.

2.5.2.5. XML import/export

2.5.2.5.1. Schema and configuration

Along with subgraphs generated by the user and imported using components (schemas with useful data), a Knowledge Graph, by extension, is also comprised of diverse other modules (configurations) that extend, configure or work with this subgraph in functional terms. Schemas and configurations are referred to jointly as configurations within the context of this menu item.

Numerous configurations of a Knowledge Graph can be systematically exported and imported.



Preparation of schema for object transfer

For transfer of specific semantic elements - especially instances (individual objects, attributes and relations of respective types) - and for controlling the export and import behavior, preconfigured XML attributes are required.

Preparation of XML attributes

To generate the XML attribute types, the " **Update** " option of the **Admin tool** adds the Boolean attribute types to the Knowledge Graph (if they do not yet exist in it) as follows:

- *XMl-Schematransfer: Export all objects*

- *XML-Schematransfer: Export direct objects*
- *XML-Schematransfer: Do not overwrite*
- *XML-Schematransfer: Do not export type and all subtypes*
- *XML-Schematransfer: Do not export subtypes*

These attribute types are required to select which elements and element types of the configuration type Knowledge Graph found in this configuration should be exported or not. To do so, these attribute types are attached to suitable object types using the Knowledge Builder and are given suitable attribute values.

If nothing has been configured using these attribute values, then the export applies for every object type, but not its objects. If an object or object type is exported, all attributes and relations directly connected to it and their attribute or relation types respectively are also exported.

The **configuration overview** is a list providing an overview of all configuration types in a Knowledge Graph that can, in principle, be transferred by means of the operations described in the following. Able to be transferred by principle are

- individual, registered mappings of data sources (*mappings of data sources*)
- individual search fields configured by administrators and are user-defined (*queries*)
- individual data source access settings for use for mappings of data sources (*data sources*)
- the print configuration (*print configuration*)
- the set of all modules defined within the category *Determination* of view configuration (*view configuration determination*)
- individual index filters (*index filter*)
- individual index configurations (*indexes*)
- the LDAP authentication (*LDAP*)
- the license for the Knowledge Graph (*license*)
- individual, registered collections of semantic objects (*collection of semantic elements*)
- individual, registered scripts (*scripts*)
- the working folder (*organizing folder*)
- the set of all modules defined within the triggers category (*Triggers*)
- individual subgraphs (*Knowledge Graph*) and
- the set of all modules defined within the rights category (*access rights*).

Display

The **configuration overview** also manages all configurations specifically intended for export. Configurations intended for export appear as an expandable list of subitems of their respective configuration types. If these configurations require other configurations for successful export, these other configurations are, in turn, listed in the form of an expandable list of subitems of the respective configurations. Configuration types without their own configurations are marked in italics, configuration types with their own configurations are marked in bold and show the number of configurations assigned to them in brackets. Configuration und configurations of each configuration type are sorted in alphabetical order respectively.

Navigation

Expanding and collapsing lists of subitems in the **configuration overview** is carried out by clicking on the triangle symbols to the left of the listed items. Alternatively, this can be implemented using a context menu, which can be accessed by right-clicking a list item:

- **Expand** opens all directly listed subitems in the list item selected.
- **Expand fully** opens all directly and indirectly listed subitems in the list item selected.
- **Contract fully** collapses all listed subitems in the list item selected.

Adding/removing configurations

Add adds a configuration of the configuration type selected there to the **configuration overview** . If more than one configuration exists in the Knowledge Graph for the configuration type selected, then a selection option follows in a separate window. Selection is carried out there by either clicking individually on the respective configurations in a list, or collectively by using the **Select/deselect all** button.

Remove either deletes all configurations of the configuration type selected in the **configuration overview** or the configuration selected in the **configuration overview** .

Add all adds all configurations existing in the Knowledge Graph to the **configuration overview** and distributes them among the respective suitable configuration types.

Maintenance scripts

The **...** buttons can be used to access the file system of the operating system in order to load a maintenance script (file name: *[Maintenance script].kss*). Maintenance scripts are produced on a case-specific basis in the programming language Smalltalk and permit operations that cannot be implemented using the predefined functions of the Admin tool or using the KEM or JS interfaces.

If a maintenance script loads, the file name of the maintenance script selected appears in the text field positioned to the left of the respective button. If configurations are imported afterwards, then the maintenance script is executed. If configurations are exported afterwards, the maintenance script is also exported and only executed when these configurations are imported. The exact time of execution of the maintenance script in relation to the import process depends on which of the two **...** buttons was used to load it. It is either before the import process starts, or after the import process finishes.

Export and import

Export exports the configuration selected in the **configuration overview** . An export as one single archive file in the archive format *tar* or as individual files in a folder can be selected. The export method is selected in a separate window:

- The free text fields **File** or **Directory** can be used to specify the name of the archive file (file name: *[Knowledge Graph].tar*) or the folder respectively (no default name). The archive file or the folder respectively is created in the same folder as the Admin tool. Alternatively, **Select** can be used to open a saving dialog to define any name and location used to save the archive file or the folder respectively.

Import imports configurations to the Knowledge Graph after confirming a prompt. An import from one single archive file in the archive format *tar* or from individual files in a folder can be selected. The import method is selected in a separate window:

- The free text fields **File** or **Directory** can be used to specify the name of the archive file (file name: *[Knowledge Graph].tar*) or the folder respectively (no default name). The archive file or the folder respectively is searched for in the same folder as the Admin tool. Alternatively, **Select** can be used to access the file system of the operating system to select an archive file or a folder respectively from any location.
- If the archive file or the folder to be imported respectively is selected, an overview of the configurations it contains appears in an additional window. This overview can be copied to the clipboard of the operating system (**Copy to clipboard** button) or be exported to any location as a text file that can be given any name using a saving dialog (**Save** button). The **Import** button starts the import process. The window also features a context menu of its own, which can be opened by right-clicking:
    - **Search** allows a string to be input in a separate window, and next appears in accordance with the read direction in relation to the position of the cursor set by clicking the mouse. The query is case-sensitive.
    - **Mark all** marks the entire text. Alternatively, the mouse pointer can be used to mark any text segment.
    - **Copy** copies the selected text area to the clipboard of the operating system.

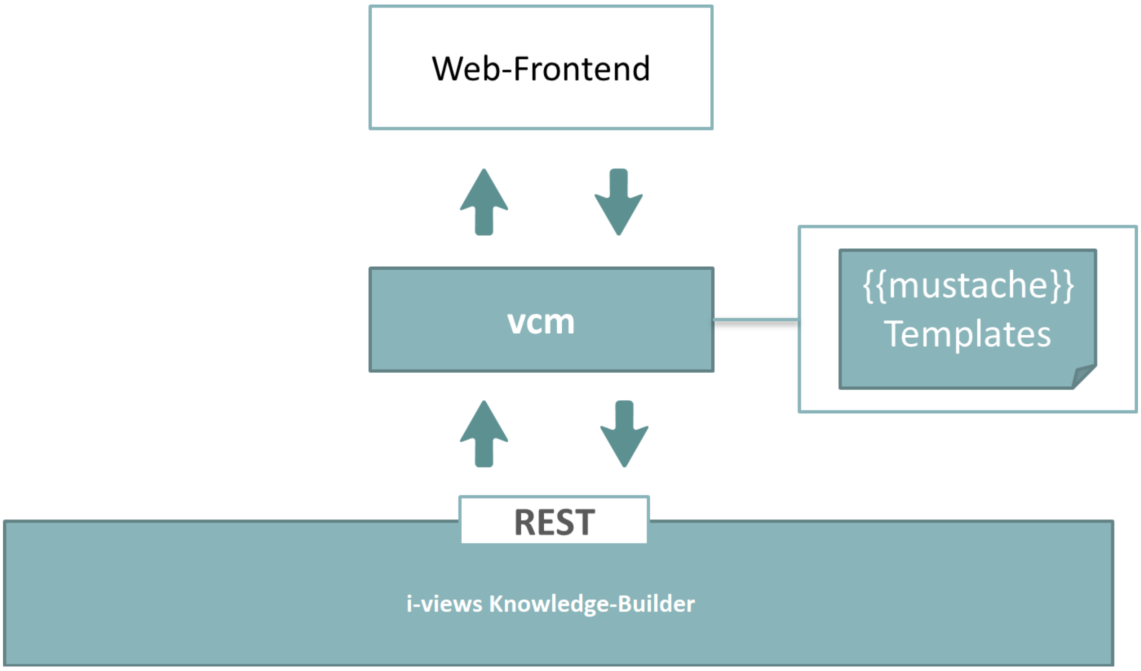
Save saves the configurations currently selected in the configuration overview for this Knowledge Graph as an XML file. A saving dialog is used to define a name and the location of the XML file (default file name: *instruction.xml*).

Load accesses the file system of the operating system to load a previously saved selection of configurations for this Knowledge Graph from an XML file (default file name: *instruction.xml*).

3. View Configuration Mapper

3.1. Introduction

View configurations can be transported into a web front-end and be displayed here in a straightforward way using the ViewConfiguration Mapper (VCM for short). To do so, the JSON generated in the view configuration is transported to the front-end via the REST interface in i-views and is translated into HTML there using mustache templates.



In addition, standard interactions such as content maintenance are supported directly, and the option is provided to execute user-specific actions in the front-end that were defined in the view configuration using VCM.

The ViewConfiguration Mapper is a single-page application that runs in the client’s browser. It uses ractive (ractive.js.org) for an interactive and reactive application that is based on mustache templates. (mustache.github.io/) .

3.2. Interaction patterns

When creating user interfaces with the i-views web GUI framework, you will have to deal with at least two different major design aspects: **static** and **dynamic** behaviour.

Static behaviour describes the way in which elements of the Knowledge Graph are displayed, how they are ordered and filtered, mapped to widgets, arranged on the page and the like. Defining static behaviour requires good domain knowledge as well as graphic designer’s skills.

Dynamic behaviour on the other hand side is closer to the work of a programmer as it describes the flow of interaction, data manipulation, handling of state, refresh of display areas and so forth. Describing dynamic behaviour often requires programming (i.e. scripting in JavaScript) and is more difficult to capture. Usually an application developer must browse through several scripts and configuration settings to understand the dynamic behaviour of an application.

Interaction patterns help to cope with the task of designing the dynamics of an application. At the same time, they help users in understanding the behaviour of an application by providing well known mechanisms which re-appear in many other applications.

Well known patterns include for example:

- navigation bar
- shopping cart
- wizards
- simple search
- etc.

This guide is not meant to be a comprehensive list of interaction patterns – such collections can be found in literature. Though, we would like to show how selected patterns can be realized using the i-views web GUI.

In the first part of this guide, we present those components that take part in the dynamic behaviour – either by controlling interaction flow or by being influenced or controlled.

In the second part we discuss application state.

In the third part we show how selected patterns can be implemented with the i-views web GUI framework.

3.2.1. Building blocks of dynamic behavior

3.2.1.1. Panels

Panels and views are mainly elements of static behaviour. As panel contents and visibility may change over time, panels are frequently part of dynamic application behaviour as well.

First, panel contents depend on the type of panel: **Layout panels** contain other panels whereas **view panels** contain views – either statically or dynamically determined.

All types of panels may carry one specific " **domain model** " at a given time. The **domain model** may be an element of the Knowledge Graph, a list of elements, a (parametrized) search definition, or search parameters. Panel contents (= domain model) are determined according to one of the following cases:

<table>
<tr><th>Possible case</th><th colspan="2">Additional option</th><th colspan="2">Additional option</th></tr>
<tr><td><b>a)</b> Resulting from an action (see chapter below) ("Show result in panel" or action within the same panel)</td><td rowspan="2">+</td><td rowspan="2">e) Optional: computed by a script (" <b>Script for target model</b> ") in addition to a) or b)</td><td rowspan="3">(+)</td><td rowspan="3">f) Optional, but not recommended in the first place: computed by a script (" <b>Script for context element</b> ") in addition to e) or c)</td></tr>
<tr><td><b>b)</b> Passed through on panel dependency activation ("influences")</td></tr>
<tr><td colspan="3"><b>c)</b> Passed through on panel activation cascade (see section below)</td></tr>
<tr><td colspan="3" rowspan="2"><b>d)</b> No action or activation (contents may be determined inherently by panel (sub-)configuration, e.g. 'Search' or 'Graph')</td><td colspan="2"><b>Alternative option</b></td></tr>
<tr><td colspan="2">g) Optional, but not recommended in the first place:
a configured fixed element of the knowledge graph ("context element")</td></tr>
</table>

Panels exist in the two states: **visible** (or active) and **invisible** (or inactive). The state of a panel can be changed by activating or deactivating the panel. This process is initially triggered by an action (see chapter below). After that, a cascade of further activations and deactivations is conducted depending on panel structure and configured dependencies.

The following rules apply with respect to panel activation:

- **Rule A** ("static activation"): The **main window panel** of the application is always active when an application starts (for the web frontend: application = view config mapper (VCM))
- **Rule B** ("action activation"): The **execution location** (location at which an action is triggered by a user, e. g. by clicking on a button or onto a table row) determines **which panel becomes active** when the action is executed

Based on A/B, there are subsequent activations based on these rules:

1. Influenced panels are activated (e.g. by relation "influences")
2. Panels with a specialized function (e.g. window title) are activated automatically by their

superordinate panel in the corresponding hierarchy (e.g. main panel or dialog panel)

3. Subpanels are activated
4. In the case of a panel with a changing layout: Sister panels of the active subpanel are deactivated
5. Continue with 1. until no further panels can be activated (an integrated cycle test prevents endless loops)
6. Make sure that all parent panels of activated panels are activated as well

Subsequent activations of step 1 - 3 pass the **domain model** (context) from one panel to the next. If, for example, panel A shows the element "Mr. Meyer", then the activated subpanel B also shows "Mr. Meyer". This default behaviour may be altered according to panel content rules (using scripts or a fix context element; see cases e), f) or g) above).

The so-called " **Activation mode** " can be used to optimize the calculation of the panel contents in step **B** (action activation) and in step **1** (influencing). This avoids the recalculation of panel contents that are currently not displayed despite activation, because they are not visible (e.g. a shopping basket). The available activation modes are as follows:

- The option " **Refresh model and view** " updates the panel contents only if panel is active
- The option " **Refresh view only** " updates the view contents (friends of Mr Miller), keeps view state (table page 4) and domain model (Mr Miller)
- The option " **Default** " is the fallback setting when neither of the other options described above were selected (update the panel contents and activate the panel)

3.2.1.2. Actions

Actions are the main driver for **dynamic behaviour** . They are triggered by **user interaction** in the web frontend, i.e. whenever the user activates a **button** , menu item, or hyperlink. Actions may change the state of the Knowledge Graph or they purely are of navigational nature - thus changing only application states (e. g. current visibility of panels, user selections etc.).

The action definition (= configuration of the action) comprises the direct effect of the action as well as the changes in display contents thereafter. The action effect depends on the selected action type, parameters, and action target (panels) which will be determined on run-time.

Changes in display contents as consequence of an action are more complicated to understand. The following rules apply:

1. The panels configured as panels to be activated ("show result in panel") are activated with the **domain model returned as action result** – optionally modified by a script ("script for target model") and optionally disabled by another script ("script for activation").
2. If the former rule does not apply, the **panel configuration** containing the action may **determine the panel to be activated** ("show action results in panel"). This configuration is inherited along the parent-child structure of the panels.

3. If the former two rules do not apply, the **panel containing the action** is activated.
4. In addition, panels to be activated or deactivated can be set by the **action script** using functions "actionResult.addActivation()"and "actionResult.setCurrentPanel()"

After applying these rules, subsequent activations will be conducted according to the activation rules in the panel section above.

3.2.1.3. Scripted actions

Action configurations with user-defined action scripts offer the broadest range of possible action behaviour.

The i-views JavaScript-API allows full access to and modification of the Knowledge Graph – considering the user’s access rights restrictions, of course. Additionally, the **current state** of the application can be accessed and modified as well as the **current view, session, user, and panel** are available to the action script. The following parameters or functions are provided:

- **Action:** the current action is the first parameter of the action script
- **View:** the view is the "this" object of the action script
- **Session:** can be accessed by "action.session()" or as shown below in the section about sessions
- **Panel:** can be accessed by "this.panelView()" – the panel is only available to the action if the configuration option "panel contents required" is selected

3.2.1.4. Actions and views

Usually, the receiver of an action is the view the action is attached to. This is the case for all actions that are configured as member of a menu of a view and for special actions directly configured for the view, e.g. the “click action” of a table. When a menu is configured “stand-alone”, e.g. as a **navigation bar** , all actions of the menu have their own view, which is of type “ActionView”.

3.2.1.5. Built-in actions

Built-in actions are executed whenever no (custom) action script is present. The **action type** ("action type") determines what will happen on action execution and what the result domain model of the action will look like. Built-in actions are usually specialized to specific views and require correct parametrization.

Action type " **save** " deals with form data from edit views, writing data back to the Knowledge Graph. The web frontend will automatically detect the corresponding edit view to a given "save" action if there is only one edit view visible. If you have more than one edit view visible at the same time, use "view roles" to link an action to a corresponding view.

NOTE An action can handle only one view role, whereas a view can be related to different view roles.

Action type " **read** " has the same effect as no action type and the same effect as an empty action script – it does nothing and the result domain model is the current domain model of the view.

Action type " **select** " has the same effect as action type " **read** " but the resulting domain model is the element specified by the parameter " **selectedElement** " (set by the web frontend).

3.2.1.6. Transactions / Action Sequences

Actions modifying the Knowledge Graph automatically run a transaction ensuring a consistent all-or-nothing modification. However, there are situations in which changes that the user makes to the Knowledge Graph are split into a sequence of consecutive actions – especially when user interaction is necessary to determine *further action parametrization* or to abort of the process so far.

Example: A new object needs to be created within a dialog. To allow the user aborting the creation of the new object by pressing a cancel button of the dialog, the dialog invoking action starts a transaction, the cancel button cancels the transaction (action type: "Cancel") and a save button commits the transaction.

In order to encapsulate a sequence of actions into one transaction, you mark the first action with "Transaction - **begin** " and the final action with "Transaction - **commit** ".

Caution:

- **Risk of data loss caused by never-ending transaction.** When actions are configured with "transaction - begin" only, a single never-ending transaction will be created. A never-ending transaction has the potential to grow continuously once started, recapturing all actions since the begin of transaction, until the system first becomes slower and then breaks down completely. Additionally, changes never will be saved since saving is triggered at the end of a transaction only in order to keep up consistency.

To avoid these effects, make sure to set a " **transaction - commit** " as soon as the sequence of actions is complete enough for achieving the required state.

Only set an action to "transaction - begin" if you also set a subsequent action to " **transaction - commit** ".

- **Risk of losing data integrity in case of repetitive transaction execution**

Transactions must not be executed on elements with variable order.

The transaction history of a transaction records which action is executed on which semantic element of a particular order. When executing a further action within a transaction, the transaction history with its fixed order of actions is repeated and the new action is supplemented to that history.

If the order of semantic elements varies when the transaction is repeated (e. g. by creating elements by means of a script or by determining elements executing a query), this results into a misalignment of the order of actions to their respective element of a particular order, leading to actions being processed on wrong elements.

When processing semantic elements in a transaction sequence, therefore make sure that the order of elements keeps deterministic. To keep a deterministic order, the elements need to be sorted after a fixed property.

Example: An action **A** executes a query for meals. The search result is “Pudding” and “Fish”. The action additionally creates two rating objects and links the ratings to “Pudding” and “Fish”. The user is enabled to write a comment (attribute) on each rating. The next action **B** saves the comments at the rating objects and ends the transaction. The execution of action is as follows: **A** , **A** + **B** . Action **A** therefore is executed twice. An important aspect is the deterministic order of found meals: since the result of a query has no specific order, the assignment of the first comment to “Pudding” or to “Fish” happens by coincidence. Therefore, the query result needs to be assorted first to ensure a correct assignment of first and second rating object to the relevant meal. Only this makes it possible to prevent the pudding from receiving the rating “Very tender, but too many fish bones“.

The transaction commit can also be brought about dynamically via the " **setTransactionCommit()** " script function.

If the transaction is to be cancelled, you can achieve this by means of an action of the " **Cancel** " type. Cancelling means that all previous changes to the Knowledge Graph conducted within the transaction are undone. The " **setFailed()** " script function can be used to dynamically initiate a cancellation.

As a transaction is always coupled with the duration of a session (see below), a transaction is cancelled automatically when the session ends in which the transaction was started.

If, for example, you open a dialog at the start of the transaction and the dialog is closed before the transaction was completed, the transaction is cancelled automatically. This does not apply to dialogs that are opened while a transaction is already running: opening a dialog creates a new session on the session stack which is independent from the currently running transaction. Dialog sequences (one dialog is closed, and another dialog is opened immediately) do not interrupt the transaction either.

NOTE	Only one transaction can be processed at once. A transaction within another transaction is not supported.

3.2.1.7. Recall actions

Sometimes an action needs to start a sequence of actions and after the last action in the sequence wants to come back to the original context for finalization. This mechanism can - but does not have to - be combined with a long-running transaction as described above.

The desired behaviour can be achieved by configuring a **recall script** ("Script (recall)") which is activated when calling the function "action.recallMarkedAction()" in the last action of the sequence. The recall script is then executed with the same environment (view, action, parameters) present when the action was first executed.

The environment necessary to run a recall script is stored on the current session and will thus be dropped on session end. The function "action.dropMarkedAction()" allows removing the environment from the session in the case that the whole sequence of actions shall be aborted.

3.2.1.8. Sessions

In sense of the view configuration, a **session** serves as temporary storage for variable values which can be read from and written to within scripts of the view configuration. This in turn allows representing the current state of an application.

Sessions are run-time objects, instantiated while running an i-views web application. Sessions form a stack. The first session lasts the entire duration of the web session; that is from the time the application is called until the respective browser window is closed. You can always call up the first session by using function " **$k.Session.main()** ".

Opening a dialog generates a new session on the stack. The closing of the dialog removes the corresponding session from the stack again.

The activation of panels, which are marked with a " **Session boundary** ", also generates a new session on the stack which lasts until the panel is deactivated. The element of the new session is set to the current element of the panel and can be used in the future by using the " **element()** " function on this session.

Use the function " **$k.Session.actual()** " to access the top session of the stack.

Values are written to a session variable by means of " **$k.Session.actual().setVariable()** " and are read from a session variable by means of " **$k.Session.actual().getVariable()** ".

3.2.2. Application state

The application state comprises the activation states of the following:

- panels
- panel contents
- session stack
- session variables

Actions allow application designers to change application states. Unfortunately, as explicated above, there are numerous options and parameters that influence especially panel activation and contents.

As a result, the desired effect is often not achieved or is spoiled by unwanted side effects. To make applications’ dynamic behaviour simpler to understand and maintain, it is therefore necessary to use clear, modular building blocks keeping action effects as local as possible.

Here, the session stack together with session variables plays an important role by providing a local, temporary context to such a building block.

System architecture considerations

There are two main players in the i-views web GUI framework: a JavaScript application running in the web browser and the i-view REST interface running at server-side.

As the REST interface is stateless by design, the **application state** resides completely in the **front-end** (web browser).

At the same time, **application logic** (static and dynamic behaviour) is exclusively available in the **back-end** (Knowledge Graph) and applied when calling the REST API.

As a result, all necessary application state must be provided by the front-end when calling the REST API. Usually this is done automatically by the framework. For example, the **session stack** is always being provided and is thus available to back-end scripts.

For efficiency reasons, *only the state of the view* an action is attached to will be provided when the action is executed. Sometimes this is not sufficient and configuration options like " **panel contents needed** " have to be set.

3.2.3. Interaction patterns and recipes

For the needed information about usage and rationals of interaction patterns for your user interface, see ui-patterns.com first. The website describes the needed patterns, whereas the implementation of the very solution is supplemented in here.

The following subchapters show recipes on how to implement the i-views specific solution of certain patterns for user interfaces using the view configuration mapper.

3.2.3.1. Navigation bar

Similar to the visualization of an *Alternative* view, panel tabs with a navigation bar can be used. The advantage of panel tabs with navigation bar in comparison to the alternative: panel tabs can contain further sub panels, allowing configuration of more specific layouts as well as using all the panel related functions a view doesn't come with (view models, session boundaries, interaction etc.).

In order to configure panel tabs with a navigation bar, proceed as follows:

1. Configuration of panel structure:
    - Create a panel containing a menu residing on top or at the side of the screen.
    - Depending on the intended layout of the navigation bar menu, select menu type *Tool bar* for horizontal layout or *List* for vertical layout.
    - Create a button for each section to be displayed.
    - Create another panel of type *Switching layout* that covers the remaining part of the display area.
    - Create a sub-panel of this panel for each section and mark each panel as *Session boundary*

(checkbox set to true).

2. Linking action to panel:
    - Link each button (= action) to the corresponding section panel using the relation *Show result in panel*. This causes a panel to be activated when its button is pressed.
    - Link each button (= action) to the panel of the menu in which the action itself is located in. This causes an update of the button styling when the button is pressed.
3. Creating style for action:
    - For a better usability, create a *Style* `buttonActive` that gives a visual indication of the button selection. First create the style at one action and then reuse (assign) the style to the other actions as well.
    - Add the following script at *class (script)* to each button:

```
function additionalPropertyValue(element) {
    const isActive = isActiveForSession($k.Session.actual(), this
.getPanelsToActivate())
    return isActive ? "yourButtonClass buttonActive" :
"yourButtonClass"
}

function isActiveForSession(session, panelsToActivate) {
    const sessionBoundaryActivatedConfig = session
.panelConfiguration()
    let isActive = panelsToActivate.indexOf
(sessionBoundaryActivatedConfig) > -1
    if(!isActive && session.parent()) {
        isActive = isActiveForSession(session.parent(),
panelsToActivate)
    }
    return isActive
}
```

    Replace `yourButtonClass` by the name of the class that is intended to be used for the button.

4. Defining the CSS class for the style:
    - For the style, add a class for the active button to the Options Resource of the REST service of the "viewconfig" application.

      To do so, use the organizer in the Knowledge Builder to navigate to *TECHNICAL* > *REST*. In the *REST Service* object list on the right side, select the service with the ID "viewconfig". In the detail editor of the service, select "vcm/options" and edit the entry for *CSS*.

Let’s assume a class `.navigation-button` is already in use for the buttons. Then a further class `.navigation-button.buttonActive` is needed for styling of the active state of the button:

```
.navigation-button {width: 200px;}
.navigation-button.butttonActive {background-color: red
!important;}
```

5. Refresh interfaces of REST and VCM:

   - Update REST-Service and ViewConfig and reload the web frontend (since panels have been created).

     Result: When clicking on a button, the representative panel is shown and the buttons is styled with an active style.

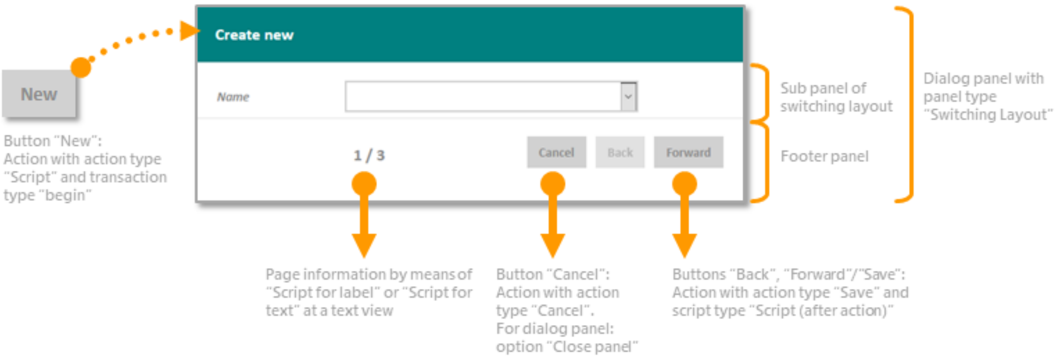
3.2.3.2. Dialog (modal)

Configure a dialog panel. Make sure the panel has proper title and a menu with a button labelled "X" to close the dialog. Check the *Close panel* option for the close action. Configure the body part of the dialog as desired. Optionally, configure a footer panel with menu buttons "Ok" and/or "Cancel" – both with *Close panel* option checked as above.

Finally, configure an action to open the dialog. Connect the action to the dialog panel using the relation *Show result in panel*. Make sure that the action result is the domain model you want to present in the dialog.

3.2.3.3. Assistent

A wizard is used to guide the user throughout an input process that contains several steps. The following wizard example configuration is as follows:



Button "New": Action with action type "Script" and transaction type "begin"

Sub panel of switching layout

Footer panel

Dialog panel with panel type "Switching Layout"

Page information by means of "Script for label" or "Script for text" at a text view

Button "Cancel": Action with action type "Cancel". For dialog panel: option "Close panel"

Buttons "Back", "Forward"/"Save": Action with action type "Save" and script type "Script (after action)"

- An action button returns the semantic element to be edited or created and displays it on another panel or opens a dialog. While doing so, the action begins a transaction so that editing can be aborted anytime.

- Within the panel or dialog, each step is presented on a separate subordinate page.
- A sub configuration displays progress information and is equipped with "Back"/"Forward"/"Save"/"Cancel" buttons. In case of a dialog, the dialog footer panel is suitable for such a purpose.

Configure a panel of type *Switching layout*. Each step of the wizard is then presented in a sub-panel of this panel. Embed the switching panel into a dialog panel or make sure that there are no other means of navigation despite "Forward/Save", "Back", and "Cancel". Configure a (footer) panel below the switching panel with navigation buttons "forward" and "back".

The wizard operates with an *Edit* view in each sub panel of the switching layout. The superordinate panel (e.g. dialog panel) is activated by means of an action starting a *transaction*: when the wizard is abandoned without saving ("Cancel"), the changes are aborted. Each step initiates an intermediate save action ("Forward"/"Back") within the transaction.

NOTE

Each sub panel of the switching layout *must* contain an edit view. Otherwise, the action buttons won’t operate. If introductory text is needed for a step, it therefore cannot be placed solely in a separate sub panel without edit view.

This wizard can be used with either the buttons placed in the same panel as the edit view or in a separate panel different from the edit view (e.g. dialog footer panel).

Register a script with the key `wizard`:

```
$k.define([], function () {
    function currentPageIndex() {
        const index = $k.Session.actual().getVariable('currentPageIndex')
        return index || 1
    }

    function numberOfPages() {
        const switchingConfig = $k.Session.actual().panelConfiguration()
        const switchingPanel = $k.PanelConfiguration.from(switchingConfig)
        return switchingPanel.subPanels().length
    }

    function nextPage(view) {
        const currentPage = $k.PanelConfiguration.from(view.panelView
().configurationElement())
        const switchingPanel = currentPage.parent()
        const pages = switchingPanel.subPanels()
        const currentIdx = pages.indexOf(currentPage)
        let nextPage = null
        if (currentIdx < (pages.length - 1)) {
            nextPage = pages[currentIdx + 1] // index based on start value 1
            $k.Session.actual().setVariable('currentPageIndex', currentIdx + 2)
        }
        return nextPage
    }

    function previousPage(view) {
        const currentPage = $k.PanelConfiguration.from(view.panelView().configurationElement())
        const switchingPanel = currentPage.parent()
        const pages = switchingPanel.subPanels()
        const currentIdx = pages.indexOf(currentPage)
        let previousPage = null
        if (currentIdx > 0) { // 0 based
            previousPage = pages[currentIdx - 1] // index based on start value 1
            $k.Session.actual().setVariable('currentPageIndex', currentIdx)
        }
        return previousPage
    }

    return {
        currentPageIndex,
        numberOfPages,
        nextPage,
        previousPage
    }
})
```

This script contains the functions as follows:

`currentPageIndex()`, `numberOfPages()`, `nextPage(view)` and `previousPage(view)` returning the current page, the number of pages, the next page and the previous page. Use this information for the action, label, and enablement scripts of the "Forward" and "Back" buttons.

For the footer panel, add a text view for displaying the pages and a menu. At the menu, add an action for the forward button: to save intermediate changes, select the action type *Save*.

Label script for **forward** button:

```
function label(element) {
    const wizard = $k.module('wizard')
    if (wizard.currentPageIndex() == wizard.numberOfPages())
        return "Save"
    return "Forward"
}
```

Action script for **forward** button — e.g. action type *Save* with *Script (after action)*:

```
function postAction(element, action) {
    const nextPage = $k.module('wizard').nextPage(this)
    if (nextPage == null) {
        action.setClosePanel(true)
        action.setTransactionCommit(true)
    } else {
        action.result().activatePanelConfiguration(nextPage)
    }
}
```

If the last step is reached (= last sub panel visible), the "Forward" button acts as a save button and commits the transaction — leading to all changes done in the dialog being saved. Committing the transaction will also save all intermittently saved changes that happend during the transaction.

Action script for **back** button — e.g. action type *Save* with *Script (after action)*:

```
function postAction(element, action) {
    const previousPage = $k.module('wizard').previousPage(this)
    if (previousPage !== null)
        action.result().activatePanelConfiguration(previousPage)
}
```

Enablement script for **back** button:

```
function actionEnabled(element) {
    const wizard = $k.module('wizard')
    return wizard.currentPageIndex() > 1 // index based on start value 1
}
```

Add a *Label* to the text view in the **footer** panel showing the current page number and total for progress indication. Provide a label script as follows:

```
function label(element) {
    const wizard = $k.module('wizard')
    return `${wizard.currentPageIndex()} / ${wizard.numberOfPages()}`
}
```

Configure further functionality for each step of the wizard as needed.

3.2.3.4. Transaction

Configure the first action as *Transaction: **begin*** and the final action as *Transaction: **commit***. Every in-between action should have an alternative action allowing users to abort the transaction. Configure abort actions with action type *Cancel*.

To clearly indicate the scope of the transaction use Dialog (modal) or Assistent patterns.

3.2.3.5. Guided input

Attach a menu to the property configuration for which you want to provide input guidance. Add an action to the menu and configure the action to do whatever is necessary to initiate the process. Configure a *Script (recall)* that will be executed after the guided process has finished:

```
function customActionRecall(action, actionResult) {
    this.setNewValue(actionResult.element())
    actionResult.activatePanel(this.panelView())
}
```

In this script, the input value will be written to the property input field (function `setNewValue()`) and the action result will be equipped with the contents of the current panel which is necessary as we otherwise might lose other input fields' values on the same panel.

Connect the action to a dialog panel or a sister switching panel using the relation *Show result in panel*. Configure the targeted panel to guide through the process of input value determination (see e.g. patterns Assistent or Dialog (modal)).

The final action of the process must invoke the recall script and make sure that possible dialog panels are closed. Additionally, the input field’s value must be passed to the recall script e.g. by setting the action result accordingly.

```
function customAction(action, actionResult) {
    actionResult.setModel(action.selectedElement())
    action.recallMarkedAction()
}
```

```
function customAction(action, actionResult) {
    action.dropMarkedAction()
}
```

```
function domainModel() {
    return this.domainModel().elements()
}
```

Provide the user with the ability to abort the process. The aborting action must remove the recall action from the session:

```
function customAction(action, actionResult) {
    action.dropMarkedAction()
}
```

3.2.3.6. Search and Filter

Searching for a specific element of the knowledge graph is often a complex task that has to be supported by various user interface elements and functionalities. First, the search has to be parameterized to the user’s needs. After initiating the search, the results are visualized in a way that allows the user to distinguish between different elements and finally select one or more. Optionally, further filtering of search results is needed.

In any case, the configuration of a search compoundis required, which on the one hand defines the search configuration to be used and on the other hand brings together the states of all the views involved, which are usually spread across several panels.

3.2.3.6.1. Parameter

Parameters are entered using form inputs. In order for the user input to be processed in the appropriate search parameters, the form input must be configured as part of a search compound as described in the chapter on form input.

Web applications often have a globally available search with a single parameter in a header bar that is always visible. More complex parameterized search functionality requires more space for the parameters and is therefore only made visible when required. In any case, it has proven to be advisable to place the parameter form inputs on a separate panel.

3.2.3.6.2. Triggering the Search

The search process can be triggered as soon as the user has specified all the required search parameters. If there is only one parameter, it can be triggered directly by an "Accept action" on the form input. Otherwise, a separate button is required.

The action for triggering the search can either have the action type "Script" or no action type. It is important that the required values from the form input are available for the parameters when the search is triggered. To do this, the action must be executed on a view that is located on the same panel as the form entries for the parameters.

The action for triggering the search must also activate the panel with the view of the search result or the panel with the view for filtering the search result.

3.2.3.6.3. Filtering

The user can use a facet view to further narrow down the search result. To do this, the facet view must be part of a search compound. An interaction configuration of the facet view is not required. If the facet selection is changed, the panel is automatically recalculated with the updated facet selection.

In order for the search result view to be adapted to the facet selection, the panel of the facet selection must influence the panel with the view of the search result. This is of course not necessary if both are on the same panel. For performance reasons, however, it is recommended to configure separate panels.

3.2.3.6.4. Search result

Various views can be configured as output in the search compound to display the search results:

- Table
- Graph
- Layout
- Alternative

In the case of Layout and Alternative, a *Script for domain model* is required, which distributes the elements of the search result to the subviews:

```
function domainModel() {
    return this.domainModel().elements()
}
```

3.2.3.7. Mirrored State

If the user navigates to a new panel, the views contained on it are recalculated on the basis of an "empty" base state. If you want to start at this point with a state that the user has already created by manipulating the views of other panels, you can configure the corresponding views as the source and target of a *Mirror compound*.

Panel A is located on the main page of the application and contains a form input F1 for a global search. Panel B contains the search result and also shows the entered search parameter in a form input F2. If you connect F1 via the *Mirrored from* relation and F2 via the *Mirroring from* relation with a mirror compound, the state of F1 is mirrored when F2 is initialized.

Mirroring is unidirectional, but can also be configured bidirectionally if required. In this case the views involved are both "mirroring" and "mirrored".

3.2.3.8. Customized relation target dialog

The standard relation target dialog provides default functionalities:

- A list of the possible relation targets displayed by their primary name. Clicking onto a list entry closes the dialog and creates a relation to the selected target.
- A dropdown form entry allows selection of relation target types. A "New" button offers to create a new object of the selected type to be used as relation target.
- A "Cancel" button closes the dialog without further action.

However, it might be the case that the dialog does not fit the needs for every web frontend. For example, the table might need to show other properties than the primary name or the dialog offers too much functionality — e.g. creating new objects of a type by an average user must be prohibited.

A customized relation target dialog can be created easily as follows:

1. At the property configuration for the relation, add a custom menu.
2. For the menu, select the menu type "View specific actions".
3. Add an action to the menu, select the action type *Choose relation target*.
4. In the ViewConfig Mapper tree, select the node *Dialog panels* and create a new dialog. In the dialog, choose the template `RelationTargetDialog`.
5. A follow-up dialog asks for a name which will be used to creating the configuration names for all components of the dialog panel, enhanced by the suffix `.relationTargetDialog`.
6. On the *Context* tab, create a new relation *Show result of action* with the previously created action as target.
7. Adjust the ViewConfig elements (table, menu actions etc.) according to your needs. For example, if a "New" button is not needed, remove it. If the both type selection and "New" button are not needed, remove the whole corresponding footer panel.

3.2.3.8.1. Assigning a new default dialog for relation targets

The standard relation target dialog already is preconfigured as a dialog panel of the View Configuration Mapper. It has the default view role `RelationTargetDialog` and it is displayed when selection of a relation target dialog is initiated by clicking onto the search button "+" of the property view.

By re-assigning the view role `RelationTargetDialog` to a customized dialog panel, this panel can be used as default instead.

Tipp: When using a default dialog panel with the role `RelationTargetDialog`, no custom action is needed at the property edit.

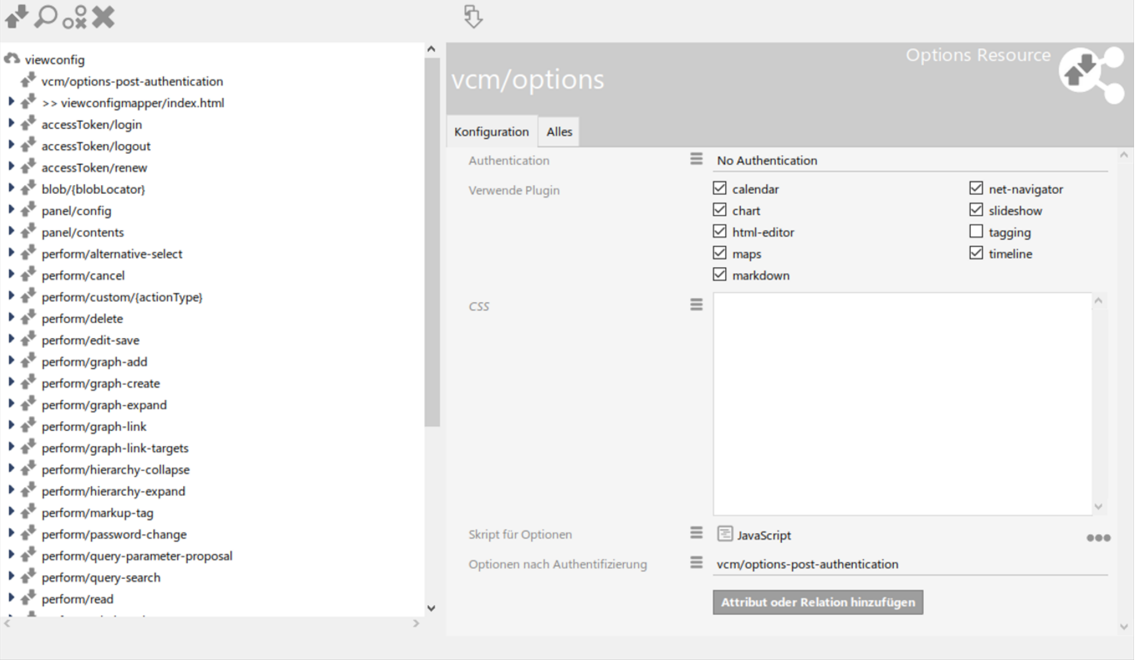
3.3. Configuration

The usual procedure involves activation of the ViewConfiguration Mapper components in the Knowledge Graph and the creation of a modification project, into which vcm is integrated. In order to modify the look & feel, making changes in CSS alone may be sufficient. vcm supports LESS (lesscss.org/). The templates can also be changed or supplemented for more complicated modifications.

Grunt (gruntjs.com/) is used as the TaskRunner, and as a Package Manager Bower (bower.io/). More detailed information and a list of the Grunt tasks is available in the README.md in the project.

3.3.1. Frontend configuration

The configuration of the web frontend is done by means of the *vcm/options* resource at the *viewconfig* REST service:



- viewconfig
  - vcm/options-post-authentication
  - >> viewconfigmapper/index.html
  - accessToken/login
  - accessToken/logout
  - accessToken/renew
  - blob/{blobLocator}
  - panel/config
  - panel/contents
  - perform/alternative-select
  - perform/cancel
  - perform/custom/{actionType}
  - perform/delete
  - perform/edit-save
  - perform/graph-add
  - perform/graph-create
  - perform/graph-expand
  - perform/graph-link
  - perform/graph-link-targets
  - perform/hierarchy-collapse
  - perform/hierarchy-expand
  - perform/markup-tag
  - perform/password-change
  - perform/query-parameter-proposal
  - perform/query-search
  - perform/read

vcm/options

Authentication: No Authentication

Verwende Plugin: ☑ calendar ☑ chart ☑ html-editor ☑ maps ☑ markdown ☑ net-navigator ☑ slideshow □ tagging ☑ timeline

CSS:

Skript für Optionen: JavaScript

Optionen nach Authentifizierung: vcm/options-post-authentication

There are two phases for which the options can be set:

1. Before authentication
2. After authentication

By default, only phase 1 is configured. If options need to be configured for the time after authentication (e. g. setting the frontend language depending on the user account), a further options resource must be created and then linked via " *Optionen nach Authentifizierung* ".

3.3.1.1. Options script

User-defined options

User-defined options can be set using the function setCustomOption at the VCMOptions object.

Option	Standard value	Description
disableUnloadWarning	*false*	Deactivates the warning when leaving the site (e. g. via the browser back button or by clicking on an external link)
history.enabled	*true*	Activates/deactivates the rewriting of the URL (bookmarking)
history.initialContent	*true*	Activates/Deactivates the initial loading of panel content

Example:

```
function configure(options, request) {
  options.setCustomOption('disableUnloadWarning', true)
}
```

Translations

Translations can be set using the function setTranslations at the *VCMOptions* object. When doing so, it is important to use " **base** " as property name and not the language abbreviation " **en** " (see following example). If this is not being considered, the standard translation texts for English are not found anymore.

Key	Description
login.form.message	Shows a a message within the login mask
login.form.title	Title of the login mask
login.form.submit.label	Label of login button
login.form.username.label	Label for user name text field
login.form.username.placeholder	Placeholder for user name text field
login.form.password.label	Label for password text field
login.form.password.placeholder	Placeholder for password text field

Example:

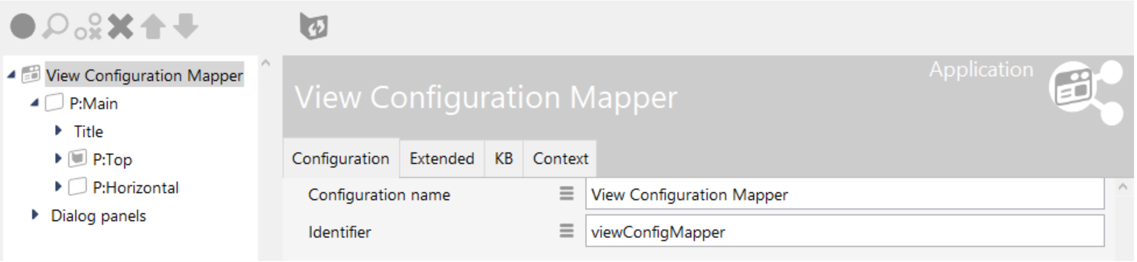
```
function configure(options, request) {
  options.setTranslations({
    de: {
      login: {
        form: {
          message: 'Bitte benutzen Sie ihr E-Mail als Login'
        }
      }
    },
    // base is the property name for english translations
    base: {
      login: {
        form: {
          message: 'Please use your e-mail as login'
        }
      }
    }
  }
}
```

3.3.2. View configurations for the View Configuration Mapper

The View Configuration mapper interprets all view configurations created in i-views. However, there are several differences between processing in the Knowledge Builder and in the View Configuration Mapper, which this chapter will discuss.

3.3.2.1. Panel configuration

If the web application is supposed to be based on a panel configuration, the application must be linked to the panel configuration.



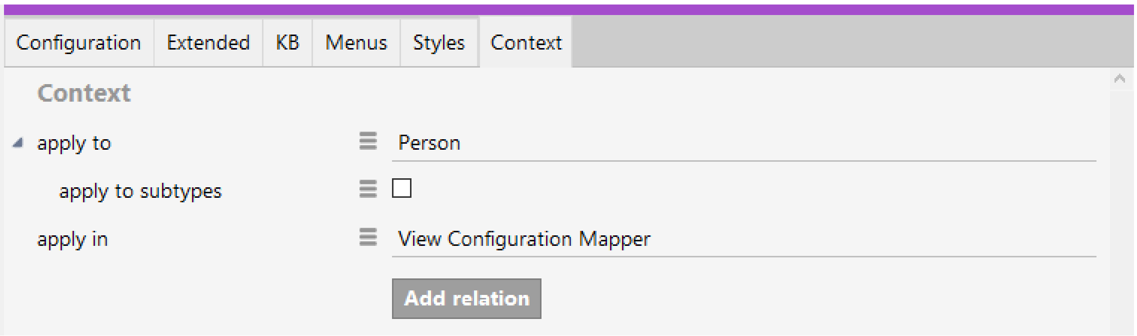
Configuration name: View Configuration Mapper

Identifier: viewConfigMapper

To do this, an object of the main window panel is appended to the application. All other panel configurations can then be appended to this object. Additional panels (e.g. dialog panels) are optional. However, if they are used in the web front-end, they must be connected to the application in this way. It does not suffice to merely define it e.g. as a target window of an action because it would not be taken into account for the display of the application otherwise.

3.3.2.2. Apply in

In order to determine a suitable view configuration for a semantic element, it is necessary to look to the type of the element and to the context in which the view configuration is to be used. This context is determined via the “apply in” relation. If a view configuration is to be used in vcm, it should therefore be ensured that the relation was sourced accordingly.



Context

apply to: Person

apply to subtypes: □

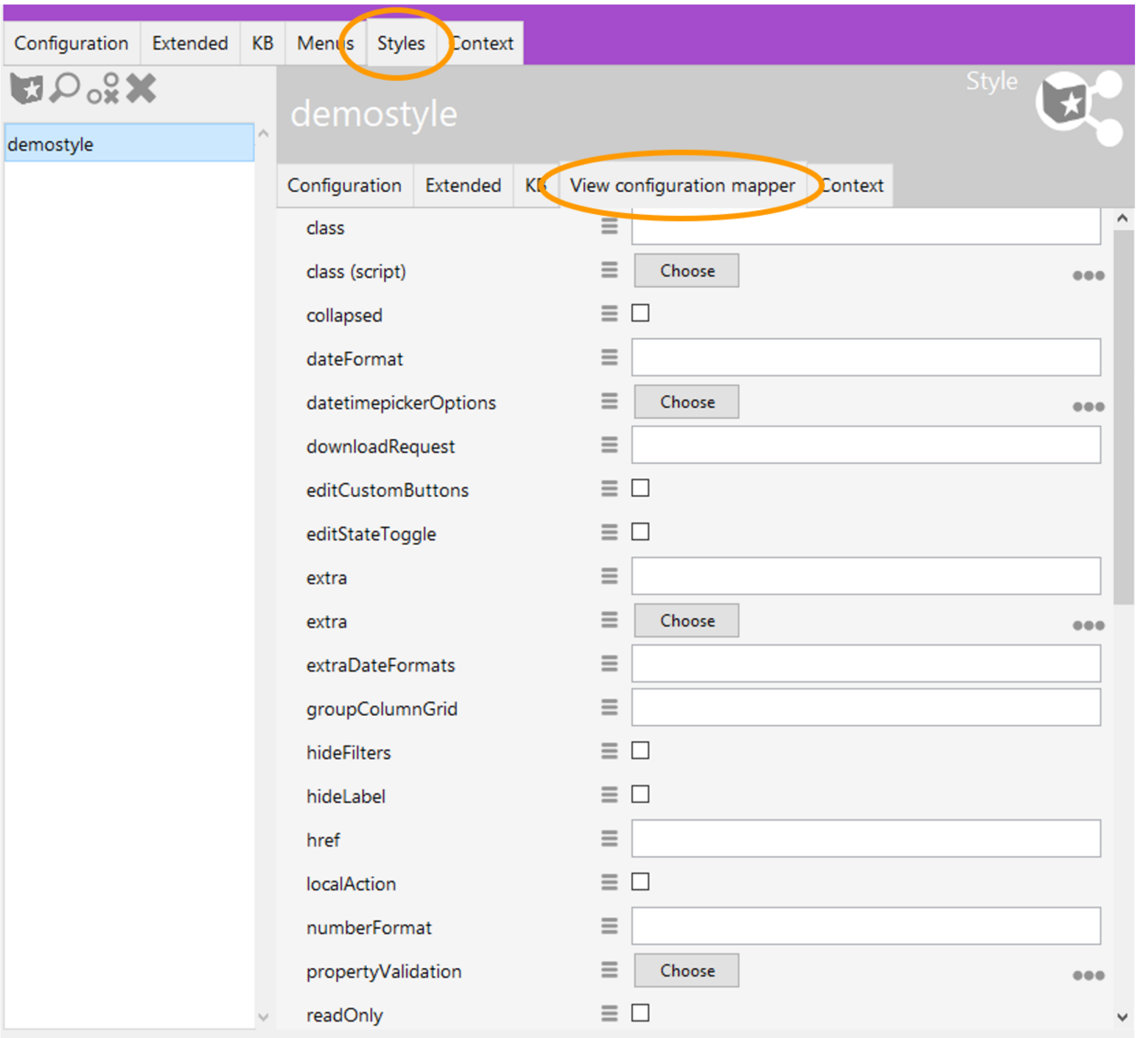
apply in: View Configuration Mapper

3.3.2.3. Style

To influence the display of a view, it is possible to use so-called “styles”. They can be used, for example, to configure whether a heading is to be displayed, or whether data should be highlighted in a specific way.

The setting for the styles for the display in the web front-end by means of the view configuration mapper are available on the “View configuration mapper” tab. The prerequisite for this is that a view configuration mapper component has been installed in the KB.

There are multiple setting options for the styles (see figure):



demostyle

class	
class (script)	Choose
collapsed	□
dateFormat	
datetimepickerOptions	Choose
downloadRequest	
editCustomButtons	□
editStateToggle	□
extra	
extra	Choose
extraDateFormats	
groupColumnGrid	
hideFilters	□
hideLabel	□
href	
localAction	□
numberFormat	
propertyValidation	Choose
readOnly	□

There are a number of Style elements that are already defined in i-views. The following section explains what these elements are and how these style elements are created in the Knowledge Builder so that they can then be linked to individual elements of the view configuration of an application.

In the view configuration, you first have to select the element with which one or more style elements are to be linked. Depending on the type of the view configuration element, various tabs are available for configuring the styles ("Actions and styles" → "Styles" or just "Styles"). Once you have chosen this tab, you can either define a new style element or link and existing style element . When defining a new style element it is first necessary to assign it a configuration name. You can then configure it on the right side of the editor.

The following section describes the individual configuration options for style elements:

Name	Attribute type	Configuration type	Description
class	String	CSS class	Styling through specification of a predefined CSS class i the CSS of the ViewConfiguration Mapper or in the “viewconfigmapper.config.GET” script
class (script)	Reference to script		Definition of CSS styling in the form of a script return value
collapsed	Boolean		
dateFormat	String		
datetimepickerOptions	Reference to script		
downloadRequest	String		
editCustomButtons	Boolean		
editStageToggle	Boolean		
extra	Reference to script		Can be used to create a user-defined behavior for an action with the help of the script and render mode. Example: A script that returns URL attribute values is used with the “external” renderMode and with a parameter specification in the “href” line to define an external web link for the action of a button.
extra	String		
extraDateFormats	String		
hideFilters	Boolean		Hides the table query filters in the table header
hideLabel	Boolean		Hides the label of a view configuration element (label on the tab of an alternative remains)
href	String	Hyperlink	Link to a website or folder path as per the HTML standard. Alternatively, you can enter a parameter name in curly brackets which is then equipped with a URL under “extra” by means of a script.
localAction	Boolean		Limits the effect of an action to the current panel
numberFormat	String		
readOnly	Boolean	Properties	The properties of the view configuration element can only be read in the application, not edited. That is why no “Edit button” is displayed.

Name	Attribute type	Configuration type	Description
renderMode	Selection	Property	See the “RenderModes” sub-chapter
renderMode	String	Property	See the “RenderModes” sub-chapter
style	String		Here you can define CSS properties that are only used for those views that are linked to this style.
style	Reference to script		Here you can use a script to define CSS properties that are only used for those views that are linked to this style.
target	String		
tooltip	String	Context help	Note that is displayed during mouse hover
vcmDetailed	Boolean		
vcmMarkRowClick	Boolean		If activated clicks on a table row will display them as marked. This style has to be linked to a table.
vcmPluginCalendarOptions	Reference to script	VCM plugin	Default values that can be defined by script, e.g. start date when the calendar view is called
vcmPluginChartDataColumns	String	VCM plugin	
vcmPluginChartDataMode	String	VCM plugin	This is used if the data of the underlying table is to be read out either by row (“rows”) or by column (“columns”) for the chart to be displayed; if not specified, the default data mode is “rows”
vcmPluginChartHeight	String	VCM plugin	Absolute height of a chart in pixels (e.g.: “300px”)
vcmPluginChartLabelColumn	String	VCM plugin	
vcmPluginChartOptions	Reference to script	VCM plugin	Script that can be used to control the display of components of the chart: Display of keys, scaling of axes etc.

Name	Attribute type	Configuration type	Description
vcmPluginChartType	Selection	VCM plugin	Selection options for the “chart” RenderMode (applicable for tables): • bar • doughnut • line • pie • pole • radar
vcmPluginChartWidth	String	VCM plugin	Absolute width of a chart in pixels (e.g.: “380px”)
vcmStateContext	Selection		Selection options: • global • page • none
vcmStateContext	String		
vcmTruncate	String		

NOTE

For each view configuration element separate styling possibilities are available which are described in detail in the respective sub chapter. For example, a properties view can be further adjusted regarding the layout of the labels and their values using specific parameters.

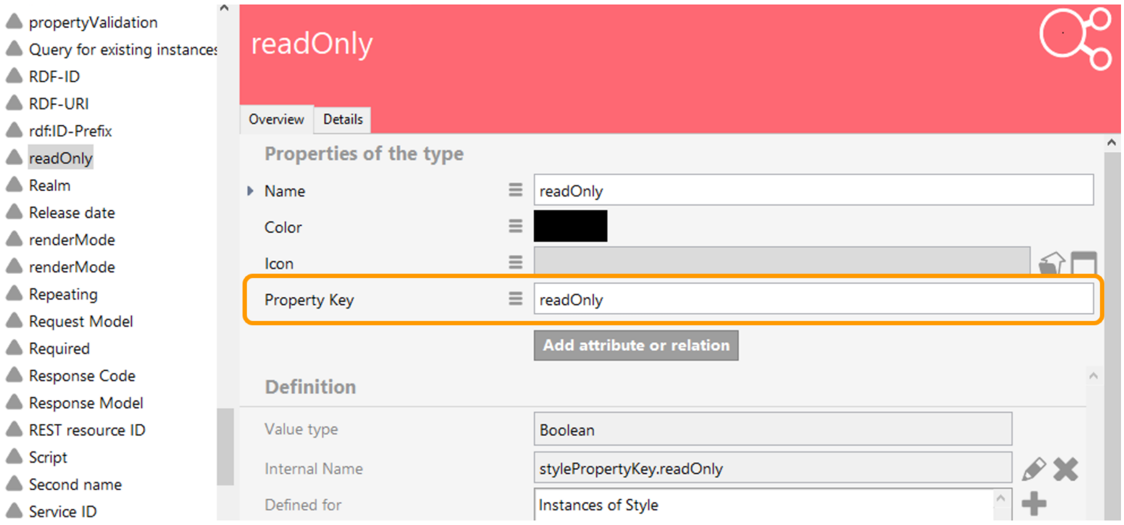
3.3.2.3.1. Definition of style attributes

You can define your own style attributes in addition to those predefined by the application.

You can create the attributes of the styles under View configuration → Attribute types.

To ensure the style attribute is also written to the JSON output, an addition must be added to the attribute in the schema. You get to the schema by clicking on “Schema” in the ☰ menu of the attribute. In the schema, you then have to maintain the attribute “Property key” and enter the name of the attribute there.

“Objects of style” must be entered in this “defined for” field. You add an entry by clicking on the Plus icon (“Add” button). Once you have entered “Style” as the search term, a list appears from which you select the entry “Style” (view configuration). Following that, you have to select the additional tab page in which the new style element is supposed to be displayed.



readOnly

Properties of the type

Name: readOnly

Color

Icon

Property Key: readOnly

Definition

Value type: Boolean

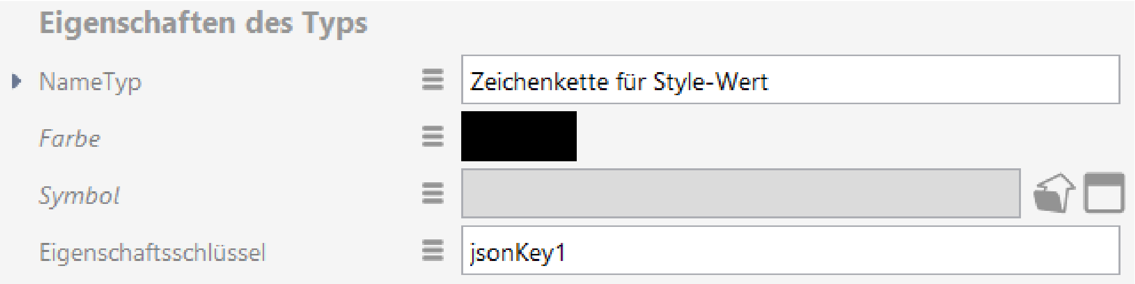
Internal Name: stylePropertyKey.readOnly

Defined for: Instances of Style

In the JSON output, the key and value pairs (*StylePropertyKey* → Style property) are output as an array under *additionalConfig* .

Example

Configuration of the type *String for style value*



Eigenschaften des Typs

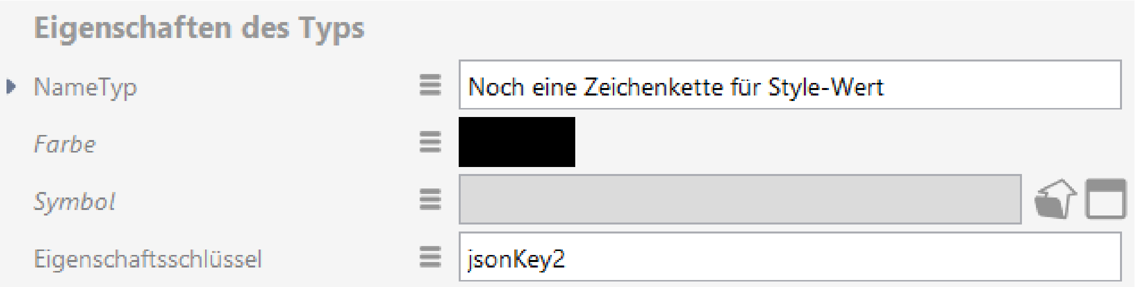
NameTyp: Zeichenkette für Style-Wert

Farbe

Symbol

Eigenschaftsschlüssel: jsonKey1

Configuration of the type *Additional string for style value*



Eigenschaften des Typs

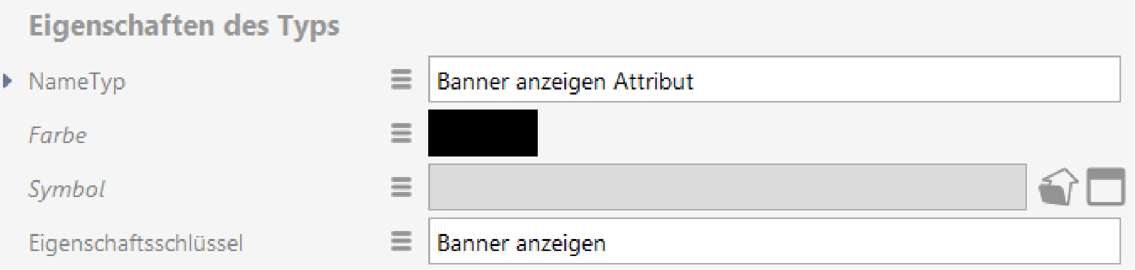
NameTyp: Noch eine Zeichenkette für Style-Wert

Farbe

Symbol

Eigenschaftsschlüssel: jsonKey2

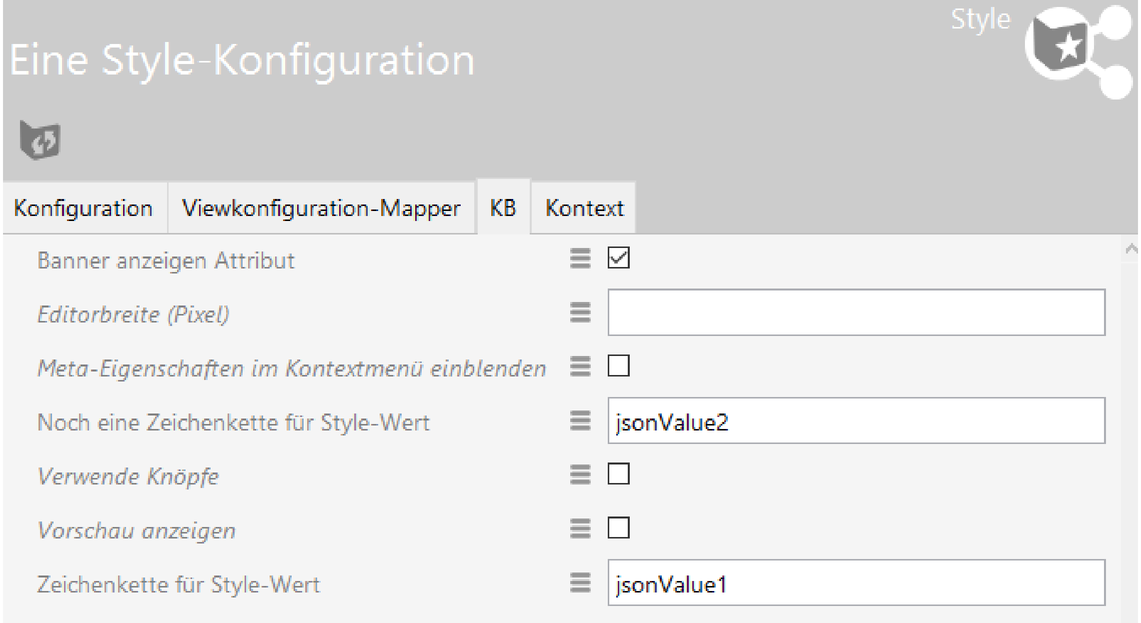
Configuration of the type *Display banner attribute*



Eigenschaften des Typs

NameTyp	Banner anzeigen Attribut
Farbe	■
Symbol	
Eigenschaftsschlüssel	Banner anzeigen

Configuration of the object *One style configuration* of the type *Style*



Eine Style-Konfiguration

Banner anzeigen Attribut	☑
Editorbreite (Pixel)	
Meta-Eigenschaften im Kontextmenü einblenden	□
Noch eine Zeichenkette für Style-Wert	jsonValue2
Verwende Knöpfe	□
Vorschau anzeigen	□
Zeichenkette für Style-Wert	jsonValue1

JSON output:

```
"properties": [{
    "values": [{ ... }],
    "label": "First name",
    "additionalConfig": {
        "jsonKey1": ["jsonValue1"],
        "jsonKey2": ["jsonValue2"],
        "Display banner": ["true"]
    },
    "viewId": "ID34304_461524079",
    "schema": { ... }
}
```

3.3.2.3.2. Render modes

RenderModes can be used to apply additional predefined style properties.

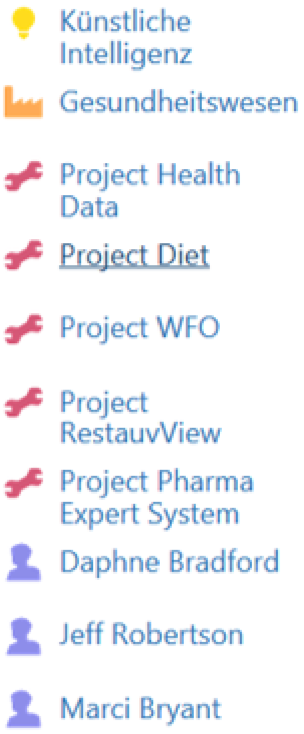
RenderModes are available in the styles in the view configuration on the “view configuration mapper” tab, once via drop-down menu and additionally via input line. Here the freely selectable value entered via the input line takes precedence, which means that it overwrites a value that was selected via drop-down.

The following renderModes are available in the drop-down menu:

renderMode	Explanation	Applicability
breadcrumb	Displays the hierarchy and path navigation	Hierarchy
calendar	Displays date information in a calendar view; the basis for this is a table containing the attributes of the value type *time* , *date* , *date and time* , *flexible time* or *interval* with the date and time type.	Table
chart	Displays the data from a table in a chart. Under *vcmPluginChartType* you can select the type of chart. Under *vcmPluginChartOptions* you can use a script to format the chart more precisely, e.g. axis scaling, display of keys etc.	Table
download	Link to file download	Action

external	Generates an external link in connection with href; can be used, for example, in combination with *icon* and *tooltip* . For dynamic links, an identifier in curly brackets can be used in the href attribute. If the *extra* script provides a JavaScript object with a value for the identifier, this is entered automatically. You can, for example, trigger a Google search for the name of the current object in the following manner: *href* : https://www.google.com/search?q={search} *extra* script
	`function additionalPropertyValue(element,`
	`context) {`
	`    return { search: element.name() }`
	`}`
		Action

html	Shows the string without masking	String property
markdown	Converts text sections equipped with mark-ups into text with highlights by means of in-line formatting	Text or string attribute

renderMode	Explanation	Applicability
medialist	Displays the table entries as an HTML text link; displays the element with their icons  Künstliche Intelligenz Gesundheitswesen Project Health Data Project Diet Project WFO Project RestauvView Project Pharma Expert System Daphne Bradford Jeff Robertson Marci Bryant	Table
multiline	Necessary to display the input field for a string in multi-line view in an edit view.	Property
nolink	The relation target is not linked, but instead shown only as text.	Relation property
pre	Displays the string as a pre-formatted and scrollable text	String property
timeline	Display of a data record in the form of a timeline; can be arranged vertically or horizontally.	Script-generated view in group
translations	Displays language variants (with the relevant flag icons in case of the string attribute). **NOTE** This render mode cannot be combined with another Style containing the render bode "Multiline".	Property

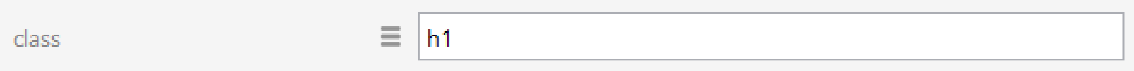
The renderModes available in the input line are related to Bootstrap. They include the following renderModes, for example:

renderMode	Explanation	Applicability
email	Creates a link to the email address	String property
image	Displays an icon on the action	Action

3.3.2.3.3. Usage of CSS

The view configuration mapper supports the use of Cascading Style Sheets (CSS). In addition to that, it includes a predefined set of CSS properties to which you can refer in the style of the views. It also offers you the option to define your own CSS properties.

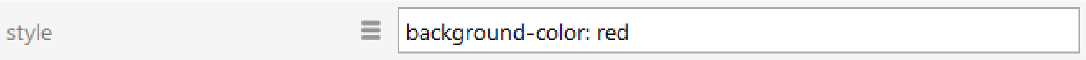
The predefined set is based on the CSS classes defined the front-end framework bootstrap (getbootstrap.com/docs/3.4/css/). To use these, they can be referenced in a style using the *class* property (e.g. "h1" as the value for a heading).



class ≡ h1

Separate CSS properties can be defined using the following values:

- The attribute *style* or *style (script) is available on a style.* Here you can define a CSS that applies only to views to which this style is linked.

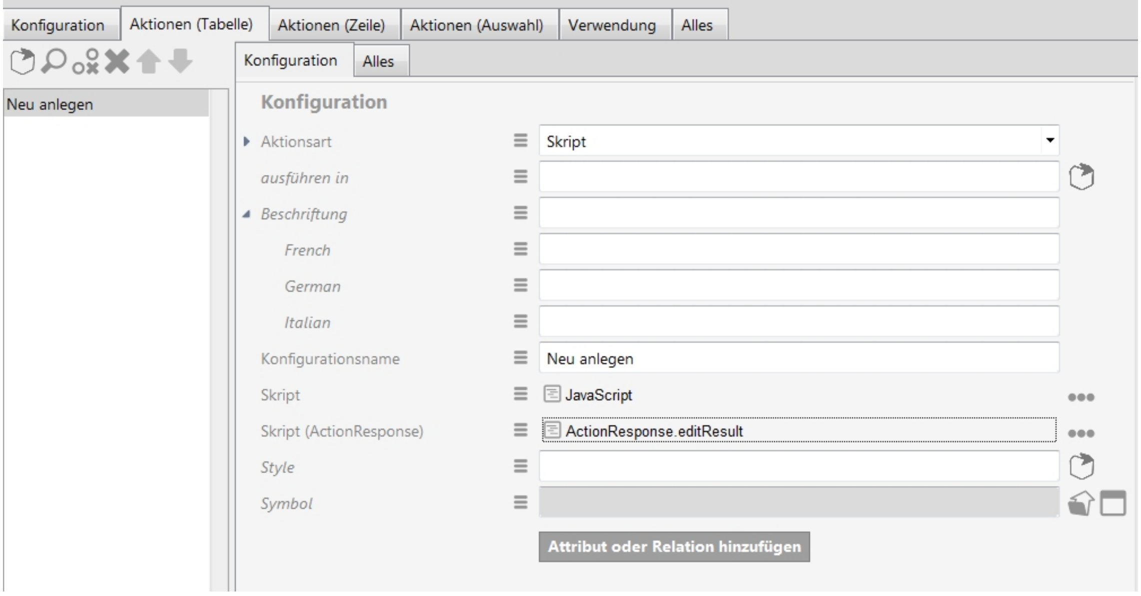
  

  style ≡ background-color: red

- CSS properties that are supposed to apply to entire applications can be defined in the script “viewconfigmapper.config.GET.” If separate CSS classes are defined there, you can access these in the styles via the *class* attribute.

3.3.2.4. Execute in

When you create a user-defined action, you can also fetch the relation “execute in.” This has the effect that the returned data is not applied to all VCM contents but that the change only relates to a certain view. This view must be set as the relation target of “execute in.”



Aktionsart: Skript

ausführen in:

Beschriftung:

French:

German:

Italian:

Konfigurationsname: Neu anlegen

Skript: JavaScript

Skript (ActionResponse): ActionResponse.editResult

Style:

Symbol:

3.3.3. Login configuration

3.3.3.1. JWT authentication

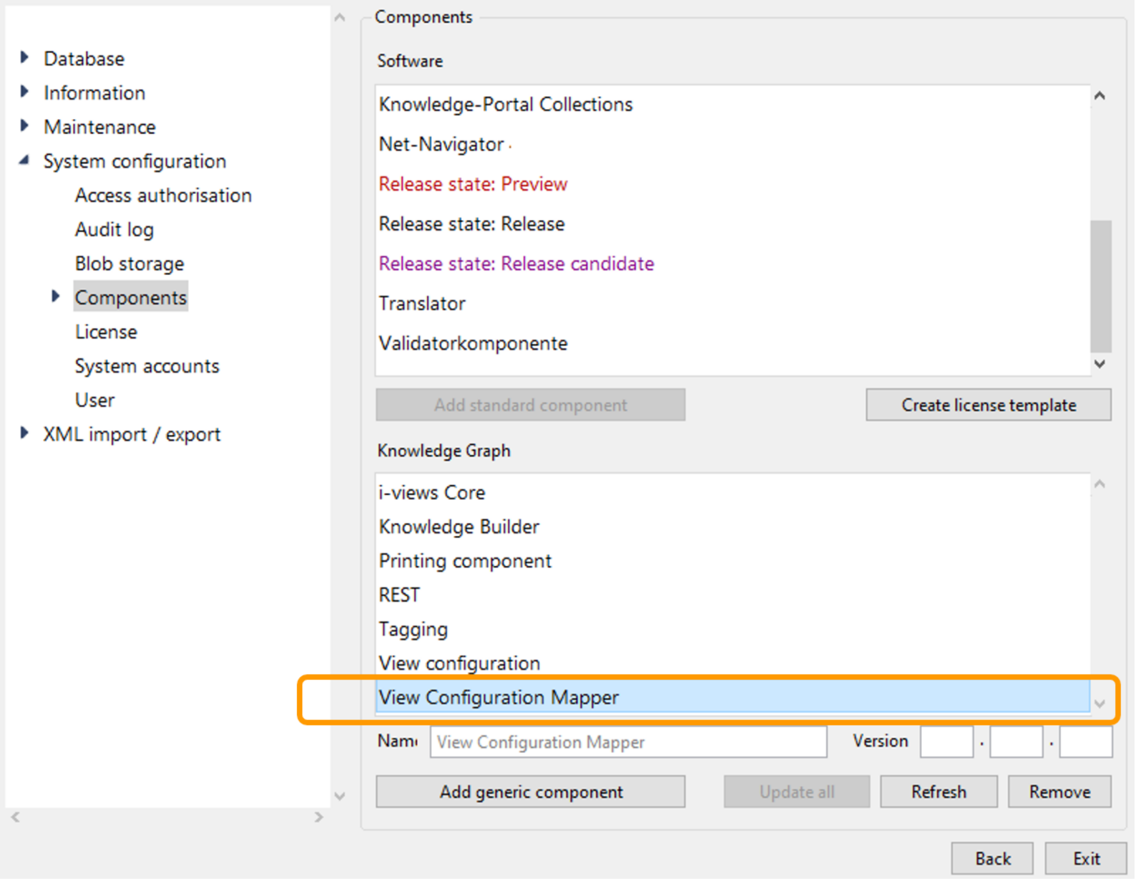
3.3.3.1.1. Modify the login form

The login form can be modified using the following translation key:

Key	Description
login.form.title	Title of the form
login.form.message	Descriptive/welcome text
login.form.username.label	Label of the user name field
login.form.username.placeholder	Placeholder of the user name field
login.form.password.label	Label of the password field
login.form.password.placeholder	Placeholder of the password field

3.3.4. The View Configuration Mapper component

To use the ViewConfiguration Mapper, activation of the corresponding components first in the Admin tool is a prerequisite.



Components

Software

Knowledge-Portal Collections
Net-Navigator
Release state: Preview
Release state: Release
Release state: Release candidate
Translator
Validatorkomponente

Knowledge Graph

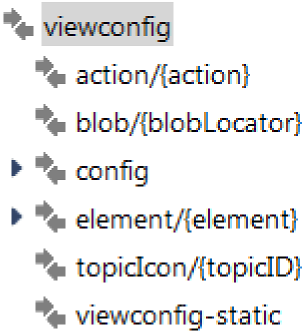
i-views Core
Knowledge Builder
Printing component
REST
Tagging
View configuration
View Configuration Mapper

Nam: View Configuration Mapper
Version . .

The component ensures the specific properties required are created in the view configuration and also creates all REST services that the vcm requires.

NOTE

All requests are preconfigured so that they expect an authentication. The attribute Password and Login is required for an authentication on the object of the user, with its schema generated by the component. Linking the user in the settings for the Knowledge Builder is not necessary for this.


viewconfig
- action/{action}
- blob/{blobLocator}
- config
- element/{element}
- topicIcon/{topicID}
- viewconfig-static

These are, specifically:

- action

- blob
- config
- element
- topicIcon
- viewconfig-static

“action” and “element” perform all communication between the ViewConfiguration Mapper and i-views. “blob” and “topicIcon” are responsible for delivery of the media data within a Knowledge Graph. “viewconfig-static” defines the area of the REST bridge in which the VCM front-end files (scripts, templates, etc.) are found. “config” is called during the initialization of vcm to configure basic configurations (such as language and start topic). All REST services are preconfigured so that modifying them is not always required. However, modifying the “config” request is recommended:

```
function respond(request, parameters, response){
    //Personalize your viewconfigmapper configuration here
    var options = {
        "application" : "viewConfigMapper",
        "user" : {
            "login" : $k.user().name()
        },
        "startElement" : $k.rootType().idString(),
        "language": getRequestLanguage(request),
        translations: getTranslations()
    };
    response.setText(JSON.stringify(options, undefined, "\t"));
}
```

Values to be modified are

- **application** : The application configured in the view configuration for the ViewConfiguration Mapper. This is, by default, “viewConfigMapper” and therefore does not have to be modified.
- **user** : User configuration. The current version of vcm only reads the configured name of the user for display in the front-end.
- **startElement** : ID or internal name of the topic that should be displayed initially when the start screen is called up. The root type of the Knowledge Graph is preconfigured. This should be modified.
- **language** : The language of the browser making the request is preconfigured. This attribute should be configured for specific language settings. The relevant I18N settings are foreseen in the front-end templates and can also be expanded in the attribute “Translations”. Modifications to this should be made in these templates. At this point, only the language is being defined.
- **translations** : I18N templates are located in the front-end and should be modified there. Their function can be extended at this point.

3.3.5. Create a project with the View Configuration Mapper

To easily create an adjustment project, a project template is available in the Git under gitlab.ivda.i-views.de/product/viewconfigmapper/grunt-init-viewconfigmapper.git. The README.md file of the project explains all further steps. Initialization requires certain parameters. For example, you will be asked for the basic path of the request and the name of the application. This data should be available when first called.

3.3.6. Modify templates

The project template contains the directories components/ and partials/ in the webroot/ directory. Both directories contain examples of ViewConfigMapper components and partials. You can add new templates here. The basic templates of ViewConfigMapper remain available, so you only need to create templates for special adjustments.

The js/ directory contains a JavaScript file where the ViewConfigMapper is initialized.

```
var vcmOptions = {
    config: {
        router: {
            urlRewrite: true
        },
        application: "{%= name %}",
        ajaxBasePath: "{%=ajax_base_path %}",
        instanceId: "vcm_{%= name%}"
    },
    partials: partials,
    components: components,
    translations: translations
};

var vcm = new ViewconfigMapper("#viewconfigmapper", vcmOptions);
```

The ViewConfigMapper receives the configuration settings, partials, components and translations. The position in which the content is to be rendered is also specified (in this example: <div id=viewconfigmapper"/>). For partials and components it is only important that they are located in the relevant directories, because there are grunt tasks that extract the files and unload them to separate JavaScript files.

Values for application, ajaxBasePath and instanceId would be set during the initialization call of the project template.

3.3.7. Operate the frontend

The front-end can be built using grunt. The files required for operation are found in the /webroot directory following generation. It is accessed, if not configured otherwise, using the start screen index.html.

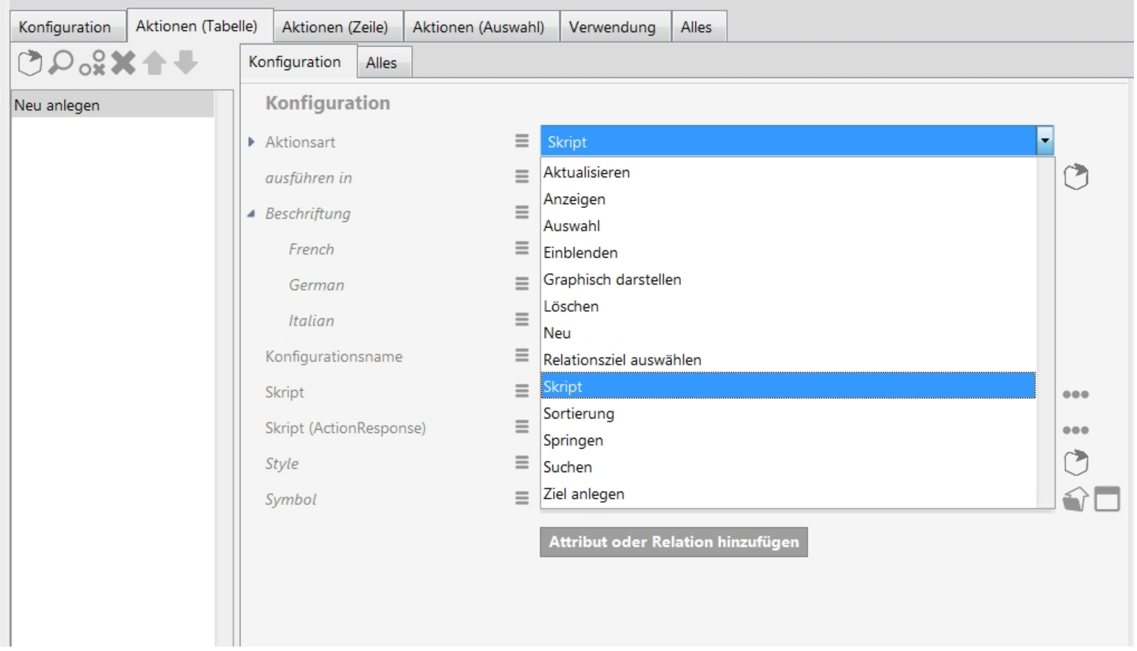
In the most straightforward case, the files are found locally and can then only be used on the client side.

There are several ways to make the front-end accessible. The component ViewConfiguration Mapper automatically generates a REST service that can deliver static files. This can be used by placing the files in the webroot directory in the corresponding directory in the REST bridge being used (default is viewconfig-static). After this, the front-end can be addressed in the default configuration via HOST:PORT/viewconfig/viewconfig-static/index.html. In addition, it is also possible to deliver the files using a corresponding server.

3.4. Actions

The VCM supports standard interactions, such as the editing of contents without these having to be configured separately. However, it is possible to define user-defined actions in a view configuration. These are actions of type "Script".

Selection is made via a drop-down menu.



For a script action, you have to select “Script” in this menu and create a “Script” under the “Script (user defined)” entry in the list.

For customized VCM builds it may be necessary to create an action result that feeds the need of a customized view. To achieve this you must use “Script (Action Response)”. Notice that you are not allowed to modify the knowledge graph in an action response script.

3.5. Panels

Panels are configuration elements that separate the application interface into sections. They are used to build the basic layout of an application.

Panels contain further panels or view configurations and can be nested in each other. They can mutually affect each other.

Panels usually contain exactly one start element (an object or a type) during activation (= becoming visible), which they pass on to their sub-configurations. Panels that contain view configurations that display a set of objects (table, facet selection, graph) can also process a set of start elements.

Panels themselves have no other functions. These can only be defined with the help of actions and view configurations.

There are different types of panels:

- Main window panels
- Dialog panels
- Window title panel
- Footer panels
- Normal panels

For each application there must be precisely one so-called *main window panel* , which can be divided by means of subordinate panels. In addition, it can be allocated a *window title panel* specifying the title and logo (*Favicon*) of the application.

It is also possible to assign additional *dialog panels* to the application; these panels can be displayed as a pop-up on top of the main window. Next to additional panels, they can also contain window title and footer panels.

A specific panel type must be selected for each panel .

- Layout panels (contain additional panels):
    - Linear layout (all subordinate panels are displayed in horizontal or vertical order)
    - Switching layout (only one of the subordinate panels is displayed at the same time)
    - Variable-sized Layout (only for printing)
- View panels (contain view configuration(s)):
    - Defined view (contains only one single defined configuration element)
    - Flexible view (multiple views possible, depending on the type of start element)

Setting options

Name	Value
Show action results in panel	All actions that are shown in the source panel cause the target panel to be displayed with the respective transferred object (example: every click in the panel object list causes the result to be shown in the details view panel).The action setting "show result in panel" overrides this setting. Moreover, the setting has no effect on "save" actions.
Influences	Here you can specify a panel that is influenced by the current panel (example: the objects displayed in the search results affect which facets are displayed correspondingly).
Inherit to subpanels	Boolean, meta-attribute of "Influences". This also allows subpanels to activate the influenced panel when activated (example: You have a navigation panel that should display the same for each subpanel of a panel with a switching layout when it is activated).
Script for target object	With the help of scripts you can specify not only panels but also conditions under which specific panels are affected by the current panel.

Setting options for layout

Name	Value
class	CSS classes for the panel (considered only for web applications or in the ViewConfig mapper)
Width/height	The precise dimensions of the panel can be set here in percent or down to the pixel.
Maximum width/height	Alternatively, you can enter the maximum dimensions of the panel here. The panel takes up as much space as possible without exceeding these values.
Flex-grow/shrink	Here you can specify the values for the relevant CSS property for the growth or shrink factor of the panel. An element with a value of 2 for flex-grow, for example, receives twice as much value as an element with a value of 1.
overflow-x/y (scrollbar)	This can be used to define how scrollbars are displayed if the content of the panel does not fit into its horizontal (x) and vertical (y) dimensions. The available options are *auto* , *scroll* and *hidden* .
Style	CSS styling rules for the panel (considered only in web applications or in the ViewConfig mapper)

3.5.1. Activation of panels

Panels exhibit two basic conditions: “active” and “inactive”. A panel is visible when it is active.

The activation of panels functions using the following mechanisms:

1. The main window panel of the application is always active when an application starts
2. The execution location determines which panel become active when an action is executed

Based on A/B, there are subsequent activations based on these rules:

1. Panels influenced are activated
2. Panels with a specialized function (e.g. window title) are activated, and this from all panels in the corresponding hierarchy
3. Subpanels are activated
4. In the case of a panel with a changing layout: Sister panels of the active subpanel are deactivated
5. Continue with 1. until no further panels can be activated (an integrated cycle test prevents endless loops)

Subsequent activations transport the model displayed respectively. If, for example, panel A shows the object “Mr. Meier”, then the activated subpanel B also shows “Mr. Meier”.

Last of all, this ensures that all panels above the activated panel are also active. However, their content is not calculated again.

Advanced activation mechanisms (version 5.2 or higher) :

So-called “Activation mode” can be used to optimize the calculation of the panel contents in step A (action activation) and in step 1 (influencing).

This avoids the recalculation of panel contents that are currently not displayed because despite activation, they are not within the visibility area (e.g. a shopping basket). The options “Refresh model and view” and "Refresh view only" are provided for this case.

The option “Default” is the fallback setting when neither of the two options described above were selected and leads to panel activation and evaluation of activation chains.

3.5.2. Layout panels

The application is divided into different areas using layout panels.

Linear Layout

Linear layouts arrange subordinate panels either next to each other or one above the other.

Name	Value
Orientation (only available if panel type " *Linear Layout* " has been selected before)	• horizontal: display order from left to right • vertical: display order from top to bottom

Switching Layout

Switching layouts permit alternative displays on the same visualization panel, with only one of the subordinate panels being displayed at the same time.

Setting options for configuration

Name	Value
Activate the first by default (for *changing layout* only)	If a checkmark is set, this means that the first subordinate panel is activated by default (the example below shows the start screen)

3.5.3. View panels

View panels serve as containers for individual views. They can however contain no further panels.

Setting options

Name	Value
Context element	Here it is possible to specify a concrete object or concrete type that serves as the source element from which further paths can be pursued through the Knowledge Graph.
Cannot be overwritten by external context element	If this option is activated, the configured context element is always used. Influence from other panels has no effect in this case.If no context element has been configured, the context element remains empty.
Script for context element	The script determines the start element. The external context element is transferred as the argument.The “Cannot be overwritten by external context element” option has no influence, and the script is always executed.
Sub-configuration (only for *defined view*)	Here it is possible to specify the one view configuration that is used for the defined view.

3.5.4. Dialog panels

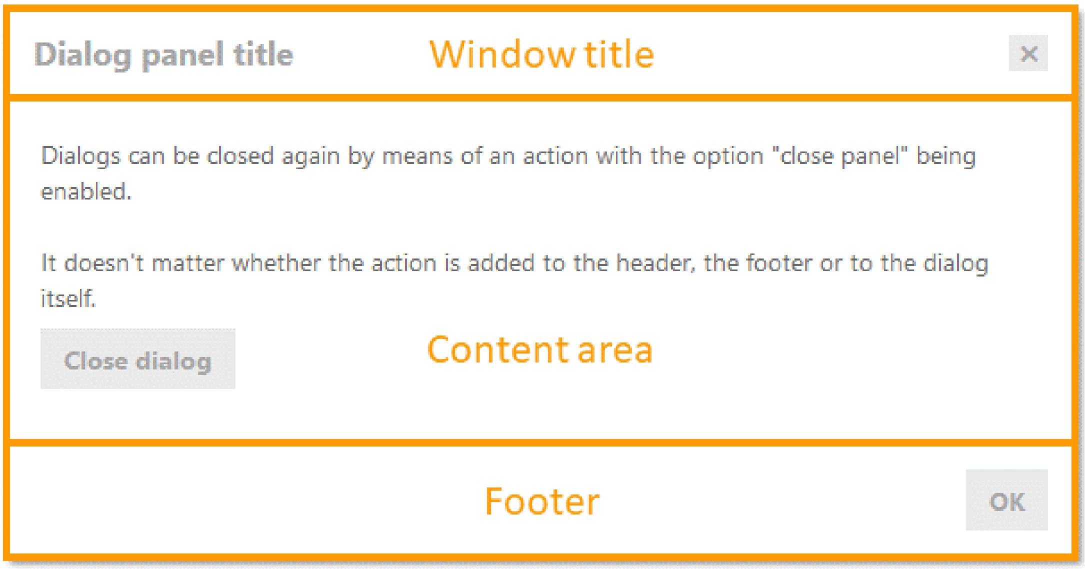
Dialog panels are special display areas whose contents are displayed in a dialog box. Dialog boxes appear automatically when the corresponding dialog panel is activated. Just like with other panels, activation is also possible via certain actions (see relation “Show result in panel” in Action configurations) or generally on activation or updates of other panels (see relations “Show actions in panel” and “influences” in other panel configurations).

Actions also have to be used to hide (“close”) dialog boxes. If the “Close panel” attribute is selected in an action configuration, executing this action in a dialog box has the effect that the window is closed. Hence, the action must be linked to a menu that is displayed in the dialog panel or one of its subordinate panels.

Content-wise, dialog boxes are divided into the following three areas:

- Window title
- Content area
- Footer

The contents and the layout within the three areas can be specified using a panel configuration for each. The dialog panel itself represents the content area. To configure the window title and footer, a sub-configuration of the type window title or footer panel must be created on the dialog panel (see the example below).



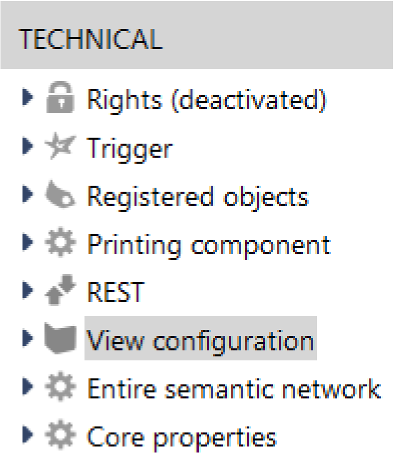
Dialogs can be closed again by means of an action with the option "close panel" being enabled.

It doesn't matter whether the action is added to the header, the footer or to the dialog itself.

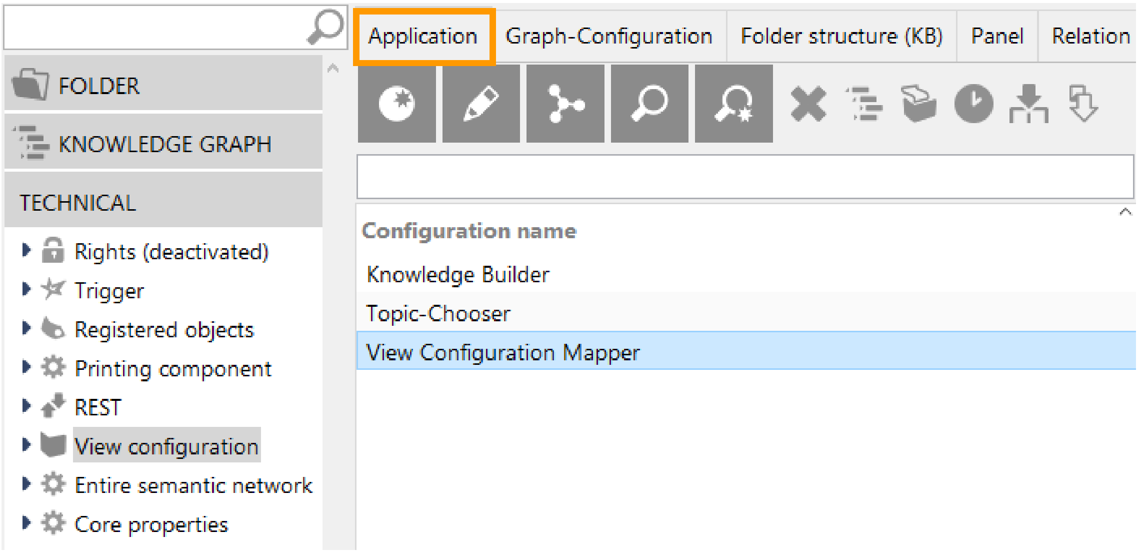
You can use the “Panel type” attribute on the actual dialog panel and on its window title and footer panels to determine whether the respective panel provides layout or view functions. Detailed descriptions of the different panel types are available in the preceding chapters.

Dialog panels can be created as follows in Knowledge Builder:

1. Use a user account that has administrator rights to log on to Knowledge Builder
2. In the navigation area, on the left, open the “Technical” category and select the sub-item “View configuration.”

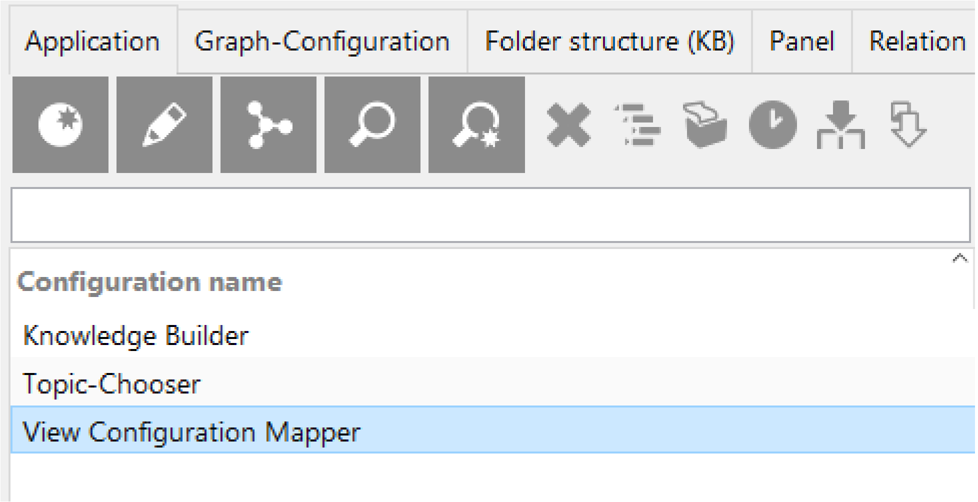


1. Select the “Application” tab on the right window.



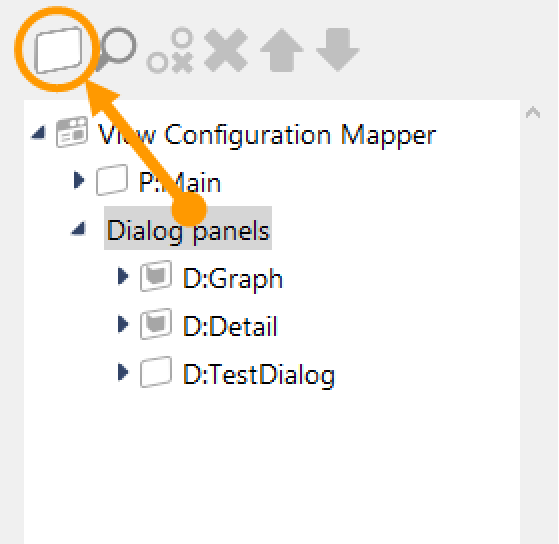
Configuration name
Knowledge Builder
Topic-Chooser
View Configuration Mapper

1. In the list underneath, select the application to which you would like to add the dialog panel (usually “View configuration mapper”).

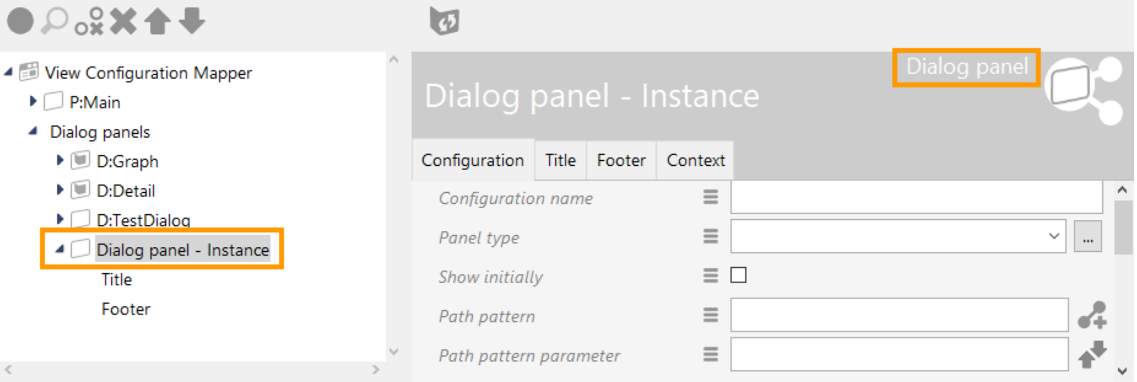


Configuration name
Knowledge Builder
Topic-Chooser
View Configuration Mapper

1. Select the dialog panel section in the panel tree below and click on the Create icon



1. The newly created dialog panel is automatically selected in the panel tree and the details view is displayed to the right of the panel tree



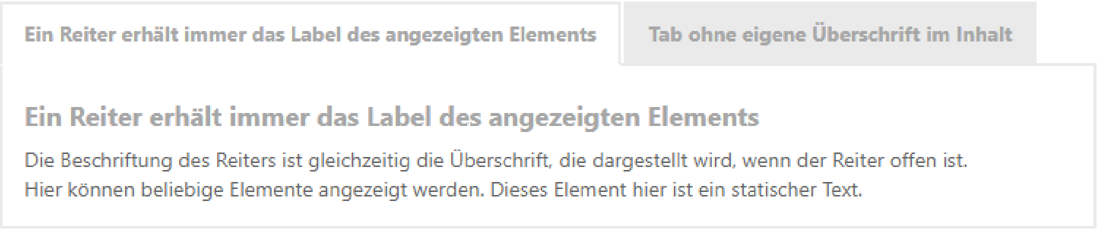
To create a window title or footer panel, you have to select the dialog panel in the panel tree, and click on the icon for creating sub-configurations . Following this, a selection window appears in which the entry “Window title” or “Footer” can be selected. Depending on the panel type of the dialog panel, additional subelements can also be created in this way. These, however, then refer to the content area of the dialog box.

3.6. Viewconfig elements

3.6.1. General

3.6.2. Alternative

An alternative view is a collective view for other views. That is, this type of view can be used to group views that show data for a shared object (e.g. a Properties view with the life data of an artist or a table view that lists the works of the artist). Unlike in a layout view, the summarized views are not shown simultaneously, but instead in alternating order (e.g. via tabs).

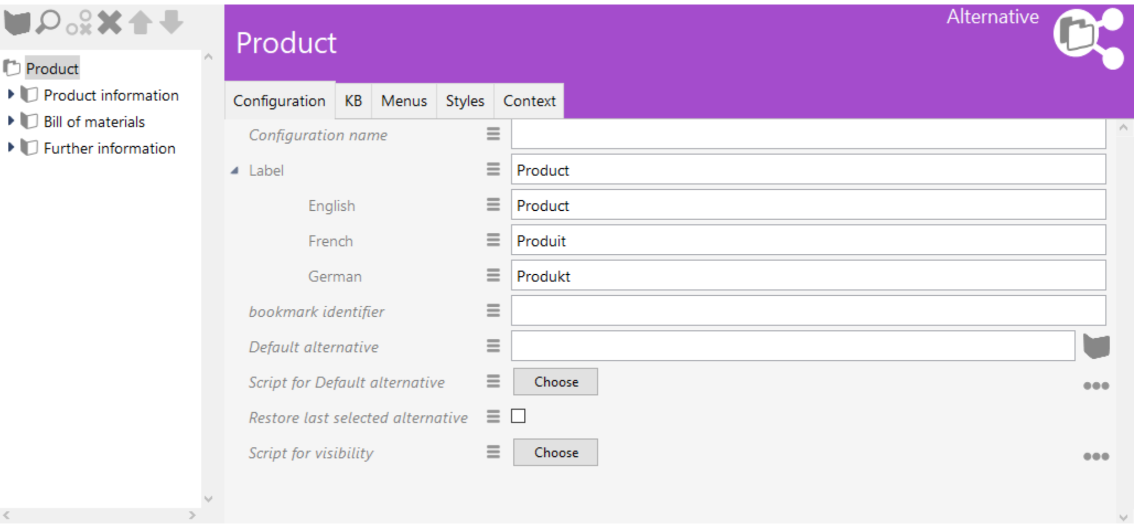


Ein Reiter erhält immer das Label des angezeigten Elements | Tab ohne eigene Überschrift im Inhalt

Ein Reiter erhält immer das Label des angezeigten Elements

Die Beschriftung des Reiters ist gleichzeitig die Überschrift, die dargestellt wird, wenn der Reiter offen ist.
Hier können beliebige Elemente angezeigt werden. Dieses Element hier ist ein statischer Text.

To group views, the corresponding views are appended to the alternative view as subviews. Their position decides the order in which they are displayed. Hence, the arrow buttons can be used to change their positions.



Product

Configuration name:
Label: Product
English: Product
French: Produit
German: Produkt
bookmark identifier:
Default alternative:
Script for Default alternative: Choose
Restore last selected alternative: □
Script for visibility: Choose

The “Configuration” and “Extended” tabs feature options for specifying the general display of the list:

Configuration name	The configuration name can be used to identify views and panels.
Label	The value entered here appears as the heading of the alternative

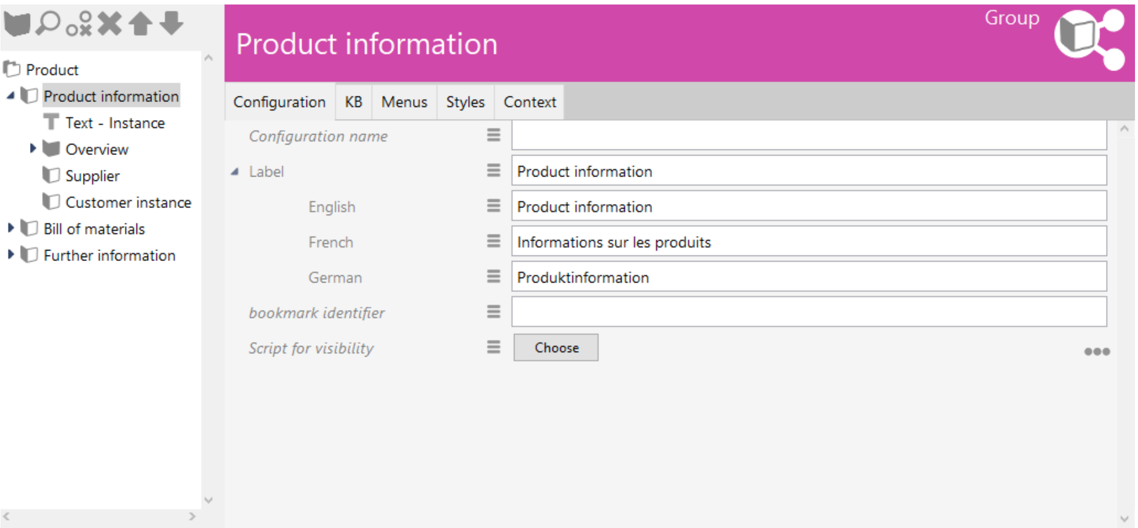
Default alternative	By default the first attached view is displayed. If you prefer the view on the third tab to be displayed first, for example, you can specify this view here. The front-end remembers the last displayed view within a session, so that the user always lands on the tab they looked at most recently if they look at one alternative view several times within a session.
Restore last selected alternative	
Script for label	As an alternative to the “Label,” the title of the alternative can be determined in a script.
bookmark identifier	
Script for Default alternative	
Script for visibility	This script is used to define whether the alternative should be displayed, and under what conditions.

Actions can be configured for the alternative in the “Menus” tab, while the “Styles” tab allows certain display options to be selected. The “KB” tab features options that only apply to Knowledge Builder and are not used in the web front-end. The “Context” tab can be used to configure for which object types the alternative view is to be used and in which application contexts.

An alternative view should be used when several views are based on the data of an object or type, but are to be displayed not simultaneously but alternatively.

3.6.3. Layout

A layout view is a collective view for other views. That is, this type of view can be used to group views that show data for a shared object (e.g. a properties view with the life data of an artist or a table view that lists the works of the artist). To group views, the corresponding views are appended to the layout view as subviews. Their position decides the order in which they are displayed. Hence, the arrow buttons can be used to change their positions.



Product information

Configuration name:

Label: Product information

English: Product information

French: Informations sur les produits

German: Produktinformation

bookmark identifier:

Script for visibility: Choose

The “Configuration” and “Extended” tabs feature options for specifying the general display of the layout:

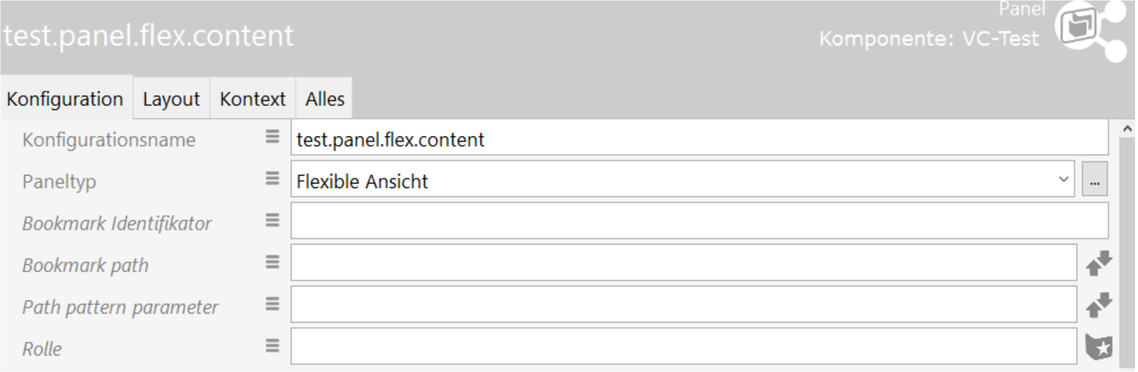
Configuration name	The configuration name can be used to identify views and panels.
Label	The value entered here appears as the header of the layout
Script for label	As an alternative to the “Label,” the title of the layout can be determined in a script.
Orientation	Determines the orientation of the sub configuration elements in the web frontend.
Resizable	When enabled, a slider is displayed in the web frontend that allows the user to resize the area of the layout view.
Boomark identifier	
Role	A view role is used to link an action to a corresponding view.
Script for visibility	This script can be used to specify whether the layout is supposed to be displayed.

The “Menus” tab lets you configure actions for the layout, while the “Styles” tab lets you select certain display options. The “KB” tab features options that only apply to the Knowledge Builder and are not used in the web front-end. The “Context” tab can be used to configure for which object type the layout view is to be used and in which application contexts.

A layout view is to be used when several views, which are based on the data of an object or type, are to be displayed simultaneously and grouped. In contrast to this, there is the alternative that displays the contained views for an object alternatingly (e.g. as tabs).

3.6.4. Flexible view

A flexible view determines its content dynamically based on the underlying model. Unlike layouts, flexible views do not have a direct subconfiguration. If a view is linked to the flexible view via the 'Apply to' relation, this view is displayed whenever the flexible view has a knowledge graph element that matches this view as a model. Whether a view matches a knowledge graph element is controlled via the 'Apply to' relation of the view.



test.panel.flex.content

Panel

Komponente: VC-Test

Konfigurationsname: test.panel.flex.content

Paneltyp: Flexible Ansicht

Bookmark Identifikator

Bookmark path

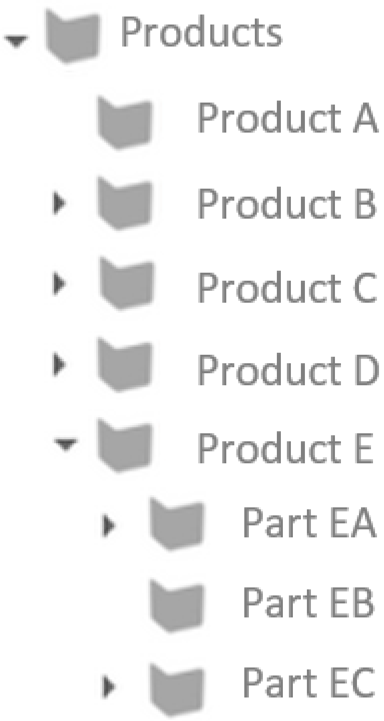
Path pattern parameter

Rolle

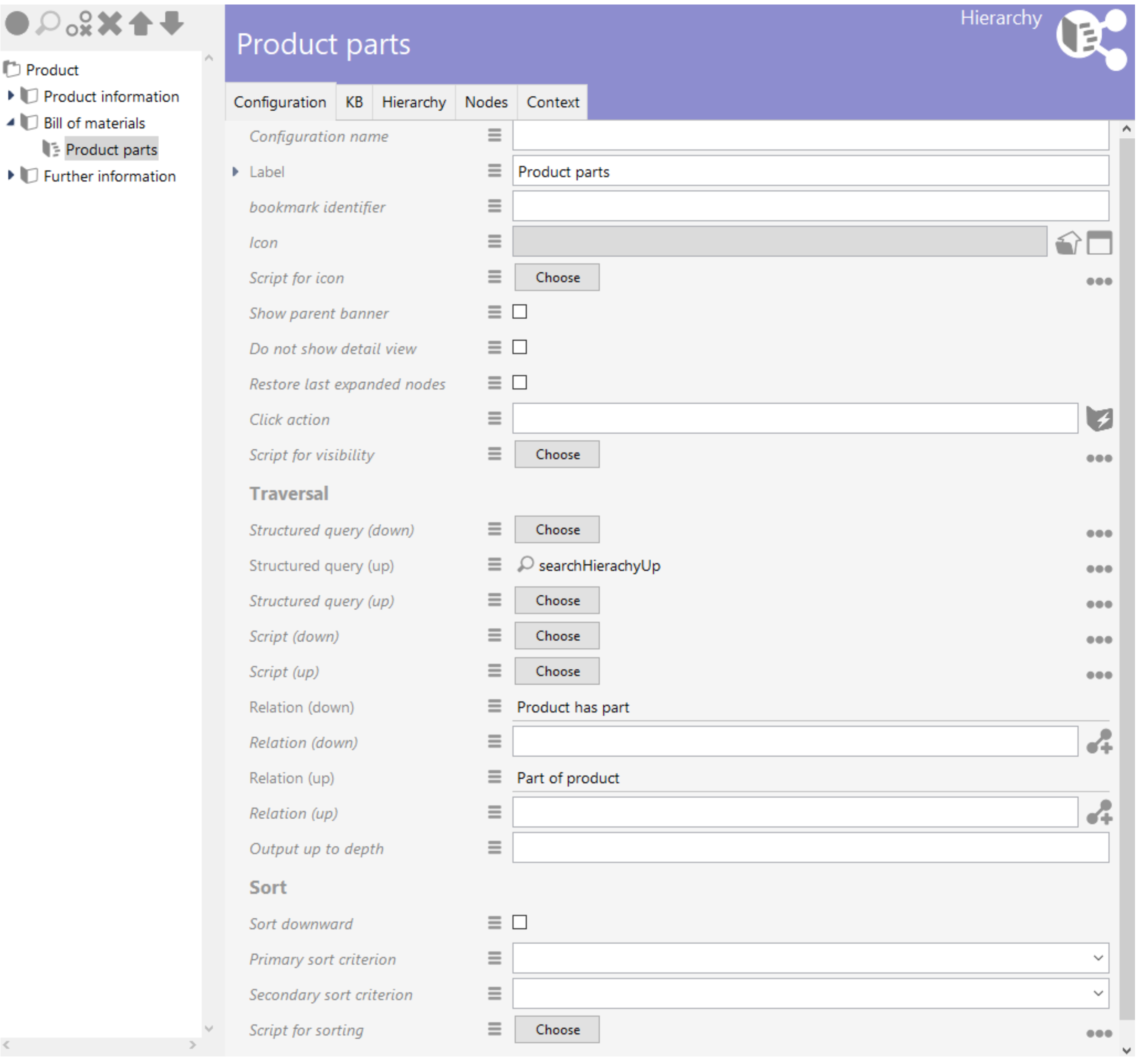
For more complex rules for determining the content of the flexible view, the detector system can be used.

3.6.5. Hierarchy

A hierarchy view is a hierarchical representation of the configurable aspects of an object.



The configuration is performed in the Knowledge Builder by creating a hierarchy view.



Product parts

Configuration name
Label: Product parts
bookmark identifier
Icon
Script for icon: Choose
Show parent banner: □
Do not show detail view: □
Restore last expanded nodes: □
Click action
Script for visibility: Choose

Traversal

Structured query (down): Choose
Structured query (up): searchHierachyUp
Structured query (up): Choose
Script (down): Choose
Script (up): Choose
Relation (down): Product has part
Relation (down)
Relation (up): Part of product
Relation (up)
Output up to depth

Sort

Sort downward: □
Primary sort criterion
Secondary sort criterion
Script for sorting: Choose

The “Configuration” tab provides options for determining the general display of the hierarchy:

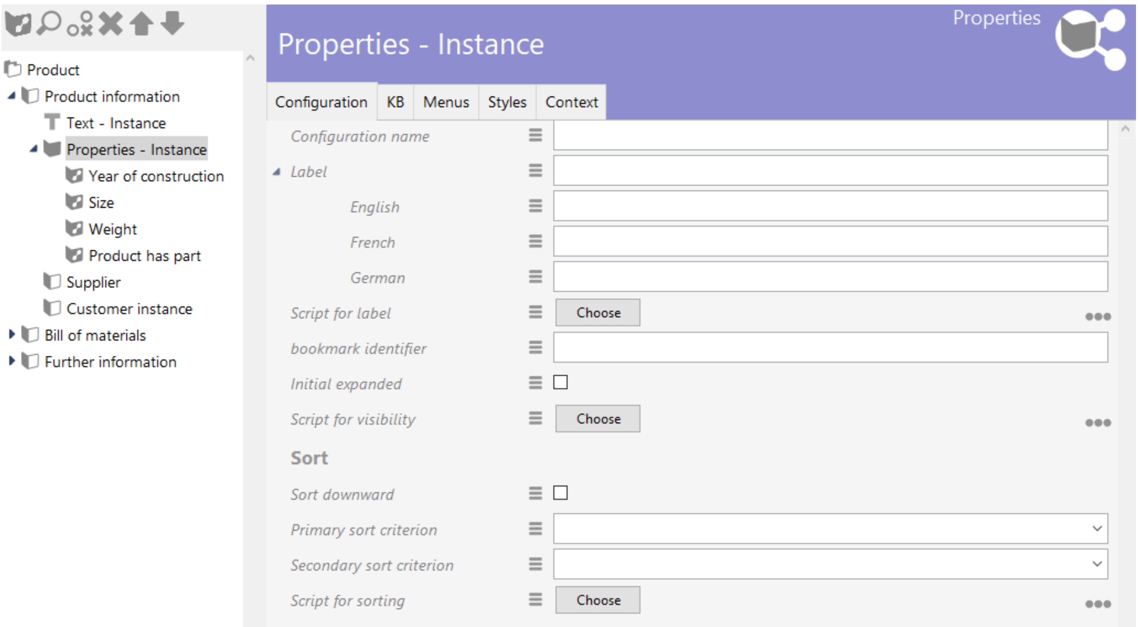
Configuration name	The configuration name can be used to identify views and panels.
Label	The value entered here appears as the heading of the hierarchy
bookmark identifier	
Script for icon	
Show parent banner	
Do not show detail view	
Restore last expanded nodes	
Click action	
Script for visibility	This script is used to define whether the list should be displayed.

Structured query (down) Structured query (up) Script (down) Script (up) Relation (down) Relation (up)	The hierarchy view starts with an **object as the basis**. This object is passed to the hierarchy either by the context element on the higher-level panel or by influencing it from another panel.Which nodes and branches should be shown for this object can be configured in both ascending and descending order. A relation defined in the Knowledge Graph can be selected as a connection between the nodes, however a structured query or even a script can too. A combination of these three types is possible, i.e. it is possible to specify a relation in a descending order, for example, and a structured query in an ascending order. Specifying both directions in optional, however it is also possible to configure the ascending order or the descending order only. In the first case, the object on which the hierarchy is based would be the node at the bottom. And in the second case, the base object of the hierarchy would then be the root node of the hierarchy.
Output up to depth	
Sort downward	The hierarchy is sorted in ascending order by default. Activating the checkbox reverses this sort order.
Primary sort criterion	The sort criterion is used to determine the aspect used to sort the hierarchy elements on one level.
Secondary sort criterion	Like “Primary sort criterion,” except this is only used if the position computed from “primary sort criterion” is the same for two or more attributes.
Script for sorting	This script is used if “Script for sorting” was selected as the primary or secondary sort criterion.
Disallow manual sorting	This option is used to disable the option of allowing the user to re-sort a hierarchy. This option is only used in the Knowledge Builder.

It is possible to configure actions and styles on the entire hierarchy, or to only apply them at node level. This is why there is a “Hierarchy” tab with the sub-items “Menus” and “Styles” and a “Nodes” tab with the same subitems. Actions can be configured for the list in the “Menus” tab, while the “Styles” tab allows certain display options to be selected. The “KB” tab features options that only apply to the Knowledge Builder and are not used in the web front-end. The “Context” tab can be used to configure for which object types the hierarchy view is to be used and in which application contexts.

3.6.6. Properties

A Properties view is a list of the attributes and relations of an object.



Properties - Instance

Configuration name

Label

English

French

German

Script for label

bookmark identifier

Initial expanded □

Script for visibility

Sort

Sort downward □

Primary sort criterion

Secondary sort criterion

Script for sorting

The “Configuration” tab features options for specifying the general display of the list:

Configuration name	The configuration name can be used to identify views and panels.
Label	The value entered here appears as the heading of the list
Script for label	As an alternative to the “Label,” the title of the list can be determined in a script.
bookmark identifier	
Initially expanded	If there are a great many properties, they are not displayed directly in the Knowledge Builder, but instead in expandable form. Activating this option expands them directly.
Script for visibility	This script is used to define whether the list should be displayed.
Sort downward	Generally the contained attributes/relations are displayed in the order specified by the order of the included property view. As it is however possible to specify higher-level types (e.g. “User relation”) here, the properties grouped in this way are sorted by name in ascending order. You can change this order by activating the “Sort downward” check-box.
Primary sort criterion	Generally the contained attributes/relations are displayed in the order specified by the order of the included property view. This option can be used to change this behavior. The available values are “Position”, “Script for sorting” and “Value”. In case of “Value”, sorting is performed by attribute value, and not by the name of the attribute.

Secondary sort criterion	Like “Primary sort criterion,” except this is only used if the position computed from “primary sort criterion” is the same for two or more attributes.
Script for sorting	This script is used if “Script for sorting” was selected as the primary or secondary sort criterion.

Actions can be configured for the list in the “Menus” tab, while the “Styles” tab allows certain display options to be selected. The “KB” tab features options that only apply to the Knowledge Builder and are not used in the web front-end. The “Context” tab can be used to configure for which object types the Properties view is to be used and in which application contexts.

Actions can be configured for the list in the “Menus” tab, while the “Styles” tab allows certain display options to be selected. The “KB” tab features options that only apply to the Knowledge Builder and are not used in the web front-end. The “Context” tab can be used to configure for which object types the Properties view is to be used and in which application contexts.

For the read view, the Properties view can be used on its own, but it is often also used in layout or alternative views. In order to allow object properties to be modified, a Properties view must be included in an Edit view.

The attributes and relations to be displayed for an object can be configured. For that purpose, it is necessary to add property views to the Properties view which can be used to select the relevant attribute/relation and determine in detail how these should be displayed.

3.6.6.1. Styling of a property view

Für individuelle Eigenschaften-Konfigurationen kann es vorkommen, dass die Aufteilung des Layouts geändert werden muss, weil für eine darin befindliche Eigenschafts-View andere Platzverhältnisse benötigt werden (Label vs. Eigenschaftswert). Dies lässt sich durch eine Anpassung mit einem neuen Style unter "Style" > "Viewconfiguration-Mapper" > "class" erreichen.

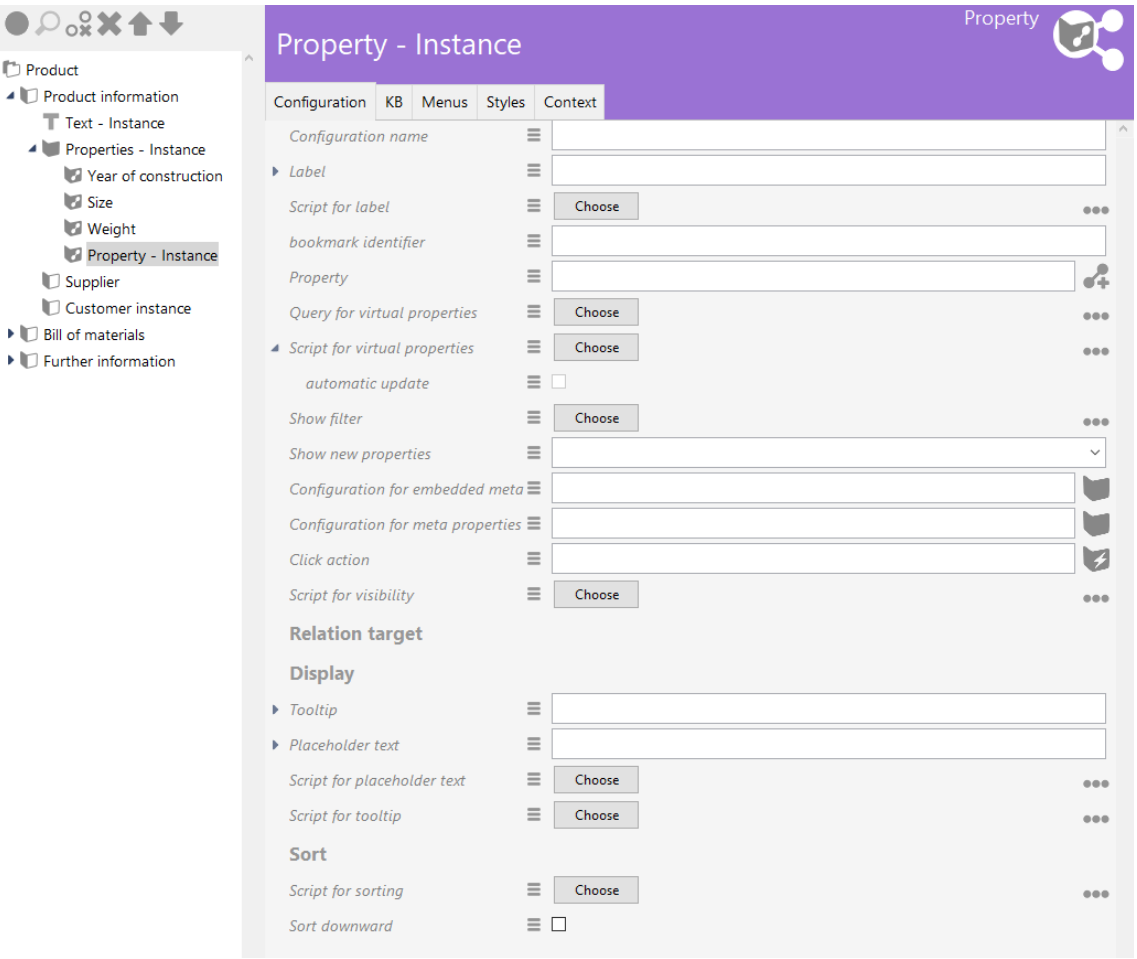
Für den "class"-Eintrag gibt es die Klasse "list", die die Aufteilung zwischen Label und darzustellendem Eigenschaftswert bestimmt. Voreingestellter Wert ist "list-5-6". Die Eigenschaften-View ist in ein gedachtes Raster von zwölf Einheiten unterteilt, wobei die letzte Einheit für die Aktion an einer Eigenschaft reserviert ist. Daraus ergibt sich ein Eintrag mit "list-N-M", wobei N+M = 11 ist. N steht für die Breite des Labels, M steht für die Breite des Eigenschaftswerts.

Wenn beispielsweise das Label einer untergeordneten Eigenschaft aufgrund der Benennung mehr Platz benötigt, kann unter "class" der Wert "list-8-3" eingegeben werden.

Wenn das Label gar nicht dargestellt werden soll und durch die Option "hide label" deaktiviert ist, kann unter "class" der Wert "list-0-11" eingegeben werden.

3.6.7. Property

A Property view is a display configuration of an attribute or a relation to an object. A Property view can only be used underneath a Properties view.

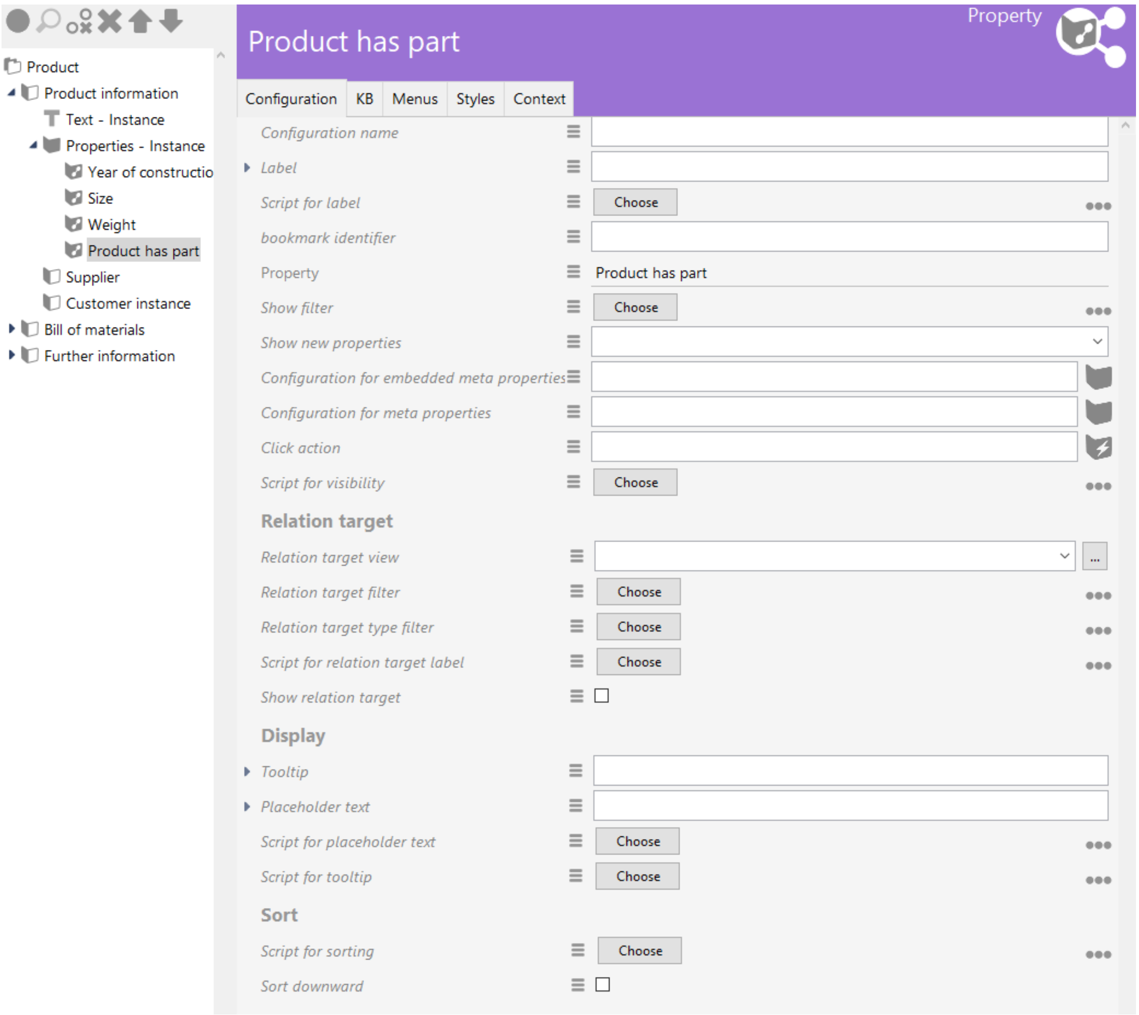


Configuration name	The configuration name can be used to identify views and panels.
Label	The value entered here appears as the heading of the list
Script for label	As an alternative to the “Label,” the title of the list can be determined in a script.
bookmark identifier	
Property	
Query for virtual properties	
Script for virtual properties (automatic update)	
Show filter	
Show new properties	Like “Primary sort criterion,” except this is only used if the position computed from “primary sort criterion” is the same for two or more attributes.
Configuration for embedded meta properties	

Configuration für meta properties	
Click action	
Tooltip	
Placeholder text	
Script for placeholder text	
Scipt for tooltip	
Script for visibility	This script is used to define whether the list should be displayed.
Script for sorting	This script is used if “Script for sorting” was selected as the primary or secondary sort criterion.
Sort downward	Generally the contained attributes/relations are displayed in the order specified by the order of the included property view. As it is however possible to specify higher-level types (e.g. “User relation”) here, the properties grouped in this way are sorted by name in ascending order. You can change this order by activating the “Sort downward” check-box.

Actions can be configured for the list in the “Menus” tab, while the “Styles” tab allows certain

There are additional options for relations:



Product
Product information
Text - Instance
Properties - Instance
Year of constructio
Size
Weight
Product has part
Supplier
Customer instance
Bill of materials
Further information

Product has part
Property

Configuration | KB | Menus | Styles | Context

Configuration name
Label
Script for label: Choose
bookmark identifier
Property: Product has part
Show filter: Choose
Show new properties
Configuration for embedded meta properties
Configuration for meta properties
Click action
Script for visibility: Choose

Relation target
Relation target view
Relation target filter: Choose
Relation target type filter: Choose
Script for relation target label: Choose
Show relation target: □

Display
Tooltip
Placeholder text
Script for placeholder text: Choose
Script for tooltip: Choose

Sort
Script for sorting: Choose
Sort downward: □

Relation target view	By default, a link or relation target editor is displayed in edit mode. However, it can make sense to display e.g. a drop-down list with pre-filtered relation targets instead. These alternative views can be configured here.
Relation target filter	To assist users with their selection of a suitable relation target, a filter query can be placed here.
Relation target type filter	If several object types have been defined as the target of a relation, a filter on the displayed types can be configured at this point.
Script for relation target identifier	By default, the name of the relation target object is displayed. This can be adapted here by means of a script.
Show relation target	

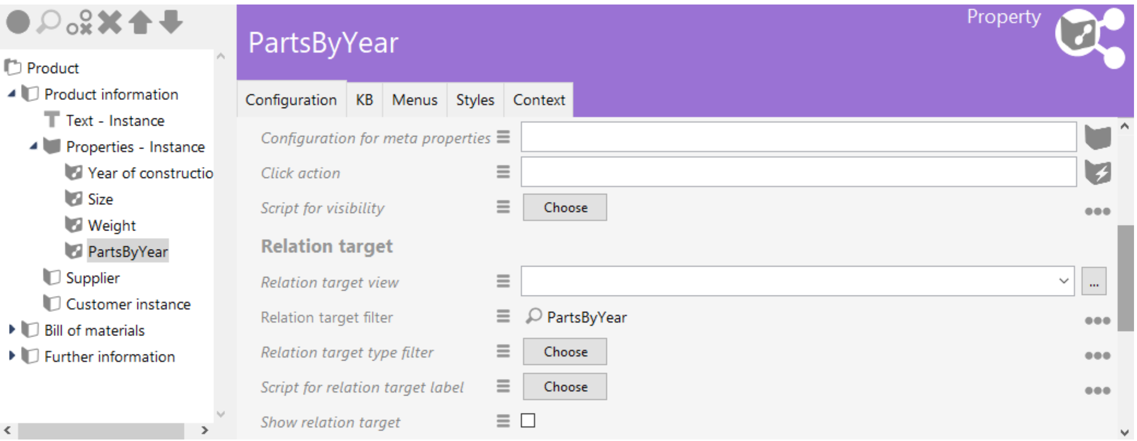
In the “Menus” tab, you can configure additional actions for the property, while the “Styles” tab lets you select certain display options. The “KB” tab features options that only apply to the Knowledge Builder and are not used in the web front-end. You can use the “Context” tab to trace in which view the Property view is used.

3.6.7.1. Relation target filter

To support the user in finding the suitable relation target, a query can be defined for filtering possible relation targets by means of the option "Relation target filter". When the user clicks on the magnifier symbol, a filtered amount of relation targets will be shown.

Example:

A user wants to select product parts by year as a relation target. If only certain products (with parts used at a certain year) need to be presented in the relation target selection, the query for filtering possible relation targets must comprise these conditions.



PartsByYear

Configuration for meta properties

Click action

Script for visibility: Choose

Relation target

Relation target view

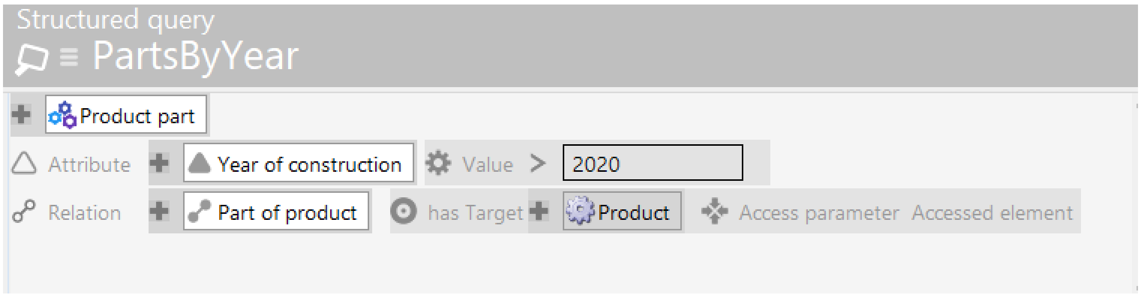
Relation target filter: PartsByYear

Relation target type filter: Choose

Script for relation target label: Choose

Show relation target: □

In the query, the accessed element (product) for specifying the conditions can be identified as usual.



Structured query
PartsByYear

Product part

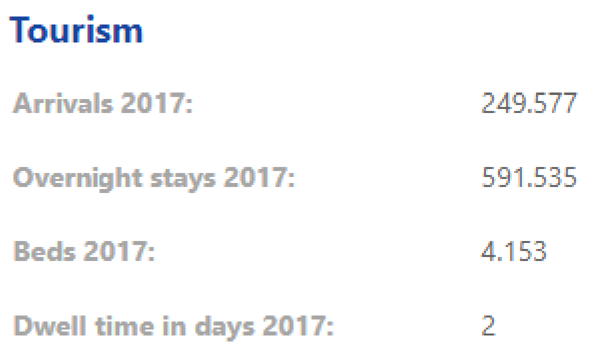
Attribute: Year of construction — Value > 2020

Relation: Part of product — has Target: Product — Access parameter: Accessed element

By standard, relation targets are shown in a simplified table, listed by their name. If a more detailed table is needed, it can be configured and assigned to the property view (in this example "PartsByYear") via the relation "apply in".

3.6.7.2. Styling of a property view

A property in a properties-list is displayed by default as follows:



Tourism

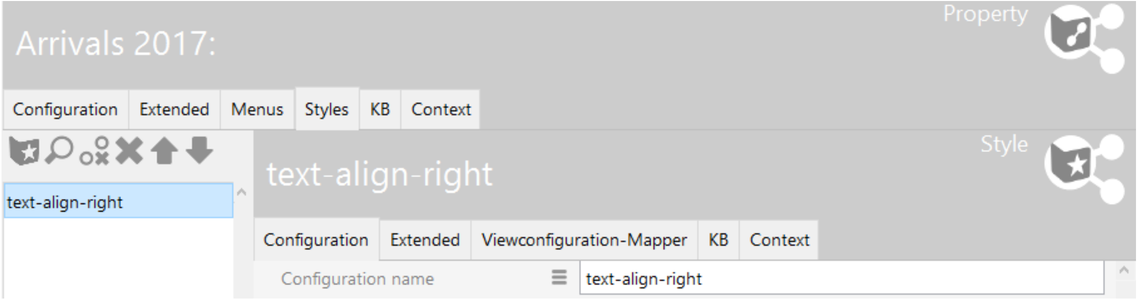
Arrivals 2017:	249.577
Overnight stays 2017:	591.535
Beds 2017:	4.153
Dwell time in days 2017:	2

The label of a property is on the left side and the value is on the right side. As all view configurations a property view can be styled, too. In the following you can see how to style a property with an example.

For example, if you want to display the values right-aligned, you must first create the appropriate css class:

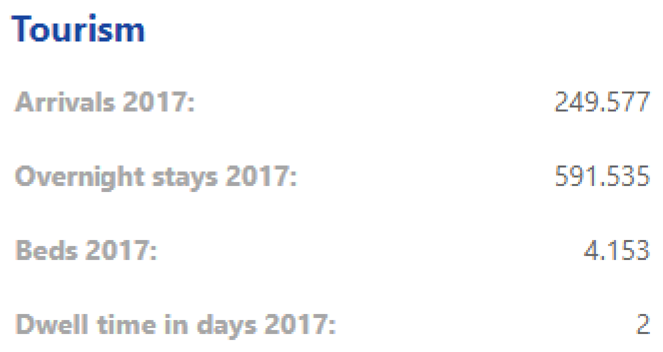
```
.text-align-right .property-value {text-align: right;}
```

This must then be passed as style to the individual properties for which this class should apply:



text-align-right

Configuration name: text-align-right



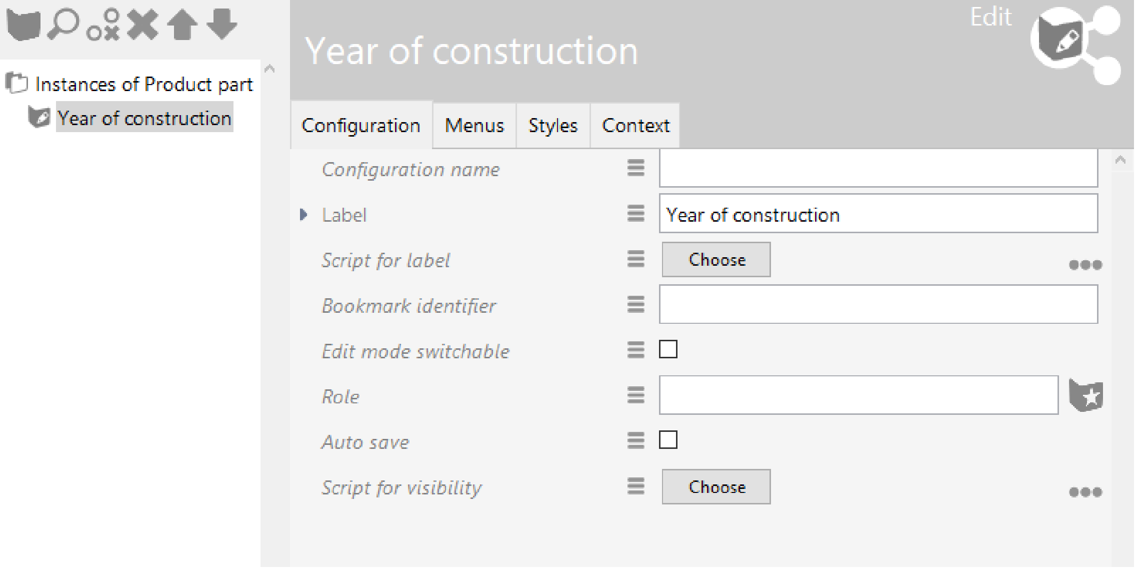
Tourism

Arrivals 2017:	249.577
Overnight stays 2017:	591.535
Beds 2017:	4.153
Dwell time in days 2017:	2

The result of the four styled properties

3.6.8. Edit

And edit view is used to manage user modification of attributes or relations.



Instances of Product part
Year of construction

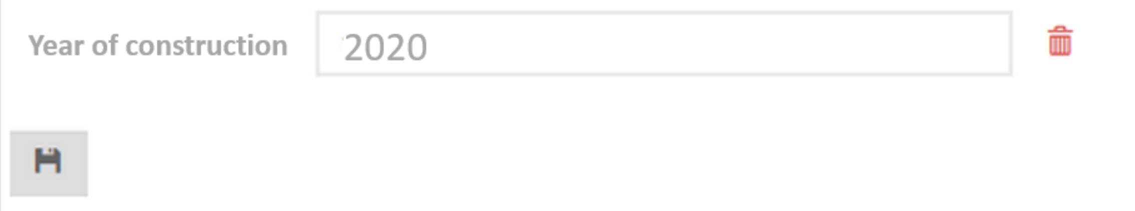
Year of construction

Edit

Configuration | Menus | Styles | Context

Configuration name:
Label: Year of construction
Script for label: Choose
Bookmark identifier:
Edit mode switchable: □
Role:
Auto save: □
Script for visibility: Choose

In the process, all child configurations of the properties type are displayed as form fields. An edit view can contain exactly one child view, which is either a properties configuration or a structuring view (layout, alternative) containing properties configurations. Changes can be synchronized with the Knowledge Graph by means of a Save button.



Year of construction: 2020

The “Configuration” tab features options for specifying the general display of the edit view:

Role

In order to use custom buttons outside the same panel (e. g. within the footer panel of a dialog), the view role can be used to assign the actions of custom buttons for the edit view. For this purpose, a menu with actions must be configured and its actions must be interrelated via the view role to the edit view.If no custom role is specified, the implicit role of an edit view is "edit".

Auto save	This option is available since i-views 5.4. It is also known as "Micro-edit" and enables the automatic saving of changes being made, without the need for a button press (meaning: without the need to trigger an action of the action type "save"). **NOTE** To avoid low performance and erratic behavior of property edits, the option "Auto save" should not be used in combination with a long running transaction. Since a transaction leads to new entries being added onto the web frontend session stack each time a save action is triggered, the performance decreases due to increasing data amounts transferred back and forth.
Edit mode switchable (not available anymore)	Since i-views 5.4 and on, this option is not available anymore. This option enables the form mode to be "switchable". That means, Properties are first shown in read mode only. A Switch button then allows the user to switch to edit mode.
Only custom buttons (not available anymore)	Since i-views 5.4 and on, the option "Only custom buttons" is not available anymore. Instead, every button (except for the entry delete buttons) needs to be configured. For example, a button with an action of the action type "Save" must be configured for saving actions if the option "Auto save" is not enabled.

Layout of property groups in an edit view

If a different layout is needed for edit views, there are following possibilities:

- Several *Properties* views can be arranged underneath an *Edit* view by means of an intermediate *Layout* view. This allows horizontal or vertcal orientation of input elements.
- The pattern of label vs. value can be modified so that, for example, the label gets more space. This is done by applying a style onto the properties view, containing a class reference "list-n-m", whereas n+m = 10.

NOTE

In contratst to the properties view used without an edit view, the properties view used within an edit allows a layout pattern of **10 units** in total instead of **12 units** . When using 12 units for n+m in list-n-m, the edit might be scattered.

3.6.9. Form inputs

Form input views serve for retrieving user input values which are independent from the existence of a semantic element. The input of the form input fields can be fetched and processed by means of an action using a script, e. g. by saving as an attribute value, or used as the input of a search compound to provide a search with parameters.

It should be noted that in contrast to Edit views, an action with the action type "Save" has no effect on form views and will not persist the values.

The following form input types are available:

Form input for	Value capturing
Boolean	Checkbox for input of boolean values
Date and time	A date picker that can be customized to accept either a date, a time with hours and minutes, or both
Number	A number spinner that can be customized to accept either integers or decimals
Choice	Specification of a script which returns an array of character strings or semantic elements for selection in forms of a drop-down entry. The selection/display can be preset with an initial value using the script.
String	Input field for character strings
Input with proposals	An input field that assists the user by proposing suitable values.

Reading out form inputs using actions or scripts

To process the values from the input fields, an action or a script needs a way for addressing them. An action can either be located in a menu at the form input itself or at a different location, whereas a reference needs to be set up from the action to the form input by means of role assignment ("perform by"). A role assigned to a form input can also be used by a script to access the corresponding view, which will provide the input value.

NOTE

When using a view role, the identifier of the role must not contain any whitespace. Since one view can have several roles, the roles are processed in a whitespace-separated form. A single view role with an identifier containing a whitespace-separated string therefore would be misinterpreted as several roles, leading to errors.

Following application scenarios are possible:

1. Reading out **multiple form inputs under a common layout** using a single action, but process each entry individually: Relating the action via a role to the form view, addressing each individual value via an individual role by means of "this.viewsWithRole(*roleName*)[0].value()".
2. Reading out only **one form input** : Relating an action via a role to the form entry view, addressing the entry value by means of "this.value()".
3. Reading out **all form inputs** at once - provided the values being of the same type or the order of values is not important: Relating the action via a role to the layout view, assigning one role to all form inputs and addressing all values at once by means of "this.viewsWithRole(*roleName*)", then processing the array items.

NOTE

Addressing all form inputs by assigning one role to all form inputs and the action by means of "this.value()" will not work since roles must be uniquely assignable in the web frontend.

Example:

A layout view contains the form inputs for "choice", "boolean" and "string".

- If an action only needs to access one dedicated form input, e. g. the string input field, the input view gets a role called "inputField" and the action is related to the role "inputField" via the relation "perform by". Then the action of the action type "Script" gets a custom script called "Script (custom)". In every case, "this" is the view the action is located in or - if a role is assigned - it is the view interrelated via the role. The value of the input field is then read out by means of "this.value()".
- If all three entries need to be read out individually by means of one action at once, the action needs to be related to the layout view via a role ("form") and the individual form inputs each get their own role. To address the input field again in this case, the action is related to the "form" role via the entry/relation "perform by". The action has the action type "Script" and the "Script (custom)". Now, "this" is the layout view. To access the input field within the script, the view with the assigned role "inputField" needs to be addressed. The value of the input field is then read out by means of "this.viewsWithRole('inputField')[0].value()". Since "viewsWithRole" returns an array, the one and only input field view is the first (one and only) array element with the index number 0.

Forms as input for search compounds

Forms can also be used to act as parameter input for a search. To achieve that, a form input needs to be be linked to a search compound by the "Input of" relation. Furthermore, the "Parameter name" must be configured. It determines which search parameter the form input relates to. If the form input is marked as "Required" and no user input is provided, the search will not be executed. Otherwise, the search parameter is deactivated if the input is empty. For further details refer to the chapter on Search compounds.

Input validation

Form inputs can be marked as "Required". In this case the input field receives a designation to indicate to the user, that an input is mandatory. This also influences how a missing parameter is handled if the form input is part of a search compound (see above).

The second validation mechanism is the "Script for validation" that can be configured for any form input. Consider the following example for a script that validates the input of a number field:

```
function validateFormValue(value) {
    if (value < 0 || value > 10) {
        this.setValidationErrorMessage('Only values between 0 and 10 are allowed.')
        return false
    }
    return true
}
```

The validation error message is presented to the frontend user who can then correct his input. If the validation script returns "false", the invalid value will not be accessible by any scripts and will also not be passed to the query, if the form input is part of a search compound.

It is also possible to access the values of other form inputs by using any of the methods described above. It should be taken care though that there is no circular dependency between the validation of multiple form inputs. In that case they will not be able to access each other’s values and will receive undefined instead.

Input with proposals

For an input with proposals, there are two ways to configure how values should be derived from user input. When specifying a "Query for proposed values", the user will be presented with the query results. To make the query results dependand on the current user input, the predefined parameter "searchString" can be used. Alternatively, a "Script for proposed values" can be provided:

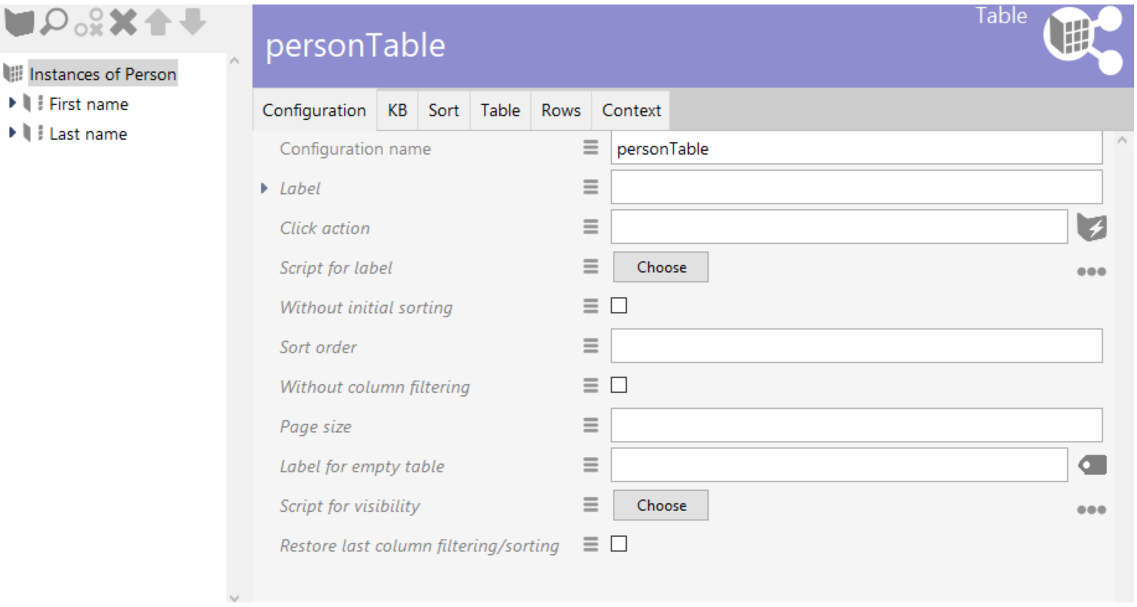
```
function valueProposals(searchString) {
  return [
    // a function applied to the user input
    new $k.TypeAheadProposal(searchString.toUpperCase()),
    // a static number with label
    new $k.TypeAheadProposal(42, 'forty-two'),
    // a semantic element of the knowledge graph
    new $k.TypeAheadProposal($k.Registry.element('myElement'))
  ]
}
```

In any case, there are two additional configuration options:

- Threshold: Defines the number of characters the user must type, before the first request for proposals is sent. For expensive queries, this value should be chosen sufficiently high to reduce performance impact. The default threshold value is 3.
- Restrict input to proposals: If this checkbox is set, the user will not be able to submit any value that was not proposed to him. Otherwise, the user is free to edit the chosen proposal before submitting.

3.6.10. Table

A table view is a display configuration of a list of objects. A table view can be used independently at different points and its content depends on the context.



Instances of Person
First name
Last name

personTable
Table

Configuration | KB | Sort | Table | Rows | Context

Configuration name: personTable
Label:
Click action:
Script for label: Choose
Without initial sorting: □
Sort order:
Without column filtering: □
Page size:
Label for empty table:
Script for visibility: Choose
Restore last column filtering/sorting: □

The “Configuration” tab features options for specifying the general display and behavior.

Action (selection)	The action configured here is executed if a row is selected in the front-end (e.g. by clicking).
Number of rows (page size)	This specifies the maximum number of rows that are displayed on one page.
Automatic search	Options: * Automatic search * Automatic search up to limit * No automatic search
Label	A table is displayed with the heading in the front-ends. By default, the name is generated from the context. You can use “Label” to display a value other than the name.
Configuration name	The configuration name can be used to identify views and panels.
Without column filter	Here you can determine whether a column filter is supposed to be displayed between the table header and table content. The column filter can be used to filter the query result for the column by entering a term.
Script for label	Instead of using the “Label,” the displayed attribute name can be determined in a script.
Table of	This references the view whose results are displayed in the preceding table. This can be a query, of a query result view or another table.

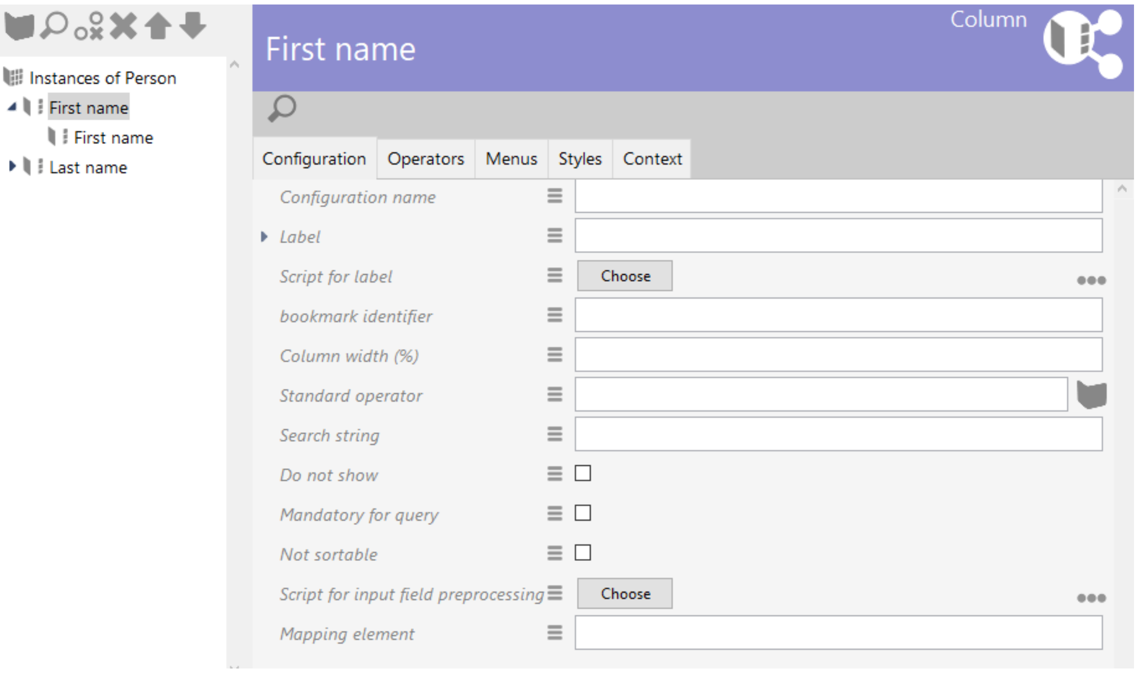
On the “Sorting” tab, you can configure the sort response using the columns.

The “Table” tab has two sub-items: “Menus” and “Styles.” In the “Menus” tab, you can configure additional actions for the table, while the “Styles” tab lets you select certain display options that affect the entire table. In the next tab, “Columns" > "Styles" you then select the display options for columns accordingly.

The columns of the table are defined using sub-configurations, which are explained in the next section. The order of the columns can be changed using arrow buttons in the tree view on the left side.

The column view represents the configuration of an entire column. Here you can influence the display and the response (e.g. filtering).

The content of the cells (“column element”) in turn is defined by the sub-configuration as described in the next section.



Configuration name

Label

Script for label

bookmark identifier

Column width (%)

Standard operator

Search string

Do not show □

Mandatory for query □

Not sortable □

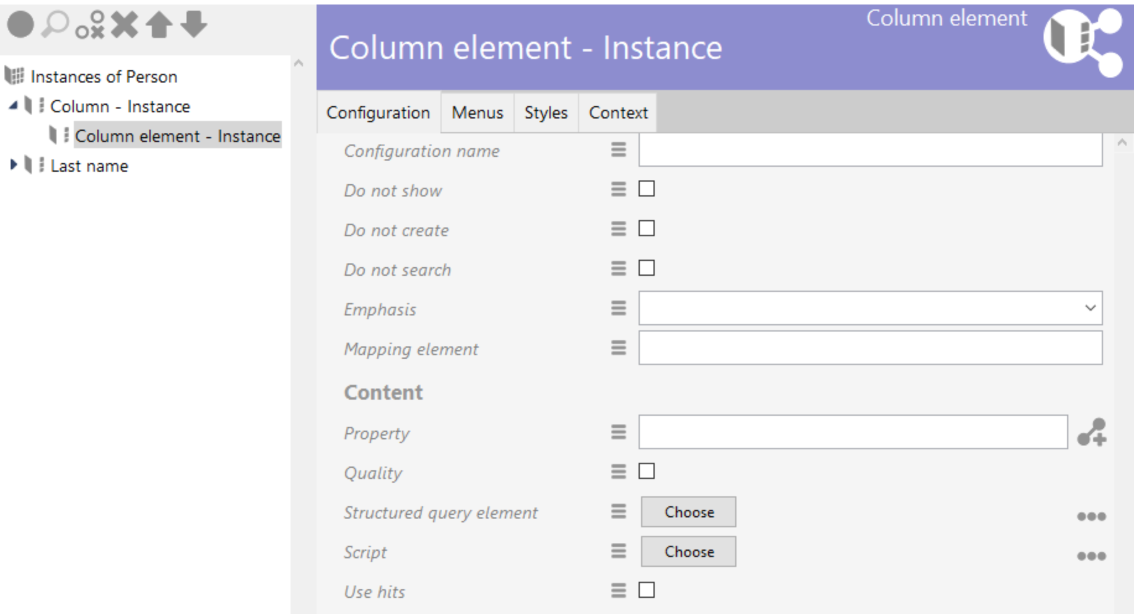
Script for input field preprocessing

Mapping element

Configuration name	The configuration name can be used to identify views and panels.
Label	Column name displayed
Script for label	Instead of using the “Label,” the displayed attribute name can be determined in a script.
bookmark identifier	
Column width (%)	Width of the column in percent of the width of the table
Standard operator	This is where the default is selected from the possible filter operators If nothing is configured, the first one in the list is selected.
Search string	

Do not show	This is used to hide a column. It is nonetheless calculated in the background and can be used e.g. for sorting.
Mandatory for query	
Not sortable	In the default setting, the columns can be sorted by clicking on the header. This function can be deactivated here.
Script for preprocessing input fields	The text that was specified in the column filters can be influenced via a script here.
Search text	The text for column filtering can be specified in advance here.

The column element sub-configuration determines the content of the column. The content is typically derived from the elements to which this table refers.



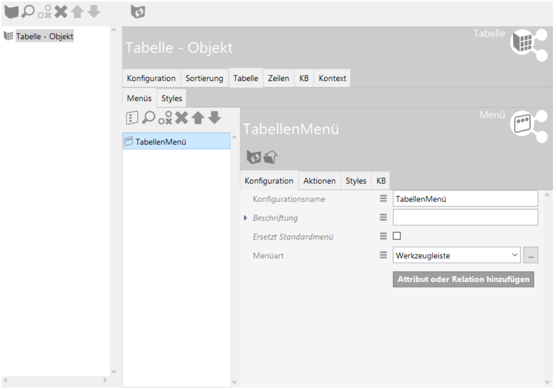
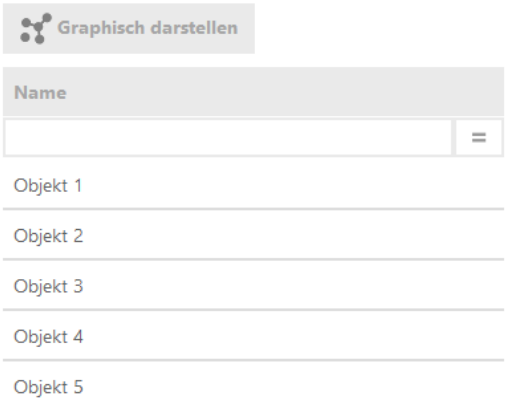
Configuration name
Do not show □
Do not create □
Do not search □
Emphasis
Mapping element
Property
Quality □
Structured query element
Script
Use hits □

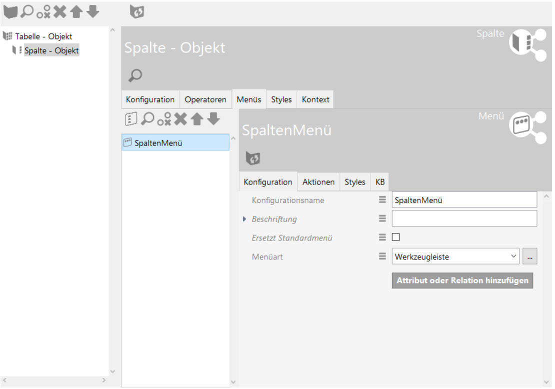
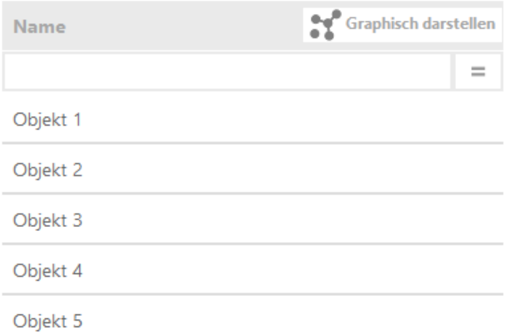
Configuration name	The configuration name can be used to identify views and panels.
Do not show	This is used to hide the column element. This is nonetheless calculated in the background and can be used e.g. for sorting or filtering.
Do not create	
Do not search	
Emphasis	This lets you choose if the content of the column element is to be highlighted by underlining it.
Mapping element	
Property	The property of the element to be displayed in this column
Quality	

Structured query element	As an alternative to “Property,” the content to be displayed can also be determined using a structured query.
Script	As an alternative to the first two method, the content to be displayed can also be derived from the element via a script.
Use hits	Allows the use of all meta properties of a search result (“hit”), such as quality, cause etc.If the search results are processed further by a script, JavaScript object $k.SemanticElement or $k.Hit is forwarded.

3.6.10.1. Menus in tables

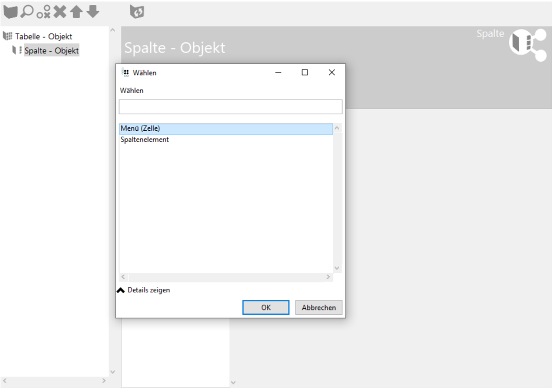
Menus can be configured at different points of a table. The selection of the configuration location determines whether a menu is available for the entire table, for the column of the table or for every column element:

Configuration location	Menu with actions for the element
Table: "Table" tab > “Menu” tab  Tabelle - Objekt Konfiguration · Sortierung · Tabelle · Zeilen · KB · Kontext Menüs · Styles TabellenMenü Konfiguration · Aktionen · Styles · KB Konfigurationsname: TabellenMenü Beschriftung: Ersetzt Standardmenü: □ Menüart: Werkzeugleiste Attribut oder Relation hinzufügen	Actions for the entire table:  Graphisch darstellen Name Objekt 1 Objekt 2 Objekt 3 Objekt 4 Objekt 5

Configuration location	Menu with actions for the element
Column: “Menus” tab  Tabelle - Objekt Spalte - Objekt SpaltenMenü Konfigurationsname: SpaltenMenü Beschriftung: Ersetzt Standardmenü: ☐ Menüart: Werkzeugleiste	Actions are displayed in the *column description* of a table:  Name Graphisch darstellen Objekt 1 Objekt 2 Objekt 3 Objekt 4 Objekt 5

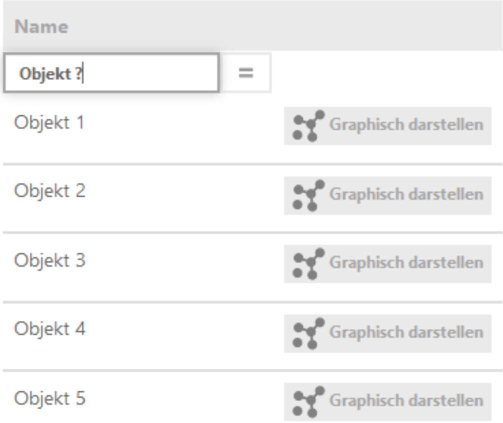
<table>
<tr><th>Configuration location</th><th>Menu with actions for the element</th></tr>
<tr><td>

Column: Menu as a subelement of a column



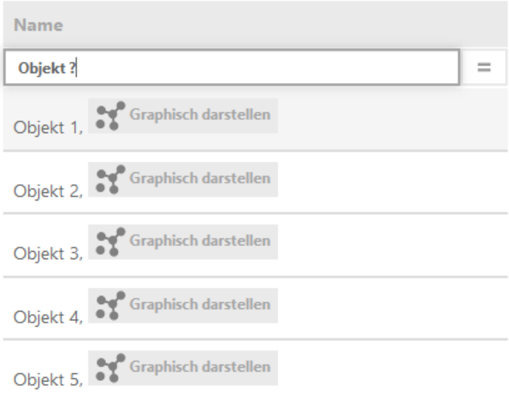
</td><td>

Actions are output *in every row* in a column: Menu in a separate column:



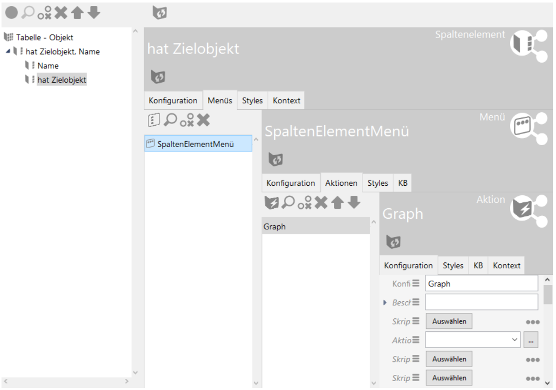
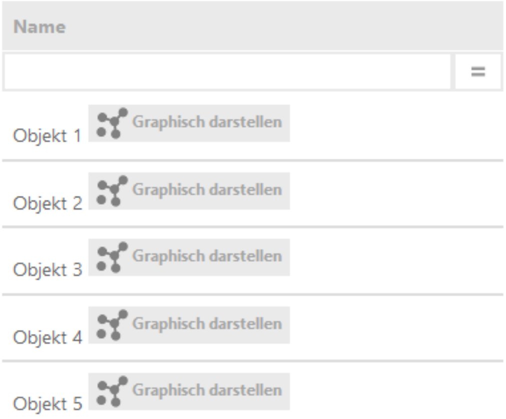
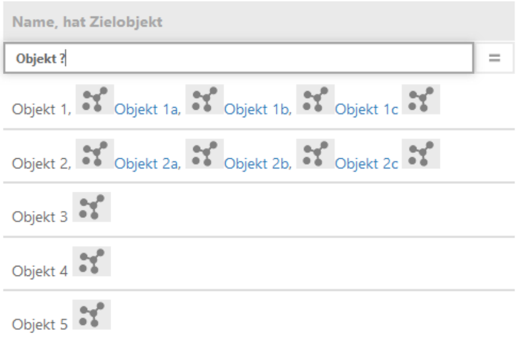
Name
Objekt ?
Objekt 1 Graphisch darstellen
Objekt 2 Graphisch darstellen
Objekt 3 Graphisch darstellen
Objekt 4 Graphisch darstellen
Objekt 5 Graphisch darstellen

Menu element in the same column as the column element to be displayed:



Name
Objekt ?
Objekt 1, Graphisch darstellen
Objekt 2, Graphisch darstellen
Objekt 3, Graphisch darstellen
Objekt 4, Graphisch darstellen
Objekt 5, Graphisch darstellen

</td></tr>
</table>

Configuration location	Menu with actions for the element
Column element: “Menus” tab  Tabelle - Objekt hat Zielobjekt, Name Name hat Zielobjekt Spaltenelement hat Zielobjekt Konfiguration Menüs Styles Kontext SpaltenElementMenü Menü SpaltenElementMenü Konfiguration Aktionen Styles KB Graph Aktion Graph Konfiguration Styles KB Kontext Konfi Graph Beschr Skrip Auswählen Aktio Skrip Auswählen Skrip Auswählen	The action is output after every value: Output for one object per column element:  Name Objekt 1 Graphisch darstellen Objekt 2 Graphisch darstellen Objekt 3 Graphisch darstellen Objekt 4 Graphisch darstellen Objekt 5 Graphisch darstellen Output for several objects per column element, e.g. in the display of target objects of a relation. The target objects are comma-separated (configuration as shown on the left). In this case, you should preferably use icons to save space; alternatively, the label can be replaced with a tooltip (mouse-over display).  Name, hat Zielobjekt Objekt ? Objekt 1, Objekt 1a, Objekt 1b, Objekt 1c Objekt 2, Objekt 2a, Objekt 2b, Objekt 2c Objekt 3 Objekt 4 Objekt 5

NOTE For relation targets, the link to the target object can be suppressed by using the “no link” style attribute.

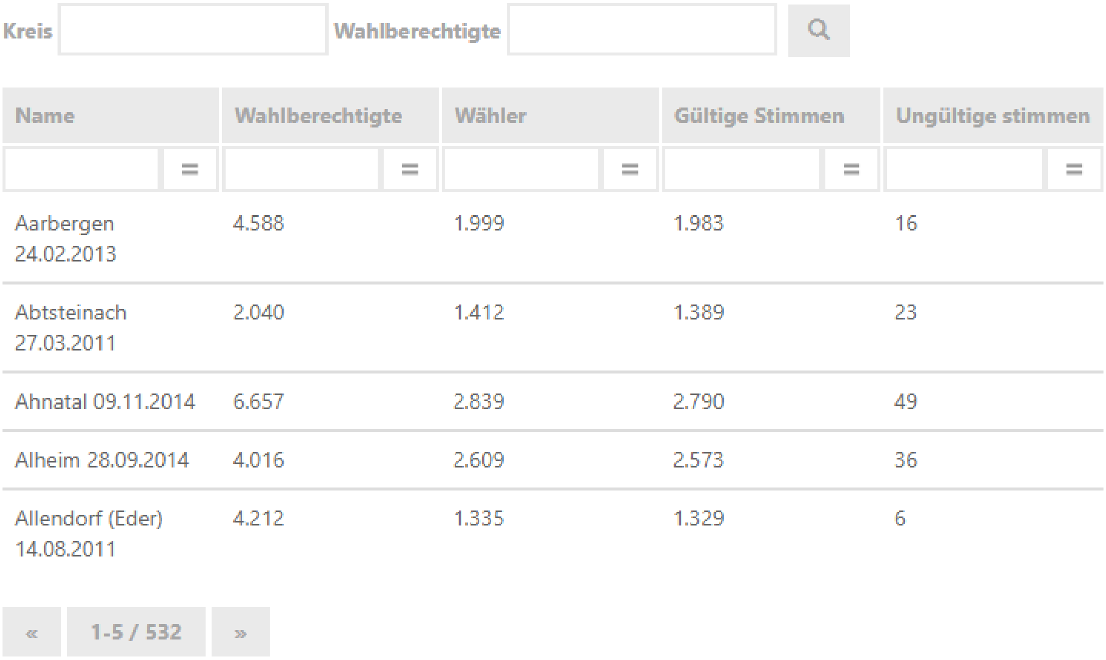
3.6.11. Search

This section describes various views that can be used to implement a search - from the "all in one" search to more complex scenarios with specialized views that are distributed across several panels.

Since version 5.8, the so called search compound has been available, which makes the search field view and the search results view no longer necessary.

3.6.11.1. Search view

A search view allows search pages to be created on which the search query and the search results are displayed at the same time. If the search does not have any parameters, or only optional ones, then the search is run immediately and the results displayed directly. If there are obligatory parameters, then the search is only run following a user input.

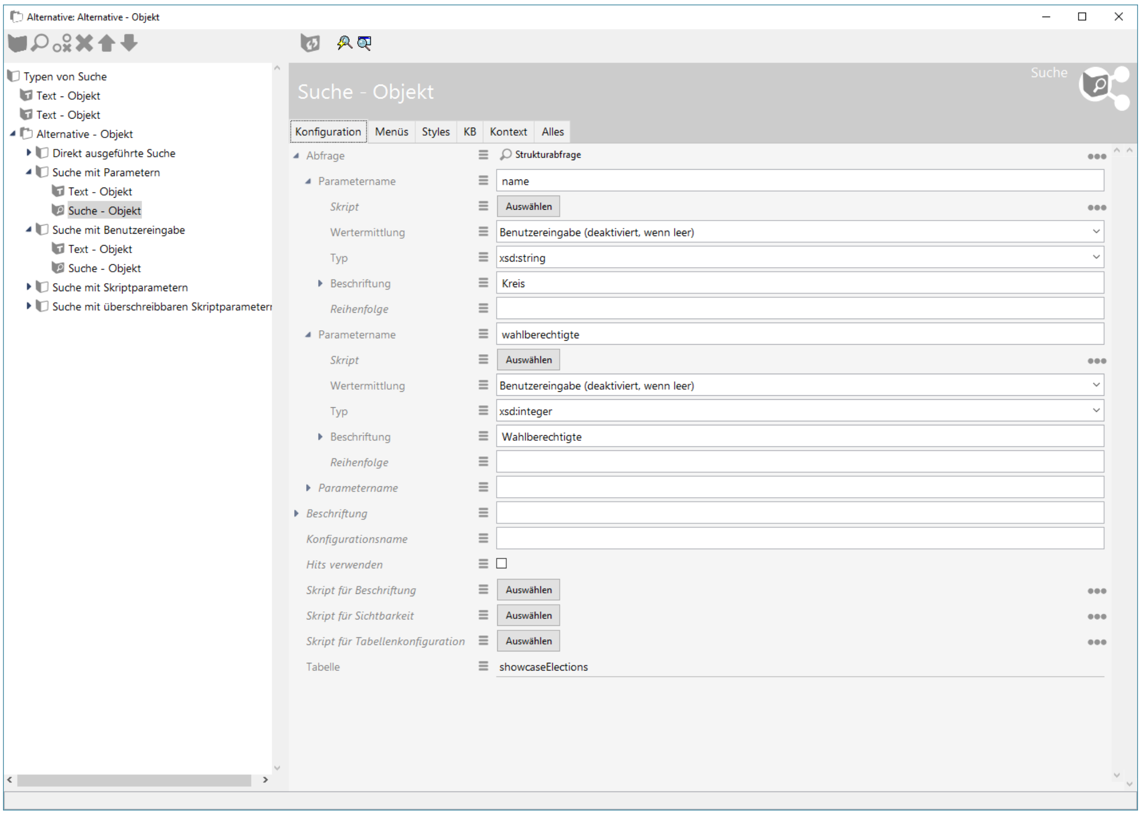


Kreis
Wahlberechtigte

Name	Wahlberechtigte	Wähler	Gültige Stimmen	Ungültige stimmen
=	=	=	=	=
Aarbergen 24.02.2013	4.588	1.999	1.983	16
Abtsteinach 27.03.2011	2.040	1.412	1.389	23
Ahnatal 09.11.2014	6.657	2.839	2.790	49
Alheim 28.09.2014	4.016	2.609	2.573	36
Allendorf (Eder) 14.08.2011	4.212	1.335	1.329	6

« 1-5 / 532 »

A search view is created in the Knowledge Builder for a simple search page.



Abfrage: Strukturabfrage
Parametername: name
Skript: Auswählen
Wertermittlung: Benutzereingabe (deaktiviert, wenn leer)
Typ: xsd:string
Beschriftung: Kreis
Reihenfolge:
Parametername: wahlberechtigte
Skript: Auswählen
Wertermittlung: Benutzereingabe (deaktiviert, wenn leer)
Typ: xsd:integer
Beschriftung: Wahlberechtigte
Reihenfolge:
Parametername:
Beschriftung:
Konfigurationsname:
Hits verwenden: □
Skript für Beschriftung: Auswählen
Skript für Sichtbarkeit: Auswählen
Skript für Tabellenkonfiguration: Auswählen
Tabelle: showcaseElections

The “Configuration” tab provides options for determining the general display of the search:

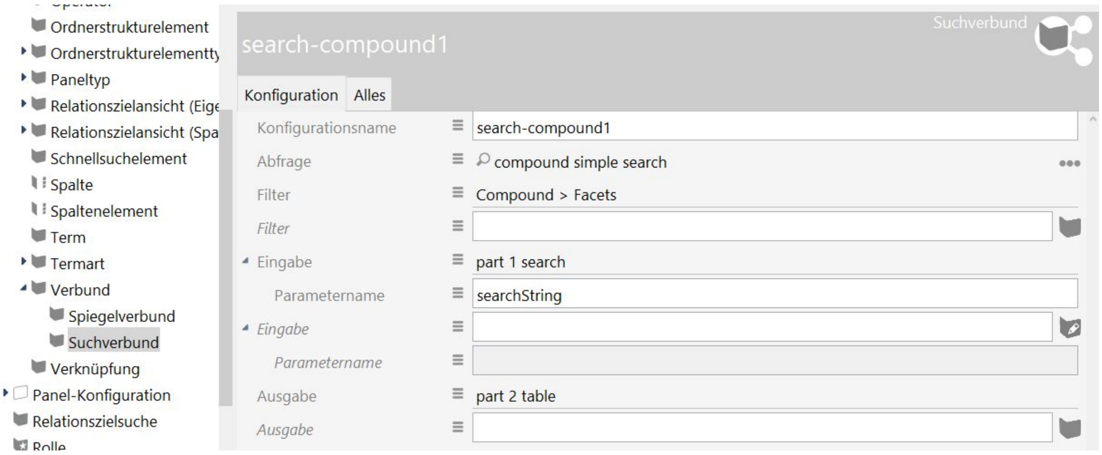
Query	This is where you configure the query that is to be executed when the query is executed.
Parameter name	Name of a search parameter. All parameters that are configured in the search must also be configured at this point to ensure no errors occur in the search.
Script	If the parameter value is to be determined via a script, this has to be configured here.
Value determination	Here you specify how the parameter value is to be determined. * “Script” (value determined via script) * “Script, can be overwritten” (the value is determined via script, but is overwritten by user input on the front-end) * “User input (optional)” (the parameter value is copied from the user input if it is set. It is displayed to the user as optional in the front-end. Please note that the search is then configured in such a way that this parameter does not have to be set) * “User input (obligatory)” (the user must enter a value in the front-end, otherwise the search is not executed) * “User input (deactivated if blank)” (the parameter is set for the search if there was no user input. Otherwise the parameter is deactivated when the search is executed)

Type	Data type of the parameter
Label (Parameter)	Name of the parameter in the front-end
Order	The order in which the parameters are displayed in the front-end
Label (Search View)	The value entered here appears as the heading of the search
Configuration name	The configuration name can be used to identify views and panels.
Use Hits	Determines whether topics or hits are generated.
Script for label	As an alternative to the "Label," the title of the group can be determined in a script.
Script for visibility	This script can be used to specify whether the group is supposed to be displayed.
Script for table configuration	As an alternative to "Table", a script can be used to determine the table displayed at this point.
Table	The search results are displayed in the front-end in the table configuration that is configured here.

Actions can be configured for the search in the "Menus" tab, while the "Styles" tab allows certain display options to be selected. The "KB" tab features options that only apply to the Knowledge Builder and are not used in the web front-end. The "Context" tab can be used to configure for which object types the search view is to be used and in which application contexts.

3.6.11.2. Search compound

For synchronizing the state of multiple search related views use a so called Search compound. A simple search compound consists of a query definition and a table view that is used as its output.



search-compound1

Konfigurationsname: search-compound1
Abfrage: compound simple search
Filter: Compound > Facets
Filter:
Eingabe: part 1 search
Parametername: searchString
Eingabe:
Parametername:
Ausgabe: part 2 table
Ausgabe:

If input is required, one or multiple form input fields can be used, replacing the search field views used in the past. For each input field, the corresponding parameter name that is used in the query definition must be specified. By using the various types of input fields, the user input can be guided, for example by providing a conveniant date input.

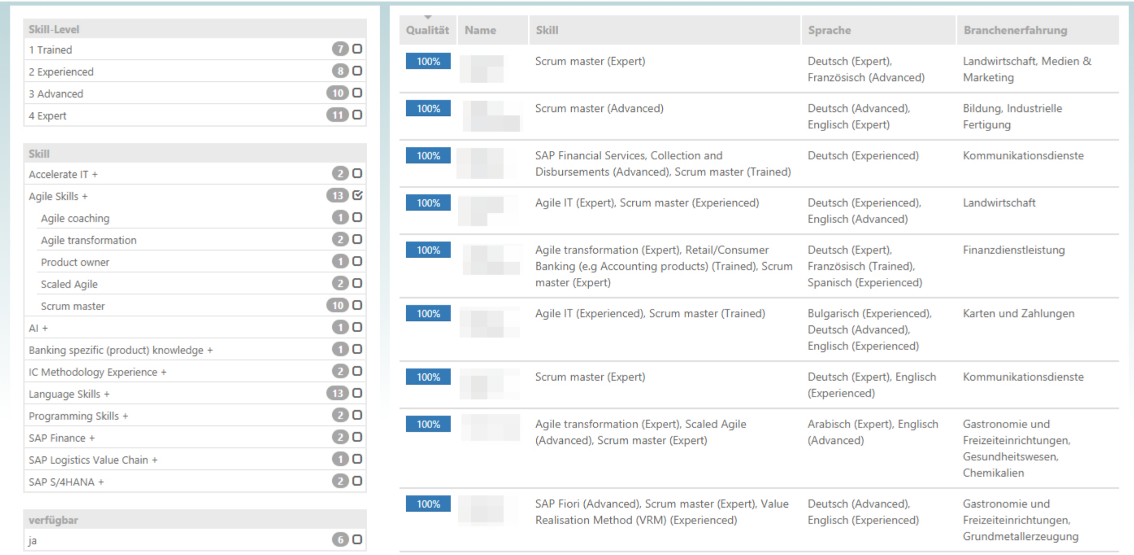
Filtering of search results is done by means of a facet view.

Any number of inputs, outputs and filters can be specified, even views that are not always visible.

New search compounds can be created from the context tab of any of the views involved.

3.6.11.3. Facet view

Display



Skill-Level		
1 Trained	7	□
2 Experienced	8	□
3 Advanced	10	□
4 Expert	11	□

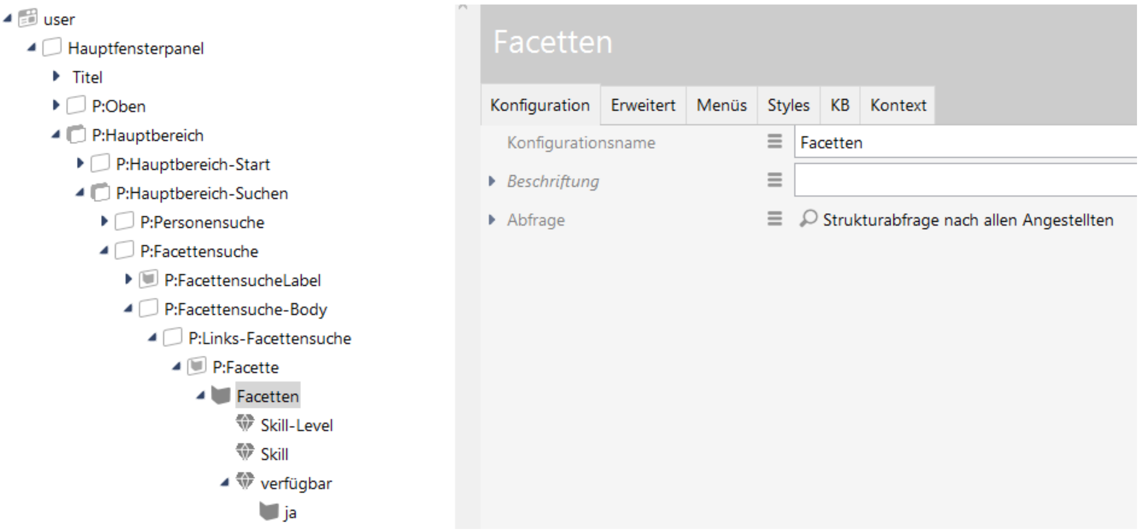
Skill		
Accelerate IT +	2	□
Agile Skills +	13	☑
Agile coaching	1	□
Agile transformation	2	□
Product owner	1	□
Scaled Agile	2	□
Scrum master	10	□
AI +	1	□
Banking spezific (product) knowledge +	1	□
IC Methodology Experience +	2	□
Language Skills +	13	□
Programming Skills +	2	□
SAP Finance +	2	□
SAP Logistics Value Chain +	1	□
SAP S/4HANA +	2	□

verfügbar		
ja	6	□

Qualität	Name	Skill	Sprache	Branchenerfahrung
100%	[redacted]	Scrum master (Expert)	Deutsch (Expert), Französisch (Advanced)	Landwirtschaft, Medien & Marketing
100%	[redacted]	Scrum master (Advanced)	Deutsch (Advanced), Englisch (Expert)	Bildung, Industrielle Fertigung
100%	[redacted]	SAP Financial Services, Collection and Disbursements (Advanced), Scrum master (Trained)	Deutsch (Experienced)	Kommunikationsdienste
100%	[redacted]	Agile IT (Expert), Scrum master (Experienced)	Deutsch (Experienced), Englisch (Advanced)	Landwirtschaft
100%	[redacted]	Agile transformation (Expert), Retail/Consumer Banking (e.g Accounting products) (Trained), Scrum master (Expert)	Deutsch (Expert), Französisch (Trained), Spanisch (Experienced)	Finanzdienstleistung
100%	[redacted]	Agile IT (Experienced), Scrum master (Trained)	Bulgarisch (Experienced), Deutsch (Advanced), Englisch (Experienced)	Karten und Zahlungen
100%	[redacted]	Scrum master (Expert)	Deutsch (Expert), Englisch (Experienced)	Kommunikationsdienste
100%	[redacted]	Agile transformation (Expert), Scaled Agile (Advanced), Scrum master (Expert)	Arabisch (Expert), Englisch (Advanced)	Gastronomie und Freizeiteinrichtungen, Gesundheitswesen, Chemikalien
100%	[redacted]	SAP Fiori (Advanced), Scrum master (Expert), Value Realisation Method (VRM) (Experienced)	Deutsch (Advanced), Englisch (Experienced)	Gastronomie und Freizeiteinrichtungen, Grundmetallerzeugung

Configuration

A facet view can be created as a sub configuration of a panel, but not within another view configuration elements. The panel of the facet view needs to influence the search result panel.



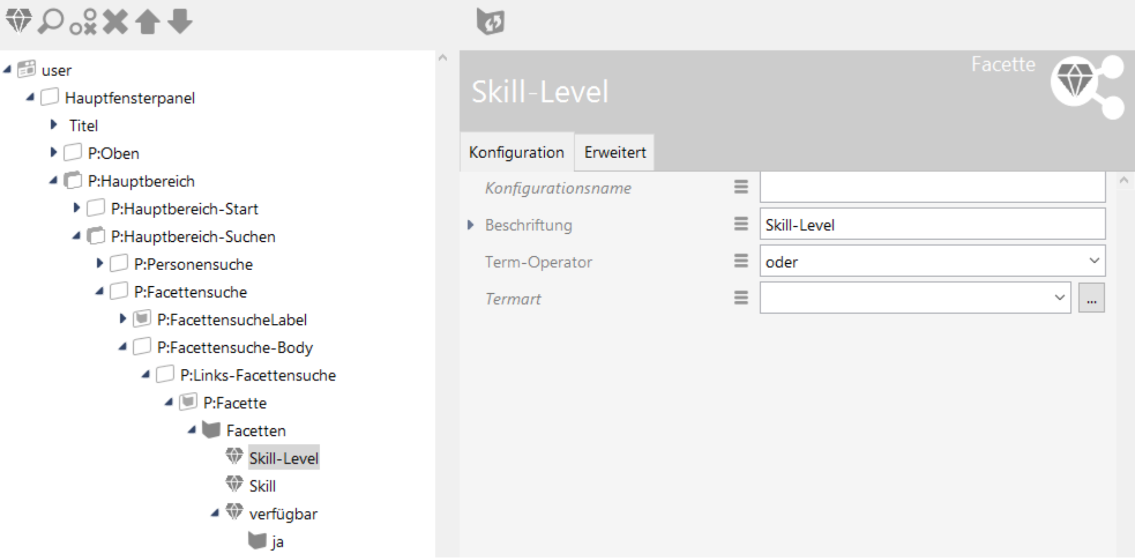
```
user
  Hauptfensterpanel
    Titel
    P:Oben
    P:Hauptbereich
      P:Hauptbereich-Start
      P:Hauptbereich-Suchen
        P:Personensuche
        P:Facettensuche
          P:FacettensucheLabel
          P:Facettensuche-Body
            P:Links-Facettensuche
              P:Facette
                Facetten
                  Skill-Level
                  Skill
                  verfügbar
                    ja
```

Facetten

Konfigurationsname	Facetten
Beschriftung	
Abfrage	Strukturabfrage nach allen Angestellten

Query	Here a query must be configured when the facet view is not linked with a search field view. If, for example, the facets are intended for influencing a search result table containing employees, the query must output the employees as source for the facets. If the facet view is linked to the search field view, no query needs to be defined.
Label	The title to appear above the facet view in the front-end.
Configuration name	Configuration names can be used to identify views and panels.
Script for label	As an alternative to a permanent label, the title can also be set via a script (to be found in the tab "Extended").

In order to configure facets, it is necessary to create facet views and attach them to the facets view. These can be arranged in multiple hierarchical orders.

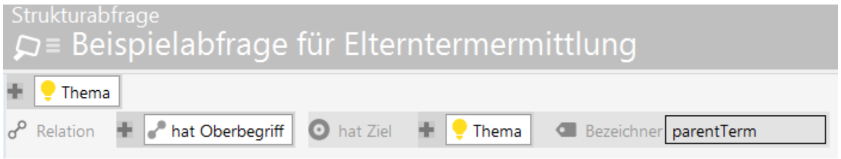
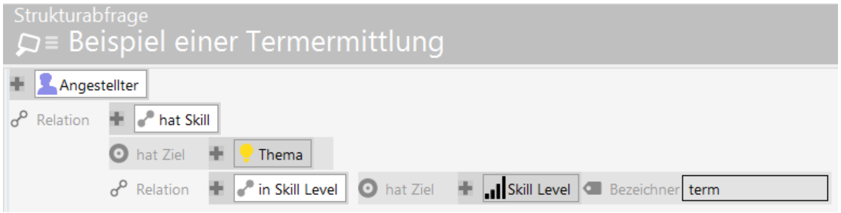


Konfigurationsname

Beschriftung: Skill-Level

Term-Operator: oder

Termart

Query for determining the parent term	In case a term hierarchy is needed, the parent term must be configured by this query. The child element is used as input element here fore. In the query, the label "parentTerm" identifies the parent element.  Strukturabfrage Beispielabfrage für Elterntermermittlung Thema Relation hat Oberbegriff hat Ziel Thema Bezeichner parentTerm **NOTE** - For the facet hierarchy to be able of being built up, the "query for term detection" needs to be configured for comprising both terms and parent terms. The herein contained parent terms are subsequently used for building up the hierarchy by means of the "query for parent term detection". Therefore, testing the queries is advised. - At the moment, only terms of the same type can build a hierarchy. - As usual in hierarchies, you can not display infinite loops.
Query for determining the term	Structured query that is used to form the facet. This query is obligatory when the standard behaviour comes into account or when it is set dynamically (which means that it keeps empty in case of static mode). The query must be specified as follows: For narrowing down the search results, facets can be defined for relation targets. The input element type is equal to the type of the search results from the query of the query view. The terms to be found must be identified by the label "term".  Strukturabfrage Beispiel einer Termermittlung Angestellter Relation hat Skill hat Ziel Thema Relation in Skill Level hat Ziel Skill Level Bezeichner term In principle, everything is possible like in all structured queries. It is also possible that the label "term" is used several times within one structured query. In this case, the behaviour of the terms specified by the values of "Term operator".
Hide from number of terms	The facet is hidden if the search results underlying the facet exceed this number.
Label	Ideally, a label is always specified. If not set, the name of the input element of the query is used.

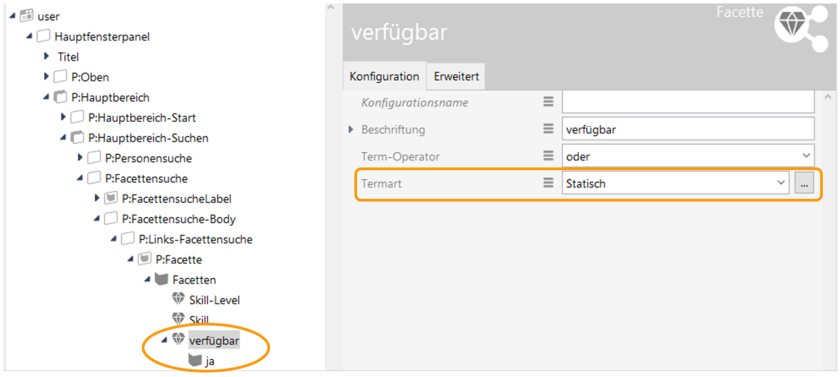
Display child terms initially	If the facet has a hierarchical structure, you can use this option to define whether the sub-facets should be displayed initially. Per default, the child elements are displayed after the parent element has been selected.
Configuration name	Views and panels can be identified via a configuration name.
Display blank terms	If no results are found for the facet, it is hidden by default. This option ensures it is displayed nonetheless.
Maximum number of terms	Describes the maximum number of terms the facet can have. per default, all terms are displayed.
Do not display term number	In the front-end, the number of found terms is displayed right behind the facet title. This option deactivates this.
Term operator	At this point it is possible to configure how the terms are linked to each other. You can use the “And” or the “Or” logic that applies on the search result regarding the selected facets.

Term type

If no term type is selected (default behaviour), the terms will be detected by the query of the facet configuration. In the query, relation targets or attribute values can be defined for terms. Additional to the default behaviour following settings are available:

- Dynamic: The value range of the terms are detected automatically. The values used for term detection must be identified by the label "terrmValue" within the "Query for term detection".
- Static: All terms to be displayed must be configured indiviudally. For every term a query needs to be configured that specifies the possible hits of the main query.

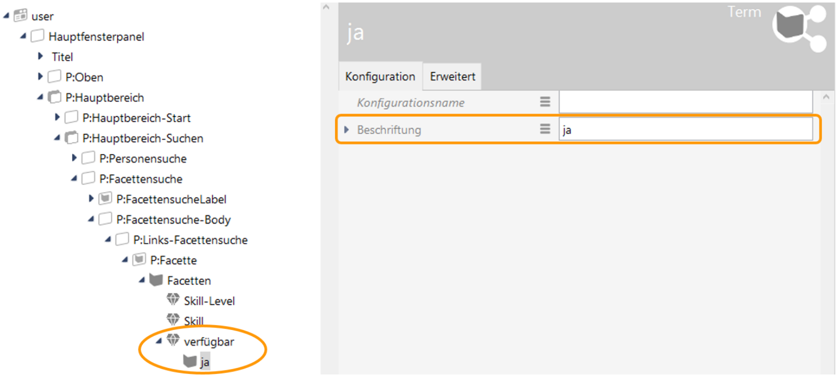
Example of a static facet:



verfügbar

Konfigurationsname:
Beschriftung: verfügbar
Term-Operator: oder
Termart: Statisch

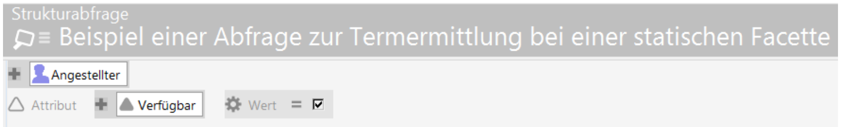
Each term of the facet needs a label for display:



ja

Konfigurationsname:
Beschriftung: ja

The query within the tab "Extended" defines the applicable criteria for the facet:



Beispiel einer Abfrage zur Termermittlung bei einer statischen Facette

Angestellter
Attribut Verfügbar Wert = ☑

Sort terms in descending order

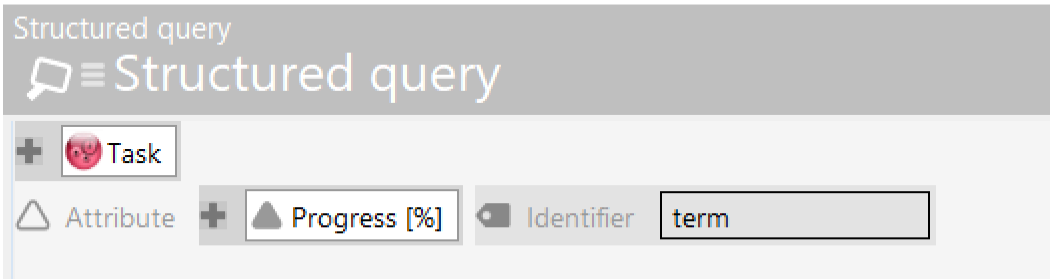
By default, the terms found for a facet are sorted in ascending order. This option reverses the sort order.

Sort terms by number	The facet terms are generally sorted in alphabetical order; with this option, they are sorted by the number of results found.

Faceting for attribute values

Search results can be faceted concerning predetermined attribute values, for which the term type " **static** " must be set. If the term type "static" is chosen, the **terms** must be added as a subconfiguration within a facette by clicking on the button "link new". For this purpose, the configuration is built up as follows:

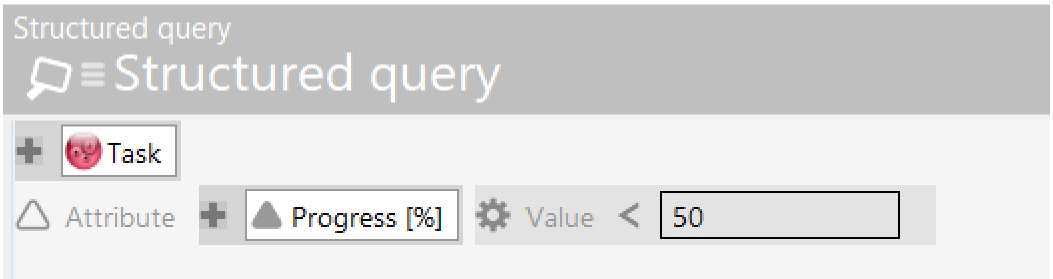
1. As usual, the structured query of the **facette** contains the elements to be filtered, including the identifier "term" at the property:



Identifier: term

Example of a query for term identification with attribute values as terms

2. The facette itself has a subordinate **term** element with a query for a more detailed definition of the terms. The structured query for the terms then only contains the conditions for the properties of the elements. An identifier is not used at this point:



Value < 50

Example of a query of a static term (predetermined attribute value)

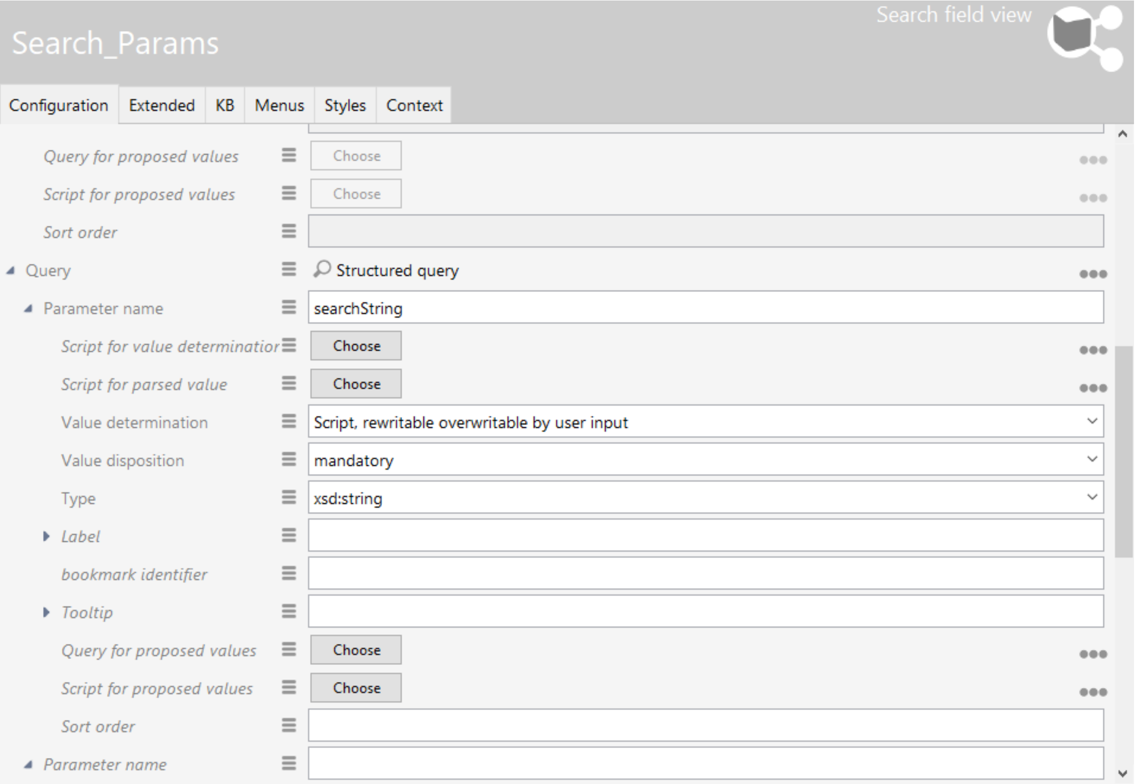
Notes:

- The labeling of the facet term sub-configutaion is obligatory. If no label is set, the facet term will not be displayed.
- For the static term, a term element is needed. If a facet element is used, the facet term will not be displayed either.

3.6.11.4. Search field view

NOTE	Search field views are deprecated in favour of Search compounds in combination with Input fields.

A search field element is used, if only a search slot and no search results is to be displayed in a certain place. Configuration takes place as for the search view but without the configuration for displaying the results.

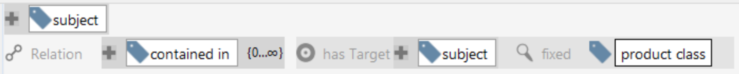


Search_Params

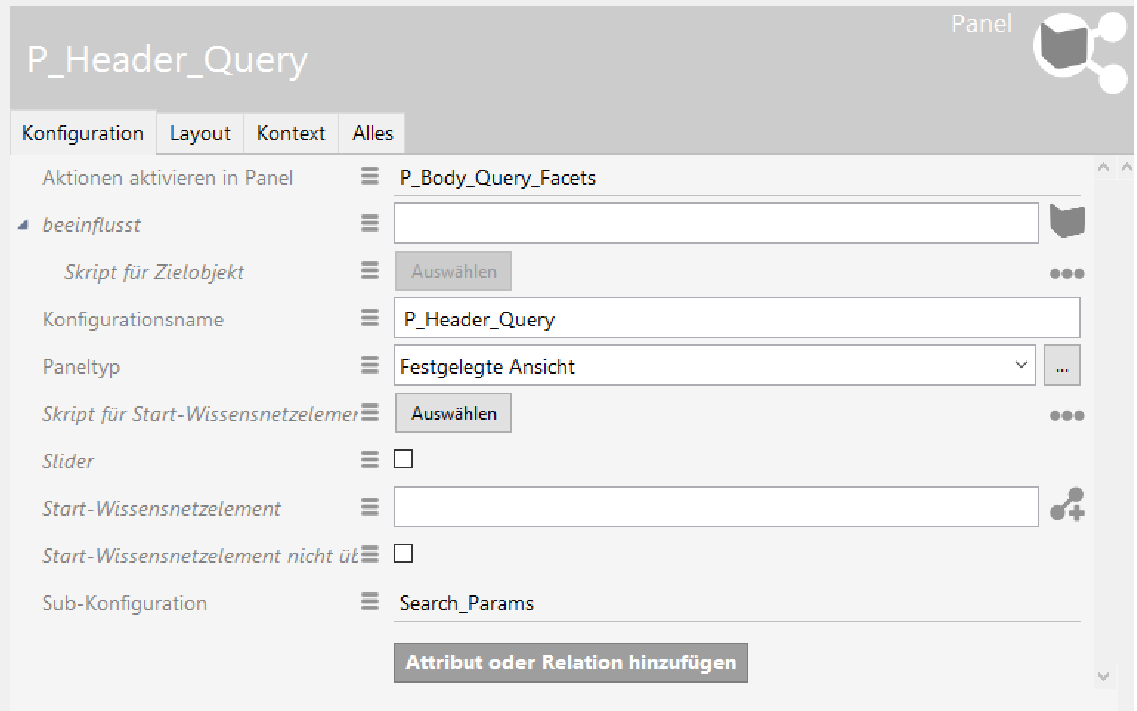
Query for proposed values	
Script for proposed values	
Sort order	
Query	Structured query
Parameter name	searchString
Script for value determination	
Script for parsed value	
Value determination	Script, rewritable overwritable by user input
Value disposition	mandatory
Type	xsd:string
Label	
bookmark identifier	
Tooltip	
Query for proposed values	
Script for proposed values	
Sort order	
Parameter name	

The “Configuration” tab provides options for determining the general display of the search field:

Query	This is where you configure the query that is to be executed when the query is executed.
Parameter name	Name of a search parameter. All parameters that are configured in the search must also be configured at this point to ensure no errors occur in the search.
Script	If the parameter value is to be determined via a script, this has to be configured here.

Value determination	Here you specify how the parameter value is to be determined. • “Script” (value determined via script) • “Script, can be overwritten” (the value is determined via script, but is overwritten by user input on the front-end) • “User input (optional)” (The parameter value is copied from the user input if it is set. It is displayed to the user as optional in the front-end. Please note that the search is then configured in such a way that this parameter does not have to be set) • “User input (obligatory)” (The user must enter a value in the front-end, otherwise the search is not executed) • “User input (deactivated if blank)” (The parameter is set for the search if there was no user input. Otherwise the parameter is deactivated when the search is executed)
Query for proposed values, script for proposed values	Proposed values are possible elements or strings that are offered to users in a list at the search slot. These in turn can be selected as search string input (also known as “type ahead”). For configuration, a query or a script can be placed on the parameter. If a structured query is used, the names of the elements found are displayed as default values on the front-end.  subject Relation contained in {0...∞} has Target subject fixed product class *In this example, only subjects belonging to "product class" would be listed as proposals, represented by their primary name.* In detail, a query allows to define which attributes of the element should be used (it doesn’t have to be the primary name in every case). A search pipeline can be used to combine arbitrary conditions (structured queries) with arbitrary attributes (queries). A search pipeleine needs a 'searchString' parameter for input. A script (see template in the Knowledge Graph) can also be used to deliver labels/strings as fixed values only (that is, without a mandatory reference to the Knowledge Graph). The "elementId" and "iconLocator" keys are optional.
Type	Data type of the parameter
Label	Name of the parameter in the front-end
Order	The order in which the parameters are displayed in the front-end
Label	The value entered here appears as the heading of the search
Configuration name	The configuration name can be used to identify views and panels.
Script for label	As an alternative to the “Label,” the title of the search field view can be determined in a script.

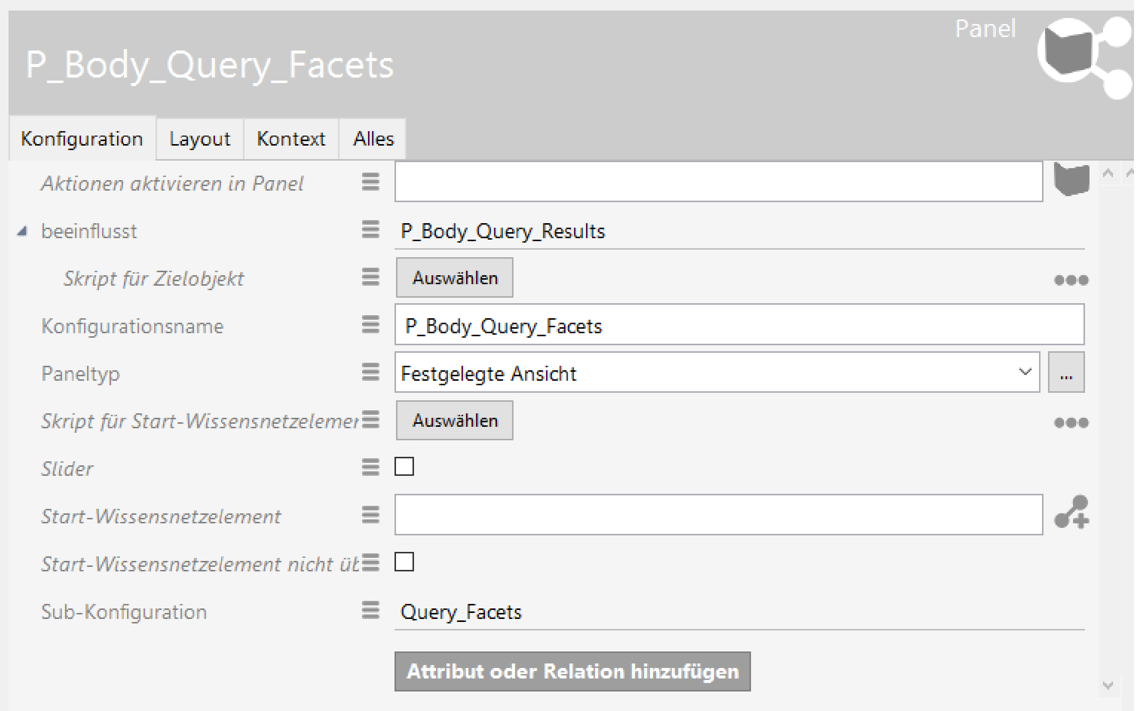
Search field elements can be combined with search result views and facet views. To ensure that the results of a search from a search field element are shown in a search result or facet view, the actions must be configured accordingly. The simplest option is to configure the panel that contains the search field element so that the actions are executed in a panel that contains a facet view or a search result view.



P_Header_Query

Aktionen aktivieren in Panel: P_Body_Query_Facets
beeinflusst:
Skript für Zielobjekt: Auswählen
Konfigurationsname: P_Header_Query
Paneltyp: Festgelegte Ansicht
Skript für Start-Wissensnetzelemen: Auswählen
Slider: □
Start-Wissensnetzelement:
Start-Wissensnetzelement nicht üb: □
Sub-Konfiguration: Search_Params

If you want to connect all three views to each other, you activate the actions of a search field element in a panel that contains a search result or facet view as described above or you configure this panel so that the other result view panel is influenced by this panel.



P_Body_Query_Facets

Aktionen aktivieren in Panel

beeinflusst: P_Body_Query_Results

Skript für Zielobjekt

Konfigurationsname: P_Body_Query_Facets

Paneltyp: Festgelegte Ansicht

Skript für Start-Wissensnetzelemen

Slider: □

Start-Wissensnetzelement

Start-Wissensnetzelement nicht üb □

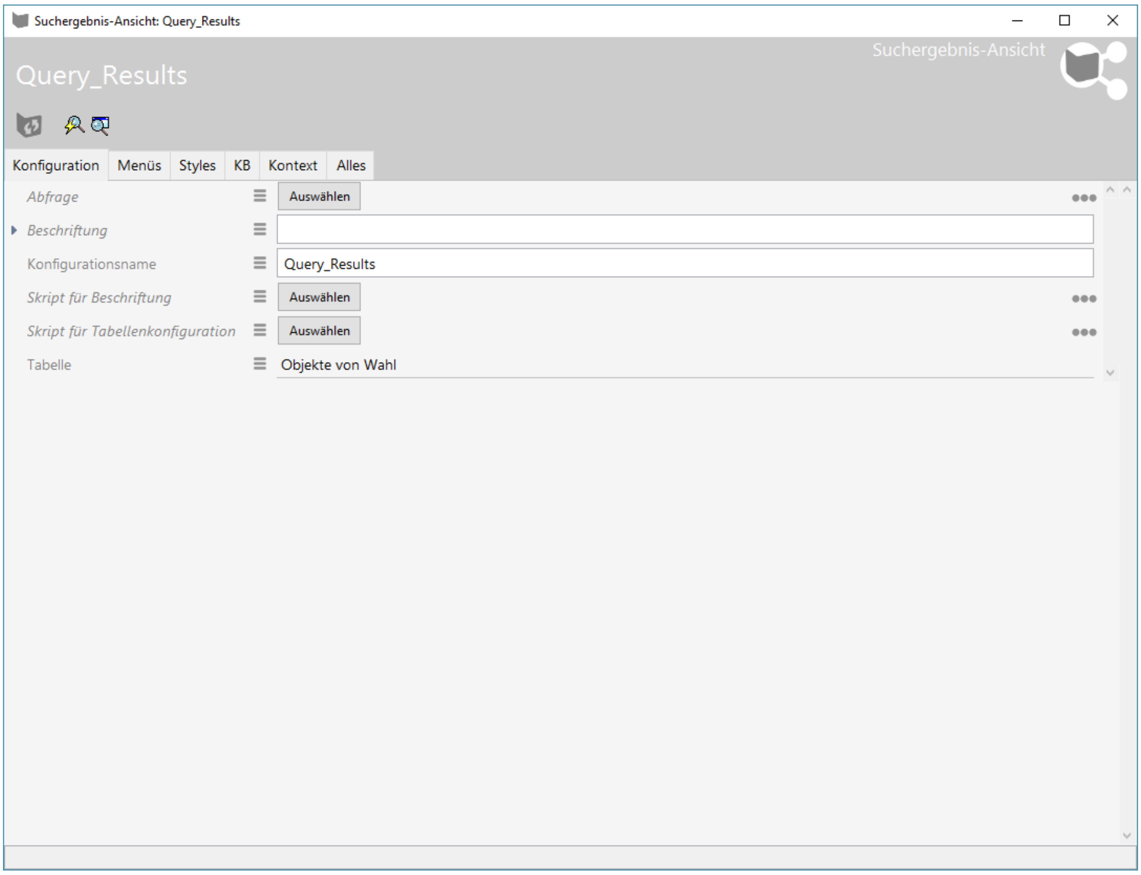
Sub-Konfiguration: Query_Facets

3.6.11.5. Search result view

NOTE Search result views are deprecated in favour of Search compounds with a configured output.

A search result view is used if a view is supposed to display only the results of the search, and not the search parameters. If the configured search has no parameters, it is enough to configure one search result view. If there are parameters, the search result view should be linked to a search field element.

It can be created in the Knowledge Builder.



Suchergebnis-Ansicht: Query_Results

Query_Results

Suchergebnis-Ansicht

Abfrage: Auswählen

Beschriftung:

Konfigurationsname: Query_Results

Skript für Beschriftung: Auswählen

Skript für Tabellenkonfiguration: Auswählen

Tabelle: Objekte von Wahl

The “Configuration” tab provides options for determining the general display of the search:

Query	This is where you configure the query that is to be executed when the query is executed.
Label	The value entered here appears as the heading of the search
Configuration name	The configuration name can be used to identify views and panels.
Script for label	As an alternative to the “Label,” the title of the search result view can be determined in a script.
Table	The search results are displayed in the front-end in the table configuration that is configured here.
Script for table configuration	As an alternative to “Table”, a script can be used to determine the table displayed at this point.

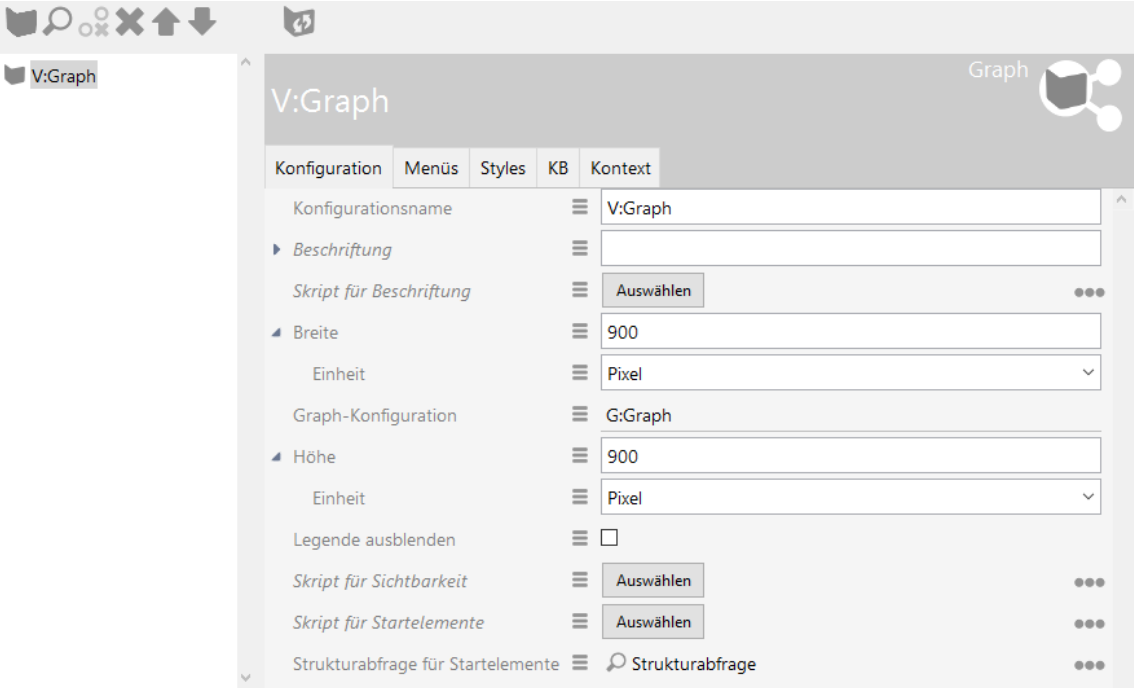
3.6.12. Graph configuration

A graph configuration is used to display objects in a graph. A first introduction to the use of graphs in the Knowledge Builder can be found under *Knowledge Builder > Basics > Graph editor* .

Details on the setting options for the different views that are required when embedding a graph in the front-end are explained under *Knowledge Builder > View configuration* > View configuration elements > *Graph* .

A **Graph view** and a **Graph configuration view** are required for display. The panel in which the graph is to be displayed contains a graph view ("V:Graph"). Up to version 5.1, the context element (called start semantic element) was optional and displayed in the graph when the application started. From version 5.2, it is obligatory to assign a context element in order to avoid triggering an error message. The object itself is not important, it is not displayed by default.

The graph view only has to contain a link to the graph configuration. The setting for the size of the graph field via the *Width* and *Height* fields is optional but usually available.



V:Graph

Konfigurationsname: V:Graph

Beschriftung:

Skript für Beschriftung: Auswählen

Breite: 900

Einheit: Pixel

Graph-Konfiguration: G:Graph

Höhe: 900

Einheit: Pixel

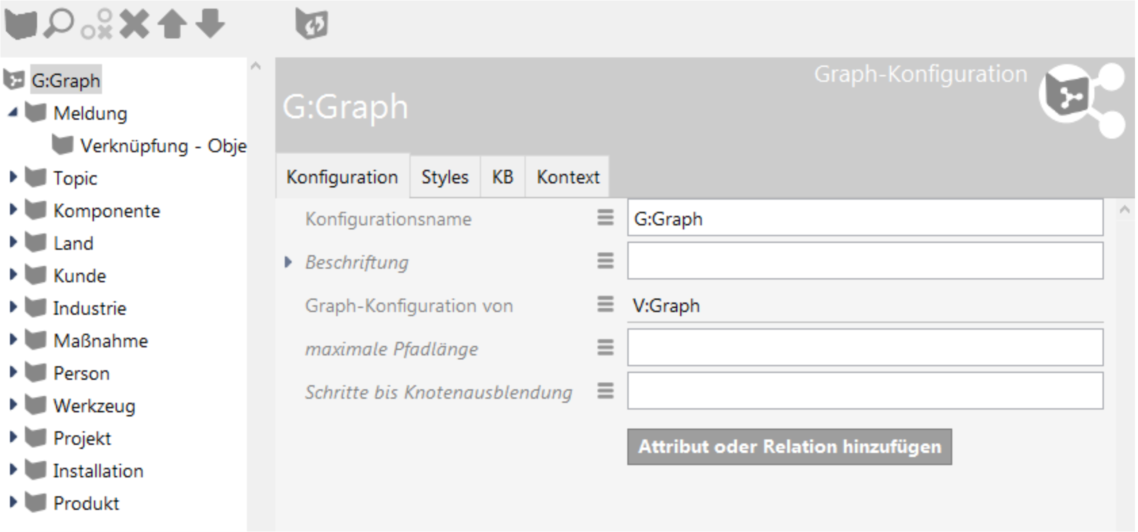
Legende ausblenden: □

Skript für Sichtbarkeit: Auswählen

Skript für Startelemente: Auswählen

Strukturabfrage für Startelemente: Strukturabfrage

The **Graph view** ensures that the graph is displayed in full. The **Graph configuration** is used to determine which nodes and relations are to be displayed.



G:Graph

Konfigurationsname: G:Graph

Beschriftung:

Graph-Konfiguration von: V:Graph

maximale Pfadlänge:

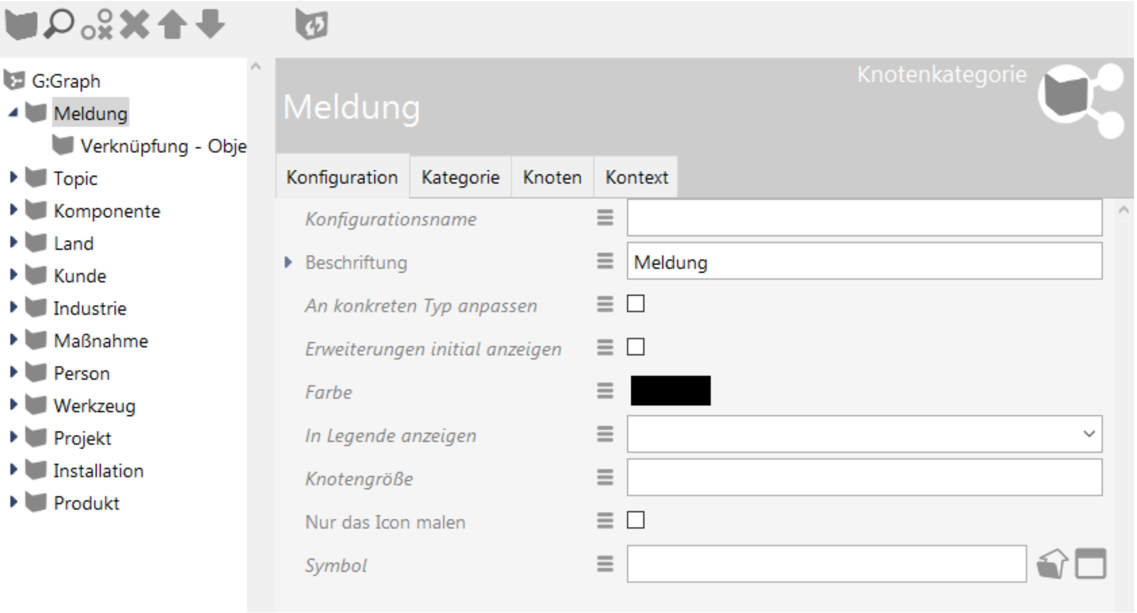
Schritte bis Knotenausblendung:

A node category must be created for every type whose objects (or types) are to be displayed. These are displayed by default as a key in the graph.

The graph displays objects that are directly attached to the type or its subtype. Use *Adapt to concrete type* to display subtypes separately in the key without having to create them individually as node categories.

In order to display types instead of objects, a checkmark must be placed by the *Apply to subtypes* box in the Context tab.

In the Nodes tab you can go to Menus and assign a satellite menu in order to continue working in the graph (see *Knowledge Builder > View configuration* > Actions > *Actions for the ViewConfiguration Mapper > NN-Expand/NN-Hide/NN-Pin actions).*



G:Graph
- Meldung
  - Verknüpfung - Obje
- Topic
- Komponente
- Land
- Kunde
- Industrie
- Maßnahme
- Person
- Werkzeug
- Projekt
- Installation
- Produkt

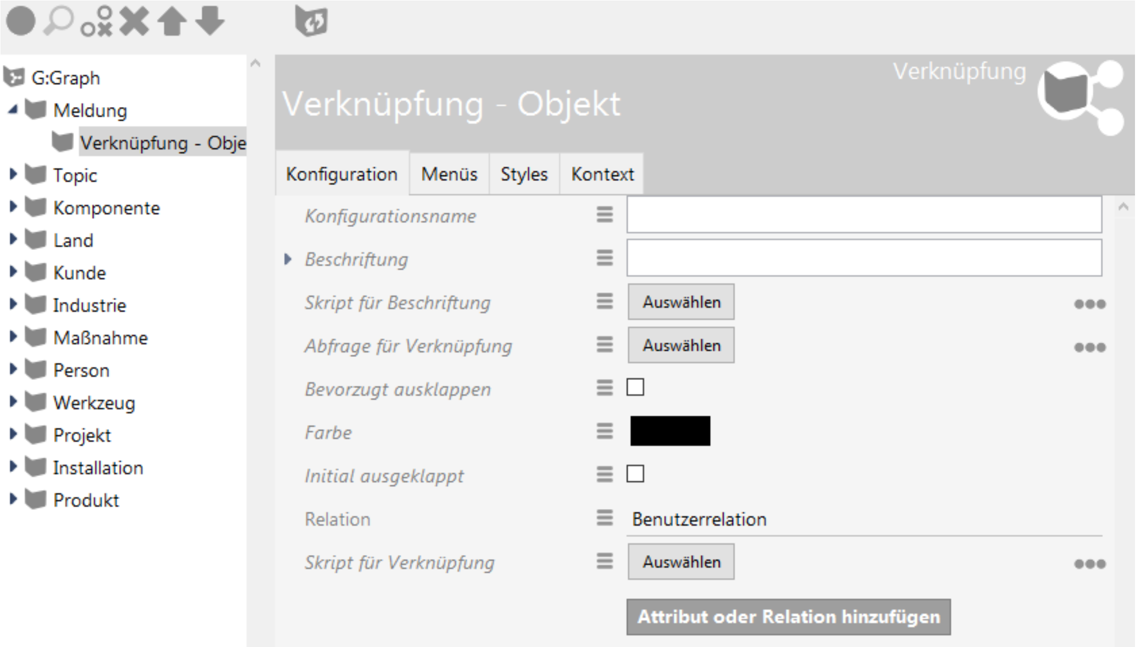
Meldung

Knotenkategorie

Konfiguration | Kategorie | Knoten | Kontext

Konfigurationsname	
Beschriftung	Meldung
An konkreten Typ anpassen	☐
Erweiterungen initial anzeigen	☐
Farbe	■
In Legende anzeigen	
Knotengröße	
Nur das Icon malen	☐
Symbol	

In order to display the relations between the nodes, a *link* is required under each *node category* . Here the relations to be displayed for this type are specified. The relations can be specified via a prompt, a script or via the direct specification of the relation. *User relation* can be assigned if all relations (apart from system relations) are to be displayed.



Verknüpfung - Objekt

Konfigurationsname

Beschriftung

Skript für Beschriftung: Auswählen

Abfrage für Verknüpfung: Auswählen

Bevorzugt ausklappen: □

Farbe: ■

Initial ausgeklappt: □

Relation: Benutzerrelation

Skript für Verknüpfung: Auswählen

For more details see the *vcm-plugin-net-navigator* chapter

3.6.13. Text

The text view can be used to display text that is either statically specified or calculated via a script.

Text	Static, multilingual text
Script for text	Script for calculation of the text
Label	Optional heading
Script for label	Optional script for calculating the heading

Example of a text script:

```
function text(element)
{
    return "Through a script in the Knowledge Graph"
           + $k.volume() + " generated text";
}
```

3.6.14. Image

Displays an image saved in the Knowledge Graph that is either statically specified or calculated by means of a script.

Image	Static image

Script for image	Script for calculation of the image. A blob attribute is expected as the return value. Dynamic blobs (e.g. through download by means of HTTP client) are not possible.
Label	Optional heading
Script for label	Optional script for calculating the heading
Width / height	Fixed width / height of the image

3.6.15. Script generated HTML

This view generates HTML via a script. Both Knowledge Builder and ViewConfigMapper show this unfiltered. Hence, the script developer is responsible for ensuring that user contents are not output unfiltered. The display options in Knowledge Builder are very limited (e.g. no CSS).

For more complex HTML you should use a script-generated view instead.

The following arguments are transferred to the script as parameters:

element	$k.SemanticElement	The element in the context of which the view is displayed
document	$k.TextDocument	Document on which HTML is output

There are two approaches for outputting HTML:

- Output the HTML source code using the print() function of the document
- Structured output using an MXL writer

The example below illustrates the use of an XML writer for outputting a heading:

```
/**
 * Render the semantic element on the document.
 * @function
 * @param {$k.SemanticElement} element The element to render
 * @param {$k.TextDocument} document Target document
**/
function render(element, document)
{
    var xmlWriter = document.xmlWriter();
    xmlWriter.startElement("h1");
    xmlWriter.characters(element.name());
    xmlWriter.endElement("h1");
}
```

3.6.16. Script generated view

A script-generated view allows custom view components to be defined. The data are generated by a script and passed on using JSON. Displaying this is the job of the front-end.

viewType	Freely selectable identifier that is output in JSON. This is used for assigning the custom components in the front-end.
Script	Delivers the data that are output in the JSON.

Two parameters are passed to the script:

element	$k.SemanticElement	The element in the context of which the view is displayed
view	object	Prefilled object with the view data. Configuration elements such as styles are already included in this.

The following script provides the data for a view that the plugin *vcm-plugin-timeline* contains:

```
/**
 * Get json object to modify.
 * @function
 * @this $k.View
 * @param {$k.SemanticElement} element
 * @param {object} json object
 * @returns {object} modified json object
 **/

function customizeView (element, view) {
  view.options = {
    layout: 'vertical'
  }
  view.events = $k.Registry.type('election').allInstances().map(function
(election) {
    return {
      elementId: election.idString(),
      name: election.name(),
      date: election.attributeValue('electionDate').toString()
    }
  })
  return view
}
```

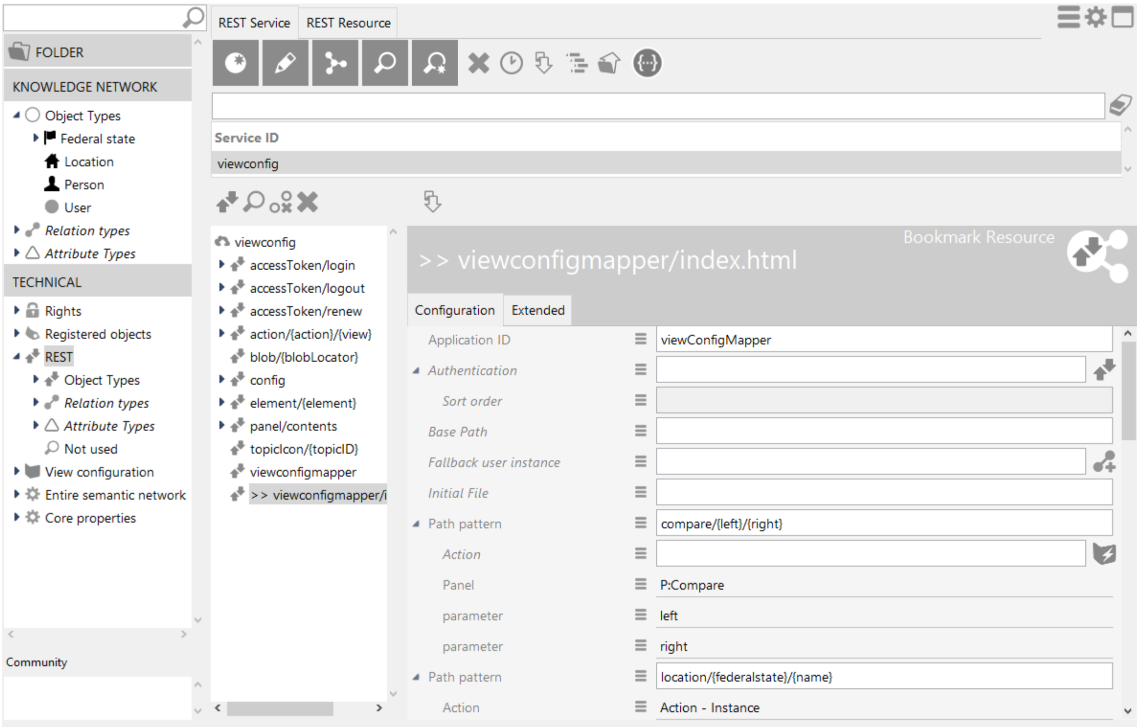
3.7. Bookmarks and history

Due to the fact of the ViewConfig-Mapper being a single-page application, the address of the application keeps always the same (http://xxx/yyy/index.html) - irrespective which content is being visualised or which panel is being displayed.

By means of defining bookmarks, the application designer in person is able to define a schema which builds up specific addresses for the currently shown content. For the user, this in turn grants direct access to a specific application state. Furthermore, bookmarking improves indexability of the application by web search engines.

3.7.1. Bookmark Resource

The definition of bookmarks has its starting point at the bookmark ressource. The bookmark ressource is situated within the REST service for the ViewConfig-Mapper. The bookmark ressource automatically is co-created when the ViewConfig-Mapper component is being added. Keep in mind that the herein described "Bookmark Resource" has to be configured to run without any authentication. This is because the ressource creates redirects which must work prior the moment of login (prior loading of the application) as well.



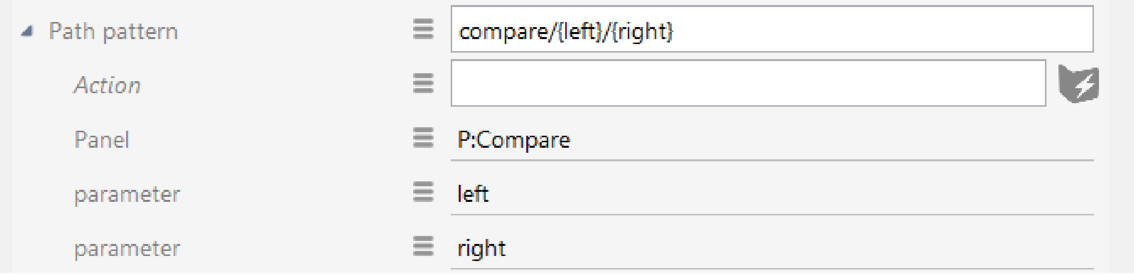
Service ID	viewconfig

>> viewconfigmapper/index.html

Application ID	viewConfigMapper
Authentication	
Sort order	
Base Path	
Fallback user instance	
Initial File	
Path pattern	compare/{left}/{right}
Action	
Panel	P:Compare
parameter	left
parameter	right
Path pattern	location/{federalstate}/{name}
Action	Action - Instance

The ressource allows the definition of any desired amount of "path patterns" - thus address patterns that can be used by the application from that point on. Path patterns must not overlap. This means, a specific address must be relatable to exactly one specific path pattern. Furthermore, overlap with other ressources must be avoided (e. g. "action" or static ressources as well).

A path pattern consists of static and variable parts. Dynamic parts are written in curly brackets (see chapter concerning REST ressources):



Path pattern: compare/{left}/{right}

Action:

Panel: P:Compare

parameter: left

parameter: right

Further examples:

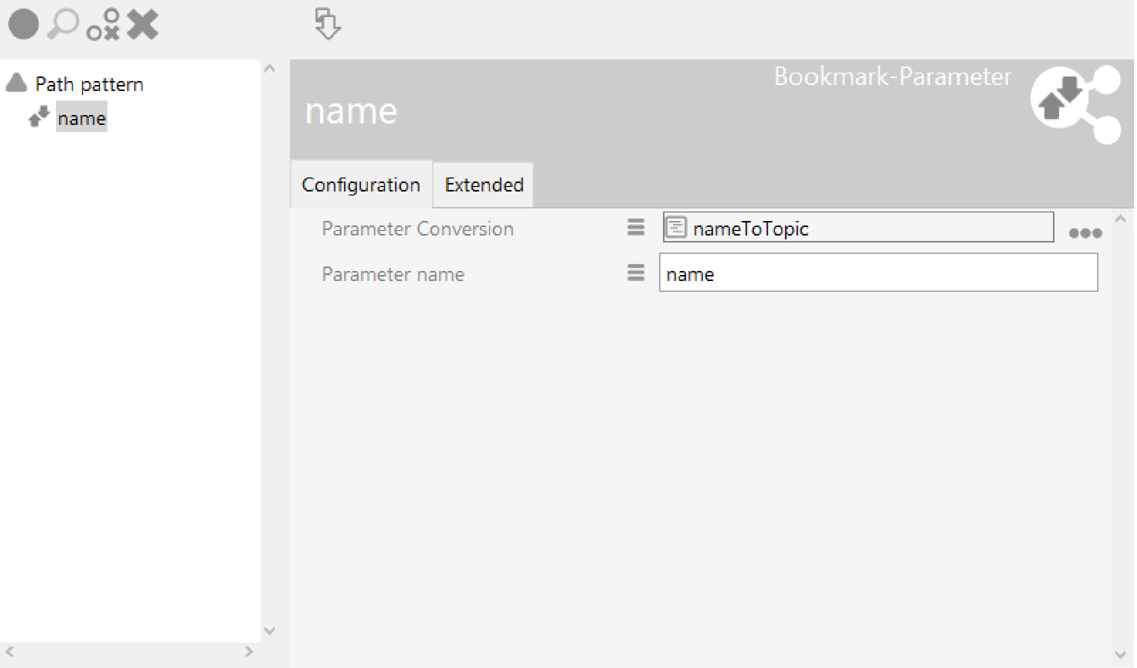
- help/{topic}
- performance/{company}/{year}

Following the definition of a path pattern, parameters have to be defined for the variable part. Parameters are meta-attributes of the path pattern attributes. A parameter normally represents an element of the Knowledge Graph and is shown in forms of the ID of the element when the address is being created (e. g. ID1527_373749).

By defining a "parameter conversion" script the default behaviour can be modified. This comes into account for following:

- representing elements in addresses in a more meaningful way
- using external IDs (e. g. part number) for addressing content
- using stable IDs that keep valid even if internal IDs change

A common use case is the indication of an object name instead of the objects' ID:



Parameter Conversion: nameToTopic

Parameter name: name

Script "Parameter conversion"

The script for parameter conversion contains two functions:

- **identifier(optionalElement):** the panel is represented as a part of the bookmark URL. This function determines in which way the semantic element of the panel will be converted into a text identifier for the URL (= processing of the "bookmark output").
- **element(parameterValue):** this function determines how an input URL is going to be interpreted to get the semantic element for being displayed in the panel (= processing of the "bookmark input").

In this example, the variable (e. g. optionalElement.name()) is accessed in the function "identifier()", in combination with assignment of the variable (e. g. $k.Registry.elementAtValue('name', parameterValue)) in the function "element()":

```
/**
 * Returns an (element-) identifier for the parameter
 * @function
 * @param {$k.SemanticElement} optionalElement The element for which the
identifier shall be returned (optional)
 * @returns {string}
**/

function identifier(optionalElement) {
    if (optionalElement)
        return optionalElement.name()
    else
        return undefined
}

/**
 * Returns an element for the given parameter value
 * @function
 * @param {string} parameterValue The parameter value
 * @returns {$k.SemanticElement}
**/

function element(parameterValue) {
    return $k.Registry.elementAtValue('name', parameterValue)
}
```

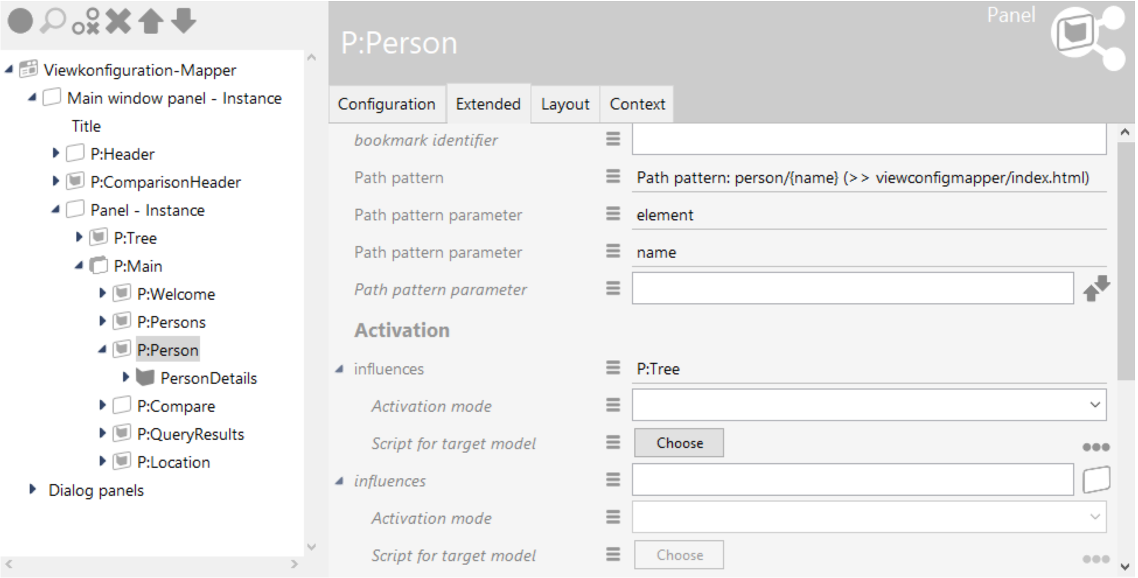
Composite parameters allow addressing of elements by means of structured descriptions (e. g. {chapter}/{version}). For each parameter fragment of the composite parameter there must be a corresponding Bookmark-Parameter object configured below the Composite-Parameter object. The Composite-Parameter object requires a Parameter Conversion script, which handles the multiple parameters.

Hint: By using parameter conversion scripts, session varaibles can be transported as well. This allows addressing an application state which itself is not defined solely by the displayed content.

Herefore the variable (e. g. $k.Session.current().getVariable("currentPersona")) can be accessed in the function "identifier()", in combination with assignment of the variable (e. g. $k.Session.current().setVariable("currentPersona", parameters.persona)) in the function "element()".

3.7.2. Link to Panels

Path patterns, as explained in the preceding chapter, can be linked to a panel (via the relation "Path pattern" of the respective panel). This means that the pattern is going to be used for construction of the address, as soon as the panel is activated (= visible).



P:Person

bookmark identifier:

Path pattern: Path pattern: person/{name} (>> viewconfigmapper/index.html)

Path pattern parameter: element

Path pattern parameter: name

Path pattern parameter:

Activation

influences: P:Tree

Activation mode:

Script for target model: Choose

influences:

Activation mode:

Script for target model: Choose

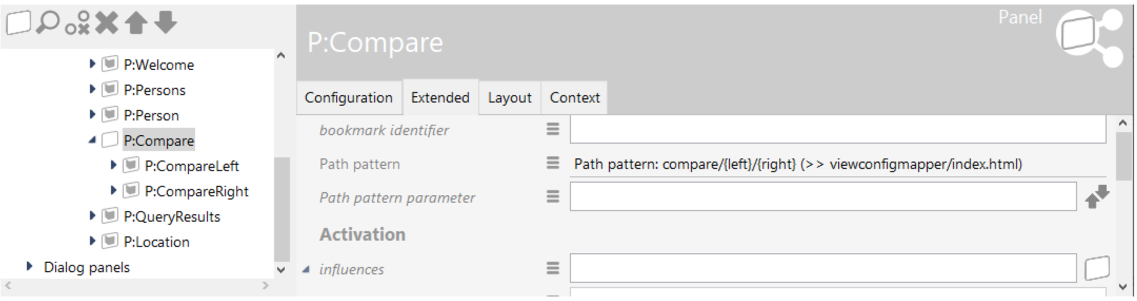
Caution: When designing the application, it is important to observe that at no time more than one panel with path pattern can be active simultaneously. Otherwise, the ViewConfig-Mapper cannot decide which address pattern has to be used.

Hint: If a more than one panel needs to be displayed when invoking a path pattern, this can be solved by assigning a path pattern to one panel and by linking the first panel via the relation "influences" to the second panel which has no path pattern (**example:** panel with navigation bar and panel with content both need to be displayed at once).

The element, which is visible in the active panel, is going to be used for parameter construction of the path pattern. It is necessary to ensure that the panel knows its element so that a parameter can be constructed. A fixed view panel usually knows the element, so it should be preferred instead of using a layout panel containing a fixed view panel.

A layout panel only knows the element if a context element is set.

If the element of another panel is to be considered for constructing parameters, the concerning panel has to be linked to the parameter via the relation "Path pattern parameter".By this, you can for example address a comparison view of two products (compare/product_A/product_B):



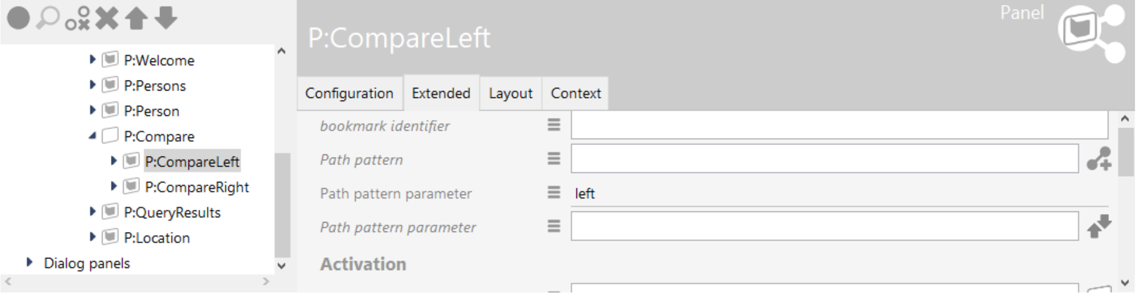
P:Compare

bookmark identifier

Path pattern: Path pattern: compare/{left}/{right} (>> viewconfigmapper/index.html)

Path pattern parameter

influences



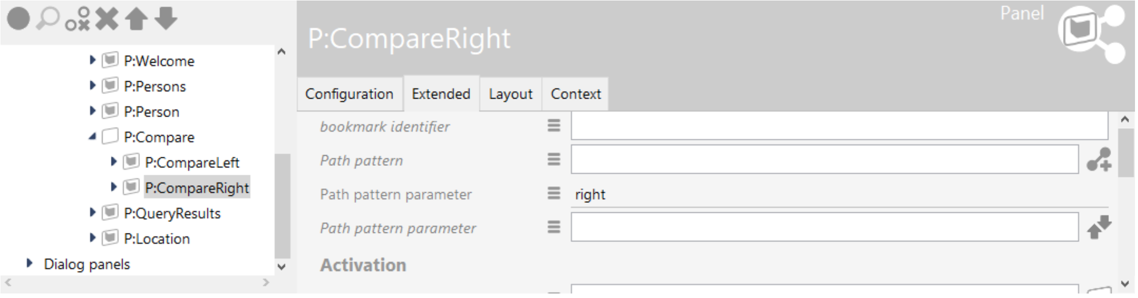
P:CompareLeft

bookmark identifier

Path pattern

Path pattern parameter: left

Path pattern parameter



P:CompareRight

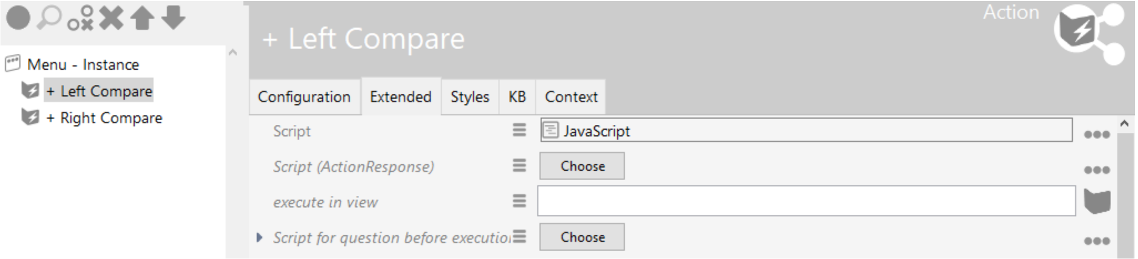
bookmark identifier

Path pattern

Path pattern parameter: right

Path pattern parameter

For the comparison action of a menu within a view, a script needs to be added:



+ Left Compare

Script: JavaScript

Script (ActionResponse)

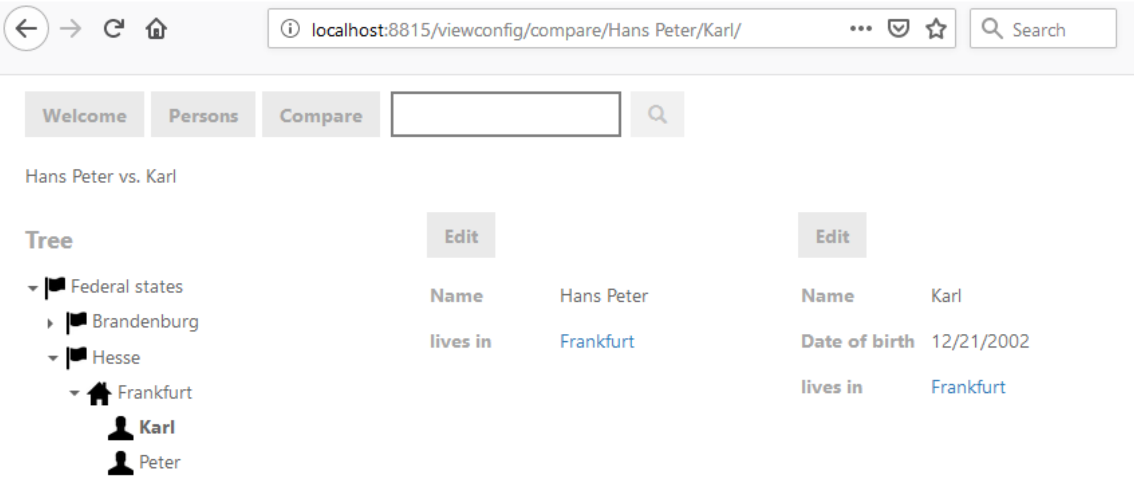
execute in view

Script for question before executio

The action script for setting the session variable is shown in the following example:

```
/**
 * Performs a custom action. Can access the UI (open dialogs etc. with
context.ui)
 * @function
 * @param {$k.SemanticElement} element
 * @param {object} context Parameters defined by the environment
**/

function onAction(element, context) {
        $k.Session.main().setVariable('comparison.left', element)

        return element
}
```

As soon as the panel with the related path pattern is activated, it shows the content which has been stored as the session variable by means of the action script.



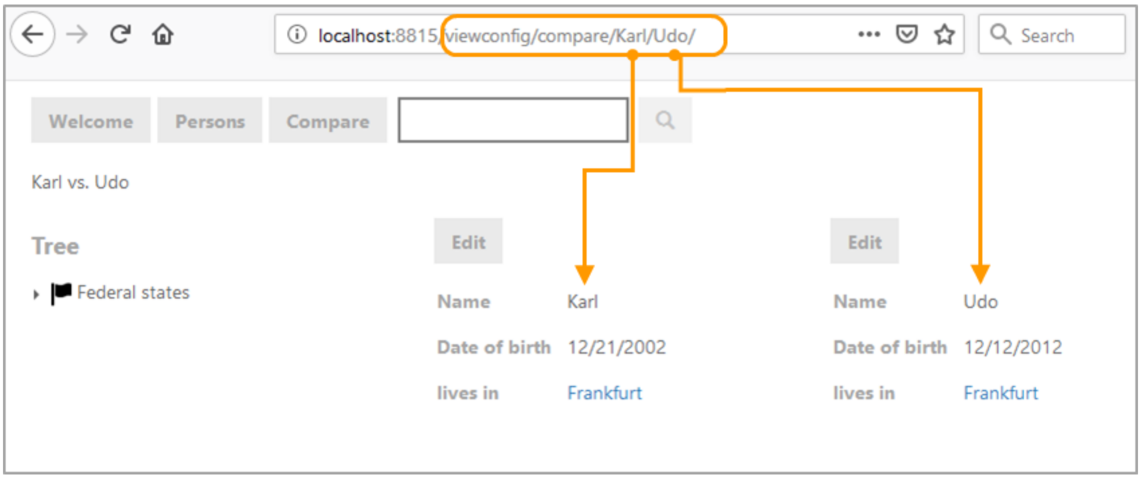
Hans Peter vs. Karl

Name	Hans Peter
lives in	Frankfurt

Name	Karl
Date of birth	12/21/2002
lives in	Frankfurt

When accessing a bookmark link by typing it into the browser input line, the configuration principle "vice versa" comes into account:

1. The apropriate path pattern is determined
2. The concerning panel is being activated and, if applicable, is being equipped with an element for indication. The indication of element itself is defined by the parameter rules.
3. Panels, which are linked by parameters, are activated as well. If applicable, the element is indicated additionally according the parameter rules.
4. The activation chains (see chapter about panel activation) are executed and the application is visible in the desired state.



Karl vs. Udo

Tree

Federal states

Name	Karl
Date of birth	12/21/2002
lives in	Frankfurt

Name	Udo
Date of birth	12/12/2012
lives in	Frankfurt

Hint: Dialogs can be addressed by means of the previously described mechanism as well. When defining the path pattern for dialogs, it is important that both the content of the dialog panel and the content underneath the dialog panel is defined by the bookmark link. This can be done by linking of a parameter with a panel of the main window panel.

3.7.3. In-app navigation with bookmarks

By means of the in-app navigation with bookmarks, an action-based navigation can be realized alternatively by using web links (panels are activated by an action and/or by exchange of content between panels).

In this case, functionalities of the browser like "open in new tab" / "open in new window" are available for the user. Furthermore, search engines can follow and index these links.

The definition is simply done by linking the action to the desired path pattern. If the parameter construction shall not (only) be executed by the element of the action, this can be adjusted by means of the script "Parameter construction".

3.7.4. Bookmark fallback

In some cases, it is possible that a bookmark saved at an earlier point in time cannot be retrieved, e.g. because the data or the structure of the application have changed, or because a bookmark is called by another user with different access privileges. In order to prevent error messages from occurring in these cases, you can configure a fallback behavior for bookmarks.

Both the general bookmark resource and the individual bookmark path objects can be linked with another bookmark path object using the *Has fallback path* relation. If an error occurs when initially opening a bookmark, the request is redirected to the fallback path instead. For example, it could be directed to the start page or a general info page for dealing with errors.

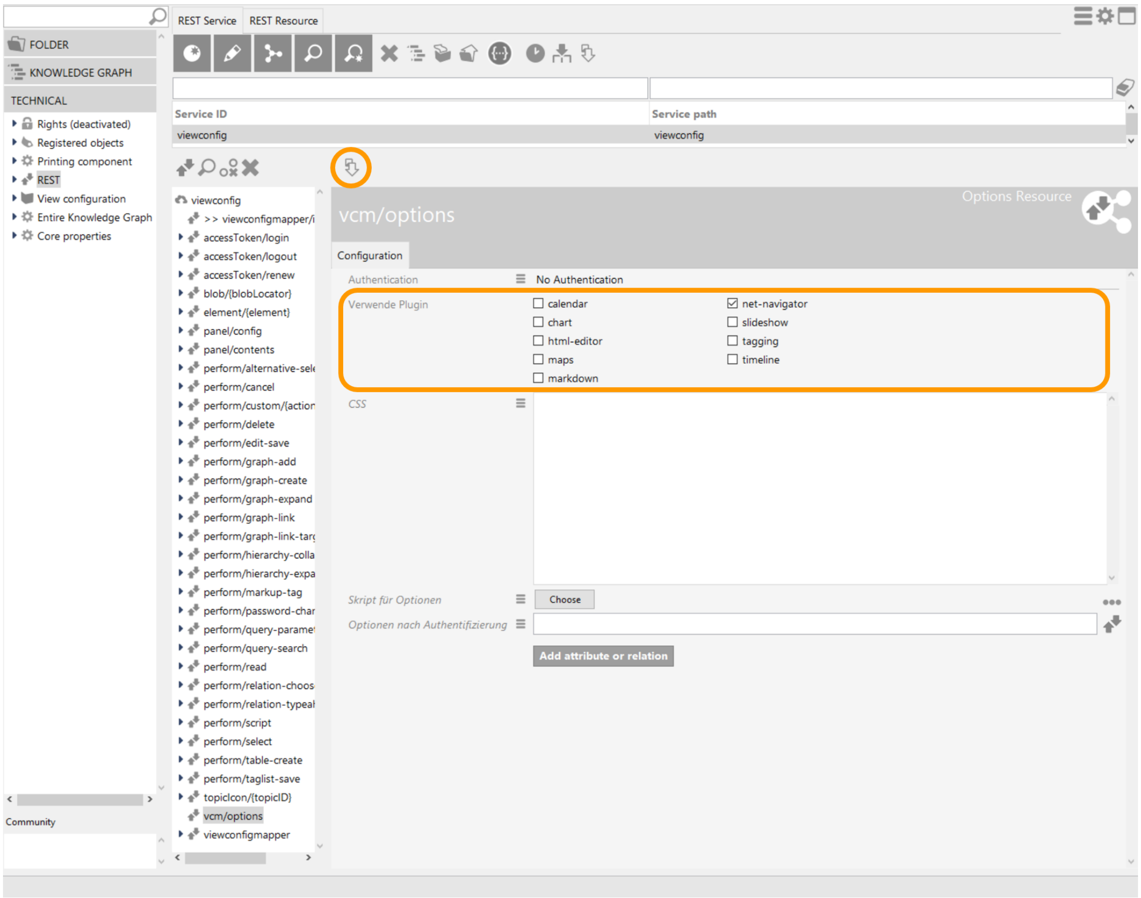
For more complex scenarios, you can configure several possible fallback paths. They then require the meta property *Script for bookmark fallback*. These scripts are used to determine from the set of possible paths the one for which the request is to be redirected. The first script that returns `true` defines the target path. You can use the parameter `originalBookmark` of the script function `bookmarkFallback()` to retrieve data about the context, such as the original path requested and the assignment of all bookmark parameters.

```
/**
* Determines whether this bookmark should be preferred over the originally requested bookmark.
* This may be the case if the user has insufficient rights to access the original path, for example
* or if a bookmark path parameter could not be resolved.
* @function
* @param {$k.Bookmark} originalBookmark The bookmark requested by the user
* @returns {Boolean} If the return value is true, the user will be redirected to this bookmark.
**/

function bookmarkFallback(originalBookmark) {
    return false
}
```

All fallback paths configured directly on a bookmark path are evaluated first. Only if no fallback can be determined there, the fallback paths of the bookmark resource are taken into account. For a fallback path determined, possible fallbacks are evaluated again if this target also causes errors or is to be accessible only under certain conditions. It is ensured that an endless cascade cannot occur.

3.8. Plugins

In order to make the following plugins applicable, they need to be activated for the options request of the ViewConfiguration mapper. At the REST service configuration of the VCM, a detail editor provides the options:



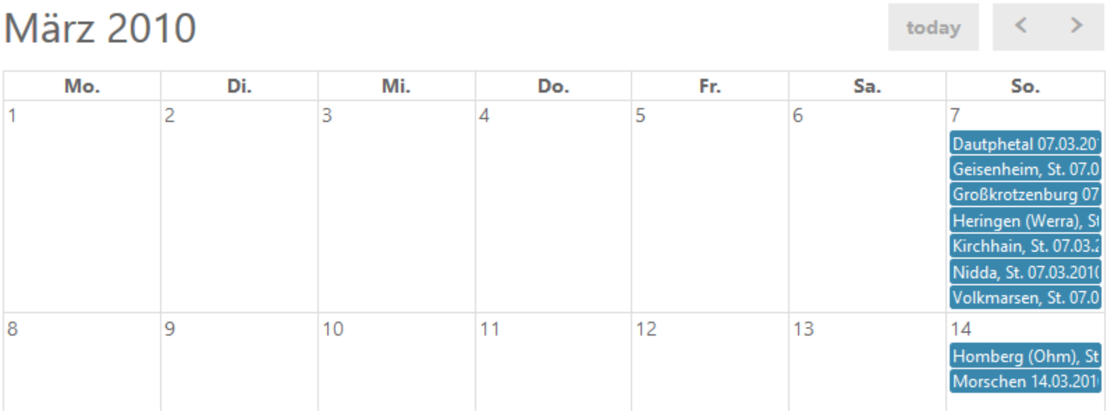
Authentication: No Authentication

Verwende Plugin:
- ☐ calendar
- ☐ chart
- ☐ html-editor
- ☐ maps
- ☐ markdown
- ☑ net-navigator
- ☐ slideshow
- ☐ tagging
- ☐ timeline

NOTE After selection of the required plugins, both REST service and ViewConfiguration must be updated/rebuilt.

3.8.1. vcm-plugin-calendar

The vcm-plugin-calendar can be used to display data in a calendar.



März 2010

Mo.	Di.	Mi.	Do.	Fr.	Sa.	So.
1	2	3	4	5	6	7 Dautphetal 07.03.20 Geisenheim, St. 07.0 Großkrotzenburg 07 Heringen (Werra), St Kirchhain, St. 07.03.2 Nidda, St. 07.03.201(Volkmarsen, St. 07.0
8	9	10	11	12	13	14 Homberg (Ohm), St Morschen 14.03.201

In order to display the data as a calendar, it is necessary to add a style element containing the *calendar* renderMode to the table configuration. The value under Number of rows (page size) specifies the maximum number of calendar entries that can be shown per view (in this case per month). The table must contain the following columns:

- *start* : A date with which the calendar entry begins.
- end: End date of the entry (optional)
- title: The title of the entry
- allDay: Boolean value that specifies whether the entry applies to the whole day (optional)
- Further options for columns can be found in the fullcalendar.io Event_Object documentation.

It is also possible to configure a select action for the columns of the table. This action is then executed when a calendar entry is clicked.

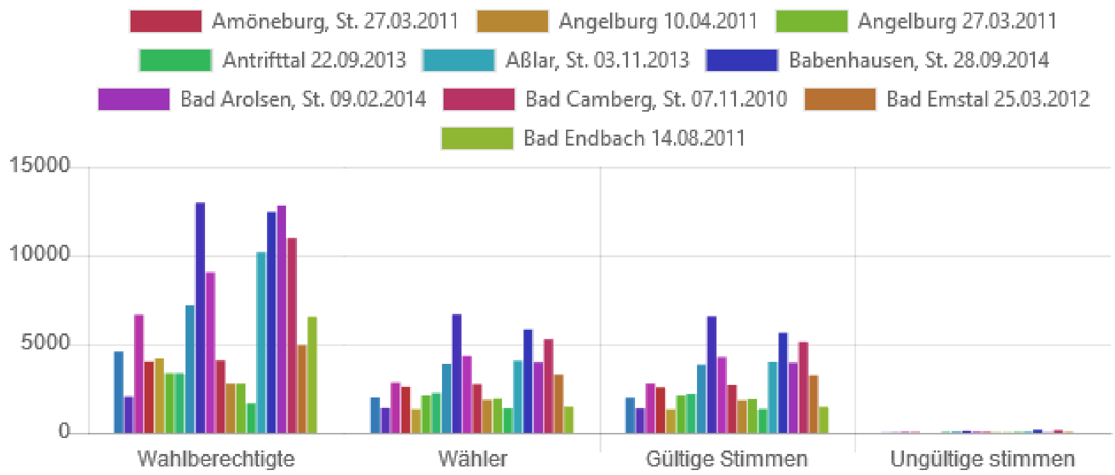
In addition, the vcmPluginCalendarOptions style attribute can be used to make additional configurations.

Further information on the plugin can be found at fullcalendar.io.

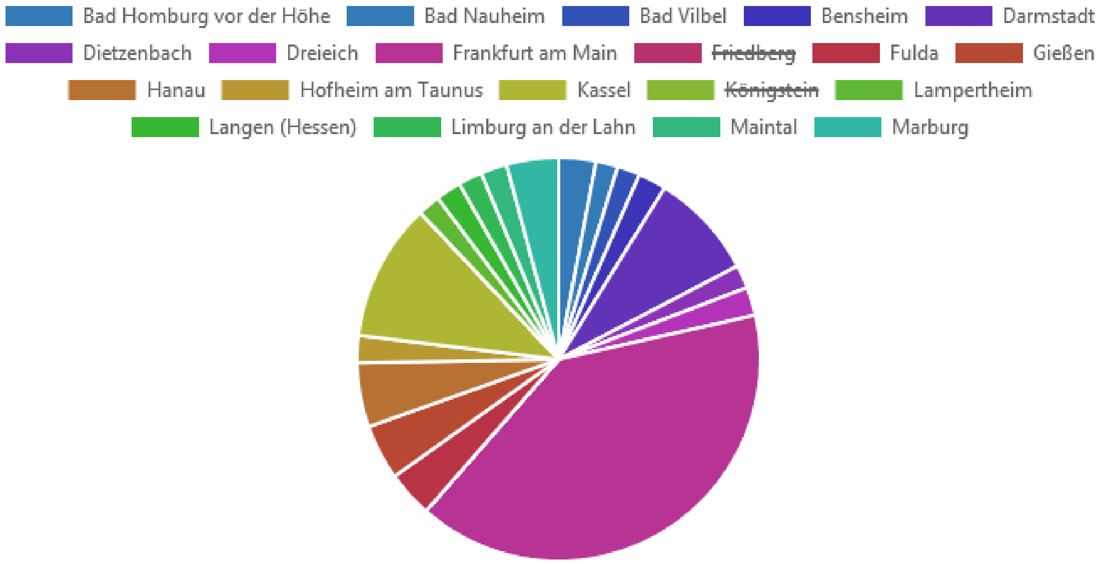
3.8.2. vcm-plugin-chart

The vcm-plugin-chart is used to display data from a table configuration on the web front-end in the form of a chart. Various chart types are available: Line, bar, pie, ring and radar charts.

Example of a bar chart:



Example of a pie chart:



Configuration example for pie chart:

1. Create a *script generated view* .
2. Important: For *viewType* , enter "chart".
3. For the script generated view, create a new script.

   The following script snippet shows a generic example for a pie chart with black border color. It displays the attributes of an attribute array, using their value and type name for showing the amount and a label:

```
function customizeView (element, view) {
    var dataEntries = [...]     // Array of numerical values or
attribute values (float or integer)
```

```
    view.chartData = {
        // static data
        datasets: [{
            data: [],
            backgroundColor: [],    // array of hexadecimal color
strings if number of values is static
            borderColor: '#000'    // hexadecimal string for border
color
        }],
        labels: []
    }

    dataEntries.forEach(function (entry) {
        view.chartData.datasets[0].data.push(entry.value())    // if
dataEntries is an array of attributes
        view.chartData.datasets[0].backgroundColor.push()      //
entry-spcific color
        view.chartData.labels.push(entry.type().name() + ': ' + entry
.value())
    })
    view.type = 'pie'
    return view
}
```

4. Add a new style to the script generated view with *renderMode* "chart" and *vcmPluginChartType* "pie". For *vcmPluginChartWidth* and *vcmPluginChartHeight* , specify width and height for the chart (values in pixel).
5. Optional step: For the style, specify a *vcmPluginChartOptions* script for legend placement and resizing behavior:

```
function additionalPropertyValue(element) {
    var value = {legend: {position: 'right'}, maintainAspectRatio:
false}
    return value
}
```

3.8.2.1. Configuration

To generate a chart, it is necessary to create in a table configuration a style with the “chart” option as its *renderMode* .

If, for example, you add an action with the “Display graphically” option to the underlying table configuration, you can then display the relevant data record additionally in the Net-Navigator by clicking on parts of the chart.

The plugin uses chart.js to generate the charts.

For vcm-plugin-chart there are multiple options for display adjustment that can be defined by means of styles:

- **vcmPluginChartDataColumns:
  - String with column numbers that are used as the data source. Default: columns 1-n
- **vcmPluginChartDataMode:
  - 'rows' or 'columns'. Default: 'rows'
- **vcmPluginChartHeight:** Specification of chart height in pixels. Default: 'auto'
- **vcmPluginChartWidth:** Specification of chart width in pixels. Default: 'auto'
- **vcmPluginChartLabelColumn:** Column number for labels. Default: 0
- **vcmPluginChartOptions:** Options for adapting how keys are displayed and axes are scaled; they are transferred to chart.js.
- **vcmPluginChartType:** Specification of the chart type: ‘line’, ‘bar’, ‘horizontalBar’, ‘radar’, ‘pie’ or ‘doughnut’. Default: 'line'

The following example shows how to use a script for vcmPluingChartOptions in order to disable the chart legend while scaling the axis to units of the size 1 and setting the axis origin to 0 instead of 1:

```
function additionalPropertyValue(element, context) {
  return {
   legend: { display: false },
   scales: { yAxes: [{ ticks: { stepSize: 1, beginAtZero: true } }] }
   }
}
```

3.8.2.2. Configuration on basis of a script-generated view

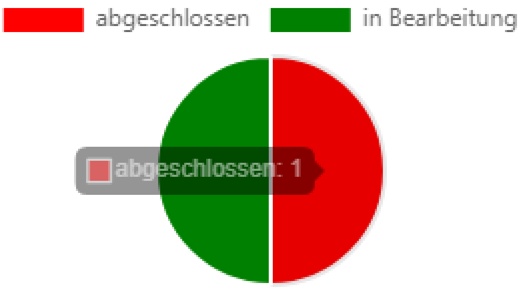
Charts can be display instead of tables using a script-generated view as well.

The prerequisite for this is that “chart” must be specified as the “viewType” in the configuration tab of the script-generated view.

Furthermore, as is the case for the table configurations, a style must be assigned that uses the property vcmPluginChartType to specify the preferred chartType (line', 'bar', 'radar', 'pie' or ‘doughnut.' Default: 'line').

The following is an example script that counts jobs according to their status and shows the set in a pie chart. Note that this script is an example of the “pie” chart type. Use the documentation for the chart.js to define the differences in the data formats of the other chart types: https://www.chartjs.org/docs/latest/

```
function customizeView(element, view) {
  var taskCount= $k.Registry.type(“job").allInstances().reduce(function
(result, job, index) {
        var status = job.attributeValueString(“statusJob");
        result[status]= (result[status]||0)+1
        return result;
    }, {})

   view.chartData = {
    datasets: [{
         data:Object.keys(jobsCount).map(function(key) {return jobsCount
[key]}),
         backgroundColor: ['red', 'green']
    }],
    labels: Object.keys(jobsCount)
  }

  view.type =  'pie'

  return view;
}
```

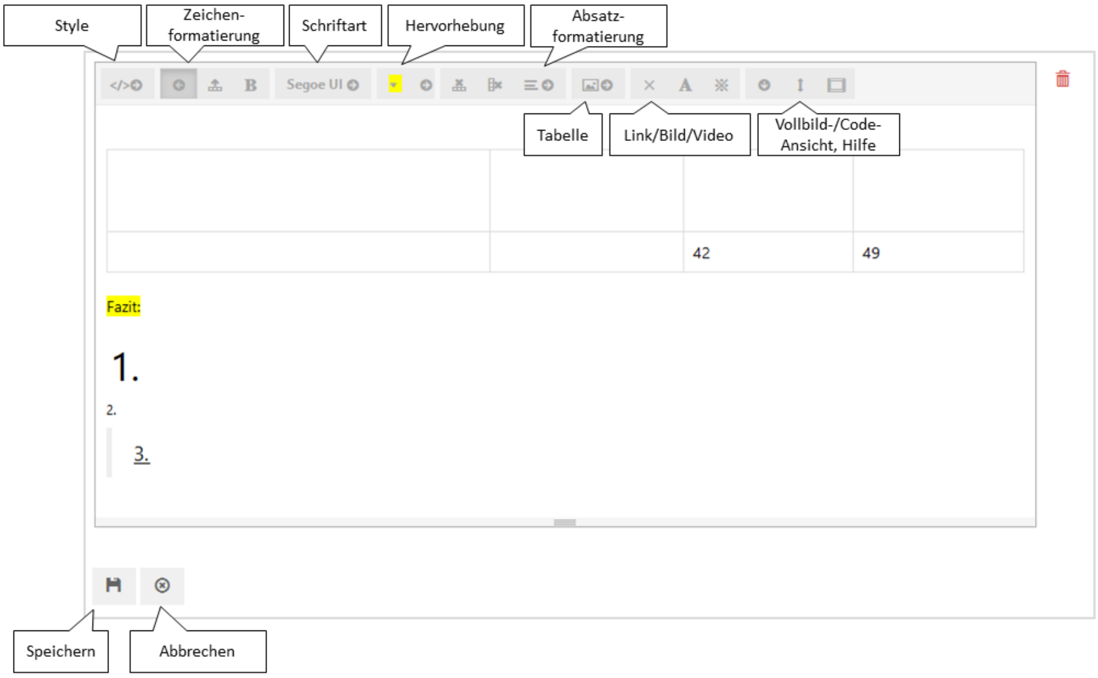


This pie chart was generated using a script that uses the chart.js plugin.

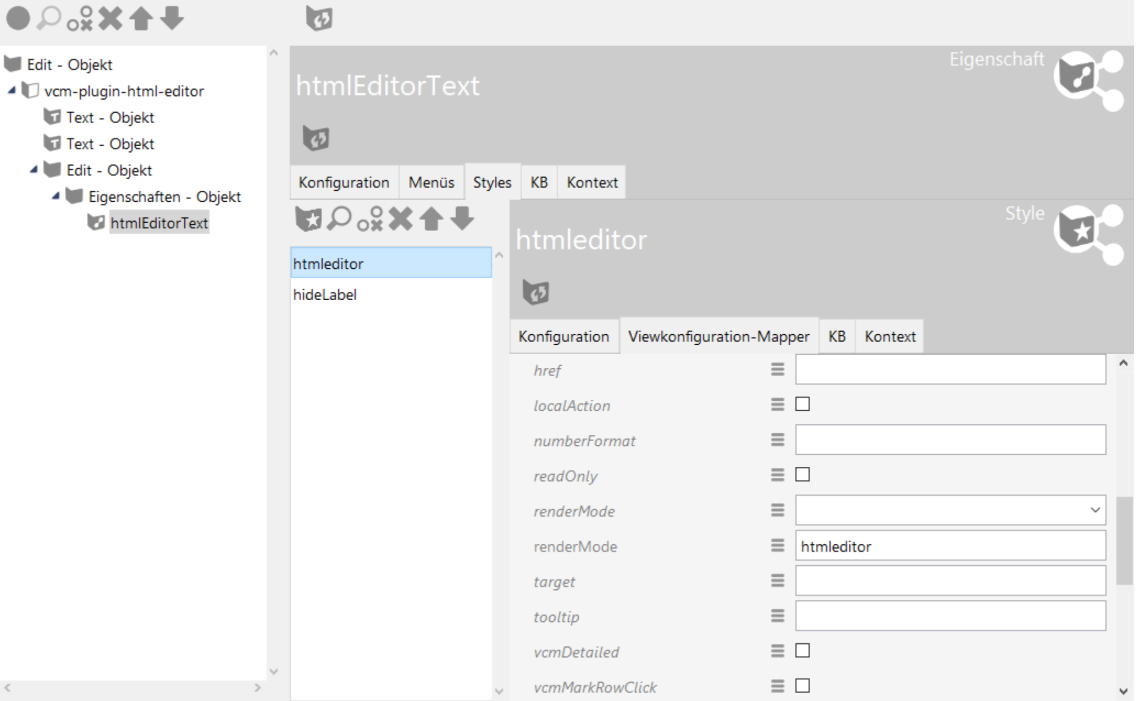
3.8.3. vcm-plugin-html-editor

Web front-end

The vcm-plugin-html-editor makes it possible to edit HTML-formatted text. For this purpose it uses the summernote WYSIWYG editor.



Configuration



htmlEditorText

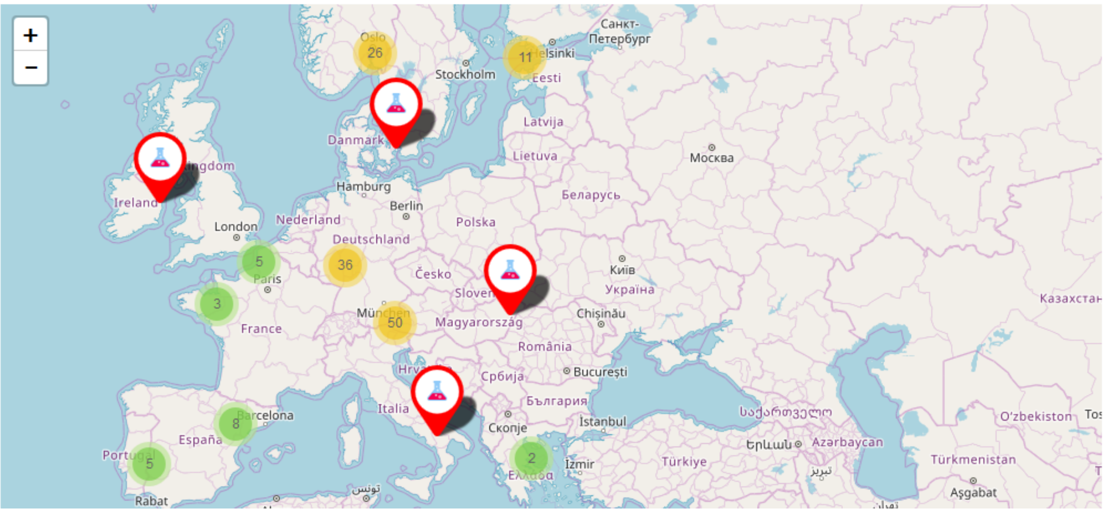
htmleditor

href	
localAction	□
numberFormat	
readOnly	□
renderMode	
renderMode	htmleditor
target	
tooltip	
vcmDetailed	□
vcmMarkRowClick	□

For the configuration, a property view for a string attribute is needed, which has a style configuration with renderMode "htmleditor". To make the content editable in the web frontend, the properties configuration needs to be embedded in an edit configuration. Otherwise, the attribute text is rendered as HTML without editor.

3.8.4. vcm-plugin-maps

The map plug-in allows for integrating a map (for geographic data) or a coordinate system (for geometric data) into the front end. For this purpose, the objects to be displayed must have an attribute of the type *Geometry* or *Geographic Position*.



NOTE

With version 6.0, the configuration of map views has changed fundamentally. The old configuration via table or script-generated view is still supported by the Material variant of the View Configuration Mapper, whereas the Vue-based variant of the View Configuration Mapper requires the new configuration elements.

The configuration elements *Map* and *Cartesian Coordinate System* are available for the configuration. Properties on these configuration elements define the display in the Web UI. In both cases, the definition of the actual data is configured via *Layers*. Both the map and the coordinate system can have any number of layers beneath them, but at least one layer is required in order to be able to display data.

3.8.4.1. Layer

The displayed content is configured with *Layers*. There are various ways in which a layer can be filled with data. In general, a layer shows *Features*. A feature consists of geometric or geographic information (which may be a point, polygon, or collection of geometries), and a set of properties. The features can either be derived from the domain model of the layer or calculated freely using the *Script for features*.

3.8.4.1.1. Features from domain model

Like almost every view, a layer can have a Knowledge Graph object, or a set of Knowledge Graph objects as a domain model. Since a feature necessarily requires geometry, the configuration of the *Geometry property* property is required in this case.

A convenient way to supply a layer with a suitable domain model is to use a *Search compound*. A layer can be linked to a search compound on the *Context* tab by means of the *Output of* property.

The search compound defines a search and any parameterization (e.g. via input fields). Details on this can be found in the chapter on Search compounds.

Point features are displayed on a map as a pin with the color and icon of the corresponding Knowledge Graph element. Areas and lines do not have an icon, but they do adopt the color. If the zoom layer causes areas and lines to be too small for the display, they are also replaced with pins with icons. All features automatically have the name of the Knowledge Graph element as a tooltip. In a coordinate system, not pins but dots are used for a better readability; the icon is not displayed here.

Combination with Elasticsearch

Elasticsearch is able to save and index geometry data in the WKT format. You can use an Elasticsearch query to retrieve geometry data from the Elasticsearch index and display them in the web UI, even if there are no corresponding geometry attributes in the Knowledge Graph. In this case, the search results are available as Elasticsearch hits. If an Elasticsearch query is linked to a layer by means of a search compound as described above, an automatic attempt is made to translate the hits into features. However, the hit cause in which the geometry data are located has to be known for this. It can be configured at each layer by means of the *Geometry hit cause* property and should be set to the name of the field in the Elasticsearch index that contains the geometry.

> **NOTE**
>
> In the Elasticsearch query configuration, the field must have it’s *Cause* checkbox checked. Otherwise Elasticsearch does not return the geometry data that lead to the search hit.

3.8.4.1.2. Features from *Script for Features*

Features can be created from any source in the *Script for features*, for example by performing a search. The script template defines a JavaScript function `features()` that must return an array of `$g.Feature` objects. The `element` parameter is assigned to the domain model element of the layer.

A feature can be defined by specifying geometry and properties:

```
function features(element) {
  const feature = new $g.Feature(new $g.Point(8.65027, 49.87167, 4326), {
name: 'Darmstadt' })
  feature.setTooltip('Darmstadt')
  feature.setColor('green')
  return [feature]
}
```

Alternatively, a feature can also be created directly from an element. The second parameter to be passed is the internal name of a Geometry property. The feature is automatically linked to the element, e.g. for click actions. The color and icon are adopted and the name of the element is set as a tooltip.

```
function features(element) {
  return [$g.Feature.fromElement(element, 'my.geometry')]
}
```

It is also possible to link the script to a search group configuration and create the features themselves:

```
function features(element) {
  return this.domainModel().elements().map(e => $g.Feature.fromElement(e,
'my.geometry'))
}
```

NOTE The *Geometry property*, *Geometry hit cause*, and *Script for features* properties are mutually exclusive. As soon as one of these properties exists, the others are not offered anymore.

3.8.4.1.3. Dynamic fetching of features

For large quantities of features, it might be useful to calculate them based on the currently visible map section and thus prevent large data volumes from being transmitted to the web UI at the beginning which are not displayed at all. The *Fetch features dynamically* configuration option can be used for this purpose.

NOTE Usually, the map or coordinate system is initialized to a zoom level that ensures the visibility of all features. However, this does not yield the desired result in combination with *Fetch features dynamically*. For this reason, initial view bounds should be configured for the coordinate view, see the Defining initial view bounds section.

If the features are defined by means of a search in a search compound, it can be parameterized with the predefined parameter *mapViewBounds*. The search is then executed every time the map section in the Web UI changes, and the parameter then contains a polygon with the current view bounds.

If the features are defined by a script, the polygon corresponding to the currently visible map section can be called by calling `this.viewBounds()`.

3.8.4.1.4. Other configuration options

The following additional configuration options are available on one layer:

Label

Name of the layer in the list of layers on the map.

Enable clustering

Allows features to be combined for cluster areas if features are very close together on the map. Clusters are only formed for elements of the same layer. The color corresponds to the color of the combined features, and the size depends to a certain extent on the number of features in the cluster.

NOTE

Lines and surfaces are only clustered when they have been replaced with pins due to the zoom level.

This setting has no effect on the Cartesian coordinate system.

Click action

Defines an action that is executed when a feature is clicked. The action is automatically parameterized with the element that is linked with the clicked feature. The exact behavior of the action can be configured in the action object.

NOTE

In order to perform the click action, it is mandatory that the feature is linked to a Knowledge Graph object. This is automatically the case when features are derived from the domain model. If features are generated by script, this must be set either by means of the `$g.Feature.fromElement()` function or by means of the `.setElement()` function on an existing feature.

3.8.4.2. Coordinate view

The map and the Cartesian coordinate system share the following basic configuration options:

Width

Width of the view in pixels or percent (relative to the width of the enclosing element).

Height

Height of the view in pixels or percent (relative to the height of the enclosing element).

NOTE

Percentage widths and heights depend on the size of the enclosing element. In particular, the coordinate view remains invisible if the enclosing element does not have any size information of its own and no other contents with absolute size.

Minimum zoom level

Limits how far the view can be zoomed out. This is an integer, unitless value. Maps show the entire world map (repeating in the east and west direction depending on the width) at level 0, while level 19 is the maximum covered by map material. For coordinate systems, this value has to be determined experimentally, if required, since it depends on the magnitude of the values to be displayed.

Has layer

This is the relation that links the coordinate view to its layers. It is created or deleted with the hierarchy on the left side of the configuration view. The order of the layers is also included in the web UI. The first layer is also first painted on the coordinate view, i.e., it is "farthest behind".

3.8.4.2.1. Map

The map is used to display geographic data. The configured attribute types of the layers (or the geometries that are used for the feature creation via script) therefore have to have a spatial reference — if none is available, WGS84 (epsg:4326) is assumed.

NOTE Currently, the map can only display data with WGS84 spatial reference (epsg:4326) in the web UI. Geographic data with a different spatial reference have to be converted before being displayed. For this purpose, an external software library is currently required, which has to be downloaded and installed separately. For more information, see the chapter on the configuration of geo-libraries.

The map currently only has one additional configuration option:

Show scale

If set, a dynamic scale bar is displayed at the bottom left of the map which can be used to estimate distances.

3.8.4.2.2. Cartesian coordinate system

The Cartesian coordinate system displays geometric data. If a geometry attribute with a spatial reference is used for the geometries to be displayed, it is ignored (without conversion).

The following additional configuration options are available for coordinate systems:

X axis

Configuration of the X axis (horizontal). If not specified, an automatic configuration with the label "x" is used. The following options are available for the axis configuration object:

Label

Defines the axis label. The default value is "x".

Scale factor

Specifies the scale ratio to the y axis. A value of 3 indicates, for example, that two raw values on the x axis are three times as far apart as the same values on the y axis. This is useful if the axes use very different orders of magnitude. Lines and polygons are distorted accordingly.

Y axis

Configuration of the Y axis (vertical). If not specified, an automatic configuration with the label "y" is used.

Background image

An image can be defined here which is shown as the background (e.g. the floor plan of a building if the features display rooms or furniture). The background image can be hidden in the web UI by means of the visible layer control, if required. Without further configuration, the image is placed in such a way that the lower left corner falls on the origin (0|0) of the coordinate system. An image pixel is set to one respective axis step. For small values, this is usually much too large, and the image might have to be moved. For this purpose, *Origin* and *Corner* can be configured as meta properties, which define the upper left and upper right corners. These are also geometry attributes which only accept points (e.g., `POINT (4 3)` for a corner at position (4|3)). If the two points do not reproduce the natural aspect ratio of the image, the background image may be distorted. Any scaling factors of the axes are also applied.

3.8.4.2.3. Defining initial view bounds

By default, the initial map or coordinate system section is selected in such a way that all features of all layers are displayed. However, the view bounds can also be set specifically, for example by means of an action or the *Initialization script* of the coordinate view. For this purpose, the `.setViewBounds()` function of the `$k.CoordinateSystemView` class is used. Any number of geometries is passed as arguments; the map section is then selected in such a way that all these geometries would be visible (but there do not necessarily have to be features with these geometries).

Here, the southwestern corner of the map section is set to N 49° E 6°, the northeastern corner to N 52° E 10°, for example:

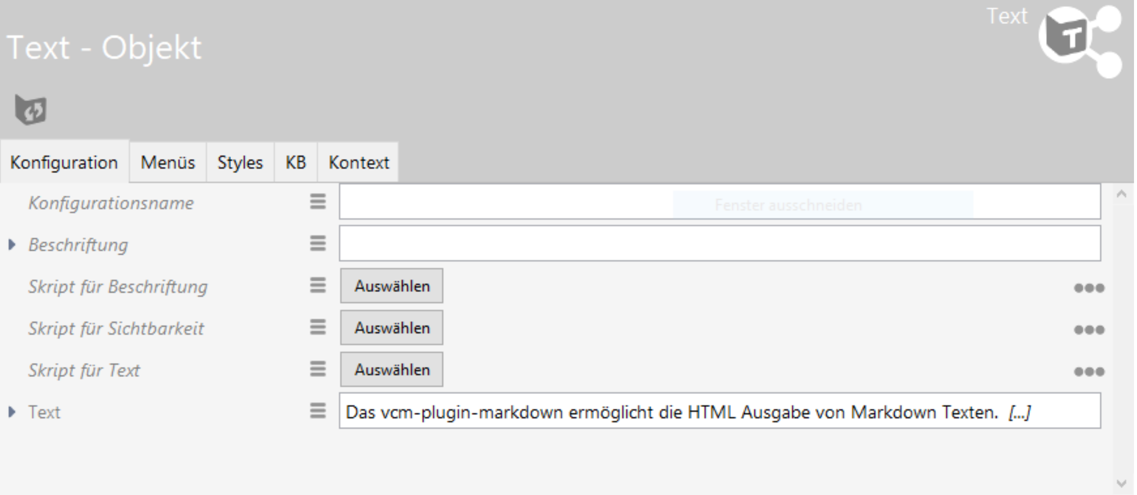
```
function initialize() {
  this.setViewBounds($g.Point(6, 49, 4326), $g.Point(10, 52, 4326))
}
```

3.8.5. vcm-plugin-markdown

The HTML output enabled by the VCM Markdown plug-in makes it possible to output Markdown texts.

It can be used by adding a style with the render mode *markdown* to one of the following configuration elements:

- *Static text* : The Viewconfig property *text* of the configuration element is interpreted as markdown.



Text - Objekt

Konfigurationsname

Beschriftung

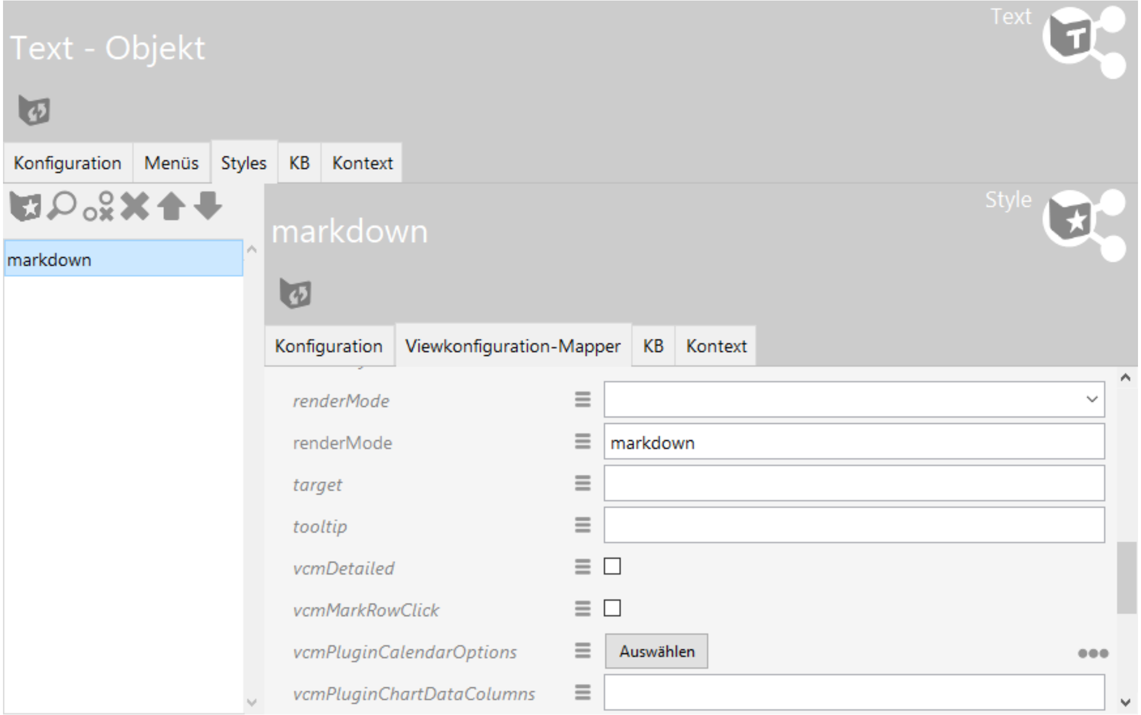
Skript für Beschriftung: Auswählen

Skript für Sichtbarkeit: Auswählen

Skript für Text: Auswählen

Text: Das vcm-plugin-markdown ermöglicht die HTML Ausgabe von Markdown Texten. [...]

To use the plug-in, the render mode called "markdown" must be entered on the “Style” tab:



Text - Objekt

markdown

renderMode

renderMode: markdown

target

tooltip

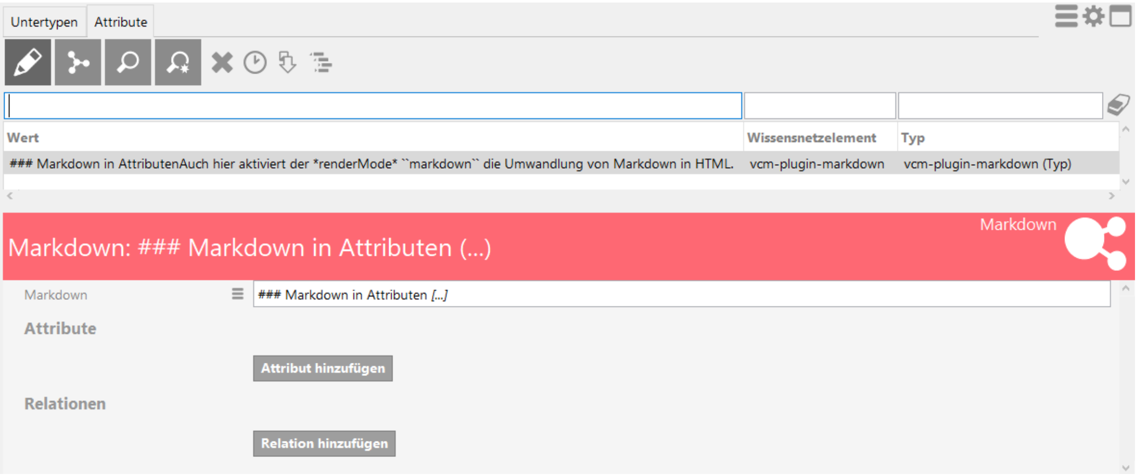
vcmDetailed: □

vcmMarkRowClick: □

vcmPluginCalendarOptions: Auswählen

vcmPluginChartDataColumns

- *Property* : This has the effect that the value of the attribute is interpreted as markdown.

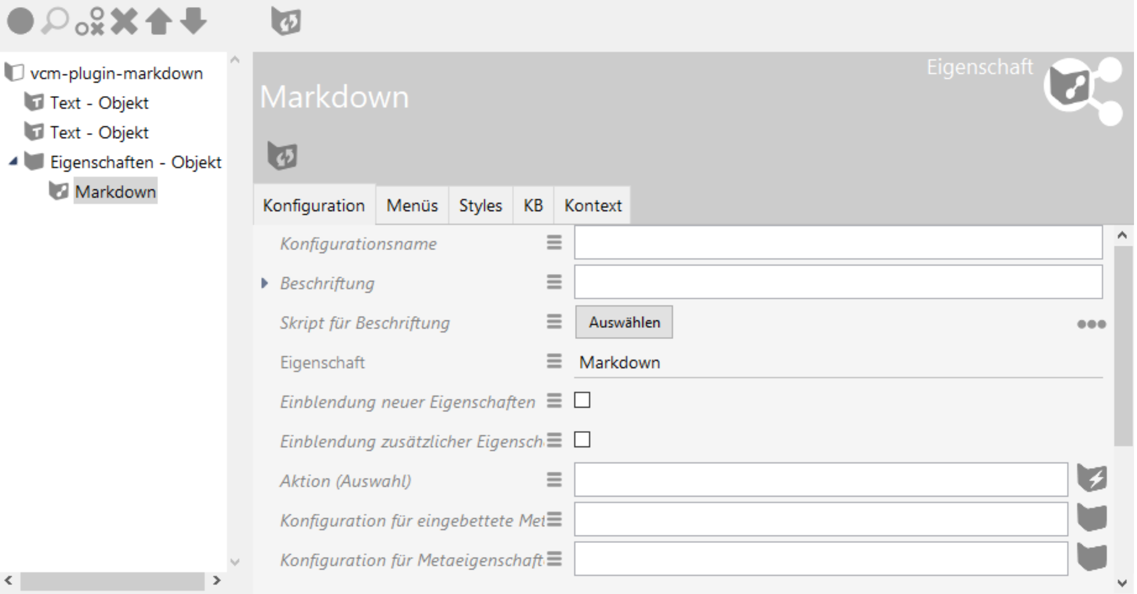


Wert	Wissensnetzelement	Typ
### Markdown in AttributenAuch hier aktiviert der *renderMode* ``markdown`` die Umwandlung von Markdown in HTML.	vcm-plugin-markdown	vcm-plugin-markdown (Typ)

Markdown: ### Markdown in Attributen (...)

Markdown: ### Markdown in Attributen [...]

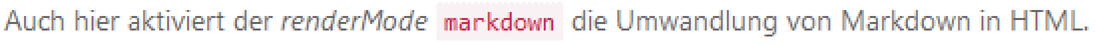
The view for the string attribute “Markdown” is configured using a property view:



Eigenschaft: Markdown

Like a text object, the property also receives the render mode “markdown.”

After rendering, the text has the following visual highlights in the web front-end:



Auch hier aktiviert der *renderMode* `markdown` die Umwandlung von Markdown in HTML.

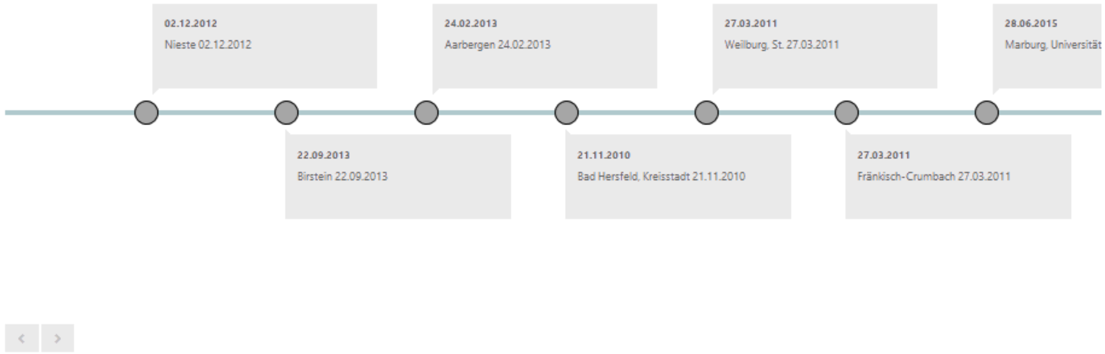
Further configuration of the plug-in is possible via the style attribute *vcmPluginMarkdownOptions* .

The plug-in uses the module markdown-it

3.8.6. vcm-plugin-timeline

Events can be displayed chronologically on a timeline using the vcm-plugin-timeline plugin.

The timeline can be horizontal or vertical. The horizontal variant of the timeline provides two additional buttons for scrolling when the timeline is wider than the space available. A scroll bar should be provided by the browser for the vertical variant in this case.



3.8.6.1. Configuration

First, a “script generated view” has to be created and its view type attribute must be set to “timeline.” In addition, a script must be placed on the view which provides data for the timeline, for example:

```
function customizeView (element, view) {    //other content   ....
  view.options = {
    layout: 'horizontal',
    // layout: 'vertical',
    itemHeight: 130
  }
  view.events = element.relationTargets('hasAlbum').map(function (album) {
    var obj = {}
    var name = album.name()
    var date = album.attributeValue('releaseDate')
    if (date) { date = date.toString() } else { date = '' }
    return obj = {name: name, date: date, elementId: album.idString()}
  })
  return view
}
```

This script can be used with the following parameters under ‘view.options’ in order to modify the appearance of the timeline:

- 'layout': determines the direction of the timeline, either 'horizontal' or ‘vertical.'
- 'itemHeight': Height of the elements on the time bar in pixels. If this is not set, all elements receive the height of the element that requires the most space.

Under 'view.events' an array has to be created which contain the results as objects. Each of these requires the attributes ‘name,' 'date' and ‘elementId.’

3.8.6.2. Styling

CSS rules are used to modify the default style of the timeline.

Depending on the alignment configured for the timeline, the following class hierarchy is available for this:

The text fields for the results can be modified using the following selectors:

```
.timelineVertical ul li
.timelineHorizontal ul li
```

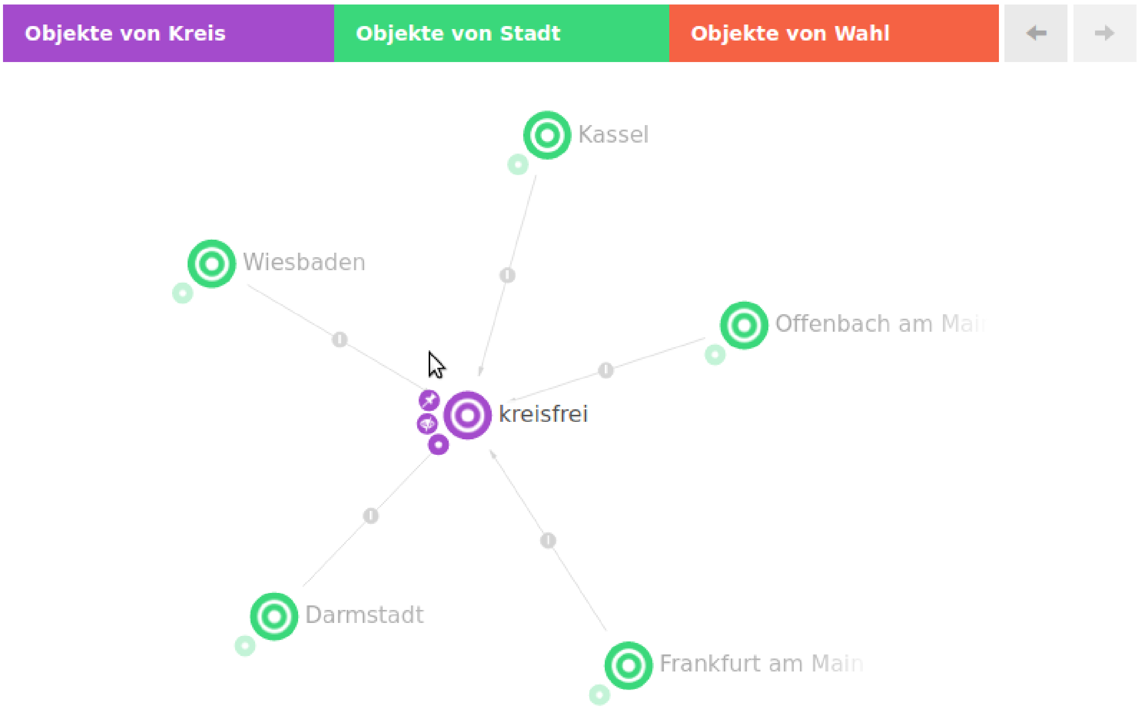
The mark points for the results can be modified using the following selector:

```
.timelineVertical ul li::after
.timelineHorizontal ul li::after
```

3.8.7. vcm-plugin-page

3.8.8. vcm-plugin-net-navigator

The vcm plugin Net-Navigator visualizes elements in a graph-like view.



3.8.8.1. Configuration

The plugin can be configured by means of styles.

Styles of the view

Style	Description
vcmPluginNetNavigatorOptions	A JSON object for the view options. See below for details
extra	Alternative to *vcmPluginNetNavigatorOptions*

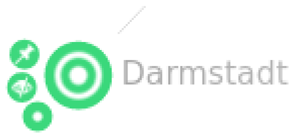
Options

Option	Description
vcmPluginNetNavigatorOptions .categories.hideLabel	Show/hide category labels
vcmPluginNetNavigatorOptions .categories.embeddedActions	Configure where actions are to be displayed. For true, they are shown next to the categories
vcmPluginNetNavigatorOptions .categories.compactActions	Combine actions in a menu
vcmPluginNetNavigatorOptions .history.enabled	Activates/deactivates the navigation history
vcmPluginNetNavigatorOptions .enableEditing	Activates/deactivates the option of creating new links between elements in the graph

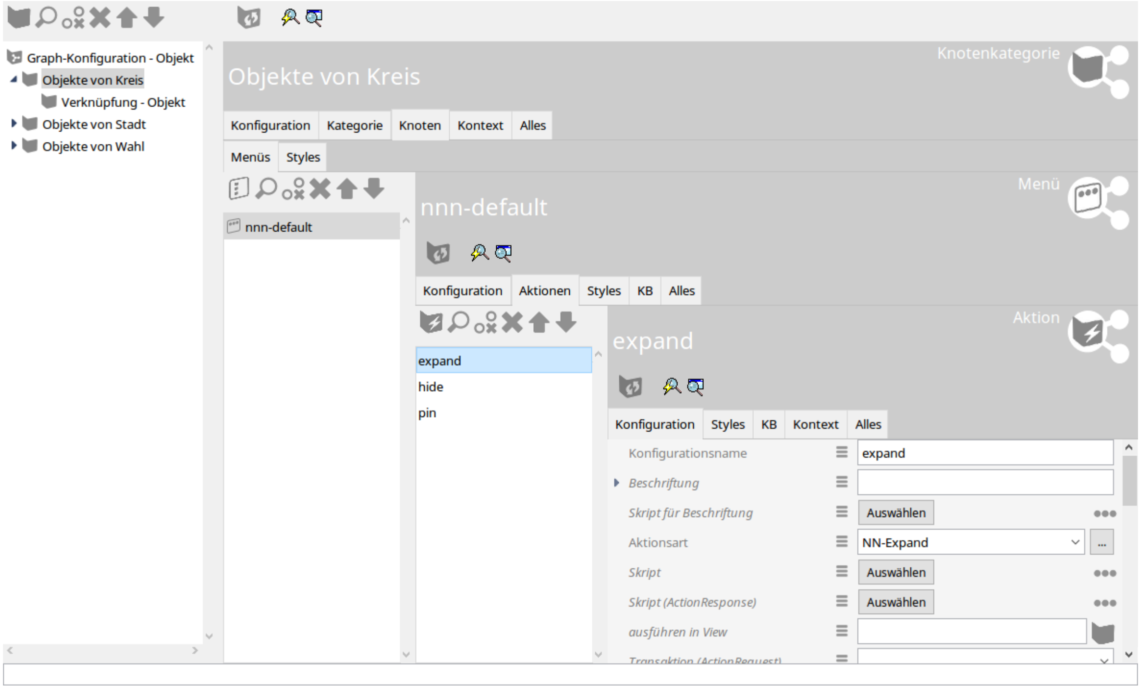
Option	Description
vcmPluginNetNavigatorOptions.nnOptions	Options for the Net-Navigator component
vcmPluginNetNavigatorOptions.nnOptions.overload.maxExpandNodes	Number of nodes that can be opened simultaneously before a query dialog regarding the relations to be opened appears. The default value is 5.
Styles of nodes	
extra	A JSON object for the node options. See below for details
Node options	
color	Overwrites the background color of the node
label	Overwrites the label of the node
icon	Overwrites the icon of the node
Styles of borders	
extra	A JSON object for the border options. See below for details
Border options	
color	Overwrites the background color of the border
label	Overwrites the label of the border

3.8.8.2. Actions

Nodes and relations can be supplemented with actions. These are arranged in a circle around a node or the relations.



Actions are configured in the graph configuration within a node category or link.



Konfigurationsname: expand

Beschriftung:

Skript für Beschriftung: Auswählen

Aktionsart: NN-Expand

Skript: Auswählen

Skript (ActionResponse): Auswählen

ausführen in View:

Transaktion (ActionRequest)

Preconfigured actions

Action type	Description
NN-Expand	A small plus symbol can be used to display neighboring nodes (for which a configuration exists)
NN-Hide	Hide a node
NN-Pin	Pin a node

Custom actions

A symbol image is always required for the display

3.8.8.3. Followups

The graph view reacts to the following follow-ups:

Follow-up	Data	Description
graph-show	{elementId: ["ID123_456"]}	Displays the elements in the graph. Elements already displayed are hidden
graph-join	{elementId: ["ID123_456"]}	Adds the elements to the graph. Elements already displayed are retained
graph-hide	{elementId: ["ID123_456"]}	Removes elements from the graph
graph-back		Moves one step back in the graph history

Follow-up	Data	Description
graph-forward		Moves one step forward in the graph history
graph-reload		Updates the elements in the graph.

Example: ActionResponse script which adds the root term to the graph view:

```
function actionResponse (element, context, actionResult) {
  var actionResponse = new $k.ActionResponse()

  actionResponse.setFollowup('graph-join')
  actionResponse.setData({
    elementId: [$k.rootType().idString()]
  })
  return actionResponse
}
```

3.9. Special configuration

This chapter covers specific application cases in the ViewConfiguration Mapper which require a combination of viewconfig element, search and/or script.

3.9.1. Switching language of web frontend

NOTE This cuntionality is available for the VCM *after* Version 11.0.0.

For switching the UI language, there are two possibilities available:

1. By means of an action with a configured *Script (ActionResponse)* and a *followup* "switch-language":

```
function actionResponse(element, context, resultModel) {
  var actionResponse = new $k.ActionResponse();

  actionResponse.setFollowup('switch-language')
  actionResponse.setData({
    language: 'en-US'
  })

 return actionResponse;
}
```

2. By means of the query parameter "lang". Examples:
    - http://localhost:8815/viewconfig?lang=en
    - http://localhost:8815/viewconfig/random/bookmark/path/?lang=en-US
    - http://localhost:8815/viewconfig/bookmark/with/query?bookmarkParam1=value&lang=de_DE

    In both cases, the "lang" parameter / "language" must be in the format of the Accept-Language Language Directive.

3.9.2. Display change history in a web frontend

Prerequisite:

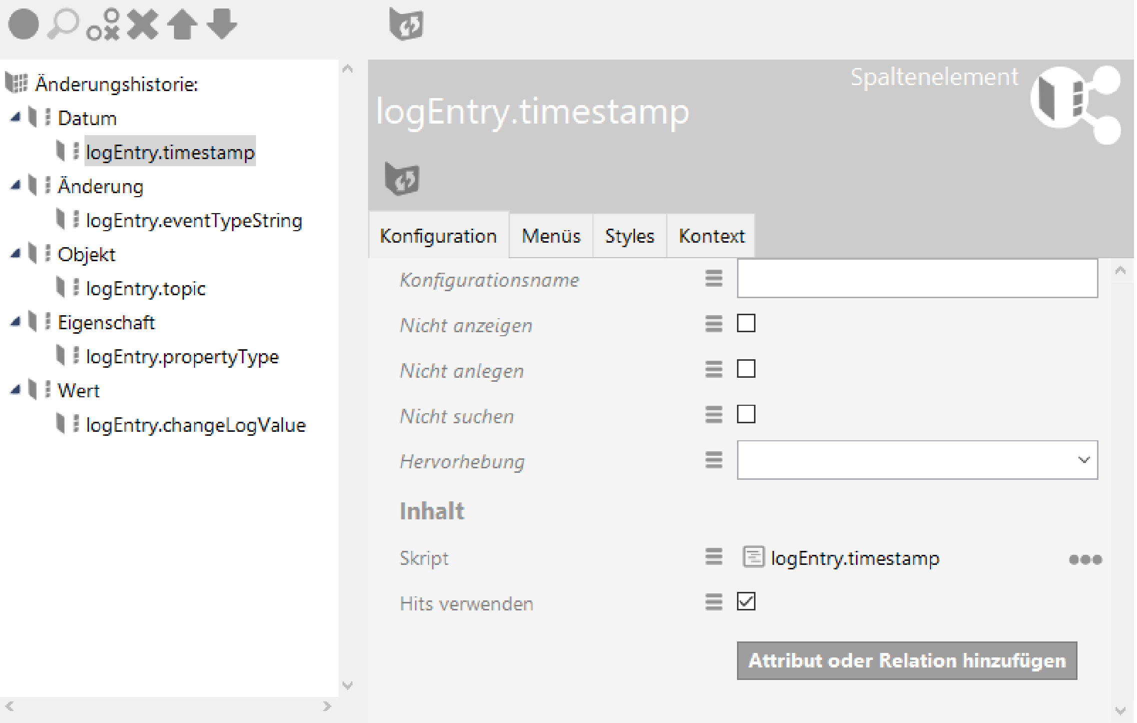
- Change history recording has been set up: For changes to elements to be recorded, it is necessary to set up a meta-attribute with the internal name “changeLog” of the “string” value type. See also the “ChangeLog Trigger” chapter.
- The table as described in the following needs to be placed within a panel which gets the changeLog attribute in forms of a domain model (context element).

NOTE Using the table within a *Query View* or a *Search result View* will not work.

- If the table needs to be placed within a grouping view, the domain model needs to be changed to the changeLog attribute. To do so, the table must be linked to the grouping view via the relation "subconfiguration of" and the relation needs a "Script for domain model" which returns the changeLog attribute (not its value).

View configuration

The view configuration of ChangeLog entries for the web front-end can be implemented in the form of a table via ViewConfiguration Mapper:



Änderungshistorie:
- Datum
  - logEntry.timestamp
- Änderung
  - logEntry.eventTypeString
- Objekt
  - logEntry.topic
- Eigenschaft
  - logEntry.propertyType
- Wert
  - logEntry.changeLogValue

Spaltenelement

logEntry.timestamp

Konfigurationsname

Nicht anzeigen □

Nicht anlegen □

Nicht suchen □

Hervorhebung

Inhalt

Skript logEntry.timestamp

Hits verwenden ☑

From the change history it is possible to read out values such as date, change, affected element, modified properties etc.

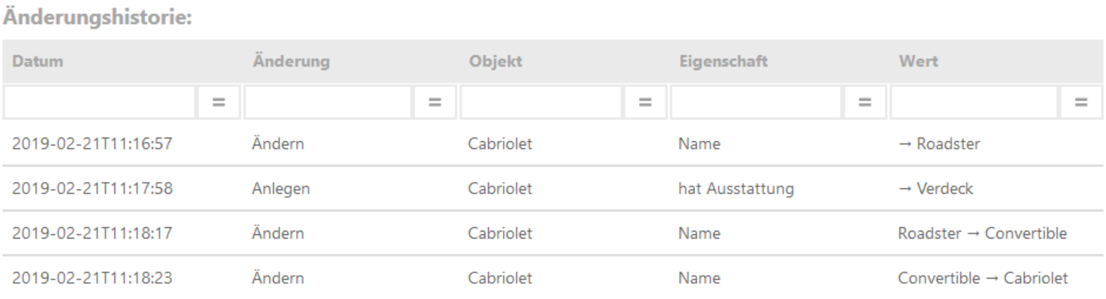
For each of these values, it is necessary to create a column configuration that contains a script as its column element. The script processes the entries as per the $k.HistoryChangeLogEntry class and returns the relevant value, filtered by value type, for the column element (for syntax, see JavaScript API). For script examples see the sections below.

As only one attribute element is generated for each semantic element of the ChangeLog attribute type to be logged, all entries in the change history are written to the string as an attribute value. Therefore, the entries of the string must be read out individually by means of the script. To ensure that the entries are even available, it is necessary to activate (tick) the “Use hits” option. For more information, see the “’Hit’ content model” chapter.

NOTE In the view configuration, the ChangeLog entries can be handled like the “hits” for

> a query. If the “Use hits” option is not activated, the semantic element is output without properties and without the corresponding ChangeLog entry (result: as many empty column elements as there are ChangeLog entries).
>
> To ensure that the view configuration of the table is told for which attribute the ChangeLog entries should be displayed, influence from another view is required, on the basis of which the context element is forwarded to the table.

The output table in the web front-end looks like this:



Änderungshistorie:

Datum	Änderung	Objekt	Eigenschaft	Wert
=	=	=	=	=
2019-02-21T11:16:57	Ändern	Cabriolet	Name	→ Roadster
2019-02-21T11:17:58	Anlegen	Cabriolet	hat Ausstattung	→ Verdeck
2019-02-21T11:18:17	Ändern	Cabriolet	Name	Roadster → Convertible
2019-02-21T11:18:23	Ändern	Cabriolet	Name	Convertible → Cabriolet

In this example, an object called “Roadster” has been created, a relation has obtained the “has equipment” relation for the “Folding top” object, and then the object has been renamed “Cabriolet”. Due to the script, each row in the “Object” column displays the *current* name of the object that was modified.

Script examples for the ChangeLog output

Change date

```
function cellValues (logEntry, queryParameters) {
    return [ convertToLocal(logEntry.timestamp()) ]
}

function filter (elements, queryParameters, columnSearchValue) {
    return elements
}

function convertToLocal (date) {
    return new $k.DateTime(date.valueOf() + (date.getTimezoneOffset() * 60
* 1000))
}
```

Change

```
function cellValues (logEntry, queryParameters) {
    return [ logEntry.eventTypeString() ]
}

function filter (elements, queryParameters, columnSearchValue) {
    return elements
}
```

Object

```
function cellValues (logEntry, queryParameters) {
    return [ logEntry.topic() && logEntry.topic().name() ]
}

function filter (elements, queryParameters, columnSearchValue) {
    return elements
}
```

Property

```
function cellValues (logEntry, queryParameters) {
    if(logEntry.propertyType()) {
        return [ logEntry.propertyType().name() ]
    } else {
        return []
    }
}

function filter (elements, queryParameters, columnSearchValue) {
    return elements
}
```

Value

```
function cellValues (logEntry, queryParameters) {
    var oldValue = logEntry.oldValue()
    if (!oldValue) { oldValue = '' } else if (oldValue.length > 100) {
        oldValue = oldValue.substr(0, 100) + '...'
    }
    var newValue = logEntry.newValue()
    if (!newValue) { newValue = '' } else if (newValue.length > 100) {
        newValue = newValue.substr(0, 100) + '...'
    }
    return [ oldValue + '  ' + newValue ]
}

function filter (elements, queryParameters, columnSearchValue) {
    return elements
}
```

3.10. Installation

The ViewConfig Mapper is a web frontend application for the Knowledge Graph and it can be provided for use as follows:

- Make ViewConfiguration Mapper available as a ZIP file via static REST resource
- Reference to VCM demo with sourcing option (link)
- Using (other) web server
- Productive operation/test operation

3.10.1. Configuration of web servers

The View Configuration Mapper (VCM) running in the browser internally requires knowledge of parameters for its functionality. If you are connecting directly to a bridge delivering VCM contents and its REST services, the VCM running in the browser is capable to derive the parameters from its first invocation.

On the other hand, if you want to run the bridge oder bridges behind some reverse proxy infrastructure (as it it common e.g. in cloud projects, with load-balancing or running multiple services behind a single virtual hostname), the services very often are not located on the root path of the URL used in the users browser.

The parameters are also relevant, if the installation is different from the defaults mentioned.

The parameters are:

Parameter name	Description	Default value
iv-root-url	The entry point of the bridge. Either a URL with schema, host and port or an absolute path. Examples: • http://localhost:8815 • https://localhost:8815 • https://i-views.com/myproject/x/y • /myproject/x/y	/
iv-path-viewconfig	Absolute or relative (to *iv-root-url*) path to the viewconfig service for this frontend. Examples: • viewconfig/ • my-service/ • /myproject/x/y/service	Path of the service belonging to the request

Parameter name	Description	Default value
iv-path-bookmarks	Absolute or relative (to *iv-root-url*) path from which the bookmark URLs are formed.	<iv-path-viewconfig >
iv-path-static	Absolute or relative (to iv-root-url) path to the static resources. Examples: • viewconfig/viewconfigmapper • viewconfig/my-statics • my-service/viewconfigmapper • /static/	<iv-path-viewconfig>/viewconfigmapper
iv-cookie-path	defines the path scope of the authzentication cookie	<iv-path-viewconfig>
iv-secure-cookies	Determines if the secure flag should be set for cookies. Possible values: *true* or *false*	*false*

Usually some kind of frontend web-server with reverse proxy capabilities is used, like Apache, nginx, Traefik, etc. In that servers configuration you can add directives to send the parameters as headers to the bridge, which will deliver them back to the VCM.

3.11. Extension project

3.11.1. Development environment

- Node.js/Webpack/etc.

3.11.2. Technical details

- Diagram showing information flow in the case of actions
- Component state

4. i-views services

4.1. General

The i-views services can be configured by command line parameters or by a configuration file.

The command line parameters have precedence over the configuration file settings.

4.1.1. Configuration file

Some settings can be specified by means of a configuration file (extension .ini). The structure of the file is as follows:

```
[Section]
parameterName1=parameterValue1
parameterName2=parameterValue2
...
```

The name of the default ini-file depends on the tool-type, e.g. a standard KnowledgeBuilder will be named "kb.exe" (or "kb.im" respectively), so by default it will search for "kb.ini" as ini-file. Please keep in mind, that renaming the file will not change the name of the default ini-file a tool is looking for.

The name of the ini file can be specified via the command-line option "inifile":

```
-inifile <File name>, -ini <File name>
```

Below is a list of configurations that can be used for any service. For service-specific settings, see the “configuration file” section of the relevant service.

4.1.1.1. Macros

Within ini files, further ini files can be included:

```
$(include:Dateiname)
```

This allows information from several files (e.g. host name) being stored into one common file.

- in one and the same line, the include instruction mustn’t be surrounded by further content; otherwise, the include instruction will not be recognized
- include corresponds to a textual replacement

- include can be nested: the included file turn may include further files
- the file name may consist of path specifications; within Windows, slashes / will be replaced by backslashes \ automatically

Including environment variables is possible as well:

```
$(env:Variablenname)
```

“`$(env:USERDNSDOMAIN)`” is translated into "i-views.com" for example.

NOTE This macro only can be used in key values, for categories / key names it will not be replaced.

Example:

`jobclient.ini`

```
$(include:../shared/host.ini)
$(include:../shared/volume.ini)
jobPools=lucene, KLuceneAdminJob
[JNI]
classPath=...
```

`bridge.ini`

```
$(include:../shared/host.ini)
[KHTTPRestBridge]
$(include:../shared/volume.ini)
port=8815
```

`../shared/host.ini`

```
host=$(env:COMPUTERNAME).$(env:USERDNSDOMAIN)
```

`../shared/volume.ini`

```
volume=master
```

4.1.1.2. Logging settings

```
loglevel = <LogLevel>
```

Configures the messages that should appear in the log:

FATAL ERROR	Critical error messages only
ERROR	Error messages only
WARNING	Warnings and error messages only
NORMAL (*default value*)	All messages excluding debug outputs.
NOTIFY	All messages including several debug outputs
DEBUG	All messages including all debug outputs

```
debug = true/false
```

Obsolete. Sets the log level to DEBUG for true, and to NORMAL for false. Only evaluated if logLevel is not set

```
nolog = true/false
```

Obsolete. If true, logTargets=null. Only evaluated if logTargets is not set

```
channels = <Channel1> [,<Channel2>,...]
```

Names of channel filters. Channel filters are used to output only the log messages belonging to the specified channel filters. The name of a channel filter indicates the topic area to which the log outputs belong. To find out which channel filters are possible, use the -availableChannels parameter in the command line.

```
channelLevels = <Channel1>:<Level1> [,<Channel2>:<Level2>,...]
```

Targeted configuration of the log level for the respective channel.

```
logTargets = <Name1> [,<Name2>,...]
```

Names of log targets. For the configuration, see the “Log targets” section.

```
logprefix = <Prefix1> [, <Prefix2>,... ]
```

Additional prefixes that are added for each log output.

Prefix	Description
pid	Process ID of the application
$proc$	ID of the current Smalltalk thread
$alloc$	allocated memory on the VM (in megabyte)
$free$	Free memory on the VM (in megabyte)
$incGC$	Status of incremental GCs
os	Information about the OS
cmd	Command line
$build$	Build version
$coast$	COAST version

If the prefix is not contained in this list the prefix is output without change.

```
logTimestampFormat = <FormatString>
```

Formatting specification for the timestamp of the log entry, e.g. “hh:mm:ss”.

```
exceptionLogSize = <Integer>
```

Sets the maximum size for the StackTrace supplied with an error message.

4.1.1.2.1. Log targets

Log targets can be used to specify different targets for logging; it is possible to configure the log level, channels, formatting and more for each of them. For each specified name from the log targets list, a configuration must be specified in section [<configuration name>]

```
[Default]
logTargets=erroroutput

[erroroutput]
type=stderr
format=json
```

```
loglevel= ERROR
```

is an example that configures the output of all error messages in the JSON format in the standard error stream.

The null log target is an exception: if logTargets=null is configured, no configuration section needs to be created. If this section is missing, this has the same significance as the following configuration

```
[Default]
logTargets=null

[null]
type=null
```

It is however possible to use null as the identifier for any log target configuration.

Generally it is possible, just as in the general configuration, to specify loglevel, debug, channels, channelLevels, logprefix and logTimestampFormat (see above). The configuration of the log target always takes precedence; if none is specified, the general configuration is used.

In addition there are several other configuration options:

```
format = <Format>
```

Specifies the output format. Possible values:

- **plain** : Standard formatting in machine-readable form if possible
- **json** : Single-line output as JSON string, above all for machine processing

```
type = <Target type>
```

Specifies the type of the output. This configuration MUST be specified, otherwise the log target is ignored. The following section contains the description and other configuration options for the different types:

file

Output in a log file.

```
file = <File name>
```

Specifies the file name of the target file.

```
maxLogSize = <size>
```

The maximum size of the log file before the old log file is archived and a new one is written. For values below 1,024, the output is to be understood in MB.

```
maxBacklogFiles = <amount>
```

The maximum amount of archived log files. When a new one begins the oldest one is deleted.

transcript

Output to the transcript, can also be redirected to a log file and therefore accepts the same configuration as "file".

stdout

Output to the standard out stream.

stderr

Output to the standard error stream.

mail

Sends the log output via email.

```
[errorMail]
type = mail
loglevel =  ERROR
;Sender address:
sender = mail@example.org
;Recipient address:
recipient = rec@example.org
;Mail server:
smtpHost = stmp.example.org
;Port of the mail server:
smptPort = 465
;If true, activates the secured connection (TLS/SSL).
;If true, the username and password must have been set.
tls = true
username = mail@example.org
password = 12345abc
```

```
;Amount of attempts to resend the email in case of failure:
retries = 3
;Waiting time between the attempts in seconds:
retryDelay = 5
```

mailfile

Like mail, however the outputs with a low log level are first collected and only sent via email when an entry with a high level is logged.

```
mailSendLevel = <LogLevel>
```

Sets the log level from which the email is sent.

syslog

Output as UDP datagram to a syslog client.

```
format = <Format>
```

Unlike with other log targets, json and plain are not supported as format; instead, the syslog version can be specified here:

- **rfc5424** : Formats the message as per RFC 5424. Most data are placed in the structured data field in structured form. Only the actual log message is transmitted in the message field .
- **rfc3164** : Formats the message as per RFC 3164. As this standard has no structured data field, the corresponding data are placed at the beginning of the message field in the same formatting.

NOTE The timestamp is specified in the local time of the sending computer as per the standard.

```
facility = <Integer>
```

The facility as an integer. For detailed information see https://tools.ietf.org/html/rfc5424#section-6.2.1

```
targetHostname = <Hostname>
```

The host name of the target system. If not specified, localhost is used.

```
targetPort = <Integer>
```

The target port. If none is specified, the syslog standard port 514 is used.

```
hostname = <Hostname>
```

The host name of the sender. If none is specified, the host name of the system is read out.

```
appname = <Name>
```

Name of the sending application. If none is specified, the name of the EXE is used.

```
maxMessageSize = <Integer>
```

The maximum message size in bytes. If none is specified, the maximum size for UDP is used. To shorten the message, structured data is removed incrementally at first, and the message is cut off if necessary. The message remains in the valid syslog format even after shortening.

null

For suppressing the log outputs. No options are read out.

4.1.1.3. Text extraction

To extract texts and meta-data from file contents, use of Apache Tika must be set up:

- Download the current Tika app (e.g. app-1.18.jar) from the website http://tika.apache.org/ and copy it into the directory of the Job-Client.
- Add the following entry to the configuration file (e.g. jobclient.ini or bridge.ini):

```
[text-extraction]
tikaJavaParams=-Xmx1024M
tikaJarPath=tika-app-1.18.jar
; Optional: Maximum size of the binary files,
;  for which text is extracted
; extractedTextSizeLimit=100000
;
; Optional: Java path, the default value is 'java'
; extractorPath=C:\Program Files\Java\jdk-9\bin\java.exe
```

4.1.1.4. HTTP Proxy configuration

Depending on the network infrastructure HTTP connections from machines are not directly possible, but need to use the existing HTTP proxy infrastructure in the network. By default i-views components do not try to use such infrastructure but they can be configured to do so.

The most simple configuration attempts to use the operating systems configured HTTP proxy infrastructure. To use this, add the following section to the tools ini-file:

```
[NetClient]
HttpClient.useProxy=true
```

With this configuration i-views will try to read and interpret the operating systems proxy configuration at the next start of the tool.

In case this does not work, i-views can also be configured to explicitely use a specific proxy setting with the following ini-snippet:

```
[NetClient]
HttpClient.proxyHost=HOSTNAME
HttpClient.proxyPort=PORT
```

The values for HOSTNAME and PORT can be obtained either by the network administration team or users can try to read the configuration from the operating system or installed browsers. Here some known sources are documented:

- Windows: in a PowerShell window execute

  ```
  [System.Net.WebProxy]::GetDefaultProxy()
  ```

  In the field "Address" you should find a hostname and port-number separated by a colon character.

- on Windows Google Chrome and Microsoft Edge/Internet Explorer use the above Windows setting.
- Firefox: Menu "Tools, Options", on page "General", item "Network Settings", entry "HTTP Proxy"

4.1.1.5. TLS configuration

If a service provides a HTTPS interface, then a certicate for TLS must be specified in the category **[tls]** . The configuration depends on the operating system.

Windows

```
[tls]
certificateName=MyCert
```

certificateName specifies the "friendly name" property of the certificate in the Windows certificate store. Note that this property is not a part of the certificate itself.

The certificate must be put in the personal certificate store of the user. This means that the service must be run under a user account, not with the local system account.

The private key must be stored, too.

A self-signed certificate for testing can be generated with Powershell:

```
New-SelfSignedCertificate -CertStoreLocation Cert:\CurrentUser\My -DnsName
"mycomputer.mydomain.org" -FriendlyName "MyCert" -NotAfter (Get-Date)
.AddYears(10)
```

Linux

```
[tls]
certificatePath=myCert.cert
privateKeyPath=myCert.key
```

certificatePath is the path to the certificate file. It must be stored in PEM format. *privateKeyPath* is the path to the private key file. It must be stored in PEM format, without password protection.

4.2. Mediator

4.2.1. General

The i-views server provides consistent and persistent data storage, and ensures that the data on the i-views clients that are connected are up-to-date.

Data is managed in an object-oriented database that uses an optimistic transaction system to allow cooperative work on the Knowledge Graph.

Functioning as a communication center, the i-views server ensures clients and services are synchronized. As a basic mechanism, it makes a shared object space and active updates available for this.

The i-views server can be operated in three modes:

1. Classic/Compact: The server starts as an individual process in this mode – the so-called “mediator”.
2. Multiprocess: The server starts at least two processes in this mode. This results in higher memory usage than in compact mode, however many jobs can be executed in parallel.
3. Distributed: The server components “stock” and “dispatcher” can be configured and operated separately in this mode. This makes it possible to distribute the server components across different computer nodes.

4.2.2. System requirements

The i-views server is platform-independent and runs on all popular operating systems, e.g. Windows and Linux. GUI components also support MacOS. Other platforms on request.

OS	Version	Processor	Supported
Windows	Windows 11, 10 22H2; Windows Server 2022, 2025	x86	Yes
Linux	Kernel >= 3.10, glibc >= 2.17	x86	Yes
Mac	macOS >= 12.x	x86, Apple Silicon	Yes

4.2.3. Operating modes

The following command line arguments generally determine the mode in which the server is started. Without these arguments, the server starts in the compact “mediator” mode.

```
-stock
```

Starts the “Stock” server component, which is responsible for persistent data storage.

```
-dispatcher
```

Starts the “Dispatcher” server component, which is responsible for the synchronization of the clients and for the distribution of “active updates”.

```
-server
```

Starts the complete server in the multiprocess mode.

4.2.3.1. Multi process mode (-server)

The start parameter **-server** automatically starts a stock and a dispatcher. The dispatcher opens a server on the default port (30068). The port of the stock is selected automatically. Authentication tokens between the two processes are generated automatically and do not have to be configured.

> **NOTE** It is important that all clients (Knowledge Builder, bridge, batch tool etc.) have access to stock and dispatcher.

If this is only possible for certain ports, stock and dispatcher must be configured explicitly. The local directory uses the same configuration files as the actual distributed mode

- **dispatcher.ini** configures the dispatcher process
- **stock.ini** configures the stock process

Other configuration files cannot be used at present.

4.2.3.2. Configuration of the Stock

The stock is responsible for storing the data on the hard drive. A simple is example of this is the configuration file **stock.ini**

```
[Default]
interfaces=cnp://0.0.0.0:4998
```

This configuration ensures that the stock listens on port 4998 and communicates via the native Coast protocol.

The following parameters can be used:

```
port=<port number>
```

Starts the stock with port number <num>. Without this entry, port 30068 is used.

This parameter is obsolete. It is replaced by the “interfaces” parameter. The entry “port=1234” corresponds to the entry “interfaces=cnp://0.0.0.0:1234.” In contrast to the start parameter, multiple values are possible here, which can be listed consecutively in comma-separated form.

```
interfaces=<interface-1>,<interface-2>,...<interface-n>
```

This parameter determines the addresses and protocols used to access the server. Several values are permissible and are separated by a comma.

Possible protocols are:

- http
- https
- cnp
- cnps

The abbreviation “cnp” stands for “Coast Native Protocol” or “Coast Native Protocol Secure.” The syntactic structure of an interface definition is equivalent to a URL with schema, host and port.

The host component is used to manage which network address(es) is/are used to access the server. For example:

- `0.0.0.0` binds to all IPv4 interfaces
- `[::1]` binds to the IPv6 loopback only

The “http” and “https” protocols can be rerouted via proxies, allowing the server to be accessed using an IIS running on port 443, for example.

```
baseDirectory=<Directory>
```

Sets the directory in which the “volumes” directory is located. If this value is supposed to end on volumes, this directory is used directly without creating an additional “volumes” directory below it.

```
volumesDirectory=<Directory>
```

The Knowledge Graphs are stored in this directory. Here, “volumes” is entered as the default value.

```
backupDirectory=<Directory>
```

Specifies the directory to which the Knowledge Graph backups are written and also read for restoring. Only complete directory names are allowed, no relative paths.

```
networkBufferSize=<Size in bytes>
```

This specifies the size of the buffer that is used for sending/receiving data. The default value is 20480. In some infrastructures you can specify

```
networkBufferSize=4096
```

to achieve a higher throughput.

```
flushJournalThreshold=<Number of clusters>
```

Specifies the maximum value that “changed cluster" + “index cluster" may reach in a saving process. If the value for “changed clusters” has already been exceeded, no “index clusters” are saved; these are kept with the journal instead.

A low value (e.g. 50) guarantees fast saving time but can potentially generate a large journal.

A value of “0” deactivates journaling. The default value is “2000.”

A “flush” of the journal is executed after complete saving at the latest. This in turn is triggered if:

- The mediator is closed
- The last client of the corresponding volume is logged off
- Saving is triggered by a full-save job (see jobs.ini)

```
autoSaveTimeInterval=<Wait interval in seconds>
```

Specifies the maximum wait time in seconds until automatic saving takes place again after the last cluster was saved. The default value is 15.

```
clientTimeout=<Timeout in seconds>
```

Specifies the time in seconds that a connected client may not have sent an Alive message before the mediator regards it as inactive and excludes it.

```
password.flavour=190133293071522928001864719805591376361
password.hash=111995451824586607054955998020526241717349657914270806386949
```

```
54247035513239844
```

The mediator password is calculated together with a random flavor to produce a (SHA256) hash value. These two pieces of information then suffice for the mediator to check an authentication request. During authentication on the server, the user name must be specified as “Server.admin.” To determine these values, you can use

```
password.update=new_password
```

Trigger the server to compute a new flavor and suitable hash value and write these to the ini file. The “password.update” entry is removed in this process.

```
password=<String>
```

The obsolete but still supported way of setting the mediator password. This variant must not be used at the same time as the SHA256 hash variant.

Changed

```
skipVolumesCheck=<true|false>
```

Specifies whether the check of the existing volume that is normally performed after starting the mediator is skipped

Changed

4.2.3.2.1. Memory settings

The following three parameters are used to configure the memory allocation and usage. You may specify values either in megabytes or actual bytes, whereby it is assumed that values under 1048576 refer to megabytes.

```
maxMemory=<Integer, in MB>
```

Maximum base memory usage permitted. A minimum of 50 MB, the total physical base memory available (under Windows) or 512 MB by default.

```
baseMemory=<Integer, in MB>
```

Base memory usage after which efforts to free up memory increase. By default 0.6 * maxMemory.

(alias: “growthRegimeUpperBound”)

```
freeMemoryBound=<Integer, in MB> [10]
```

If memory that is being used, but is no longer needed, exceeds this limit, it is freed up for use again.

4.2.3.2.2. Blob service configuration

If the mediator is supposed to be started with an integrated BLOB service so that the BLOBs are stored separately from the database on the hard drive, the following setting must be entered in the “mediator.ini” file:

```
startBlobService=true
```

For more information on this, refer to the documentation of the BLOB service.

4.2.3.3. Configuration of the Dispatcher

The dispatcher is responsible for transaction control and coordination of several clients. A simple configuration file is

```
[Default]

interfaces=cnp://0.0.0.0:5000

stockAddress=cnp://localhost:4998
stockAuthentication=dsfkhvqw3n9485z432504
```

This configuration opens a server on port 5000 to which clients can connect. The dispatcher looks for the stock under localhost:4998. This address is also the address that clients use to fetch data from stock

If dispatcher and stock are running on the same server, the dispatcher tells its clients its own host name to ensure connections via the network work.

Token dsfkhvqw3n9485z432504 is used to authenticate the dispatcher on the stock. This token must be set in the stock configuration using the "password.*” keys .

4.2.4. Installation

By principle, the i-views server does not require a specific installation, i.e. it can be started ad-hoc from any directory.

However, it must be ensured that the necessary access rights (read/write/generate) have been set for the server’s working directory and all subdirectories.

4.2.4.1. Start parameter

A range of parameters can also be transferred to the mediator process when starting. Most parameters can, however, also be specified in the mediator.ini, allowing the mediator to be started using a simple command line. When doing so, the rule is that the parameters specified on the command line take precedence over any parameters specified twice in the .ini file.

The complete list of possible start parameters is output by the mediator when called up using the parameter “-?”.

```
-interface <interface-1>
```

This parameter determines the addresses and protocols used to access the server. Possible protocols are: http, https, cnp, cnps. The abbreviation “cnp” stands for “Coast Native Protocol” or “Coast Native Protocol Secure.” The syntactic structure of an interface definition is equivalent to a URL with schema, host and port. The host component is used to manage which network address(es) is/are used to access the server. For example: “0.0.0.0”=IPv4 all interfaces, “[::1]”=IPv6 loopback only.

The “http” and “https” protocols can be rerouted via proxies, allowing the server to be accessed using an IIS running on port 443, for example.

```
-clientTimeout <sec>
```

Sets the time within which a client must automatically answer to <sec> seconds. The value should be set to a minimum of 600 (which is also the default value).

```
-baseDirectory <directory>
```

Sets the directory in which the “Volumes” directory is located. Along with the “Volumes” subdirectory, the directories for backups and downloads are created. This parameter used to be called “-volumes”.

The following parameters give commands to the mediator executable to run specific jobs, without functioning as a server for Knowledge Graph afterwards.

```
-quickRecover <volume> -recover <volume>
```

In the event that the mediator was not shut down properly (e.g. computer crash), lock files in volumes that were in use stop running. The volume will then not be able to be entered. In order to disable the lock, remove the lock by calling -quickRecover <volume>. It cannot be called when (possible) inconsistencies were found. In this case, the start parameter -recover must be used.

NOTE The working directory called must be the directory that contains the “volumes” directory. The “volumes” parameter therefore does not function in this case.

```
-bfscommand <volume> <command>
```

Executes commands that are identified by the BlockFileSystem.

Command line parameter for logging:

```
-nolog
```

Disables logging

```
-loglevel <integer>
```

Configures the messages that should appear in the log:

- 0: All messages including debug outputs
- 10 (default value): All messages excluding debug outputs
- 20: Warnings and error messages only
- 30: Error messages only

```
-logfile <file name>, -log <file name>
```

Name of the log file that is used instead of the standard log file. It is important to change this parameter when several clients are being started in the same working directory.

```
-debug
```

Switches logging to debug mode

```
-log <logname>
```

Sets the log file to <logname>.

4.2.4.2. Configuration file "mediator.ini"

A number of mediator settings can also be defined in the configuration file mediator.ini. The structure of the file is as follows:

```
[Default]
parameterName1=parameterValue1
parameterName2=parameterValue2
...
```

The following parameters can be used at this point:

Network communication

```
port=<port number>
```

Starts the server with port number <num>. Without this entry, port 30068 is used.

This parameter is obsolete. It is replaced by the “interfaces” parameter. The entry “port=1234” corresponds to the entry “interfaces=cnp://0.0.0.0:1234.” In contrast to the start parameter, multiple values are possible here, which can be listed consecutively in comma-separated form.

```
interfaces=<interface-1>,<interface-2>,...<interface-n>
```

This parameter determines the addresses and protocols used to access the server. Several values are permissible and are separated by a comma. Possible protocols are: http, https, cnp, cnps. The abbreviation “cnp” stands for “Coast Native Protocol” or “Coast Native Protocol Secure.” The syntactic structure of an interface definition is equivalent to a URL with schema, host and port. The host component is used to manage which network address(es) is/are used to access the server. For example: “0.0.0.0”=IPv4 all interfaces, “[::1]”=IPv6 loopback only.

The “http” and “https” protocols can be rerouted via proxies, allowing the server to be accessed using an IIS running on port 443, for example.

For SSL communication (cnps:// or https://), the file paths for certification and private key must also be specified in the configuration file:

```
certificate=name of the .crt file
privateKey=name of the .key file
```

Directories

```
baseDirectory=<Directory>
```

Sets the directory in which the “volumes” directory is located. If this value is supposed to end on volumes, this directory is used directly without creating an additional “volumes” directory below it.

```
volumesDirectory=<Directory>
```

The Knowledge Graphs are in this directory. ‘volumes’ is entered as the default value at this position.

```
backupDirectory=<Directory>
```

Specifies the directory to which the Knowledge Graph backups are written and also read for restoring. Only complete directory names are allowed, no relative paths.

```
networkBufferSize=<Size in bytes>
```

This specifies the size of the buffer that is used for sending/receiving data. The default value is 20480. In some infrastructures, you can specify

networkBufferSize=4096

to achieve a higher throughput.

```
journalMaxSize=<Maximum size of the journal>
```

journalMaxSize=0 can be used to deactivate journaling, which is normally active. The default value is 5242880 (5 MB).

autoSaveTimeInterval=< Wait interval in seconds>

Specifies the maximum wait time in seconds until automatic saving takes place again after the last cluster was saved. The default value is 15.

```
clientTimeout=<Timeout in seconds>
```

Specifies the time in seconds that a connected client may not have sent an Alive message before the mediator regards it as inactive and excludes it.

```
password.flavour=19013329307152292800186471980559137636l
password.hash=11199545182458660705495599802052624171734965791427080638694954247035513239844
```

The mediator password is calculated together with a random flavor to produce a (SHA256) hash value. These two pieces of information then suffice for the mediator to check an authentication request. During authentication on the server, the user name must be specified as “Server.admin.” To determine these values, you can use

```
password.update=new_password
```

Trigger the server to compute a new flavor and suitable hash value and write these to the ini file. The “password.update” entry is removed in this process.

```
password=<String>
```

The obsolete but still supported way of setting the mediator password. This variant must not be used at the same time as the SHA256 hash variant.

Changed

```
skipVolumesCheck=<true|false>
```

Specifies whether the check of the existing volume that is normally performed after starting the mediator is skipped

Logging

For the configuration options for logging, see the logging settings in Chapter 11.1.2 Configuration file.

Working memory

The following three parameters are used to configure the memory allocation and usage. You may specify values either in megabytes or actual bytes, whereby it is assumed that values under 1048576 refer to megabytes.

```
maxMemory=<integer, in MB>
```

Maximum base memory usage permitted. A minimum of 50 MB, the total physical base memory available (under Windows) or 512 MB by default.

```
baseMemory=<integer, in MB>
```

Base memory usage after which efforts to free up memory increase. By default 0.6 * maxMemory. (alias: “growthRegimeUpperBound”)

```
freeMemoryBound=<integer, in MB> [10]
```

If memory that is being used, but is no longer needed, exceeds this limit, it is freed up for use again.

BLOB service configuration

If the mediator is supposed to be started with an integrated BLOB service so that the BLOBs are stored separately from the database on the hard drive, the following setting must be entered in the “mediator.ini” file:

```
startBlobService=true
```

For more information on this, refer to the documentation of the BLOB service (see link below).

4.2.4.3. Security concept of the Mediator

The i-views server is a generic component that can be used for more than i-views. Along with the restrictions due to authentications on the server or in the database, the user can also control which applications may connect to it.

Each application (client and server) receives a pair of RSA keys that is unique for each application delivered. The public key can be obtained by using the information (KB: “Tools” menu, “Information”, then the “Copy RSA key” button) or be called up using the parameter -showBuildID for console applications. The build information exported this way includes the public RSA exponents (rsa.e_1) and RSA module (distributed across several rsa.n_X) and an MD5 checksum for this information (buildID).

Example of build information:

```
[buildID.90A1203EFB957A58C2268AD8FE3CC5A3]
build=Build 00010101
rsa.n_1=93D516DF61395258AA21A91B33E8EE67
rsa.n_2=B07C6FC5023DBB18F2201CF723C8F5DD
rsa.n_3=78941FB7C10D20988FEDFC6BD02CF3B7
rsa.n_4=E4567751843C38F055ED791AA7505278
rsa.n_5=23D94BB9EAB2E23F21DBEAA3DD2D2776
rsa.n_6=CE8B81564645DA85C85E9A78BB6E6B41
rsa.n_7=28A646D4868C38E00AE4810601B1EE9F
rsa.n_8=4FF5C35F873E6ED4F65F0FE8B4B45307
rsa.e_1=010001
```

If you would now like only a specific set of client applications to be able to connect to the server, then you must transfer the respective sections into the mediator.ini in the server. The client transfers its buildID when it connects. When the mediator receives a suitable entry, it authenticates the client. In other cases, it will only connect when there are no entries on build information in its ini file. This, for example, prevents outdated client applications or modified client applications from being able to connect to the mediator.

Conversely, corresponding buildIDs for the mediators can be entered in the respective ini file in the client application in order to prevent a compromised or outdated server from establishing a connection.

This allows an environment to be configured in which only the latest software can be used to access productive data, but also allows access to the server with the test data from a development environment. The user software, in turn, can only access the productive server or the test server.

If neither the server nor the client is configured, then the installation performs the same way as the predecessor version: Each application can connect to any server (as long as the protocol version is correct).

Server version 5.4 or higher requires the server password as a parameter in order to run administrative commands (by means of the Rest interface or by means of the administration using an administration tool). An authentication as the administrator in the volume has been sufficient since version 6.2 for actions that relate to an existing database (backup, download, garbage collection, etc.).

Conversely, it is possible to log into a volume using the server password. Details of this can be found in the Admin tool.

If no password has been configured on the server, then any password can be used to log onto the server. However, logging in on the volume is then not possible.

4.2.4.4. Audit log configuration

In a number of application scenarios, it may be necessary to log all accesses to a Knowledge Graph in an access or audit log. This audit log contains entries for all log-in and log-out processes, write and read access to Knowledge Graph contents, search requests made, printouts, exports, etc.

The log must be activated in the ‘System configuration / Audit log’ category in the Admin tool. The activation or deactivation of the log, in turn, results in a entry in the audit log.

An analysis tool can be opened in the administrator menu of the Knowledge Builder to view and search within the access log.

The log can be configured by creating a file named 'log.ini' in the data directory of the volume. This configuration file is only read when the volume is opened. If the configuration was changed while the volume was opened, then the Mediator has to be restarted.

```
[Default]
; A comma-separated list of log names. The log is configured in the
section with the same name.
applicationLog=audit

[audit]
; Create a compressed backup every 28 days and start with a new empty log
backupInterval=28
; Max size of a JSON file, in MB
maxLogSize=5
; Do not flush the log immediately, for better performance
writeBackImmediately=false
```

4.2.5. Operation

4.2.5.1. Shut down the server

The i-views server can be shut down locally by means of the Ctrl-C abort signal.

In case of installation as a Windows service, the server must be stopped using the service management.

Under UNIX and if operated as a Windows service, the server is shut down properly when the operating system is shut down.

4.2.5.2. Storage and backup of Knowledge Graphs

Directory structure

The basic directory of the i-views server has the following structure:

```
volumes/
   knowledgegraphName/
     knowledgegraphName.cbf
     knowledgegraphName.cdr
     knowledgegraphName.cfl
     knowledgegraphName.lock (if the Knowledge Graph is open)

backup/
  knowledgegraphName/
    <ten-digit number>/
      knowledgegraphName.cbf
```

```
        knowledgegraphName.cdr
        knowledgegraphName.cfl
```

Storage of Knowledge Graphs

Knowledge Graphs are stored in the file system in the “volumes” subdirectory of the basic directory of the i-views server. In this directory, a subdirectory with a corresponding name is created for each Knowledge Graph. A file with the ‘.lock’ file extension indicates that a Knowledge Graph is currently in use.

Backup of Knowledge Graphs

The Knowledge Graph directories must never by copied while the server is running. For this purpose the server has a backup service, which copies a consistent state of the Knowledge Graph to a backup area. This backup area must be backed up at regular intervals (e.g. as part of an overall backup strategy).

The location where backups are created can be specified using the entry

```
backupDirectory=<directory>
```

in the “ **mediator.ini** “ file. Without this information, the “backup” subdirectory of the basic directory is used.

The backup service of the K-Infinity server can be initiated in two ways:

1. With a direct request to the server process (e.g. from the administrator tool)
2. With entries in the `jobs.ini` file in the working directory of the server. For each Knowledge Graph, this file can contain a category [name_of_graph] with the following entries:

Example jobs.ini

```
[volume1]
;Backup of Knowledge Graph “volume1”

;Time the backup starts
backupTime=00:45

;Interval in days - daily in this case
backupInterval=1

;Keep the last 5 backups of this Knowledge Graph
backupsToKeep=5
```

‘backupsToKeep’ specifies the number of backups to be kept. This also includes backups that were created manually. The default value is 3.

When specifying the graph names in square brackets, you can use the wildcards “*” and “?”; the names are not case-sensitive.

4.2.5.3. Garbage Collection

Without Garbage Collection, the Knowledge Graph continues to grow through use. Hence, it makes sense to perform a cleanup (Garbage Collection) from time to time. Like a data backup, you can start the Garbage Collection manually at any time (e.g. with a special administrator tool) or it can be started automatically.

Depending on the size of the Knowledge Graph, the Garbage Collection might require a lot of time and memory. When running the Garbage Collection in large Knowledge Graphs, we recommend starting it without connected clients (e.g. Knowledge Builder and Job-Clients) and without other active processes (e.g. backup).

Automatic Garbage Collection: Structure of the jobs.ini file

Automatic Garbage Collection is configured through an entry in the ' **jobs.ini** ' file, e.g.

```
[volume1]
garbageCollectTime=00:55
garbageCollectInterval=7
```

This entry in jobs.ini ensures that a garbage collection in the Knowledge Graph called “volume1” is performed at “00:55” a.m. every “7” days. The default value for the interval is “1” (i.e. daily); the time of day must be specified.

When specifying the Knowledge Graph names in square brackets, you can use the wildcards “*” and “?”; the names are not case-sensitive.

Manual start of Garbage Collection

Alternatively, garbage collection can also be controlled via the Admin tool or by using the mediator REST api.

4.2.5.4. Operation in Unix

In UNIX the server reacts to the following signals:

```
SIGTERM/SIGHUP
```

Shuts down the server

```
SIGUSR2
```

The server immediately begins to back up all Knowledge Graphs that are specified for backup in the jobs.ini file (see also the section on backups).

4.2.5.5. Operation in Cluster

The mediator can be operated in a cluster. A cluster environment usually mirrors the directories and therefore the Knowledge Graph constantly. If the part of the cluster on which the mediator is running fails, a new mediator that then manages access to the Knowledge Graph is started automatically

If the first mediator fails, it is possible that the mediator no longer has time to make the Knowledge Graph consistent and that the graph thus has an inconsistency and the “lock” file of the old mediator remains in the corresponding directory. To ensure that the new mediator is able to delete the “lock” file, the following parameter must be added to the mediator.ini file.

```
host=NameOfCluster
```

In this case, all mediators with this ini entry can also unlock locked volumes of other mediators that read the same value in the mediator.ini when started. “NameOfCluster" can be selected freely but must comply with the rules that apply to host names (no spaces, colon, or the like)

A consistency check of the volume is executed automatically when the mediator is started. To the extent possible, the Knowledge Graph is made consistent and operation continues as normal.

4.2.5.6. Troubleshooting

If the i-views server was not shut down properly during operation (e.g. computer crash), then the locks remain in opened Knowledge Graphs. When a locked Knowledge Graph is opened, this lock is detected and removed, if possible.

If the mediator detects an inconsistency, then the Knowledge Graph can be checked and inconsistencies can be repaired to the extent possible by calling the mediator in the command line using the parameters -quickRecover / -recover.

If resolving the inconsistencies is, contrary to expectation, not possible, then a backup copy will need to be used.

4.2.5.7. Commands of the BlockFileSystem

The commands behind -bfscommand enable operations on the BlockFileSystem and are designed for support cases. Such a command could look as follows, for example:

```
-bfscommand quickCheck {target volume}
```

The database addressed with {target volume} is subjected to a quick structural analysis. Similarly, deepCheck can be used to perform a complete analysis.

4.3. Bridge

4.3.1. General

The i-views bridge provides interfaces for clients which do not communicate using the i-views internal protocol. The following protocols are supported:

- REST: Provides a REST interface via HTTP or HTTPS
- Message Queue
  - MQTT: Provides a MQTT interface
  - ZeroMQ: Provides a ZeroMQ interface

To specify / configure the desired mode, the following identifiers are used:

Identifier	Legacy identifier	Description
rest	KHTTPRestBridge	REST interface
mqtt	-	MQTT interface
zmq	-	ZMQ interface

4.3.2. Common command line parameters

If the bridge is started without any parameters, the required parameters are read from the ini file bridge.ini and the error messages are written to the file bridge.log.

If there is also an entry in the ini file for a call parameter, then the call parameter has a higher priority.

```
-inifile <File name>, -ini <File name>
```

Name of the ini file that is used instead of the standard ini file. The default is bridge.ini

```
-host <hostname:port>, -hostname <hostname:port>
```

Name of the mediator that acts as the data server. This applies to all activated bridge clients

```
-port <ClientName> <portnumber>
```

Parameter –port should usually be set for every client in the ini file. However, if you want to already do this in the command line, you can specify different clients by specifying the client name in front of the port number. The line above applies to one client; hence, the –port parameter must be repeated until several clients are configured.

Example: Start a REST-Bridge on port 8815:

`bridge -host server01:30000 -port rest 8815`

```
-stop <hostname>
```

4.3.3. Configuration file "bridge.ini"

All of the following entries are found below the ini file section [Default]. The entries for the individual clients follow these. Adding client-specific configuration sections also defines which clients are activated in the bridge to be configured and started.

4.3.3.1. Memory settings

The following three parameters are used to configure the memory allocation and usage. You may specify values either in megabytes or actual bytes, whereby it is assumed that values under 1048576 refer to megabytes.

```
maxMemory=<integer, in MB>
```

Maximum base memory usage permitted. A minimum of 50 MB, the total physical base memory available (under Windows) or 512 MB by default.

```
baseMemory=<integer, in MB>
```

Base memory usage after which efforts to free up memory increase. By default 0.6 * maxMemory (alias: “growthRegimeUpperBound”)

```
freeMemoryBound=<integer, in MB> [10]
```

If memory that is being used, but is no longer needed, exceeds this limit, it is freed up for use again.

```
minAge=<integer> [30]
```

Minimum duration (in seconds) in which a cluster remains in the memory. A cluster is a set of objects that are always loaded together as one (e.g. an individual with all its (meta) properties. Clusters that have not been used for an extended period are unloaded when necessary.

```
unloadInterval=<integer> [10]
```

Minimum duration (in seconds) between two clusters being unloaded

```
unloadSize=<integer> [4000]
```

Minimum number of loaded clusters after which unloading occurs

```
keepSize=<integer> [3500]
```

Number of clusters that are kept when unloading

```
useProxyValueHolder=true/false
```

The option useProxyValueHolder=false can be used to reduce the mediator workload during searches. The client then loads indexes in the base memory instead of querying the mediator by means of RPCs. The drawback of this option is that only read access is permitted.

```
loadIndexes=true/false
```

This option is also used to load indexes to the memory. However, it continues to allow write access. The option can be activated for all clients, including Knowledge Builder.

4.3.4. REST bridge

4.3.4.1. Introduction

The REST-Bridge application enables read and write access to i-views via a RESTful services architecture. The interface is available via HTTP or HTTPS.

The REST bridge runs inside the standard bridge of i-views.

The interface is fully configured by configuration individuals in the Knowledge Graph. The return value of a REST call is any string, usually in a format that the calling client can process easily (e.g. XML or JSON).

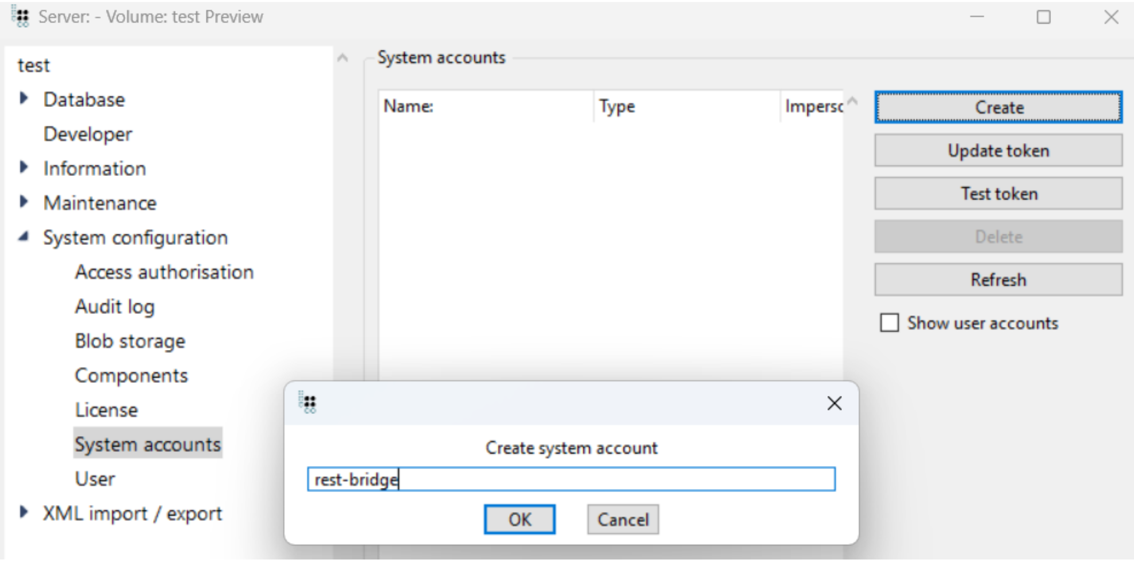
4.3.4.2. Installation

4.3.4.2.1. Prepare volume

1. Creating a system account for the bridge service

To allow a Bridge service to access a Knowledge-Graph database that is managed by a Mediator service, a system account must be created for the Bridge service in the database. This can be done with the Admin-Tool (under System configuration > System accounts) or with the Knowledge-Builder (Settings/Cogwheel > 'System' tab > System accounts). The example shows how a system account can be created using the Admin-Tool:

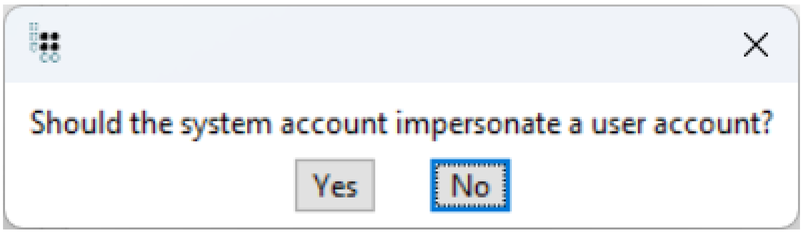
Step 1:



Create system account

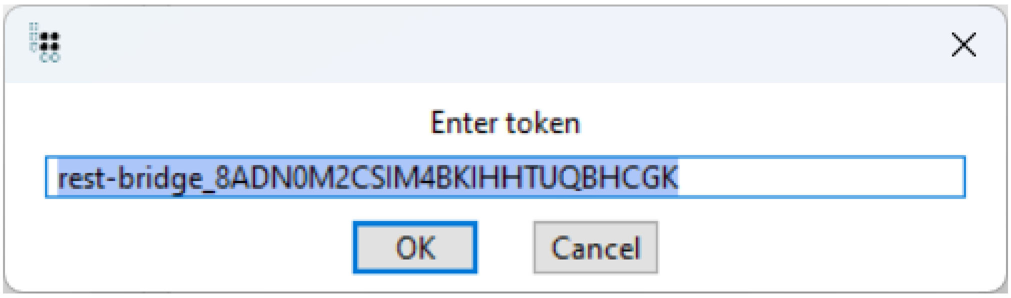
rest-bridge

Step 2:



Should the system account impersonate a user account?

Step 3:



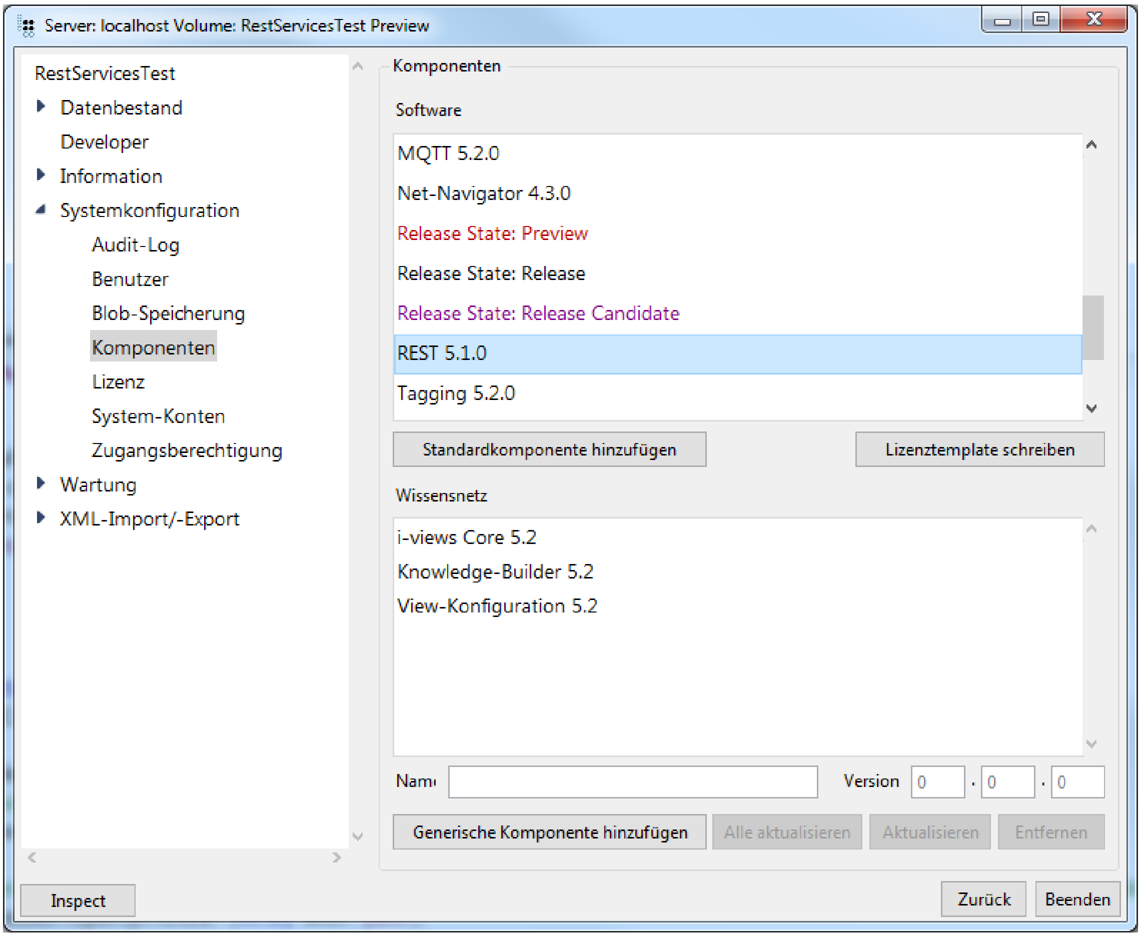
Enter token

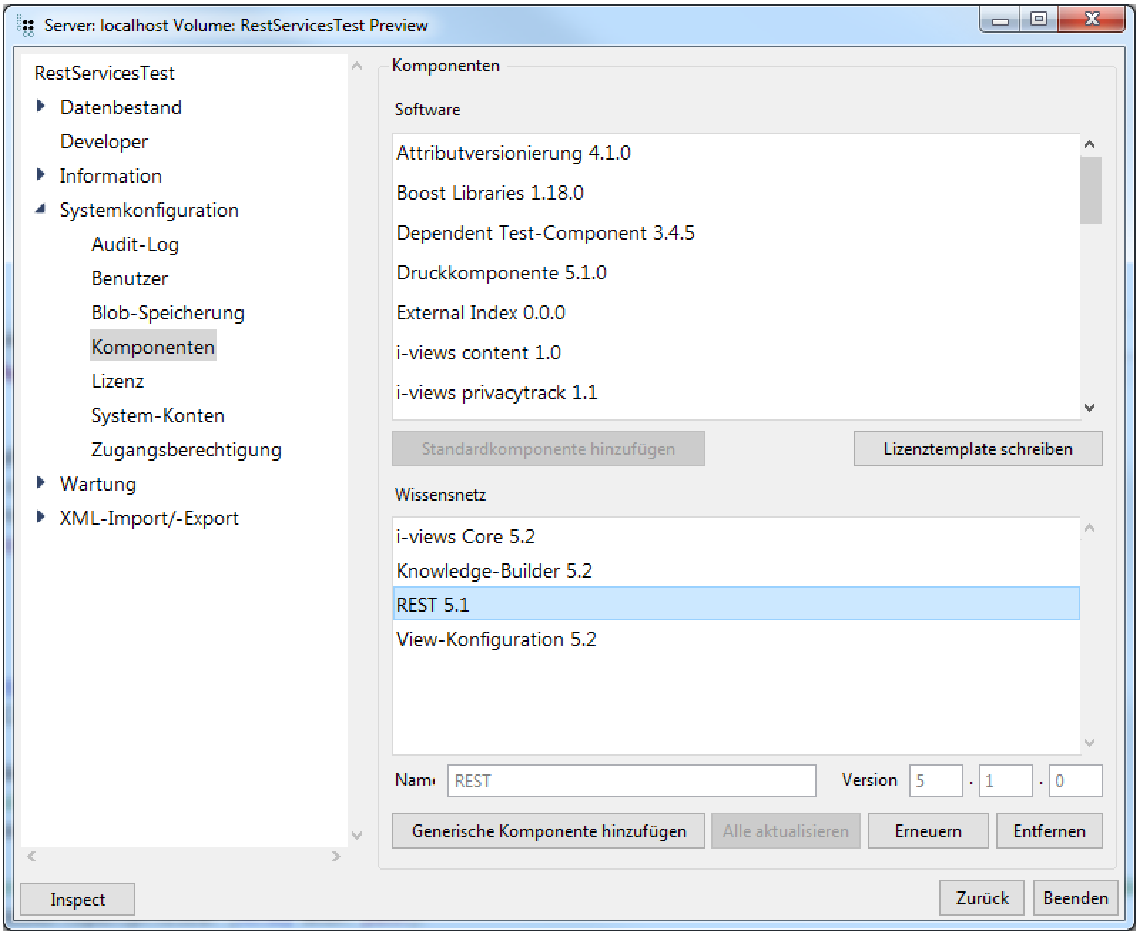
rest-bridge_8ADN0M2CSIM4BKIHHTUQBHCGK

NOTE The login token ('rest-bridge_...') displayed in the last step is required again when configuring the bridge (next chapter). The 'Enter token' window should therefore remain open or the token should be saved in a safe place.

2. Activating the REST component in the Knowledge Graph

By adding the software component “REST” in the Admin tool, the required schema is created in the Knowledge Graph.





Server: localhost Volume: RestServicesTest Preview

Software:

Attributversionierung 4.1.0
Boost Libraries 1.18.0
Dependent Test-Component 3.4.5
Druckkomponente 5.1.0
External Index 0.0.0
i-views content 1.0
i-views privacytrack 1.1

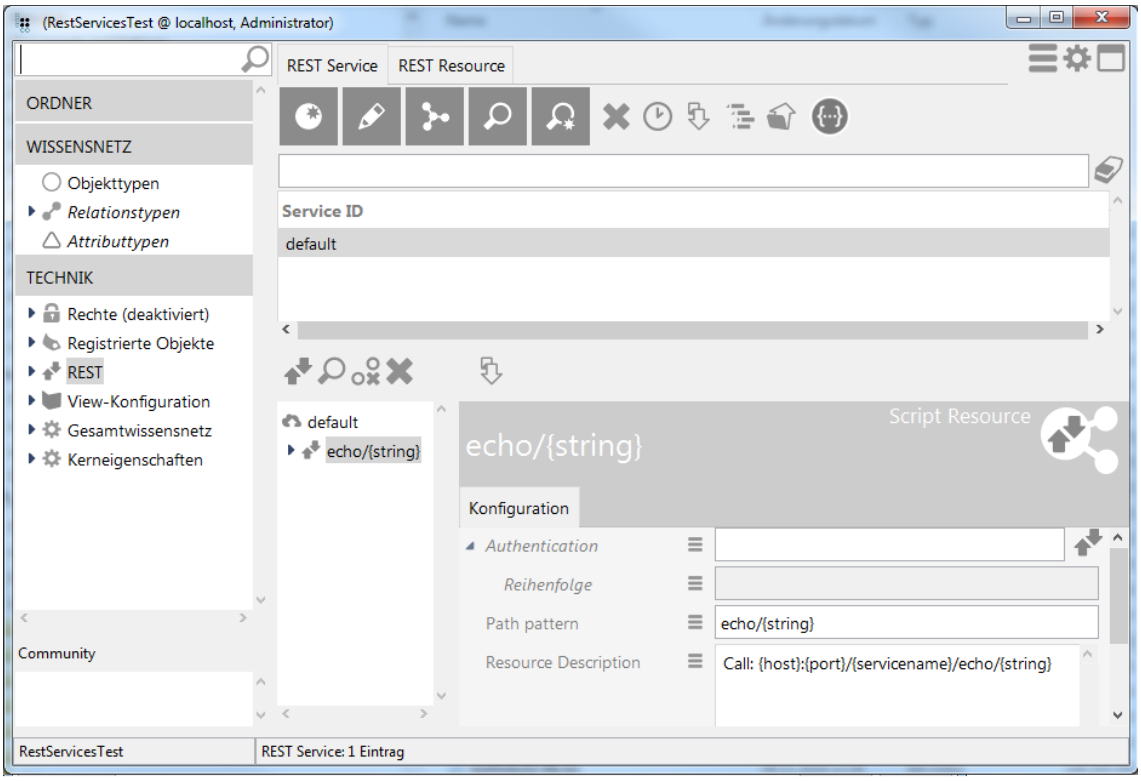
Wissensnetz:

i-views Core 5.2
Knowledge-Builder 5.2
REST 5.1
View-Konfiguration 5.2

Nam REST

Version 5 . 1 . 0

The schema is created as a subgraph of the Knowledge Graph called “REST,” which can only be edited by an administrator in the Technical section:



Service ID: default

echo/{string}

Authentication:

Reihenfolge:

Path pattern: echo/{string}

Resource Description: Call: {host}:{port}/{servicename}/echo/{string}

4.3.4.2.2. Configure bridge

The REST interface is provided by the standard bridge component of i-views, provided the corresponding configuration file contains an entry for the category `rest`:

```ini
[rest]
;Name of the Knowledge Graph
volume=example-graph
;Optional List of network interfaces
;Default value is http://0.0.0.0:8815
interfaces=http://0.0.0.0:8080
;Optional List of REST Service IDs
services=public
```

```
If an interface with the HTTPS protocol was defined, then the file paths
for the certificate and private key must also be specified in the
configuration file. These files must be stored in PEM format and must not
be password protected.

[source,ini]
```

```ini
volume=example-graph
```

```
interfaces=https://0.0.0.0:8443
;Name of the certificate file
certificate=bridge.crt
;Name of the private key
privateKey=bridge.key
```

NOTE The legacy section names `KHTTPRestBridge` or `KHTTPSRestBridge` can be used instead of the section name `rest`. KHTTPRestBridge uses the HTTPS protocol by default, if no interfaces were configured.

In the configuration section of the REST bridge you can also enter the following special configuration options:

Name	Description
realm	Name that is returned to the client as the realm name if authentication is active. Web browsers typically display the realm name as the application name in dialog boxes for authentication to ensure the user knows who is requesting the authentication. Default value: REST

4.3.5. MQTT bridge

4.3.5.1. Introduction

An MQTT bridge connects to an MQTT broker and receives messages from it. The MQTT configuration stored in the knowledge graph determines which MQTT topics are subscribed to (SUBSCRIBE) and how the receipt of these topics is handled. Javascripts are used at this point that allow the configurator to interpret the message, access the knowledge graph for reading and writing and, if necessary, formulate a response in the direction the broker (PUBLISH).

4.3.6. ZMQ-Bridge

4.3.6.1. Introduction

While MQTT always follows the PUBLISH/SUBSCRIBE pattern and the client/server topology, ZeroMQ enables far more flexible topologies and communication patterns. The integration of ZeroMQ enables the composition of runtime components using PUBLISH/SUBSCRIBE, SEND/RECEIVE and PUSH/PULL patterns. A dedicated bridge type, script functions and built-in broker support are available in the Mediator for this purpose.

4.4. Job-Client

4.4.1. General

On the one hand, the Job-Client provides services for other i-views clients to relieve them of time-consuming and data-hungry tasks. On the other hand, it is used as the bridge between i-views clients and external systems.

Normally, the client waits until a job is complete (synchronous operation).

To execute complex searches, generate statistics, batch reconciliations, data formatting, data clearing etc. the client does not have to wait for completion (asynchronous operation). The result is made available by the server and the client is notified. The result can be viewed some time later. Since the result is also made persistent, it is still available if the system is restarted or in case of a fail-over.

4.4.1.1. Operation

In the shared object space provided by the i-views mediator, the jobs of the clients are stored in pools. All i-views Job-Clients are notified of new jobs. If a jobclient is not processing a job currently, it will process the next available job. The oldest job will be processed first.

Once the job has been processed, the result is made available in the shared object space, the requesting client is informed and the result can be retrieved and displayed.

To the client it is transparent which Job-Client is executing its job. For the Job-Client it is transparent, how many parallel Job-Clients are currently active. Hence installation and maintenance of Job-Clients is very easy and flexible for administrators. Job-Clients can be scaled as designed, distributed across different computers and be connected and disconnected dynamically. External clustering or other orchestration is not necessary.

4.4.2. Configuration of the Jobclient

4.4.2.1. Configuration file "jobclient.ini"

The Job-Client is configured directly in the ini file. If this file is not specified by the call parameter “-inifile” when the Job-Client is started, "jobclient.ini” is used as the configuration file.

4.4.2.1.1. General parameters

The following parameters can be configured:

Parameter	Description	Syntax
host	Name / IP address and port of the server.	`host=<host name:port number>`

Parameter	Description	Syntax
volume	The name of the Knowledge Graph for working on.	`volume=<volume name>`

jobPools	Specifies which jobs the Job-Client is supposed to process. The names of the job pools to be started are to be specified in comma-separated form.
	The job pools should be specified by category (e.g. “index”). Alternatively, it is possible to specify indiviual pool names (e.g. ”KScriptJob”).
	See Job pool types for the complete list of possible types.
		`jobPools=<job name1> [,<job name2>, ...]`
		Example:
		`jobPools=script, query`

cacheDir	The description of the location at which the cache for the Job-Client is stored.	`cacheDir=<directory>`
maxCacheSize	Target size of the cache	`maxCacheSize=<size in MB>`
shutDownTimeout	Wait period for termination of the active job when shutting down the Job-Client. The jobs are terminated at the end of this period. The default value is 10 seconds.	`shutDownTimeout=<seconds>`
enableLowSpaceHandler	This option activates the LowSpaceHandler. This should always be activated for large Knowledge Graphs.	`enableLowSpaceHandler=true/false`
useProxyValueHolder	This option can be used to control whether the Job-Client executes index access via RPC (true) or loads indexes to memory (false). This option should be deactivated to ease the mediator load. In doing so, however, you should ensure that the Job-Client has enough memory. If the Job-Client has been configured for write jobs, this option has no effect as index access is always executed via RPC then. If you set the value to false, a message is output in the log on start-up.	`useProxyValueHolder=true/false`

Parameter	Description	Syntax
loadIndexes	Indexes are also always loaded to memory. In contrast to the useProxyValueHolder option, it continues to allow write access. The option can be activated for all clients, including Knowledge Builder.	`loadIndexes=true/false`
name	This name is used to identify the Job-Client in the Admin tool in the overview list of all Job-Clients.	`name=<Job-Client name>`

scheduledJobs	A comma-separated list of jobs that are to be scheduled.	`scheduledJobs=<Job name 1>`
		`[, <Job name 2>, ...]`

4.4.2.1.2. Memory settings

The following three parameters are used to configure the memory allocation and usage. You may specify values either in megabytes or actual bytes, whereby it is assumed that values under 1048576 refer to megabytes.

Parameter	Description	Syntax
maxMemory	Maximum base memory usage permitted. A minimum of 50 MB, the total physical base memory available (under Windows) or 512 MB by default.	`maxMemory=<integer, in MB>`
baseMemory	Base memory usage after which efforts to free up memory increase. By default 0.6 * maxMemory. (alias: “growthRegimeUpperBound”)	`baseMemory=<integer, in MB>`
freeMemoryBound	If memory that is being used, but is no longer needed, exceeds this limit, it is freed up for use again.	`freeMemoryBound=<integer, in MB> [10]`
minAge	Minimum duration (in seconds) in which a cluster remains in the memory. A cluster is a set of objects that are always loaded together as one (e.g. an individual with all its (meta) properties. Clusters that have not been used for an extended period are unloaded when necessary.	`minAge=<Integer> [30]`

Parameter	Description	Syntax
unloadInterval	Minimum duration (in seconds) between two clusters being unloaded	`unloadInterval=<Integer> [10]`
unloadSize	Minimum number of loaded clusters after which unloading occurs	`unloadSize=<Integer> [4000]`
keepSize	Number of clusters that are kept when unloading.	`keepSize=<Integer> [3500]`

4.4.2.1.3. Lucene server configuration

Lucene is integrated via a Job-Client whose jobclient.ini file has to be configured accordingly. Below is an exemplary configuration:

```
[lucene]
directory=lucene-index
port=5100
pageSize=100
; Wildcards at the start of a word are prohibited by default as they are
very slow
; Allow in this configuration
allowLeadingWildcards=true

[JNI]
classPath=lucene-6.4.1\\core\\lucene-core-6.4.1.jar;lucene-6.4.1\\
analysis\\common\\lucene-analyzers-common-6.4.1.jar;lucene-6.4.1\\
analysis\\queries\\lucene-queries-6.4.1.jar;lucene-6.4.1\\analysis
\\queries\\lucene-queries-6.4.1.jar
```

The directory *lucene-6.4.1* contains the Lucene binary files. The index is stored in the directory *lucene-index* .

4.4.2.1.4. Scheduled jobs

Jobs can be scheduled by the Job-Client. The Job-Client then creates jobs at the specified time.

NOTE

Usually, only one Job-Client should schedule jobs. Scheduled jobs are treated as regular jobs and can also be performed by other Job-Clients.

To configure individual jobs in the configuration file, a new section has to be created for each one.

These are each started with the name of the job in a pair of square brackets. This is followed by the respective parameters of the job.

The job names must be listed at the parameter `scheduledJobs`.

Example:

```
scheduledJobs=Job-Name1,Job-Name2

[Job-Name1]
<Parameter>=<value>
...

[Job-Name2]
...
```

Scheduled job parameters

Parameter	Description	Syntax
jobPool	The pool where the job should be inserted. Defines the type of job that is scheduled.	`jobPool=<Job-Pool-Name>`

time	Time at which the job should be executed for the first time.	`time=<Time>`
		Example:
		`time=22:15`

interval	Specifies how frequently the job should be executed. (d=days, h= hours, m=minutes, s=seconds)	`interval=<Exact time>`
command	For KExternalCommandJob only: Name of an external batch file that should be executed by the job.	`command=<File name.cmd>`
scriptName	For KScriptJob only: Registration key of an internal script that should be executed by the job.	`command=<Script resource>`

Parameter	Description	Syntax
unique	If a job of this type is already queued, then no additional job is scheduled	`unique=true/false`
user	Name of a user account, under which the job should be executed	`user=<User name>`
arguments	For KExternalCommandJob only: Arguments that are transferred when the script is called.	`arguments=<Argument1 [Argument2 ...]>`

4.4.2.1.5. Job pool types

The following types of job pools are available:

Category	Pool	Description
blob	KBlobGCJob	Performs a BLOB garbage collection
blob	KSwitchBlobStoreJob	Transfers BLOBs between stores
index	KAddAllToIndexJob	Add all properties of a property type to the index
index	KLightweightIndexJob	Updates the index of a property
index	KRemoveIndexJob	Removes all properties of a property type from the index
index	KSyncIndexJob	Updates all properties of a property type
index	KExternalIndexUpdateJob	Updates an external index (e.g. IAS, Elasticsearch)
lucene, luceneAdmin	KLuceneAdminJob	Manages Lucene indexes
lucene, luceneQuery	KLuceneQueryJob	Performs Lucene queries
script	KScriptJob	Performs a script of the Knowledge Graph
script	KScriptTriggerJob	Performs a trigger script
print	KPrintJob	Creates a document BLOB by applying a print configuration to semantic elements
query	KQueryJob	Performs a query and stores the result
-	KExternalCommandJob	Performs an external command (executable, shell script)
-	KExtractBlobTextJob	Extracts the text from a BLOB

Index jobs

The indexing jobs should be performed by a single Job-Client only.

KExternalCommandJob

Using the **KExternalCommandJobs** it is possible to activate executable programs that are concerned with processing or changing files, or that are simply to be called. No configuration is necessary in the ini file of the Job-Client. The job is also inserted by a script call.

The main element of the script call is the element ***ExternalCommandJob*** . The attribute *Execution* allows the user to set whether the job should be executed locally without Job-Client (value: *local*) or with Job-Client (value: *remote*). The default value is *remote* .

Note about remote execution:

Access to local programs is checked by calling a batch file. Before the Job-Client takes a KExternalCommandJob to execute, it checks whether it can execute this job. This is the case if the batch file, which is specified in the element *command* , exists in the current directory of the Job-Client. If the currently pending job is not accepted for processing by any Job-Client, then the job queue is blocked for the user who inserted the job. This job must be deleted manually.

The necessary first subelement in the script:

- **Command** : specifies which batch file should be called

```
<Command>convert.bat</Command>
```

The name of the batch file is specified in the command element. The directory and the actual program to be executed are specified in the batch file. ***Important:*** *The batch file must be located on the same level as the program (e.g. Job-Client or KB). Directory specifications in the command element are ignored.*

The other subelements are worked through from top to bottom. If the order of parameters plays a role in the external program, this should be factored in.

Script elements that form the parameters for the call:

- **OptionString** : can be used multiple times. Parameters of the external program to be called are specified as strings. The parameters entries must be specified in full.

```
<OptionString>-size 100x100</OptionString>
```

- **OptionPath** : the path expression specified is evaluated and built up in the command call as a string

```
<OptionPath path="./topic()/concept()/@$size$"/>
```

Script elements that are concerned with the handling of attributes

- **SourceBlob** : This specifies the blob attribute that is used as a data source

```
<SourceBlob><Path path=“$image$"/></SourceBlob>
<SourceBlob path=“$image$"/>
```

- **ResultAttribute** : This specifies the parameter for the generation of a new, or the change of an existing, blob attribute with the content of the file, or the file itself, that is the result of the program called externally. Attribute values: **name** : Name or internal name of the attribute **Topic** to be created: Target individual of the attribute **modifyExisting** to be created: change (*true*) or create new (*false*, default value) **filename** : File name of the blob attribute to be created

```
<ResultAttribute
      name="$image2$"
      topic="./topic()"
      modifyExisting="true"
      filename="convert_ +./valueString()">
    <Path path="$image2$"/>
</ResultAttribute>
```

Example 01:

Script:

```
<Script>
    <ExternalCommandJob execution="local">
        <Command>convert.bat</Command>
        <OptionString>-size 100x100</OptionString>
        <SourceBlob>
            <Path path="."/>
        </SourceBlob>
        <OptionString>-geometry +5+10</OptionString>
        <SourceBlob>
            <Path path="."/>
        </SourceBlob>
        <OptionString>-geometry +35+30</OptionString>
        <OptionString>-composite</OptionString>
        <ResultAttribute
```

```
            name="$image2$"
            topic="./topic()"
            modifyExisting="true"
            filename="convert_ +./valueString()"/>
    </ExternalCommandJob>
</Script>
```

Content of the batch file under Windows:

```
"C:\\Program Files\\ImageMagick-6.2.6-Q16\\convert.exe" %*
exit /B %ERRORLEVEL%
```

Content of the batch file under Linux:

```
#!/bin/bash
convert $*
```

Example 02:

Script:

```
<Script>
    <ExternalCommandJob execution="local">
        <Command>convert2.bat</Command>
        <SourceBlob path="."/>
        <SourceBlob path="."/>
        <ResultAttribute
            name="$image3$"
            topic="./topic()"
            modifyExisting="true"
            filename="convert2_" + ./valueString()/>
    </ExternalCommandJob>
</Script>
```

Content of the batch file under Windows:

```
"C:\\Program Files\\ImageMagick-6.2.6-Q16\\convert" -size 100x100 %1
-geometry +5+10 %2 -geometry +35+30 -composite %3
exit /B %ERRORLEVEL%
```

Content of the batch file under Linux:

```
#!/bin/bash

convert -size 100x100 $1 -geometry +5+10 $2 -geometry +35+30 -composite $3
```

NOTE The two examples deliver the same file as the result. The exit command is used in the Windows batch files to return the exit code of “convert” to the call.

Here is another example of an advanced conversion script that can be called using the parameters “Source file”, “Image width” and “Target file” and that only minimizes wider images to the specified width. The script also writes a log file for the conversion, whereby error messages from Image Magick are also written to the log file:

```
set MONTH_YEAR=%DATE:~-8%
echo Converting %1 to %3 (width: %2) >> convert%MONTH_YEAR%.log
convert.exe %1 -resize "%~2>" %3 2>> convert%MONTH_YEAR%.log
echo Conversion finished with exit code %ERRORLEVEL% >>
convert%MONTH_YEAR%.log
exit /B %ERRORLEVEL%
```

And here is the version for Linux (Bash):

```
#!/bin/bash
FULLDATE=`date +%c`
MONTH_YEAR=`date +%m.%Y`
LOGFILE="convert.$MONTH_YEAR.log"
echo "$FULLDATE: Converting $1 to $3 (width: $2)">>$LOGFILE
convert "$1" -resize "$2>" "$3" 2>>$LOGFILE
EXITCODE="$?"
echo $FULLDATE: Conversion finished with exit code $EXITCODE>>$LOGFILE
exit $EXITCODE
```

4.4.2.2. Performance optimizations

4.4.2.2.1. Pre-load

When starting up, Job-Clients can pre-load selectable structures if configured accordingly. This operation increases the amount of memory that the Job-Client requires, but it also enables the Job-Client to run more quickly.

The entry **keepClusterIDs** must be specified in the ini file of the Job-Client. Possible values for this entry are:

- **index** - In the settings for pluggable indexers, there is an option to set the check-mark for *Job-Client to load index into base memory* . For activated indexers, a part of their index structure is loaded.

  NOTE Only used when *useProxyValueHolder* is set to *false* . Otherwise, the Job-Client will send RPCs instead of loading the index into memory.

- **protoOfSizes** - The number of individuals for each concept is already determined at the start.
- **accessRights** - The root object of the rights system is loaded into the memory.

To improve performance, it also helps to activate the Knowledge Graph cache for the Job-Client.

Example of entries in the ini file:

```
[Default]
...
useProxyValueHolder=false
keepClusterIDs=index,protoOfSizes,accessRights
cacheDir=jobcache
maxCacheSize=1000
...
```

4.5. Batch tool

The batch tool allows to perform administrative commands and data import/export commands from the command line. The desired command needs to be passed as a command-line parameter, followed by command-specific parameters. It is also possible to perform a series of commands.

4.5.1. Common command line parameters

All commands share some common parameters:

```
-host
```

URL or host name and port of the server

```
-volume
```

Name of the volume

```
-user
```

Deprecated. Name of a user account that should be activated when running the command-line. Use an authentication token instead (see configuration file options), to avoid leaking user/password information to other processes.

```
-password
```

Password of the user

4.5.2. Configuration file options

```
host
```

URL or host name and port of the server

```
volume
```

Name of the volume

```
authentication
```

System account authentication token

4.5.3. Commands

4.5.3.1. Importing or exporting mapped data

These commands allow to import or to export data based on a mapping defined in the volume.

The following example exports data mapped by the mapping registered as `example.export` to the file `data.csv`:

```
batchtool -volume example -exportMapping example.export -file data.csv
-errorLogFile export-errors.log
```

4.5.3.1.1. Command line parameter

Either

```
-exportMapping
```

Export mapped data

or

```
-importMapping
```

Import mapped data

4.5.3.1.2. Additional command line parameters

```
-encoding {name}
```

Name of a character encoding, e.g. utf-8. Determines the connection encoding for database mappings, and the text encoding for text files (e.g. CSV files).

```
-errorLogFile {logfile}
```

Filename of the error log

```
-mapping {ID}
```

Registered ID of the data mappings

```
-triggers {true/false}
```

Enable/disable triggers during import

4.5.3.1.3. Command line parameters for file mapping only

```
-caption {true/false}
```

True if the table file contains/should contain captions

```
-file {filename}
```

Possible values: Name of the file to be imported/exported

```
-separator {separator string}
```

Cell separator string

4.5.3.1.4. Command line parameters for database mappings only

```
-binding {name value}
```

Name-value pair for database bindings. Only used by database mappings that contain a named binding in the query

```
-dbEnvironment {string}
```

Database connection string

```
-dbHostname {string}
```

Database host. For MySQL only.

```
-dbPassword {string}
```

Database password

```
-dbUsername {string}
```

Database user name

4.5.3.2. Importing or exporting RDF files

These commands allow to import or export RDF(S) or OWL files.

4.5.3.2.1. Command line parameter

Either

```
-exportRDF {filename}
```

Export an RDFS or OWL file

or

```
-importRDF {filename}
```

Import an RDF, RDFS or OWL file

NOTE The export always uses RDFS or OWL, it is not possible to export „pure“ RDF.

4.5.3.2.2. Additional command line parameters

```
-parameter {name} {value}
```

Set a parameter controlling the RDF import/export

```
-query {ID}
```

Set the query that returns the elements that should be exported

```
-queryParameter {name} {value}
```

Set a query parameter

4.5.3.2.3. Export parameters

The following export parameters can be set with -parameter {name} {value}

```
abbreviateURIs
```

Abbreviate URIs using rdf:ID and xml:base

```
baseURI
```

Base URI

```
blobHash
```

True if additional comments should be exported

```
exportFrameIDs
```

Export object frame IDs

```
exportLabels
```

Export name as rdfs:label

```
exportMeta
```

True if meta properties should be exported. They will be exported as reifications.

```
exportPropertyIDs
```

Export IDs of properties

```
exportReferencedTopics
```

True if external relation targets should be exported as stubs

```
ignoreStoredIdentifier
```

If true, the RDF locator will always be generated. If false, the rdf:about/rdf ID attribute will be used if present

```
qualifier
```

XML qualifier bound to the base URI

```
schemaNameSpace
```

Default schema XML namespace

```
schemaOnly
```

True if only types should be exported

```
updatePersistentIdentifier
```

True if the generated values for rdf:ID / rdf:about should be materialized as attribute values

```
useFrameURIs
```

True if URIs constructed from the object ID should be used to identify objects

```
useKRDF
```

True if KRDF properties (e.g. krdf:internalName) should be exported

```
useOWL
```

True if OWL vocabulary should be used

4.5.3.2.4. Import parameters

```
activateTriggers
```

True if triggers should be enabled

```
allowExtensiveRestructurings
```

Perform schema changes even if they impact a lot of objects

```
avoidDuplicateProperties
```

True if duplicate properties should be skipped

```
baseURI
```

Base URI

```
changeCompatibleInstanceTypes
```

Change instance types even if the current type is compatible with the defined type

```
enableCreateSchema
```

True if schema may be created / modified

```
enforceInverseRelationConcepts
```

true: Create inverse relation types if not defined.

false: Relations without inverse will be symmetric.

```
importCommentsAsAttributes
```

Import rdfs:comment as attribute (must be defined in the schema)

```
importReferencedResources
```

Import referenced external RDF resources

```
importValuesAsAttributes
```

Import rdf:value as attribute (must be defined in the schema)

4.5.4. Running scripts

This command allows to run arbitrary scripts written in JavaScript.

4.5.4.1. Command line parameter

```
-script {registered script ID}
```

Runs a registered script

```
-scriptfile {filename}
```

Runs a script read from the file

4.5.4.2. Additional command line parameters

```
-argument {argument name} {argument value}
```

Set the global script variable named {argument name} with value {argument value}

```
-encoding {encoding name}
```

Set the output encoding to {encoding name}

```
-errorLogFile {filename}
```

Filename of the error log

```
-output {file name}
```

Print all output to {file name}

```
-stdout
```

Print all output to stdout and errors to stderr. Log entries will only be written to the log file.

4.5.5. Importing or exporting schema

These commands allow to import or export the schema the Knowledge Graph. This includes

- Types
- View configurations
- REST service definitions
- Registered queries, scripts, mappings, folders
- Triggers
- Access rights
- Index configurations and filters

4.5.5.1. Common command line parameters

Either

```
-exportSchema {schema file}
```

Export the schema as a file

or

```
-importSchema {schema file}
```

Import the schema from a file

4.5.5.2. Additional command line parameters

```
-filter {filter file}
```

Specifies a filter file (see next chapter)

4.5.5.3. Export filter

The filter file defines which registered objects are exported. Each line defines a positive (starts with "+") or negative (starts with "-") flag, a category, and a pattern matched against the registered ID. The category and the ID pattern may contain the wildcards "*" for a partial string or "#" for a single character. A line beginning with ";" is ignored.

Objects without ID are only matched if the ID pattern is "*".

Objects that do not match any filter are exported

Schema sub nets are matched against the value of "RDF:about" of the top type (e.g. "http://www.intelligent-views.de/kinfinity/component/rest/4.0/type/rest-ConfigTopConcept").

Possible categories:

accessRights, dataConnections, editorDetection, indexers, indexFilter, ldap, license, mappings, organizingFolder, printConfigurations, queries, schema, scripts, topicCollection, triggers, unknown.

Example:

```
+ queries custom.*
- * *
```

This will export all queries whose ID match the pattern "custom.*".

4.5.6. Importing licenses

This command allows to import a license file. This might be necessary for bootstrapping, because other commands will fail if a volume has an invalid license.

Command line parameters

```
-importLicense {license file}
```

Imports the license from the file.

4.5.7. Upgrading components

This command allows to upgrade the software / model components of the volume. It is also possible to reinitialize the schema.

4.5.7.1. Command line parameter

```
-upgradeComponents {optional component names}
```

Upgrades the specified components, or all components if none are specified

Possible component names:

iviewsProducts, kintelligence, knowledgeBuilder, mqtt, netNavigator, printing, rest, translator, viewConfigMapper

4.5.7.2. Additional command line parameters

```
-updateSchema
```

Reinitializes the schema of components

4.5.8. Executing a series of commands

This command allows to perform a series of commands. This is more efficient than running the batch tool for each command separately, because data that has already been loaded by a previous command does not need to be reloaded.

4.5.8.1. Command line parameter

```
-batchFile {file}
```

Perform all commands listed in the batch file. Must be UTF-8 encoded. The batch file must not contain command line parameters (host, volume etc.)

Example:

```
batchtool -volume example -batchFile commands.txt
```

With commands.txt containing the following lines:

```
-exportMapping example.export1 -file data1.csv -errorLogFile export1-
errors.log
-exportMapping example.export2 -file data2.csv -errorLogFile export2-
errors.log
```

This will perform two exports.

4.5.9. Example: Importing per batch tool

For importing data using the batch tool, access to the import data is needed as well as a network connection to the mediator. If the batch tool is going to be executed on a different server as the mediator for import is locate on, the following entries of the batchtool.ini file must be adjusted:

- Address/URL of the server ("host=")
- Port number of the server ("port=")
- Name of the volume ("volume=")
- Token for authentication, to be created using the admin tool ("authentication=")

For a one-time call of the batch tool by means of a control file (e. g. "import.data"), the command line will be as follows:

```
batchtool -batchFile import.data
```

If the batch tool is not situated in the same directory as the working directory of the command line (or of the scheduled task), at least the ini file needs to be in the same directory as the working directory or the ini file needs to be passed on in forms of a parameter:

```
D:\\PATH\\batchtool\\batchtool -ini D:\\PATH\\batchtool\\batchtool.ini
-batchFile import.data
```

"PATH" relates to the machine from which the command line call is invoked.

The control file can be created easily as follows:

```
-importMapping MAPPING1 -file EXCELFILE1 -errorLogFile MAPPING1-errors.log
-importMapping MAPPING2 -file EXCELFILE2 -errorLogFile MAPPING2-errors.log
-importMapping MAPPING3 -file EXCELFILE3 -errorLogFile MAPPING3-errors.log
```

- `MAPPINGx` = registered name of the import mapping in i-views
- `EXCELFILEx` = file name, where necessary incl. file path

For the regular execution of the import, a "scheduled task" containing the procedure call can be configured on the operating system (known as "Task Scheduler" within Windows or as "cron" within Linux).

If the import needs to be done as a subsequent process after a previous export from another system, the calls can be encapsulated within a batch file (*.bat, *.cmd or *.ps1) to ensure that the import only starts when the export has been processed successfully.

4.6. Blob service

4.6.1. Introduction

The blob service is used to store the data of large files outside the Knowledge Graph but links to the file attributes in which these file contents are supposed to be stored. This has several advantages:

- It has the effect that the Knowledge Graph only receives the semantic information that is based on files and remains easy to backup and transfer.
- Storage locations of the Knowledge Graph and file contents can be configured differently.
- Several blob services can be connected to one Knowledge Graph, so that one storage location can be provided for each attribute definition.

The following chapter explains how to set up and operate blob services.

4.6.2. Configuration

To specify under which network address (host and port) the blob service is supposed to be reachable, the “interfaces” option must be entered in the file “blobservice.ini.” There are two options here:

1. The blob service is supposed to be reachable only from the computer on which the blob service is installed.
2. The blob service is supposed to be reachable also by other computers via the network.

Here is an a configuration example for variant 1, whereby the blob service port (30000) can be selected freely.

```
interfaces=http://localhost:30000
```

To configure variant 2, you need to enter the IP address of the network adapter via which the blob service is supposed to be reachable from the network instead of “localhost.” If you want the blob service to be reachable via all network adapters that are active on the computer, you have to enter “0.0.0.0” as the IP address. Example:

```
interfaces=http://0.0.0.0:30000
```

If the blob service is address via the network, communication should be encrypted. Encrypted communication using HTTPS can also be configured in the “interfaces” option by replacing “http://" with “https://.” Example:

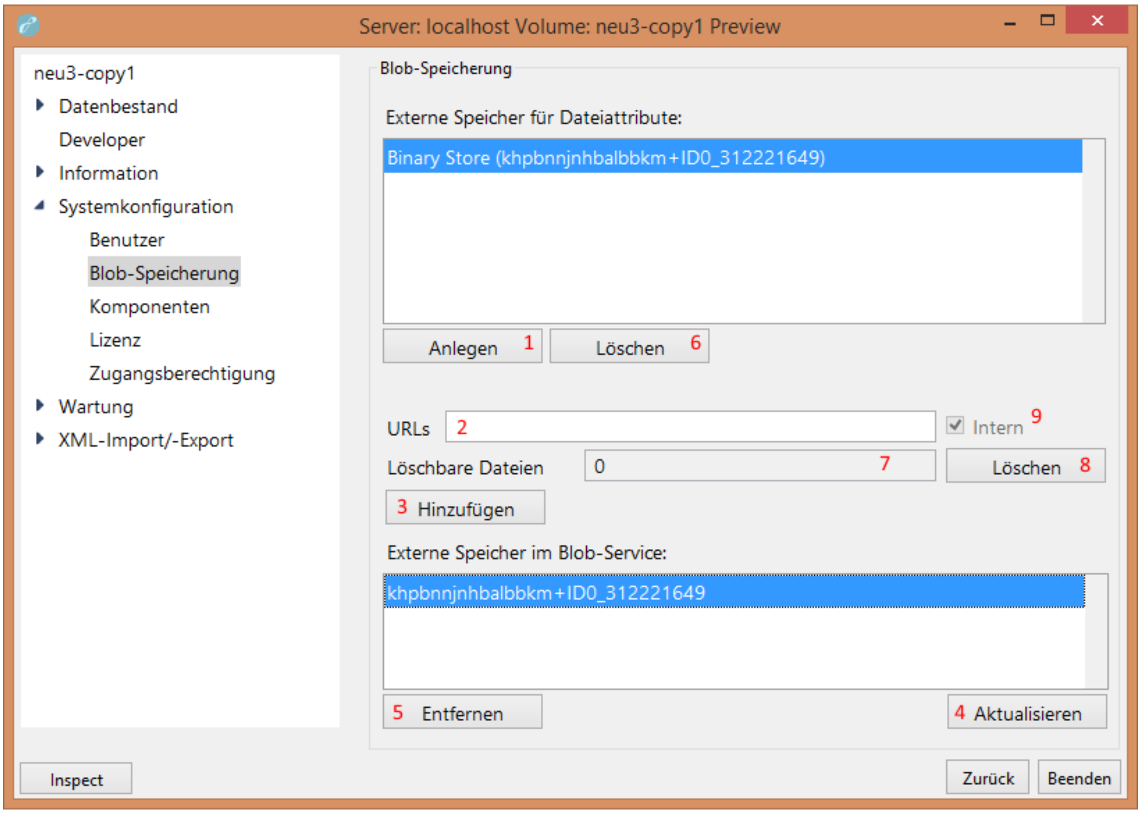
```
interfaces=https://0.0.0.0:30000
```

In relation to encrypted communication, see also the next chapter called SSL certificates.

To ensure operation, the DLL of the SQLite framework "sqlite3.dll" must also be available in the working directory. Without this DLL, the internally required administration structure cannot be generated and maintained.

Following that, the blob service can be started to make it available immediately.

To link the blob service with a blob store in the Knowledge Graph, the Admin tool offers the required tools under “System configuration - Blob storage:”


Server: localhost Volume: neu3-copy1 Preview

Externe Speicher für Dateiattribute: Binary Store (khpbnnjnhbalbbkm+ID0_312221649)

URLs: 2

☑ Intern 9

Löschbare Dateien: 0

Externe Speicher im Blob-Service: khpbnnjnhbalbbkm+ID0_312221649

Clicking on “Create” (1) creates a new logical store. After that, enter the URL (2) of the blob service specified in the ini file and then click on “Add” (3). The newly created blob store for external storage of file attributes is then linked to the blob service, which you can check by clicking on “Update” (4) in the lower display area.

You can also specify a comma-separated list of alternative URLs in the “URLs” area (2). For alternative URLs, i-views prefers a connection via a loop-back device where possible.

The “Deletable files” area (7) displays the number of files that are no longer required from the Knowledge Graph perspective. Use “Delete" (8) to de-reference them in the blob service and remove them if appropriate.

The indicator “Internal” (9) shows that this is a store that is integrated into a mediator. Internal stores are automatically transferred with the volume during a volume transfer (upload, download, copy, backup, recover).

If you want to remove the link between a blob store and a blob service, select the desired blob store in the list “External stores in the blob service” and click “Remove” (5). Following that, you can select the blob store in the top section “External storage for file attributes” and then click “Delete” (6) to remove it completely. Alternatively, you can specify a new URL to link the blob store to another blob service.

NOTE By removing a blob store’s link to a blob service, all files stored therein are lost.

4.6.3. SSL certificates

To configure the HTTPS connection, the certificate and the private key must be stored.

The certificate must be stored under **certificates/server.crt** .

The private key must be stored under **private/server.key** . Make sure that server.key is available as an RSA key, i.e. the first line of the file must be

```
 -----BEGIN RSA PRIVATE KEY-----
```

If the key is in a different format, it has to be converted, e.g. with OpenSSL:

```
openssl rsa -in input.key -out private/server.key -outform PEM
```

4.7. Login with OAuth 2.0

Users of the Knowledge-Builder and the Admin-Tool can be authorized with the OAuth 2.0 framework. This requires an external authorization server that provides the tokens to access the Knowledge Graph. It is also necessary to install a bridge service that provides a REST interface for handling user data

4.7.1. Limitations

- The only supported grant type is the authorization code flow
- Server administration tasks (e.g. upload Knowledge Graphs) cannot be authorized with OAuth
- When creating Knowledge Graphs, the initial graph administrator account is created with username and password. The administrator can be authorized either with OAuth or username and password.

4.7.2. Authorization flow

When a user opens a Knowledge Graph that has been configured to use OAuth 2, a web browser will be opened and directed to the login URI. There, the user performs the necessary steps to login, e.g. confirm the login or enter credentials.

Afterwards, a request containg a grant is sent to a redirect URI which must point to the endpoint

```
/oauth/redirect
```

of the Knowledge graph server. This endpoint then requests a token from the authorization server. The token is validated using the public keys (JWKS). The token then allows access to the Knowledge Graph.

Once a user has been authorized, then the server sends a POST request containing the data to an endpoint of the REST interface provided by the bridge service. This allows to perform additional, customizable steps to create user objects in the Knowledge Graph.

4.7.3. Configuration

The OAuth framework can either be configured for the entire server, which affects all Knowledge Graphs, or for a single Knowledge Graph.

4.7.3.1. Configuring the authorization server

The authorization server must be prepared for an authorization code flow. This usually requires registering a new application and generating a client ID and secret. It is also necessary to register a redirect URI that points to the OAuth redirect endpoint of the server.

4.7.3.2. Configuring OAuth for the entire server

The OAuth configuration for all Knowledge Graphs of a server is part of the server configuration file (mediator.ini). It can (but must not) be put in a separate file by using an include directive.

The server must provide an HTTP or HTTPS interface. The redirect endpoint is available at the path

```
/oauth/redirect
```

File **mediator.ini**

```
interfaces=http://0.0.0.0:30080,https://0.0.0.0:30443
$(include:oauth2.ini)
```

File **oauth2.ini**

```
[auth-oauth2]
clientID=12345-abcd-6789-1234-123456789
clientSecret=qwertzuioplkjhgfdsayxcvbnm
configURI=https://login.microsoftonline.com/c4cc84aa-3413-47c6-bd6e-
c38019596fbf/v2.0/.well-known/openid-configuration
redirectURI=https://exampleserver:30443/oauth/redirect
loginFinishedURI=https://exampleserver:8815/oauth/userAccount
userNameKey=preferred_username
createAccounts=true
```

The configuration is contained in the category [auth-oauth2]. The values are

Configuration	Required	Description
clientID	yes	OAuth Client ID
clientSecret	yes	OAuth Client secret
configURI	yes (*)	URI of an OpenID connect configuration endpoint.This URI is used by the mediator to get information about the openid configuration.

Configuration	Required	Description
redirectURI	yes	Public redirect endpoint of the mediator server. This is usually http(s)://SERVERNAME:SERVER_PORT/oauth/redirect (SERVERNAME and SERVER_PORT must be replaced with actual values).This is invoked from the authentication service and adresses the mediator. Pay attention to possible sub-pathing, e.g. if the mediator is reachable at `https://server/mediator`
loginFinishedURI	no	URI of the graphs REST endpoint for handling the user data. This is usually http(s)://SERVERNAME:REST_PORT/oauth/login-finished (SERVERNAME and REST_PORT must be replaced with actual values).It is possible to use the macro {volume}, which is replace by the name of the accessed Knowledge Graph.This URI is invoked by the mediator and targets a REST endpoint of the graph the user is logging on to. This is only required, if - after login - data from the authentication token should be used to fill a user topic.
userNameKey	no	Name of the token property that contains the user name, e.g. preferred_username (which is also the default value)
createAccounts	no	Boolean value that defines if new accounts should be created in the Knowledge Graph for authorized users.If false, then users can only login if an account has already been created for them. The default value is false.
scopes	no	Comma separated list of additional requested scopes. The following scopes will always be requested and do not need to be configued: openid, email, profile, offline_access

If no OpenID connect configuration endpoint (configURI) is given, then the following settings must be configured:

Configuration	Description
jwksEndpoint	URI of an endpoint that returns the public keys (JSON Web Key Sets), e.g. `https://login.microsoftonline.com/c4cc84aa-3413-47c6-bd6e-c38019596fbf/discovery/v2.0/keys`

Configuration	Description
tokenEndpoint	URI of the token endpoint, e.g. `https://login.microsoftonline.com/c4cc84aa-3413-47c6-bd6e-c38019596fbf/oauth2/v2.0/token`
authorizationEndpoint	URI of the authorization endpoint, e.g. `https://login.microsoftonline.com/c4cc84aa-3413-47c6-bd6e-c38019596fbf/oauth2/v2.0/authorize`

4.7.3.2.1. Configuring HTML pages

By default a login attempt will finally redirect the browser to a HTML page showing a success oder failure message. Without further configuration standard plain text messages will be presented. If required, this can be customized by defining templates in the **auth-oauth2** section:

```
[auth-oauth2]
htmlTemplates=oauth2-html-authorized-de,oauth2-html-authorized-en,oauth2-html-unauthorized-de,oauth2-html-unauthorized-en
; Addition configuration ommitted

[oauth2-html-authorized-de]
state=authorized
languages=de
file=html\authorized-de.html

[oauth2-html-authorized-en]
state=authorized
languages=*
file=html\authorized-en.html

[oauth2-html-unauthorized-de]
state=unauthorized
file=html\unauthorized-de.html

[oauth2-html-unauthorized-en]
state=unauthorized
file=html\unauthorized-en.html
```

htmlTemplates is a comma-separated list of unique section names. Each section can have the following entries:

Section	Description
state	must be `authorized` or `unauthorized`. Default value: `authorized`
languages	comma-separated list of languages, * can be used to match any language.Default value: *
file	HTML file. The file must self-contained, no additional files are included.
redirect	URI that is opened via a redirect (307) instead of showing a built-in HTML file. Only used when **file** is not specified.

4.7.3.3. Configuring OAuth for a Knowledge Graph

OAuth can be configured for a single Knowledge Graph in the Knowledge-Builder. This can be done by administrators only. To configure OAuth, open the settings, and select **OAuth** on the **System** tab. The settings are equal to the server configuration described above.

If a configuration is present, it has precedence over the configuration of the server.

4.7.3.4. Configuring the OAuth REST endpoint

The server only creates basic user accounts when registering new users. All additional steps must be performed by a REST endpoint provided by a bridge service. To simplyfiy the setup, add the software component OAuth login in the Admin-Tool. This will create a basic setup:

- A mapping **rest.oauth.userMapping** that maps the user data to objects of the Knowledge Graph
- A script **rest.oauth.postUserAccount** that uses the mapping to create user objects.
- An endpoint **/oauth/userAccount** which calls the script
- An OAuth configuration skeleton. This is incomplete and must be adjusted to the server setup (e.g. set host names)

4.8. Installation of i-views

4.8.1. General information

For the operation of i-views components, it is not relevant if processes run in physical hardware (servers, workstation, notebook, …) or in virtual environments (VMWare, VirtualBox, docker, …).

The operation of i-views does not require any special access rights or account permissions of the underlying operating system. Processes can run as any operating system account considered reasonable. Nonetheless, running the processes as system accounts is possible; i-views will only work in configured data areas and its product parts only communicate via TCP/IP.

Without further configuration, i-views processes read and write data and log-files in or below the working directory assigned at process startup. That means that the account the process is running as requires write access to the areas the process is running in.

There are also no special requirements for install locations. i-views can be installed in any directory matching your operating guidelines. Operating any process in a non-locally attached directory (e.g. a network share) is not recommended. Next to reduced I/O performance this can lead to problems in specific situations.

All i-views product parts are run as a combination of virtual machine (VM) and so called image. The image files contain the code of the respective product part and are independent of the operating system, therefore these are interchangeable between operating systems. The distributed virtual machines perform the abstraction of the underlying operating system. For Windows, i-views offers executables which are a pre-packaged combination of VM and image.

All product parts communicate via TCP/IP, so in principle they can be distributed across networked machines. The distribution is not described in this document, please refer to the individual tools documentation for that. If the selected TCP/IP ports are to be used from outside the local machine, the respective operating systems firewall mechanisms might need to be configured to allow such communication.

The configuration of product parts depending on the local installation is performed in ini-files. Some products parts even require such an ini-file and will issue error messages when started without one. When on the processes command line no explicit ini-file is given, the product part searches for a default ini-file in its working directory. The file name of this ini-file depends of the image’s role, e.g. a mediator image will search for a mediator.ini file. The filename of the image file is not relevant for its role, so renaming an image file will not make it look for a differently named configuration file.

All product parts can be started via command line or from file manager in foreground. Closing a window or pressing ctrl-c/cmd-c will end the processes. Also, the processes react to standard OS signals. All headless processes can also be started as services, for Windows they support this by built-in command line parameters.

Component / Feature	Description	Required for ...
Core	Core environment	required for Core
Core (de-)compression	Tools for compression	required for Core
Core with graphical interface	i-views-tools with graphical interface	required for Core with UI
Mediator checksums	Accelerates the calculation of checksums for graph contents	optional for mediator
TLS connections	Use of operating system based TLS features (5.5 an up)	required for the feature
Image scaling	Accelerated scaling of pixel-based images	optional for Core with UI
Graph editor alpha channel	Prettier display of icons with transparent areas in the graph editor	optional for Core with UI
JNI	Java Native Interface to connect to Java based software (e.g. Lucene)	required for the feature
Encryption	Cryptographically protected connections (e.g. HTTPS)	required for the feature
Enhanced text display	Display texts with better anti-aliasing and kerning in UI tools	optional for Core with UI
MySQL-connection	Connect to MySQL databases	required for the feature
ODBC-connection	Connect to ODBC data sources	required for the feature
Oracle-connection	Connect to Oracle databases	required for the feature
SQLite-connection	Connection to SQLite databases	required for the feature and BlobService/Mediator
Extended regular expressions	Extension of the available regular expressions	required for the feature
Coordinate transformation	Transformation of geographic geometry attributes from one spatial reference to another; calculation of geodesic distances	required for the feature
Spatial relations for geometries	Calculation of spatial relations of the Dimensionally Extended 9-Intersection Model with increased performance	optional for the feature

i-views is delivered as 64-bit software. When integrating external software libraries, it has to be ensured that they are also available in 64-bit architecture. It is not possible to integrate 32-bit libraries.

4.8.2. Operating Systems

i-views supports several operating systems. This chapter describes the installation on supported operating systems. At the end of the chapter, some hints to further possible operating systems are given; nevertheless those are not officially supported.

If required all data managed in i-views can be migrated to a different system environment (hardware, operating system, …) without any required conversion. When distributing the product parts across different machines, these also can run different operating systems (e.g. a mix of Windows and Linux). Also, for clients accessing the server components there are no restrictions on the used clients operating system with regards to the servers operating system.

4.8.2.1. Microsoft Windows

For Windows the product parts usually are delivered as EXE-files. These combine internally a virtual machine and image. If required, instead of EXE-files also separated virtual machines and images can be used.

All product parts, that support the operation as a service, can be installed as a service by issuing the following command line in an administrative shell:

```
PRODUCTPART.exe -installAsService SERVICENAME [SERVICE1 ...]
[SERVICEPARAMETER ...]
```

This invocation only creates the service configuration in the Windows environment and stops after that. If a service should depend on the state different services SERVICE1 to SERVICEn, this can be added directly on the command line. Parameters, which should be passed to the process started by the Windows service manager can be added at the end of the command line.

In the Windows service configuration (usually invoked by services.msc) the service then shows up in the list of services with the name SERVICENAME. Here it also can also be modified, e.g. to change the account, the start-type or description of the service. Usual command line tools (like sc.exe) can also be used to modify the service.

To deregister the service, you can use Windows tools like `sc delete SERVICENAME` or:

```
PRODUCTPART.exe -deinstallService SERVICENAME
```

Alternatively, the installation as service can also be performed by Windows service control utilities.

Feature	Library	To install
Core	-	Integrated into Windows
Core (de-)compression	-	Integrated into Windows
Core with graphical interface	-	Integrated into Windows
Mediator-checksums	coastbinary	coastbinary.dll: supplied, next to mediator.exe
TLS connections	tlsPlugin	tlsPlugin.dll: supplied, next to all executables
Image scaling	GDIPlus	Integrated into Windows
Graph-editor alpha-channel	Cairo	libcairo-2.dll from gtk.org
JNI	Java Virtual Machine	libjvm.dll is installed with a Java Runtime (jre) or Java Development Kit (jdk), e.g. from java.com
Encryption	Windows-native OpenSSL	bcrypt.dll: Integrated into Windows, libeay32.dll: via openssl.org (sometimes required because of bcrypt problems)
Enhanced text display	-	Integrated into Windows
MySQL-connection	MySQL client libraries	libmysql.dll: via mysql.com
ODBC-connection	ODBC	Integrated into Windows or installable as Windows feature
Oracle-connection	Windows Advanced ServicesOracle client libraries	advapi32.dll: Windows Feature Advanced Services, usually available. oci.dll: from oracle.com in the version matching the target database servers version
SQLite-connection	SQLite	sqlite3.dll: from sqlite.org
Extended regular expressions	Boost-Regex	boost_regex.dll, msvcp140.dll, vcruntime140.dll: from boost.org
Coordinate transformation	PROJ	proj_9.dll: either from proj.org or by using a distribution like OSGeo4W
Spatial relations for geometries	GEOS	geos.dll, geos_c.dll: either from libgeos.org or by using a distribution like OSGeo4W

When running on Windows, sometimes the libraries of the Microsoft Visual C++ Runtime need to be installed. This depends on version and type of the OS installation. If they are not present in the current setup, they need to be installed.

The recommended way to do this, is to use the redistributables offered from Microsofts official page of the Visual C++ Redistributable. Please select the correct runtime release, and the language matching the operating systems language. Using this approach not only Microsoft's update mechanisms provide new security fixes and patches for the libraries but also there is only a single copy of the libraries present on your system.

Alternatively you can copy the required msvcpXXX.dll and/or vcruntimeXXX.dll files to the same directory as the executables requiring them.

4.8.2.2. macOS

Some components are already included with macOS, other can be added from the standard code source, via MacPorts or from the respective project websites.

Feature	OS part	Filename
Core	LibC	libc.dylib, included in macOS as libSystem.dylib
Core (de-)compression	LibZ	Included in macOS
Core with graphical interface	X11 environment	Included in macOS
Mediator-checksums	coastbinary	not available for macOS, checksums are calculated without library support
TLS connections	tlsPlugin	tlsPlugins.dylib: supplied with the client app
Image scaling	FreeImage	Available at the FreeImage Project
Graph-editor alpha-channel	Cairo	libcairo.2.dylib: via Cairo Project or MacPorts
JNI	Java Virtual Machine	
Encryption	OpenSSL	libcrypto.1.0.dylib
Enhanced text display	-	Included in macOS
MySQL-connection	MySQL	libmysqlclient.dylib: from the MySQL website
ODBC-connection	diverse, z.B. iODBC	libiodbc.dylib: e.g. from iodbc.org

Feature	OS part	Filename
Oracle-connection	Oracle	libclntsh.dylib: from Oracle's website in the version matching the target database server’s version
SQLite-connection	SQLite	libsqlite3.dylib: via sqlite.org
Extended regular expressions	Boost-Regex	libboost_regex.dylib: via boost.org
Coordinate transformation	PROJ	libproj.dylib: via proj.org
Spatial relations for geometries	GEOS	libgeos.dylib, libgeos_c.dylib: via libgeos.org

4.8.2.3. Linux / Unix

i-views runs on several Unix operating systems, most installations are miscellaneous Linux variants.

Currently there are no repositories available to the public, that provide installation packages for the different package management tools. The software usually will be delivered as an archive, which can be installed in the location preferred by organisational policies or just personal taste. Most installations are placed in the /opt/-tree to signify the non-packaged nature of the installation.

The virtual machines require a small set of libraries to be present for operation. On most systems, these libraries will be installed out-of-the-box; on some mininalistic installs they might be missing. Optional i-views features often make use of external libraries, which are only required when a feature is actively used. If required libraries are not found, it will be noted in the tools log file. For specific libraries there might be log entries indicating the library as "not loadable" while the library file is present. This usually happens, when a second library the noted one depends on is missing on the system. Usually a `ldd LIBRARYFILE` can help identifying the missing prerequisite library.

4.8.2.3.1. Debian/Ubuntu (x86/amd64)

Debian/Ubuntu uses `apt` as package manager. It is recommended to use standard package sources wherever possible. Packages are installed by issuing `apt install PACKAGENAME`. If you prefer extended package managers like aptitude or synaptic, feel free to use those.

Feature	OS part	package name / library
Core	GLibC	libc / libc.so
Core (de-)compression	Utilities for compression	zlib1g / libz.so
Core with graphical interface	X11 environment	libx11-6 / libX11.so
Mediator checksums	coastbinary.so	— / coastbinary.so: supplied, next to the mediator.im

Feature	OS part	package name / library
TLS connections	tlsPlugin	— / tlsPlugin.so: supplied for all executables
Image scaling	FreeImage	libfreeimage3 / libfreeimage.so
Graph-editor alpha-channel	Cairo	libcairo2 / libcairo.so
JNI	Java Virtual Machine	— / libjvm.so: is installed with a Java Runtime (jre) or Java Development Kit (jdk), e.g. from java.com
Encryption	OpenSSL	libssl1.X.Y / libssl.so
Enhanced text display	XFT	libxft2 / libXft.so
MySQL-connection	MySQL or MariaDB	libmysqlclient20 or libmariadb3 / libmysqlclient.so
ODBC-connection	iODBC	libiodbc2 / libiodbc.so
Oracle-connection	Oracle	— / libclntsh.so: from oracle.com in the version matching the target database server’s version
SQLite-connection	SQLite	libsqlite3-0 / libsqlite3.so
Extended regular expressions	Boost-Regex	libboost-regex1.XY.Z / libboost_regex.so
Coordinate transformation	PROJ	libproj25 / libproj.so.25
Spatial relations for geometries	GEOS	libgeos-c1v5 (Debian), libgeos-c1t64 (Ubuntu) / libgeos.so, libgeos_c.so

Running Debian or one of its descendants you can find packages providing a specific feature using `apt-cache search 'NAME'`.

If a package really provides the desired files, you can verify with `apt-file list 'PACKAGE'` (from the package apt-file). The package-name for a file can be determined with "apt-file search 'FILE'".

As an alternative you can use the search functionality of your distributions web-site e.g. https://www.debian.org/distrib/packages#search_contents or http://packages.ubuntu.com/.

4.8.2.3.2. RedHat/Fedora/CentOS (x86/amd64)

RedHat-based distributions use `yum` or `dnf` as package manager. It is recommended to use the standard package sources wherever possible, since they deliver security fixes and patches. Package are installed with `yum install PACKAGENAME` or `dnf install PACKAGENAME`.

Please keep in mind, that depending on the distribution some packages might not be available from the base/core repositories but only from extended community driven repositories.

Feature	OS part	package name / library
Core	GLibC	glibc / libc.so
Core (de-)compression	LibZ	zlib / libz.so
Core with graphical interface	X11 environment	libX11 / libX11.so
Mediator checksums	coastbinary.so	— / coastbinary.so: supplied, next to the mediator.im
TLS connections	tlsPlugin	— / tlsPlugin.so: supplied next to all images
Image scaling	FreeImage	freeimage / libfreeimage.so
Graph-Editor alpha-channel	Cairo	cairo / libcairo.so
JNI	Java Virtual Machine	— / libjvm.so: is installed with a Java Runtime (jre) or a Java Development Kit (jdk), e.g. from java.com
Encryption	OpenSSL	openssl-libs / libssl.so
Enhanced text display	XFT	libXft / libXft.so
MySQL-connection	MariaDB	mysql-libs or mariadb-libs / libmysqlclient.so
ODBC-connection	iODBC	libiodbc / libiodbc.so
Oracle-connection	Oracle	— / libclntsh.so: from oracle.com in the version matching the target database server’s version
SQLite-connection	SQLite	sqlite / libsqlite3.so
Extended regular expressions	Boost-Regex	boost-regex / libboost_regex.so
Coordinate transformation	PROJ	proj / libproj.so.25
Spatial relations for geometries	GEOS	geos / libgeos.so, libgeos_c.so

In RPM-based distributions you can find packages containing a specific feature with the search function of yum/dnf `yum search STRING`. To list all files in a package you can issue `yum repoquery -l PACKAGENAME`. To find which package provides a specific file use `yum provides FILENAME`. All commands allow to use glob-patterns.

As an online alternative you can use services like https://rpmfind.net/linux/rpm2html/search.php or https://pkgs.org/ to search for files and packages for your distribution.

4.8.2.3.3. SUSE Linux Enterprise (SLES)

SUSE distributions are RPM based but use `zypper` as package manager. It is recommended to use the standard package sources wherever possible. Packages are installed using `zypper install PACKAGENAME`.

Feature	OS part	package name / library
Core	GLibC	glibc / libc.so
Core (de-)compression	LibZ	libz1 / libz.so
Core with graphical interface	X11 environment	libX11-6 / libX11.so
Mediator checksums	coastbinary.so	— / coastbinary.so: supplied, next to the mediator.im
Image scaling	FreeImage	libfreeimage3 / libfreeimage.so
Graph-editor alpha-channel	Cairo	libcairo2 / libcairo.so
JNI	Java Virtual Machine	— / libjvm.so: is installed with a Java Runtime (jre) or a Java Development Kits (jdk), e.g. from java.com
Encryption	OpenSSL	libopenssl1_0_0 / libssl.so
Enhanced text display	XFT	libXft2 / libXft.so
MySQL-connection	MariaDB	libmysqlclient18 / libmiysqlclient.so
ODBC-connection	iODBC	? / libiodbc.so
Oracle-connection	Oracle	— / libclntsh.so: from Oracle in the version matching the target database server’s version
SQLite-connection	SQLite	libsqlite3-0 / libsqlite3.so
Extended regular expressions	Boost-Regex	libboost_regex1.XY.Z / libboost_regex.so
Coordinate transformation	PROJ	libproj25 / libproj.so.25
Spatial relations for geometries	GEOS	libgeos_c1 / libgeos.so, libgeos_c.so

4.8.3. Service configuration

Running of i-views product parts depends on the distributions "init"-system. Here we currently offer sample files for systemd-based systems.

Other init-systems like e.g. upstart (CentOS 6.5) or LaunchDaemon (Mac OS X) are currently not offered here, as we do only have incomplete or not-generic files for those. Since i-views does not depend on any special operating system support, running i-views on those platforms should be easily possible.

For container based environments the build system creates standard container images, which can be easily used. Example configurations provided are:

- docker-compose
- kubernetes

Other container runtime variants like docker, podman or openshift can be derived from these templates.

4.8.3.1. systemd

For regular operation in distributions using systemd as init-system, the following configuration files can be used as a blueprint:

`/etc/systemd/system/iviews-mediator.service`:

```
[Unit]
Description=iviews mediator
After=network.target nss-lookup.target

[Service]
User=iviews
WorkingDirectory=/opt/iviews/mediator
ExecStart=/opt/iviews/vm/vwlinux86 -noherald
-=/opt/iviews/mediator/mediator.im
Restart=on-failure

[Install]
WantedBy=default.target
```

`/etc/systemd/system/iviews-bridge-rest.service`:

```
[Unit]
Description=iviews bridge rest
After=network.target nss-lookup.target iviews-mediator.service
Wants=iviews-mediator.service

[Service]
User=iviews
WorkingDirectory=/opt/iviews/bridge
ExecStart=/opt/iviews/vm/vwlinux86 -noherald
-=/opt/iviews/bridge/bridge.im -ini bridge-rest.ini
```

```
Restart=on-failure

[Install]
WantedBy=default.target
```

`/etc/systemd/system/iviews-jobclient.service`:

```
[Unit]
Description=iviews jobclient
After=network.target nss-lookup.target iviews-mediator.service
Wants=iviews-mediator.service

[Service]
User=iviews
WorkingDirectory=/opt/iviews/jobclient
ExecStart=/opt/iviews/vm/vwlinux86 -noherald
-=/opt/iviews/jobclient/jobclient.im
Restart=on-failure

[Install]
WantedBy=default.target
```

If services are to be combined to a single installation, e.g. if you want to run multiple projects in parallel on the same machine and want to be able to control whole projects with single command lines, it might be useful to create a project-specific systemd-target:

`/etc/systemd/system/iviews.target`:

```
[Unit]
Description=iviews
After=network.target nss-lookup.target
Wants=iviews-mediator.service iviews-bridge-rest.service iviews-
jobclient.service

[Install]
WantedBy=default.target
```

In this case you can remove the [Install] sections from the individual service files and add an entry to each [Unit] section:

```
PartOf=iviews.target
```

When starting or stopping a iviews.target unit, all parts will be started or stopped. You also can control individual services without affecting the iviews.target.

Names of the services and paths need to be adapted to your environment. The above service definitions can be used as a blueprint for services with further i-views product parts. After modification of these files, systemd needs to be informed about the changed services via systemctl.

If required, dependencies of services can be expressed differently. Please see the systemd homepage or your distributions man-pages.

Short cheat-sheet for controlling systemd services:

Tasks	Command
Register a service	`systemctl enable SERVICE`
Deregister a service	`systemctl disable SERVICE`
Start a service	`systemctl start SERVICE`
Stop a service	`systemctl stop SERVICE`
Status of a service	`systemctl status SERVICE`
Edit a service file	`systemctl edit --full SERVICE`
After manual changes to a service file	`systemctl daemon-reload`

4.8.3.2. docker-compose

The presented example provides a configuration file for docker-compose starting a mediator bridge and jobclient for a project. There is also an example env-file containing the secrets required in the installation. The compose-file can safely be stored in a versioning system like git. The env-file in the repository should not contain actual secrets.

In the files values enclosed in angular brackets (like "<knowledge-graph-name>") are placeholders to be filled with your projects values. Identical placeholders indicate the same value to be inserted. These are intentionally not part of the env-file in this configuration variant to enable only having a single env-file and being able to keep all relevant configuration information in a repository.

Please note that the installations does not use any INI-files, but relies on IV_ prefixed variables. The i-views tools assist in converting from ini to environment variables by using the commandline parameter -iniToEnv.

Please refer to the appendix for the configuration files.

4.8.3.3. kubernetes

Running i-views in kubernetes environments is possible. For the mediator a reliable storage driver is required, since it only supports filesystem storage.

In the files values in angular brackets (like "<knowledge-graph-name>") are placeholders to be filled with your projects values. Identical placeholder names in the files indicate the same value to be inserted. The single-file configuration below can be separated into multiple yaml files if you prefer to manage the setup in one file per resource.

Please note:

- Nearly all values mentioned here are depending on the project, the kubernetes environment and policies relevant to the environment.
- Providing storage via NFS shares is strongly discouraged due to its history of failures.
- Also avoid transferring process with permanent storage (especially the mediator) to different nodes (see PodDisruptionBudget, maxUnavailable: 0).
- Not all k8s object types are included to make services accessible, e.g. missing is an Ingress config, since these depend heavily on the infrastructure running the k8s cluster.

Please refer to the appendix for the configuration file.

4.8.4. Typical Requirements

4.8.4.1. Proxy and Load Balancing

4.8.4.1.1. Load-Balancing of multiple REST services

i-views REST-services are provided by the bridge component. They are built to scale by running multiple processes in parallel. You can start an i-views bridge in a load-balancer mode, where it starts and monitors a set of worker bridges. For that the configuration file of the bridge needs to look like the following:

```
[KLoadBalancer]
hostname=localhost
port=5001
vm=/opt/iviews/vm/vwlinux86
directory=.
image=bridge.im
configNames=REST
```

```
autoRestart=true

[REST]
bridgeClientClassName=KWeb.KHTTPRestBridge
inifile=bridge_rest.ini
bridgeLogfile=bridge_rest_<id>.log
ports=5002-5005
```

This configuration will start four REST-bridges on ports 5002 to 5005 and use the port 5001 as controlling port the workers and the controller use for communication.

The load-balancing bridge itself will not perform the HTTP load-balancing. For this, i-views installations usually make use of load-balancing features of nginx, apache with mod_proxy or haproxy. Since the started bridges will use the standard HTTP protocol, any standard compliant load-balancer can be used.

Nginx

In nginx, you can use proxy_pass directives to forward incoming requests on a specific subpath to a set of backend bridges. To achive this, extend the nginx configuration with the following lines:

```
upstream iviews-rest-bridges {
   server 127.0.0.1:5002;
   server 127.0.0.1:5003;
   server 127.0.0.1:5004;
   server 127.0.0.1:5005;
}

server {
   ...

   location ^ ~ /myRestPath/ {
      proxy_pass http://iviews-rest-bridges/;
   }
}
```

Apache HTTPD

The Apache HTTPD project offers several proxy modules, which can be activated in typical installations. To balance to the above scaled bridges, an apache configuration would look like this:

```
ProxyRequests off
<Proxy balancer://iviews>
   BalancerMember  http://127.0.0.1:5002  disablereuse=On
   BalancerMember  http://127.0.0.1:5003  disablereuse=On
   BalancerMember  http://127.0.0.1:5004  disablereuse=On
   BalancerMember  http://127.0.0.1:5005  disablereuse=On
</Proxy>
ProxyPass         /myRestPath/  balancer://iviews/  lbmethod=bybusyness
ProxyPassReverse  /myRestPath/  balancer://iviews/  lbmethod=bybusyness
```

(this uses the modules mod_proxy, mod_proxy_balancer, mod_lbmethod_bybusyness)

haproxy

haproxy is a pure proxy service and does not offer web-server features like nginx or apache. The haproxy service usually is configured in

/etc/haproxy/haproxy.cfg:

```
frontend         main  *:5000
default_backend  rest_bridge

backend rest_bridge
  balance  first
  server   rest_bridge1 127.0.0.1:5002 check maxconn 1 weight 100
  server   rest_bridge2 127.0.0.1:5003 check maxconn 1 weight 99
  server   rest_bridge3 127.0.0.1:5004 check maxconn 1 weight 98
  server   rest_bridge4 127.0.0.1:5005 check maxconn 1 weight 97
```

4.8.4.1.2. Microsoft Windows Server

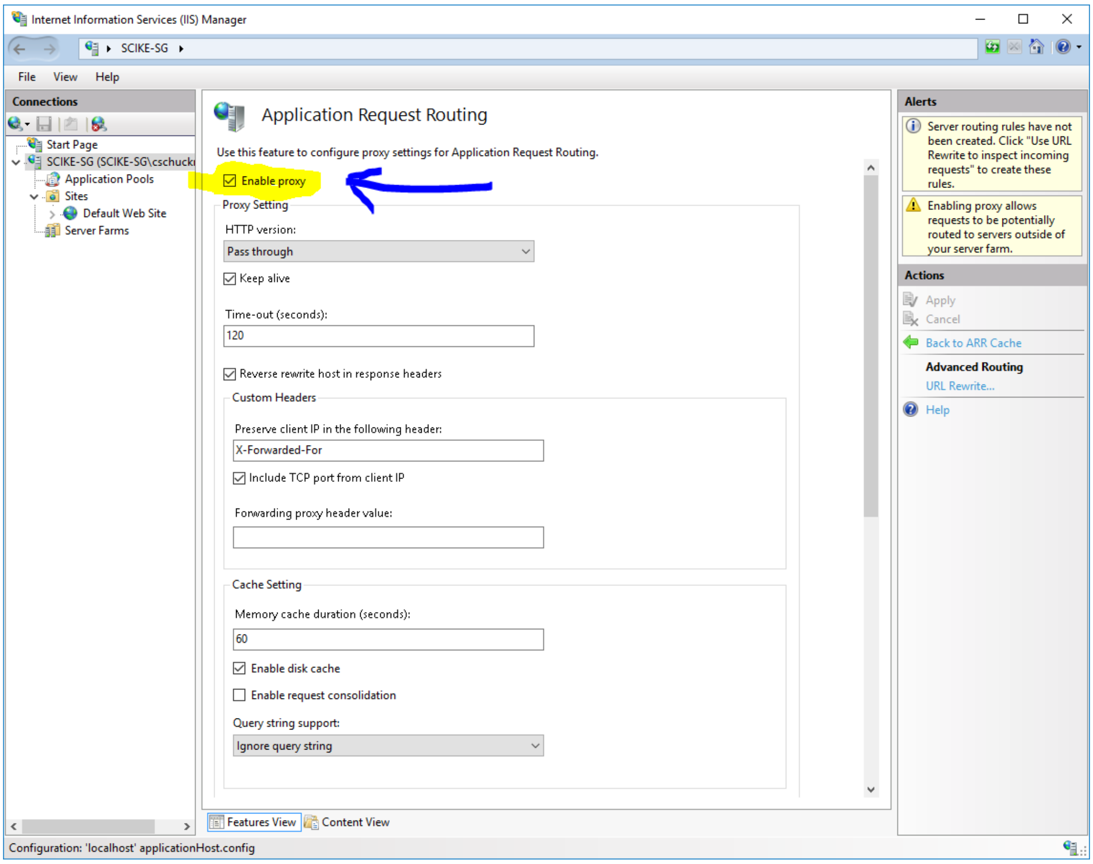
Using the Internet information Services (IIS) i-views services can be offered on a standard single port. This has the advantages to not require special ports to be opened in firewalls, that IIS can add the HTTPS wrapping using the integrated certificate management and that redirection features can be administrated centrally in IIS.

The following modules need to be installed:

- ApplicationRequestRouting (remember to activate the module)
- RewriteModule (the display name is: "Url Rewrite")
- WebSocketModule (if the mediator needs to be accessible in WebSocket mode)

With above modules, a set of redirections can be defined:

1. Ensure acitivation of Application Request Routing. (select the server in the tree on the left, double-click the icon for "Application Request Routing")



Application Request Routing

Use this feature to configure proxy settings for Application Request Routing.

☑ Enable proxy

Proxy Setting

HTTP version: Pass through

☑ Keep alive

Time-out (seconds): 120

☑ Reverse rewrite host in response headers

Custom Headers

Preserve client IP in the following header: X-Forwarded-For

☑ Include TCP port from client IP

Forwarding proxy header value:

Cache Setting

Memory cache duration (seconds): 60

☑ Enable disk cache

☐ Enable request consolidation

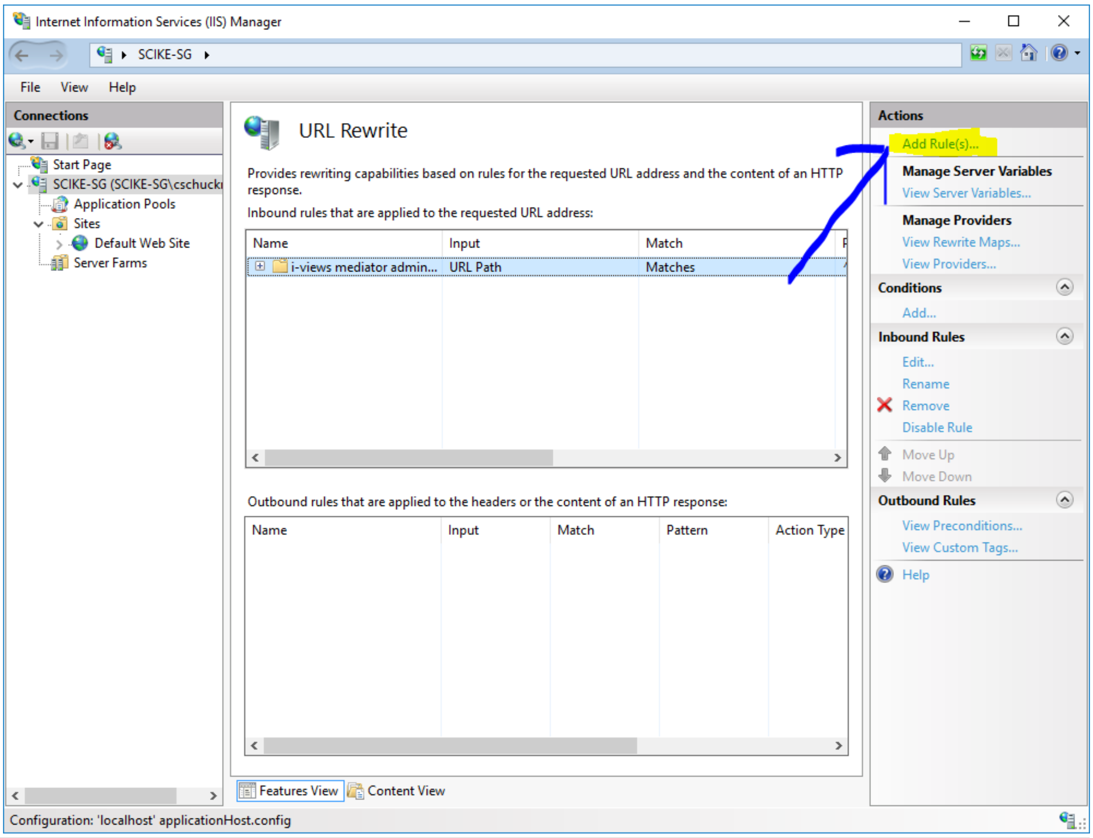
Query string support: Ignore query string

Server routing rules have not been created. Click "Use URL Rewrite to inspect incoming requests" to create these rules.

Enabling proxy allows requests to be potentially routed to servers outside of your server farm.

Configuration: 'localhost' applicationHost.config

2. Add an URL-Rewrite rule (Blank rule) (select the server in the tree on the left, double-click the icon for "URL Rewrite")



URL Rewrite

Provides rewriting capabilities based on rules for the requested URL address and the content of an HTTP response.

Inbound rules that are applied to the requested URL address:

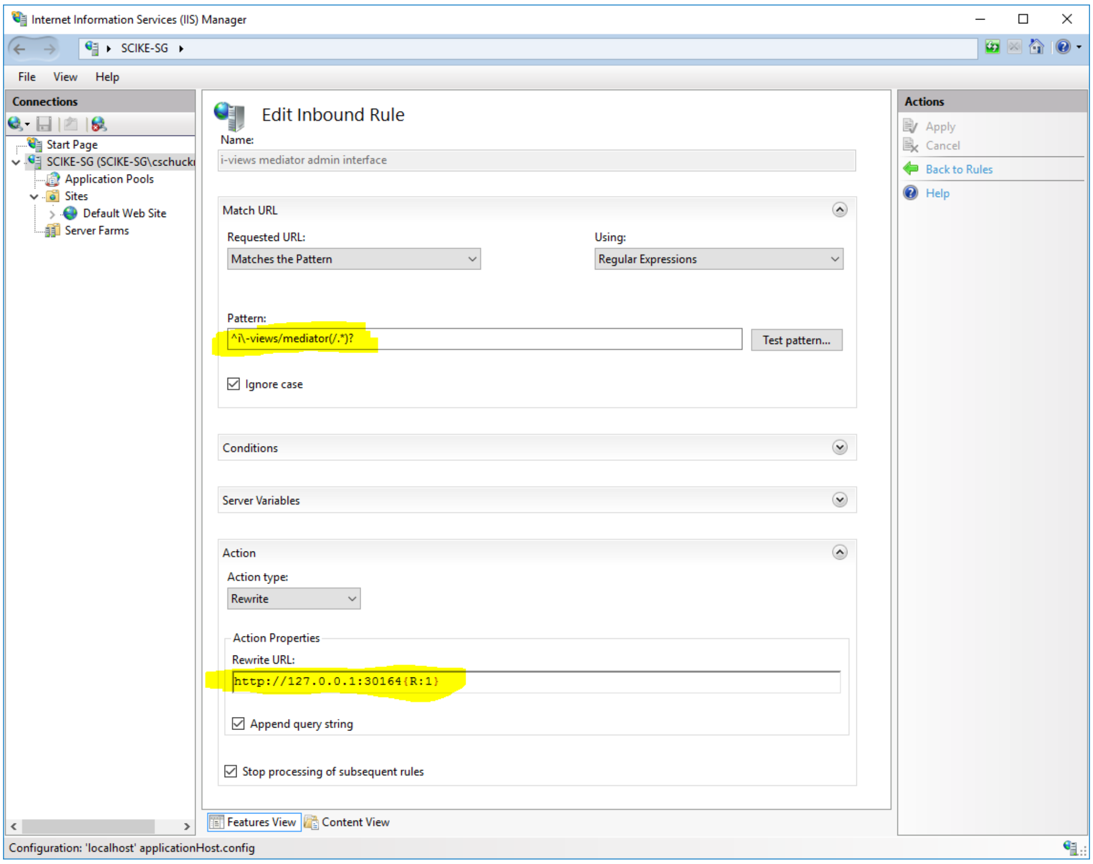
Name	Input	Match
i-views mediator admin...	URL Path	Matches

Outbound rules that are applied to the headers or the content of an HTTP response:

Name	Input	Match	Pattern	Action Type

Configuration: 'localhost' applicationHost.config

3. Add rules (in the screenshot you see a sample mediator rule)



Edit Inbound Rule

Name: i-views mediator admin interface

Match URL

Requested URL: Matches the Pattern

Using: Regular Expressions

Pattern: ^i\-views/mediator(/.*)?

☑ Ignore case

Conditions

Server Variables

Action

Action type: Rewrite

Action Properties

Rewrite URL: http://127.0.0.1:30164{R:1}

☑ Append query string

☑ Stop processing of subsequent rules

Configuration: 'localhost' applicationHost.config

Please note: On the server itself, communication should be done over HTTP even when the external traffic is HTTPS encrypted. Else you would need to handle certificates between the installed services and the IIS. Also, the encryption would bind computing resources and there is no security benefit for server-local communication.

Debugging

If requests do not flow like expected, you can use the IIS debugging functionality to investigate:

1. Select the server in the tree on the left
2. Double-click the icon "Failed Request Tracing Rules"
3. Create a rule (e.g. log all request with return codes 400-999)
4. Select the web-site in the tree
5. Click on "Configure Failed Request Tracing"
6. Select "enable" in the new dialog

The logs will be written to the folder "C:\\inetpub\\logs\\FailedReqLogFiles".

4.8.4.2. Firewall configuration

i-views processes communicate exclusively via TCP/IP. The ports used by default depend on software version used, but these defaults can be changed via local config.

Depending on the operating system and distribution, the firewall configuration needs to be performed with different tools. Therefore, here only a subset of firewall utilities can be covered. E.g. in Linux there are serveral frontends and configuration styles, all of them control the underlying iptables- or nftables-implementation in the kernel.

4.8.4.2.1. firewalld

firewalld, as it is used in newer CentOS, mainly uses services and zones in the configuration.

You can use the command line utility firewall-cmd to change the firewalls behaviour. Or you can use the XML configuration files in /etc/firewalld/services/NAME.xml.

/etc/firewalld/services/iviews.xml:

```xml
<?xml version="1.0" encoding="utf-8"?>
<service>
  <short>iviews-example</short>
  <description>i-views example installation</description>
  <!-- mediator -->
  <port protocol="tcp" port="30063"/>
  <!-- rest-bridge -->
  <port protocol="tcp" port="8815"/>
</service>
```

A new configuration can be activated for a session or permanently for the public zone by these commands:

```
firewall-cmd --permanent --zone=public --add-service=iviews
firewall-cmd --reload
```

As firewall-cmd "only" edits the zone-file you can also manually edit the file to include the service:

/etc/firewalld/zones/public.xml:

```xml
<service name="iviews"/>
```

5. Appendix

5.1. docker-compose configuration

env

```
MEDIATOR_PASSWORD=<mediator-password>
AUTH_TOKEN=<system-account>
```

docker-compose.yaml

```
version: '3'
services:
  mediator:
    image: container-registry.example.com/iviews-mediator:<image-tag-1>
    environment:
      IV_PASSWORD: "${MEDIATOR_PASSWORD:?MEDIATOR_PASSWORD not
configured}"
      IV_SCHEDULED_JOBS: "job-bu,job-gc"
      IV_JOB_BU_VOLUME_PATTERN: "<knowledge-graph-name>"
      IV_JOB_BU_BACKUP_INTERVAL: "1"
      IV_JOB_BU_BACKUP_TIME: "22:22"
      IV_JOB_BU_BACKUPS_TO_KEEP: "5"
      IV_JOB_GC_VOLUME_PATTERN: "<knowledge-graph-name>"
      IV_JOB_GC_GARBAGE_COLLECT_INTERVAL: "1"
      IV_JOB_GC_GARBAGE_COLLECT_TIME: "22:33"
    volumes:
      - "mediator-volumes:/mediator/volumes"
      - "mediator-backup:/mediator/backup"
    ports:
      - "30000:30000"
      - "30001:30001"
    restart: unless-stopped
  jobclient:
    image: container-registry.example.com/iviews-jobclient:<image-tag-2>
    environment:
      IV_HOST: "mediator:30001"
      IV_VOLUME: "<knowledge-graph-name"
      IV_AUTHENTICATION: "${AUTH_TOKEN:?AUTH_TOKEN not configured}"
      IV_JOB_POOLS: "script,index"
    depends_on:
      - mediator
    deploy:
      replicas: 1
    restart: unless-stopped
  bridge:
    image: container-registry.example.com/iviews-bridge:<image-tag-3>
    environment:
      IV_HOST: "mediator:30001"
      IV_VOLUME: "<knowledge-graph-name>"
      IV_AUTHENTICATION: "${AUTH_TOKEN:?AUTH_TOKEN not configured}"
    depends_on:
      - mediator
      - jobclient
    ports:
      - "8815:8815"
    deploy:
      replicas: 1
    restart: unless-stopped
volumes:
  mediator-volumes:
  mediator-backup:
```

5.2. kubernetes configuration

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mediator
  labels:
    app: mediator
spec:
  selector:
    matchLabels:
      app: mediator
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: mediator
    spec:
      containers:
      - name: mediator
        image: container-registry.example.com/iviews-mediator:<image-tag-
1>
        ports:
        - containerPort: 30000
        - containerPort: 30001
        env:
        - name: IV_PASSWORD
          valueFrom:
            secretKeyRef:
              name: iviews-secret
              key: MEDIATOR_PASSWORD
        - name: IV_INTERFACES
          value: "http://0.0.0.0:30000,cnp://0.0.0.0:30001"
        - name: IV_SCHEDULED_JOBS
          value: "job-bu,job-gc"
        - name: IV_JOB_BU_VOLUME_PATTERN
          valueFrom:
            configMapKeyRef:
              name: iviews-config
              key: VOLUME
        - name: IV_JOB_BU_BACKUP_INTERVAL
          value: "1"
        - name: IV_JOB_BU_BACKUP_TIME
          value: "22:22"
        - name: IV_JOB_BU_BACKUPS_TO_KEEP
          value: "1"
        - name: IV_JOB_GC_VOLUME_PATTERN
          valueFrom:
            configMapKeyRef:
              name: iviews-config
              key: VOLUME
        - name: IV_JOB_GC_GARBAGE_COLLECT_INTERVAL
          value: "1"
        - name: IV_JOB_GC_GARBAGE_COLLECT_TIME
          value: "22:33"
        volumeMounts:
        - mountPath: /mediator/volumes
          name: mediator-volumes
        - mountPath: /mediator/backup
          name: mediator-backup
      volumes:
      - name: mediator-volumes
        persistentVolumeClaim:
          claimName: mediator-volumes
      - name: mediator-backup
        persistentVolumeClaim:
          claimName: mediator-backup
      securityContext:
        runAsUser: 10000
        runAsGroup: 10000
        fsGroup: 10000
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mediator-volumes
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 20Gi
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mediator-backup
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 100Gi
---
apiVersion: v1
kind: Service
metadata:
  name: mediator
  labels:
    app: mediator
spec:
  selector:
    app: mediator
  ports:
  - name: mediator-http
    port: 30000
  - name: mediator-cnp
    port: 30001
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: mediator
spec:
  policyTypes:
  - Ingress
  podSelector:
    matchLabels:
      app: mediator
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: jobclient
    - podSelector:
        matchLabels:
          app: bridge
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: jobclient
  labels:
    app: jobclient
spec:
  selector:
    matchLabels:
      app: jobclient
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: jobclient
    spec:
      containers:
      - name: jobclient
        image: container-registry.example.com/iviews-mediator:<image-tag-
2>
        imagePullPolicy: IfNotPresent
        env:
        - name: IV_HOST
          value: "mediator:30001"
        - name: IV_VOLUME
          valueFrom:
            configMapKeyRef:
              name: iviews-config
              key: VOLUME
        - name: IV_AUTHENTICATION
          valueFrom:
            secretKeyRef:
              name: iviews-secret
              key: AUTHENTICATION
        - name: IV_JOB_POOLS
          value: script,index,query
      securityContext:
        runAsUser: 10000
        runAsGroup: 10000
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: bridge
  labels:
    app: bridge
spec:
  selector:
    matchLabels:
      app: bridge
  strategy:
    type: Recreate
  replicas: 1
  template:
    metadata:
      labels:
        app: bridge
    spec:
      containers:
      - name: bridge
        image: container-registry.example.com/iviews-bridge:<image-tag-3>
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 8815
        env:
        - name: IV_HOST
          value: "mediator:30001"
        - name: IV_VOLUME
          valueFrom:
            configMapKeyRef:
              name: iviews-config
              key: VOLUME
        - name: IV_AUTHENTICATION
          valueFrom:
            secretKeyRef:
              name: iviews-secret
              key: AUTHENTICATION
      securityContext:
        runAsUser: 10000
        runAsGroup: 10000
---
apiVersion: v1
kind: Service
metadata:
  name: bridge
  labels:
    app: bridge
spec:
  selector:
    app: bridge
  ports:
  - name: bridge
    port: 8815
---
apiVersion: v1
kind: Secret
metadata:
  name: iviews-secret
type: Opaque
data:
  MEDIATOR_PASSWORD: <encoded-mediator-password>
  AUTHENTICATION: <encoded-system-account>
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: iviews-config
data:
  VOLUME: <knowledge-graph-name>
```