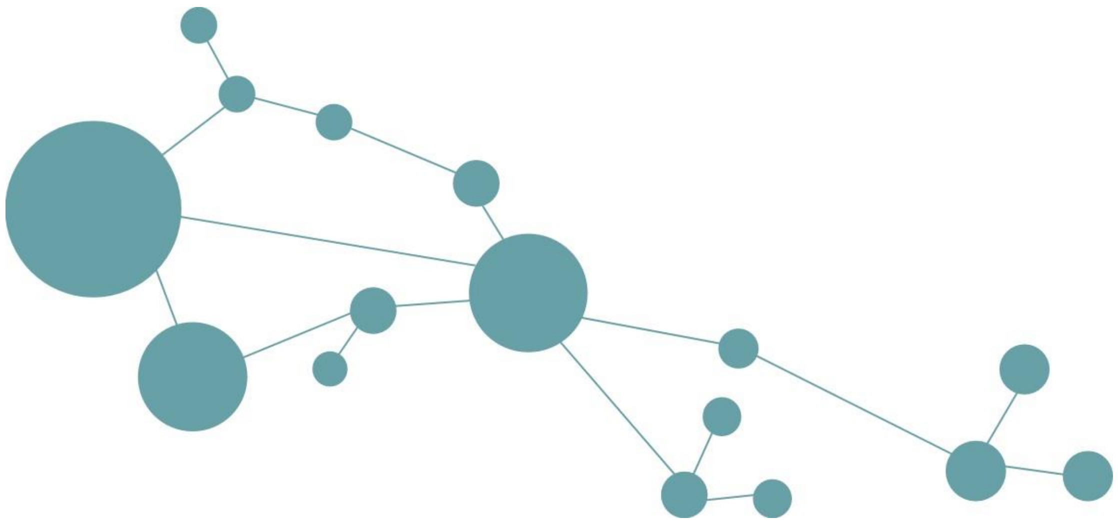


i-views 5.3

Contents

1 Knowledge-Builder

1.1 Basics

When using i-views, databases work the way people think: simple, agile and flexible. That is why in i-views many things are different than relational databases: we do not work with tables and keys, but with objects and the relationships between them. Modelling of the data is visual and oriented towards examples so that we can also share it with users from the specialist departments.

With i-views we do not set-up pure data storage but intelligent data networks which already contain a lot of business logic and with which the behaviour of our application may, to a large extent, be defined. To this end we use inheritance, mechanisms for conclusions and for the definition of views, along with a multitude of search processes which i-views has to offer.

Our central tool is the knowledge builder, one of the core components of i-views. Using the knowledge builder we can:

- define the scheme but also establish examples and, above all, visualise
- define imports and mappings from a data source
- phrase requests, traverse networked data, process strings and calculate proximities
- define rights, triggers and views

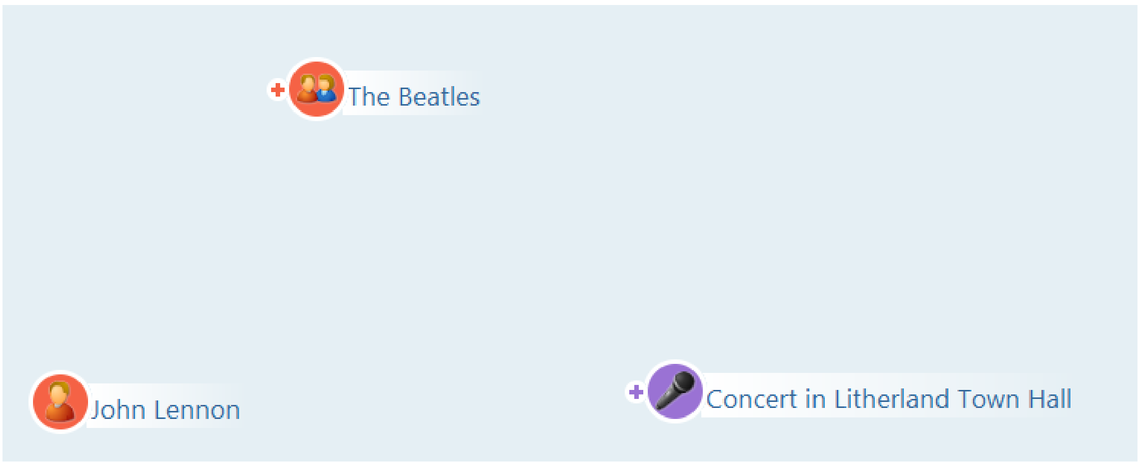
All these functions are the subject of this documentation. One continuous example is a semantic network surrounding music, bands, songs, etc.

1.1.1 Building blocks

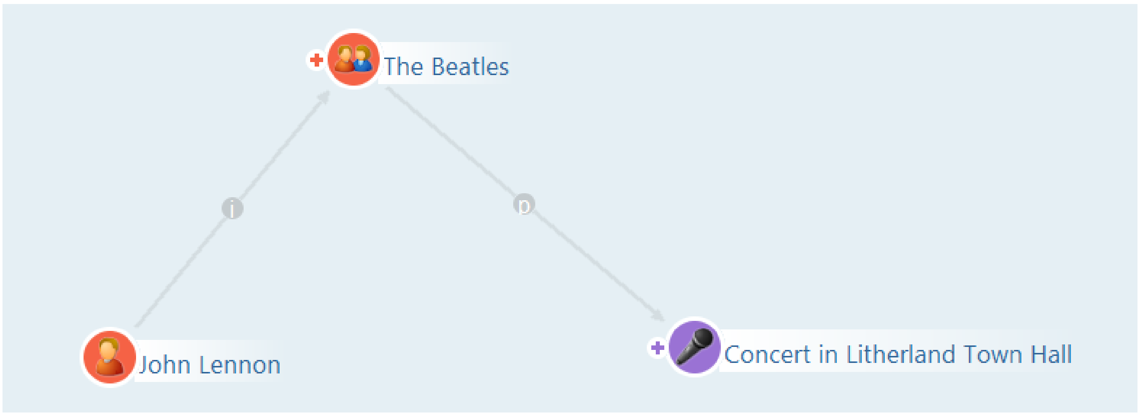
The basic components of modelling within i-views are:

- specific objects
- relationships
- attributes
- types of objects
- types of relationships
- types of attributes

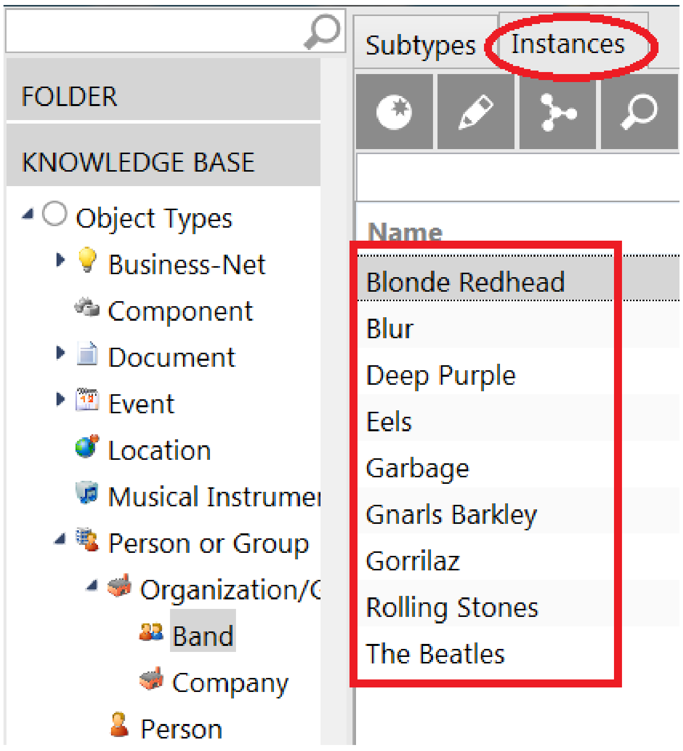
Examples for specific objects are John Lennon, the Beatles, Liverpool, the concert in Litherland Town Hall, the football world cup in Mexico in 1970, the leaning tower of Pisa, etc.:



We can link these specific objects together through relationships: "John Lennon is a member of the Beatles", "The Beatles perform a concert in Litherland Town Hall".

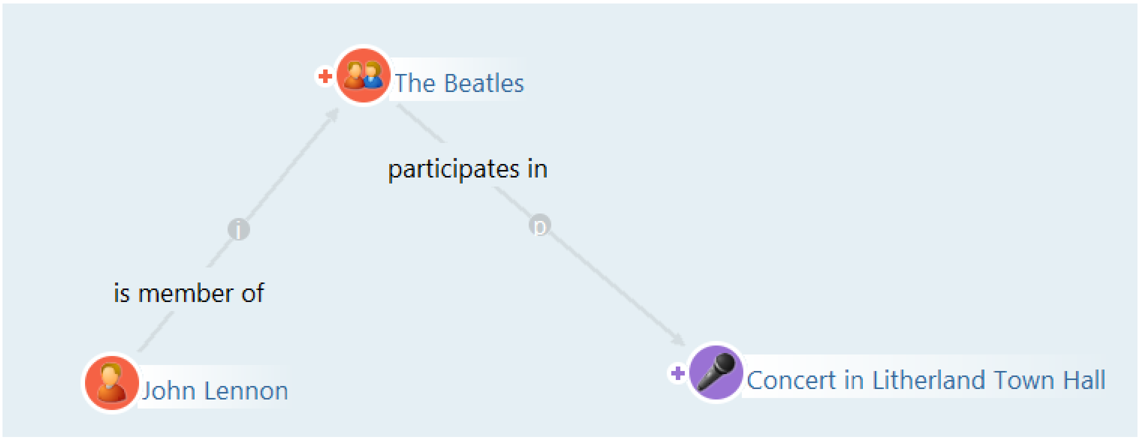


Additionally, we have introduced four types here: specific objects always have a type, e.g. the type of persons, type of the cities, the events or the bands - types which you may freely define in your data model.



The main window of i-views: on the left-hand side the types of objects, on the right-hand side the respective, specific objects - here we can also see that the types of the i-views networks are within a hierarchy. You will find out more about the type of hierarchy in the next paragraph.

Even the relationships have different types: between John Lennon and the Beatles there is the relationship "is member of"; between the Beatles and their concert the relationship could be called "performed at" - if we want to generalise more, "participates in" is perhaps a more practical type of relationship.

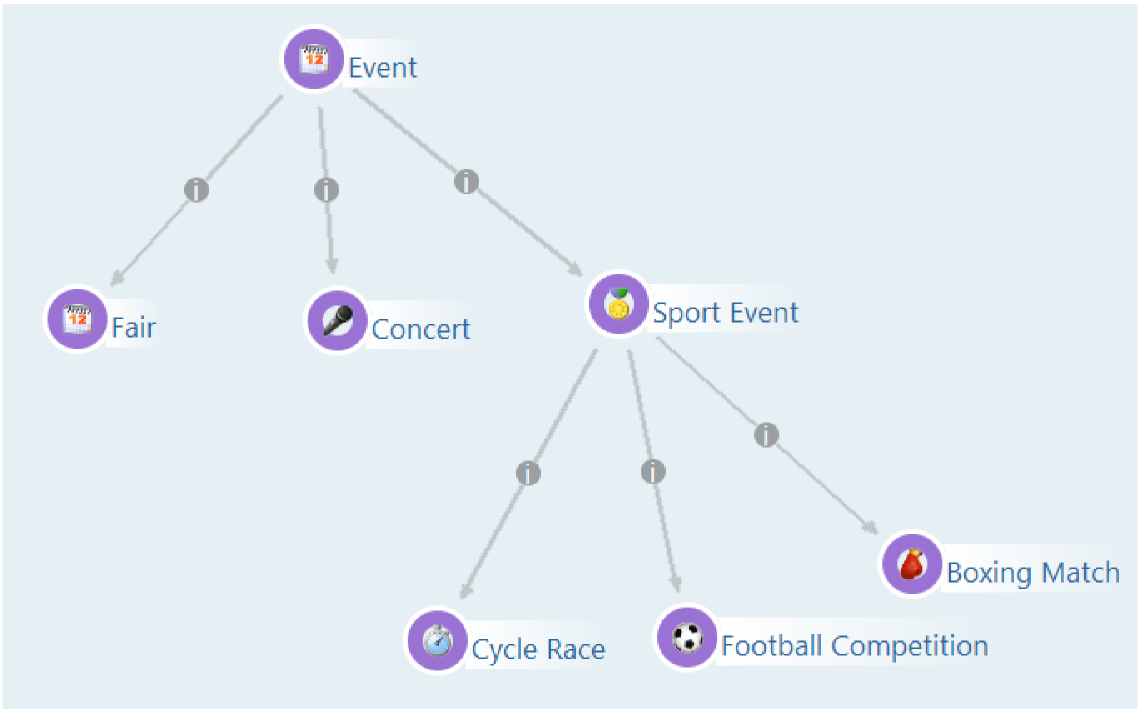


The same applies for attributes: in the case of a person these may be the name or the date of birth. Specific persons (objects of the type 'person') may then have name, date of birth, place of birth, address, colour of eyes, etc. Events may have a location and a time span. Attributes and relations are always defined with the object itself.

1.1.2 Type hierarchy - Inheritance

We can finely or less finely divide types of objects: we can put the football world cup in 1970 into the same basket as all the other events (the book fair in 2015, the Woodstock festival, etc.), then we only have one type called "event" or we differentiate between sport events, fairs, exhibitions, music events, etc. Of course, we can divide all these types of events even finer: sport events may, for example, be differentiated by the types of sports (a football match, a basket ball match, a bike race, a boxing match).

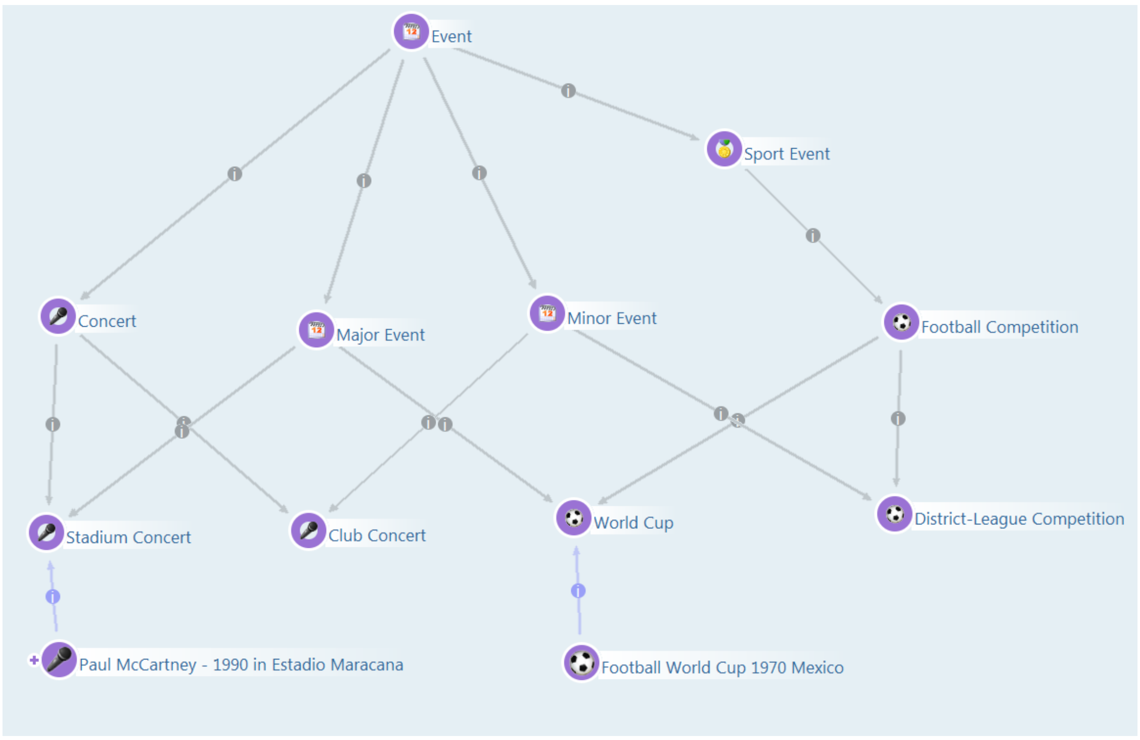
In this manner we obtain a hierarchy of supertypes and subtypes:



The hierarchy is transitive: when we ask i-views about all events, not only all specific objects are shown which are of type event, but also all sports events and all bike races, boxing matches and football matches. Hence, since the type "boxing match" is not only a subtype of "sport event", i-views will reject a direct supertype / subtype relationship between event and boxing match - with a note that this connection is already known.

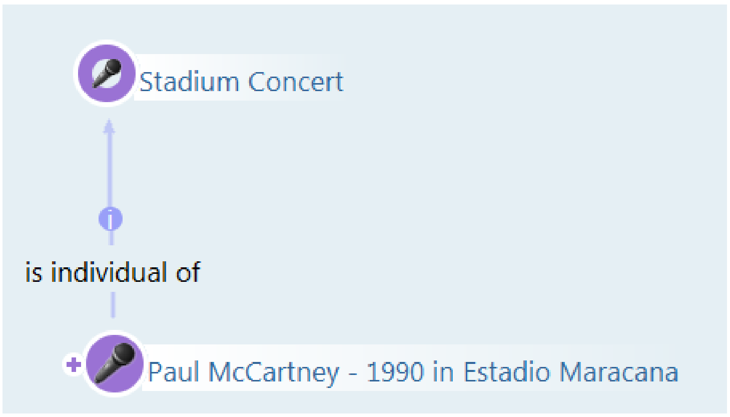
The hierarchical structure does not necessarily have to have the structure of a tree - a type of object may also have several upper types. However, an object may only have one type of object.

If we then wish to join the aspects of a concert and major event we cannot do this in the specific concert with Paul McCartney because we need the type of object "stadium concert" in order to do this:

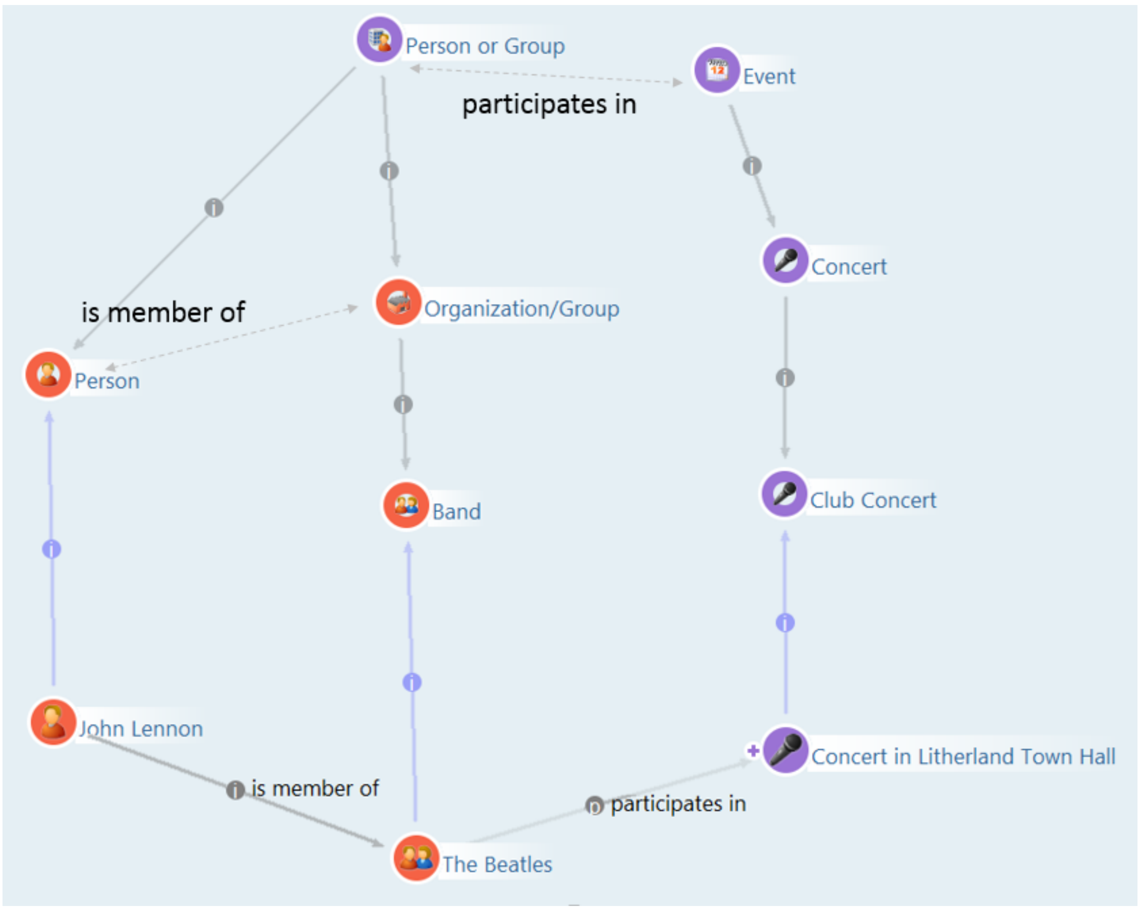


Type hierarchy with multiple inheritance

The affiliation of specific objects with a type of object is also expressed as a relation in i-views and may as such be queried:



When do we differentiate between types at all? Types do not only differ in icon and colour - their properties are also defined in the types and when queried, the types can also easily be filtered. The inheritance plays a major role in all these questions: properties are inherited, icons and colours are inherited and when, in a query, we say that we wish to see events, all objects of the subtypes are also shown in the results.

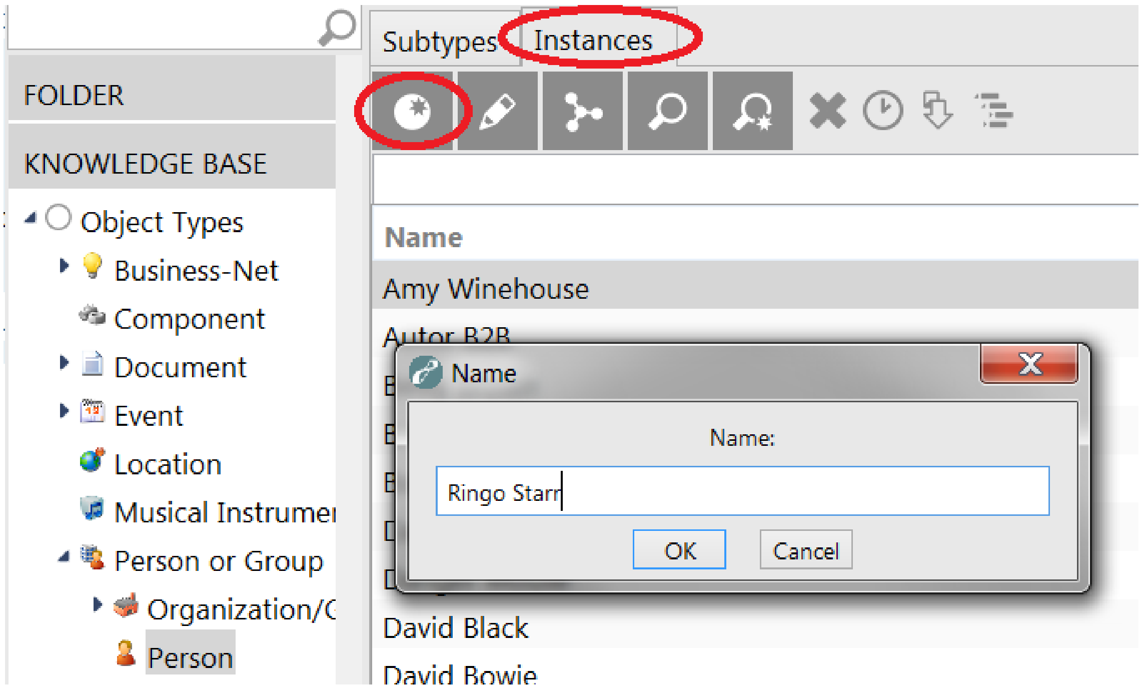


Inheritance makes it possible to define types of relations (and types of attributes) further up in the hierarchy of the object type and hence use them for different types of objects (e.g. for bands and other organisations.

1.1.3 Create and edit objects

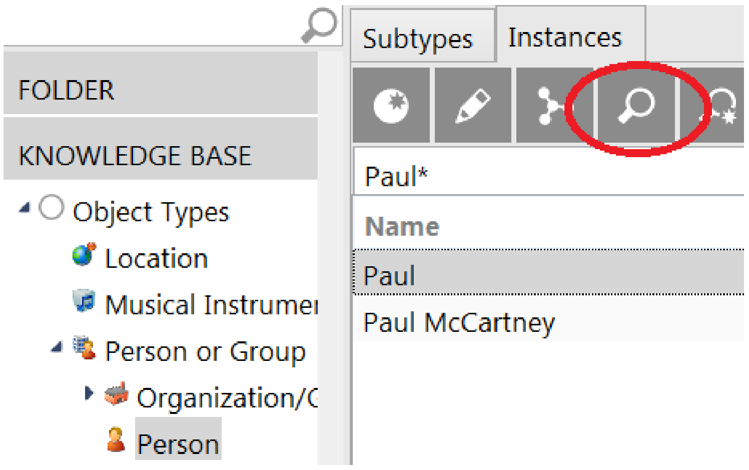
Creating specific objects

Specific objects (in the knowledge builder they are called "instances") may be created everywhere within the knowledge builder where types of objects can be seen. Based on the types of objects, objects can be newly created via the context menus.



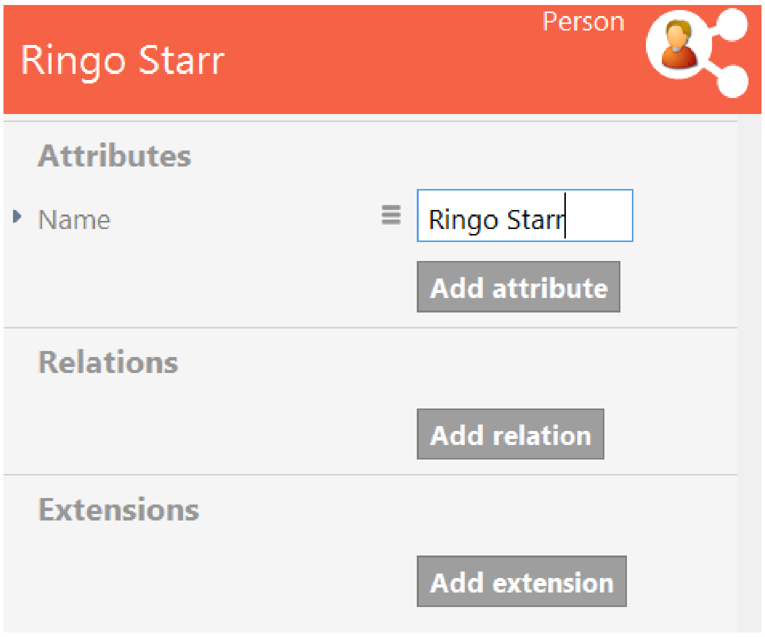
An object can be created by means of the button "new" and using the named entered

In the main window below the header there is the list of specific objects already available. In order that objects cannot inadvertently be created twice, the name of the object can be keyed into the search button in the header. The search does not, by default, differentiate between upper and lower case and the search term may be cut off left and right (supplement by placeholders "*" and "?"):



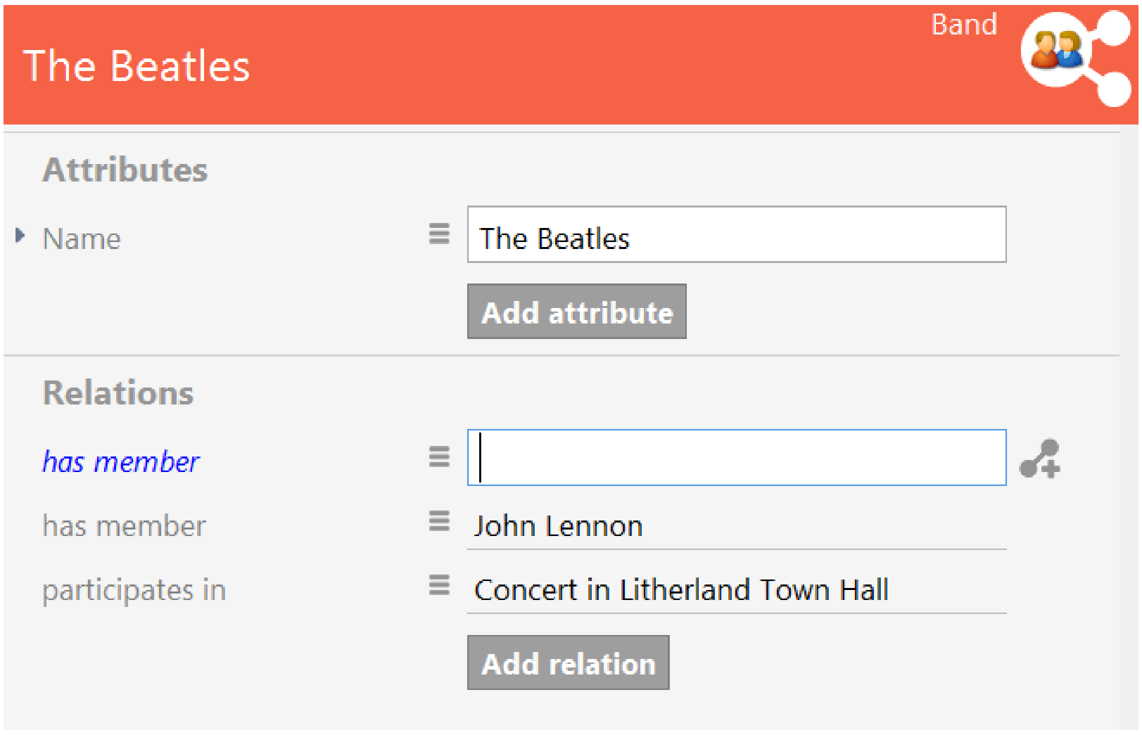
Editing objects

After entering and confirming the name of the object, further details for the object created may be keyed into the editor. The object may be assigned attributes, relations and extensions by using the respective buttons.

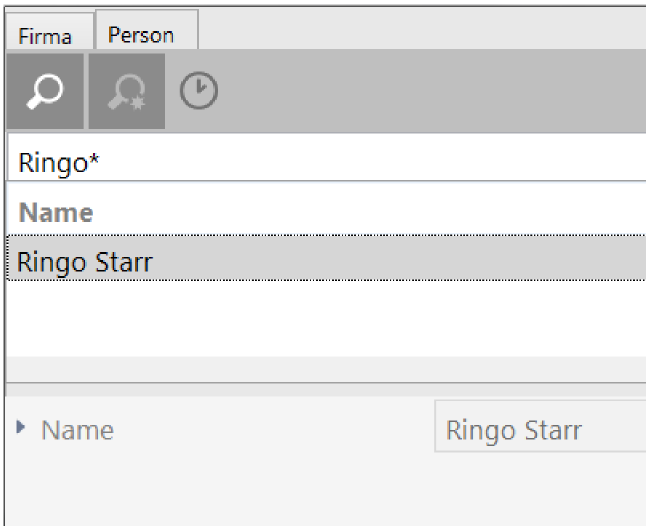


When editing an object we can, in addition to linking it to another object, also generate the target of the link if the object does not already exist.

For example, members of a music band are documented completely. Via the relation, we want to link the member Ringo Starr with the object "The Beatles". If it is not yet clear whether the object Ringo Starr is already documented in i-views you can use the search button to ascertain this,



or via the icon button, select ’Choose relation target’ from a searchable list with all feasible targets of relation.



Deleting the relation has a member may be accomplished in two different ways:

1. Delete in the context menu using the button *further actions* ≡ and the option '*delete*'.
2. With the cursor over the button *further actions* ≡ and holding down the Ctrl key.

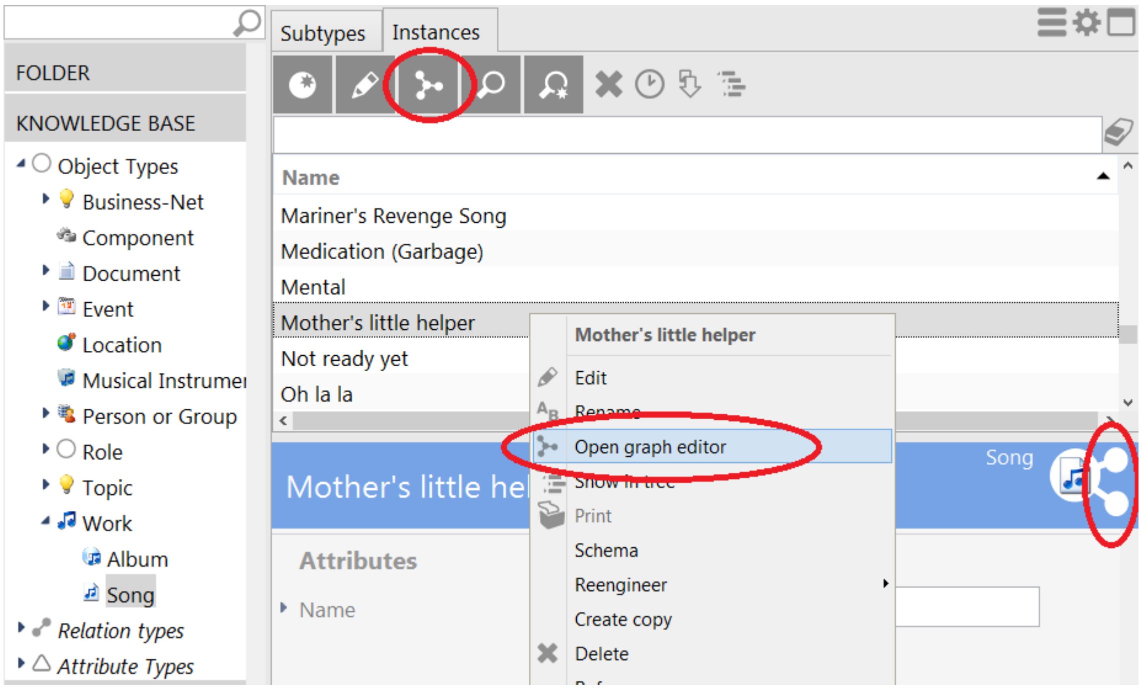
The target object of the relation itself will not be deleted as a result of this however. If an object has to be deleted this is done via the button ✖ in the main window or via the context menu directly on this object.

Objects may also be created using the graph editor. This process is described in the following paragraphs.

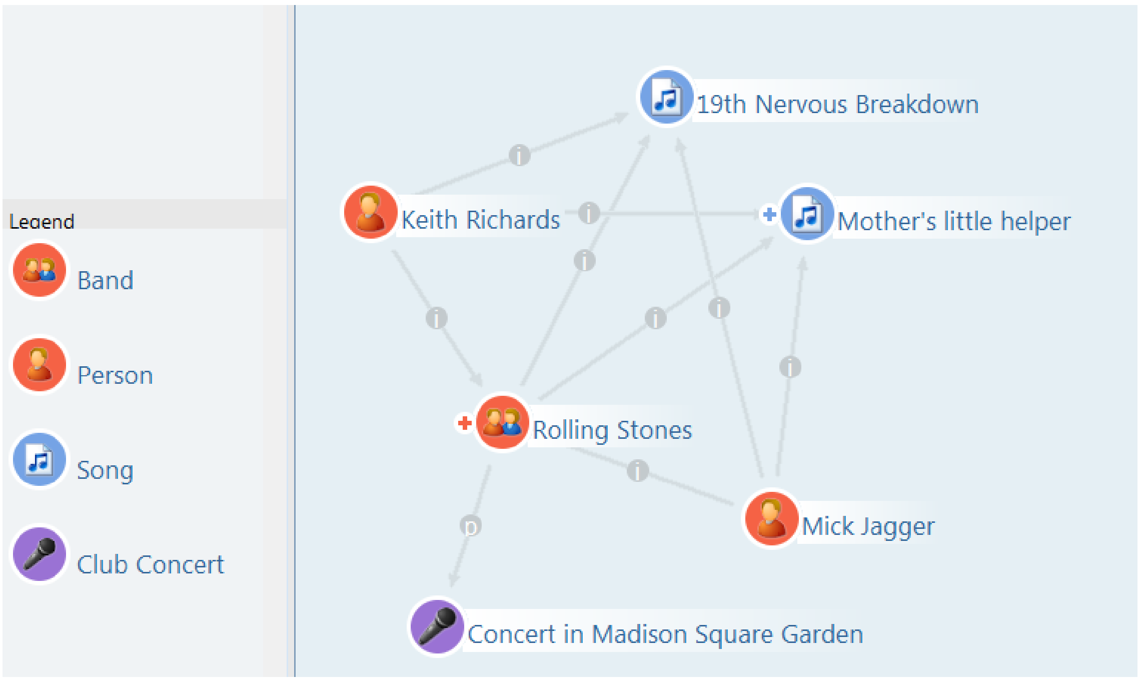
1.1.4 Graph editor

1.1.4.1 Introduction graph editor

By using the graph editor, knowledge networks with their objects and links can be depicted graphically. The graph editor may be opened on a selected object using the *graph* button:



The graph always shows a section of the network. Objects from the graph may be displayed and hidden and you can navigate through the graph.

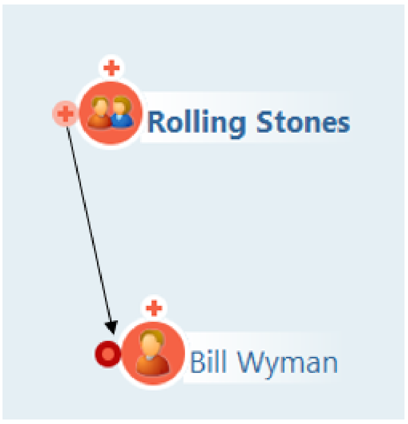


In the graph editor not only a section of the network may be displayed: objects and relations may be edited as well.

On the left-hand side of a node there is a drag point for interaction with the object. By double-clicking on the drag point all user relations of the object will be displayed or hidden.

Linking objects via a relation is carried out in the graph editor as follows:

1. Position the cursor over the drag point to the left of the object with the left mouse button.
2. Drag the cursor in a held down position to another object (drag & drop). If several relations are available for selection, a list will appear with all feasible relations. If there is only one feasible relation between the two objects, this will be selected and no list will be shown.

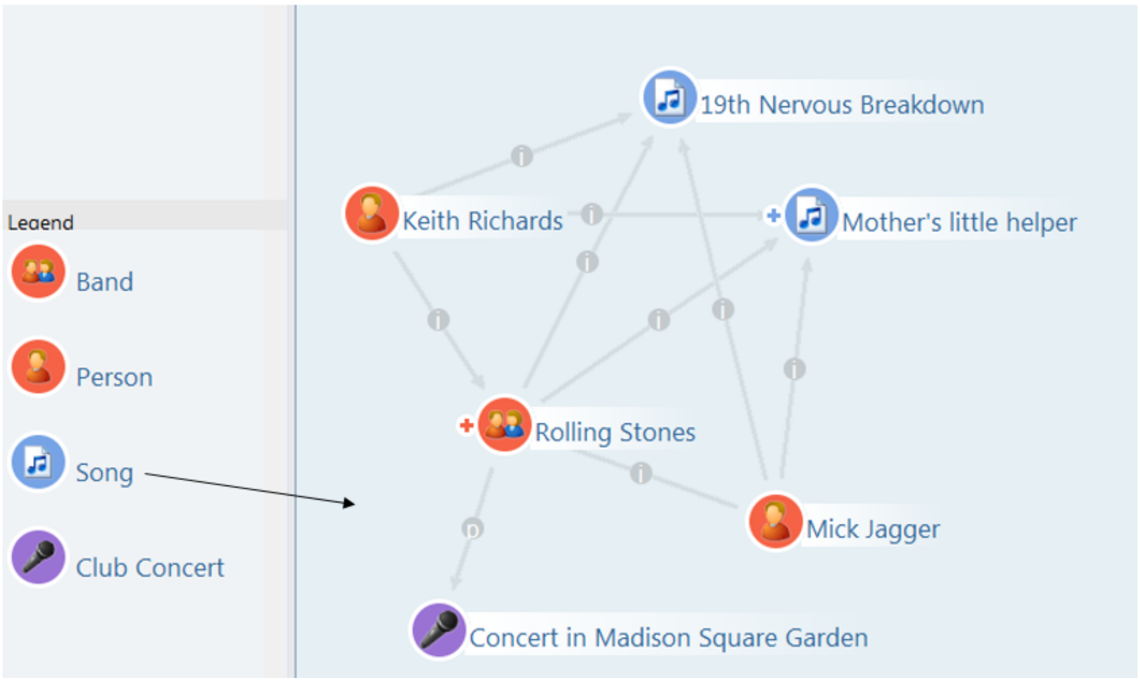


In order to display objects in the graph editor there are different options:

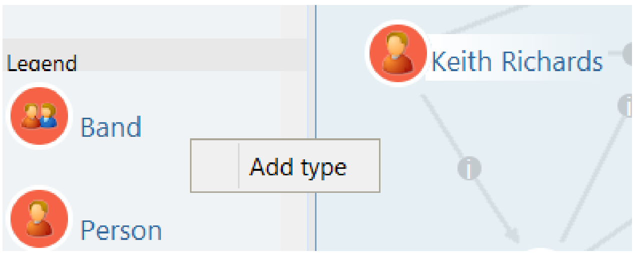
- Objects may be dragged from the hit list in the main window to the graph editor window using drag & drop.
- If the name of the object is known it can be selected via the context menu using the function "show individual".

If an object is to be hidden from the graph editor, it may be removed from there by clicking it and dragging it from the graph editor holding down the Ctrl key. In doing so, there will be no changes in the data: the object will exist unchanged within the semantic network but it will not be displayed anymore in the current graph editor section.

New objects may also be created in the graph editor. To do this we drag & drop the type of object from the legend on the left-hand side of the graph editor to the drawing area:



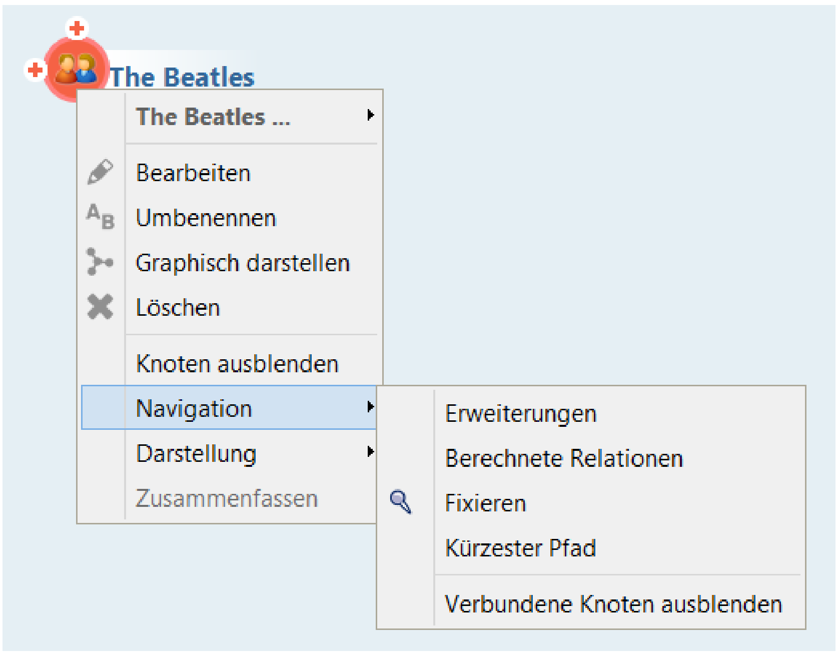
If there are no types of objects to be seen in the legend you can search for them using a right mouse click in the legend area. Following this, the name of the object will be given.



The editor will re-appear in which the possible relations, attributes and enhancements for the object can be edited.

1.1.4.2 Operations on objects in the graph editor

The name can be changed later on in the Admin tool or the Knowledge Builder. The user created in this way automatically has graph administrator rights. Right-clicking the object in the context menu allows other operations to be executed. For the most part, this context menu provides the same functions as the form editor, however also includes other graph editor-specific components.

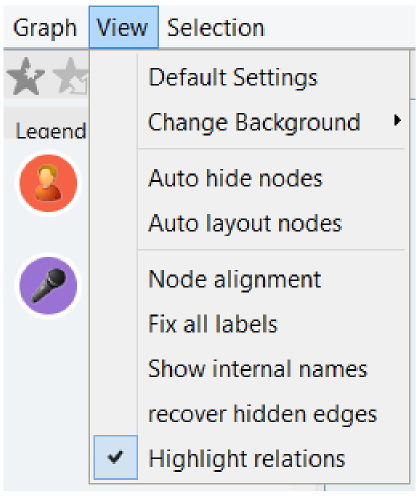


The following graph editor-specific functions are available in this context menu:

- **Hide node:** The node can be hidden here.
- **Navigation - Extensions**: Opens the extensions for an object.
- **Navigation - Calculated relations**: Opens the calculated relations for an object.
- **Navigation - Fix:** Fixes the position of a node in the graph editor, so that it is not repositioned even when the layout is restructured. The fixed node can be undone using the *Release* option.
- **Navigation - Shortest path**

1.1.4.3 View

The menu "*View*" provides many more functions for the graphic illustration of objects and types of objects:



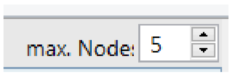
Default settings: Opens the menu with the default settings for the graph editor**.** This menu is also available in: global setting window -> register card "*personal*" -> *graph*. There you can set whether attributes, relations and enhancements should appear in a small mouse-over-window above the object and how many nodes at a maximum will be visible in one step:

- **Show bubble help with details**: if the mouse pointer stops on one node the details of the first ten attributes and relations will be displayed in a yellow window if bubble help was previously activated. (check "show bubble help with details" in the global setting window register card "personal" graph)

- **Max nodes**: if a node/object has a lot of adjacent objects it often doesn't make sense to show them all by clicking on the drag point.

Change Background: The background color can be changed or a picture can be set as background.

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. nodes" in the toolbar:


max. Node: 5

Auto layout nodes: automatically implements the layout function for newly displayed nodes.

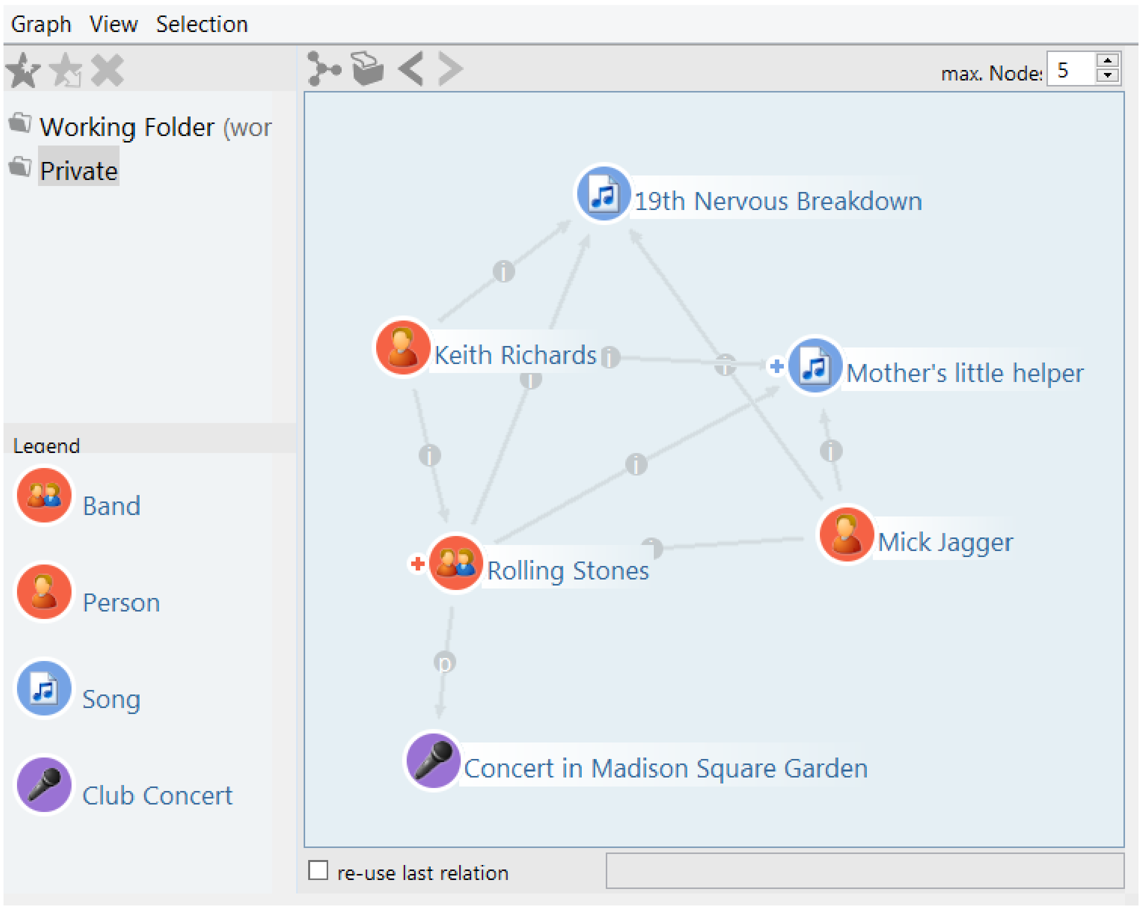
Fix all labels: using this option the names of all relations are always visible, not only when rolled over with the mouse. Alternatively, the description may be fixed directly in the context menu of a relation.

Show internal names: displays the internal name of types of in brackets

recover hidden edges: all edges hidden by means of the context menu are shown again

The window of the graph editor and the main window of the knowledge builder provide even more menu items which may offer support when modelling the knowledge network.

On the left-hand side of the graph editor window there is the legend of the types of objects.

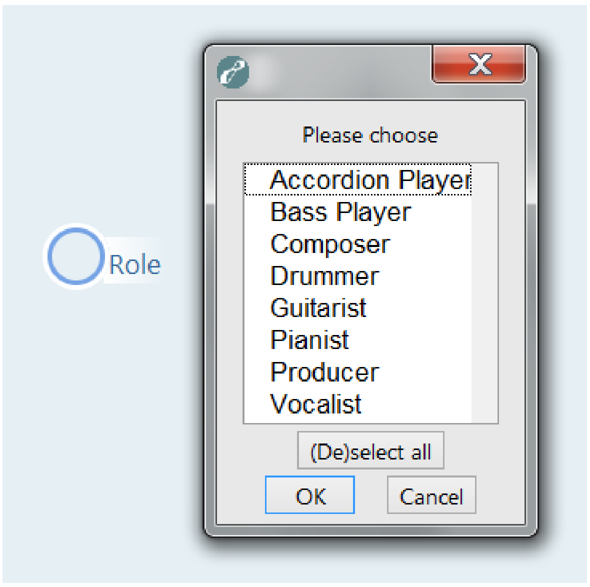


This legend shows the types of objects for the specific objects on the right-hand side.

By dragging & dropping an entry from the legend into the drawing area you can create a new specific object of the corresponding type.

Via the context menu for the legend entries all specific objects can be hidden from the image. Here you can also "hold" legend entries and add new types of objects to the legend (regardless of whether specific objects of this kind are represented in the image).

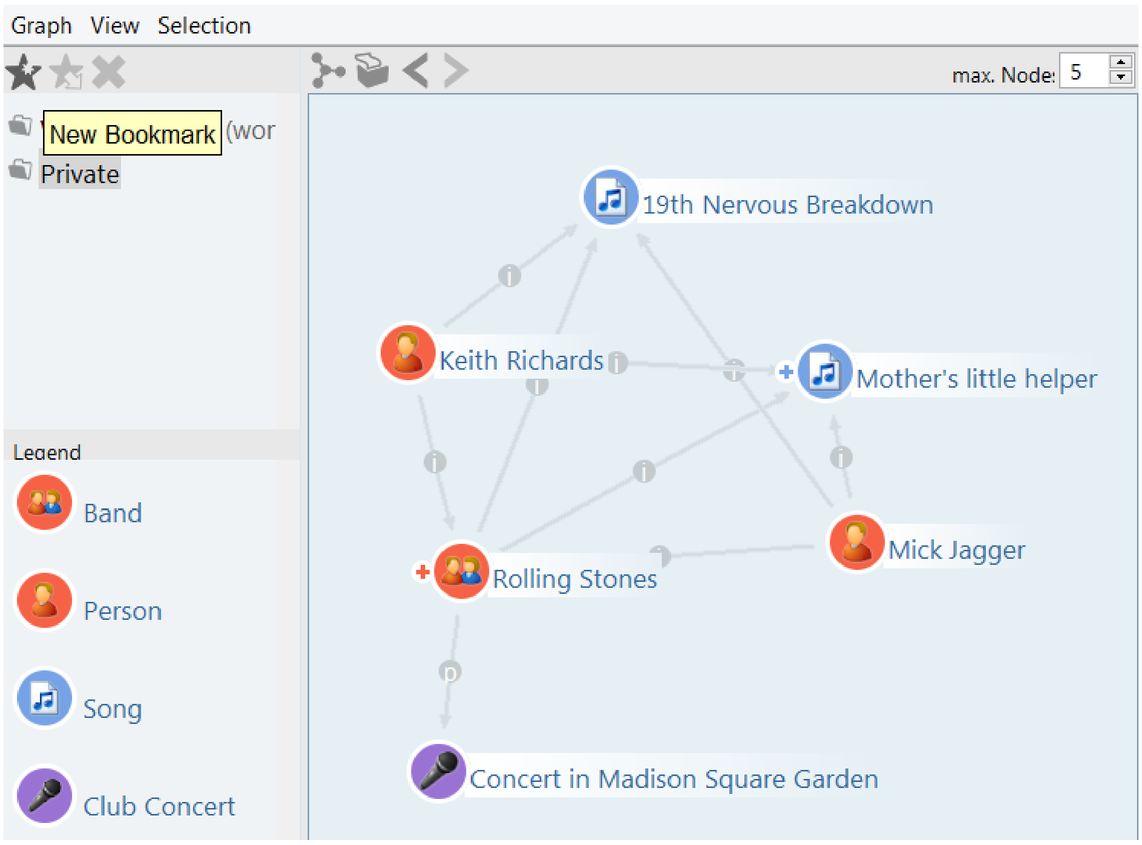
If the drag point has been clicked to show the adjacent objects a selection list will appear instead of the objects.



1.1.4.4 Bookmarks and history

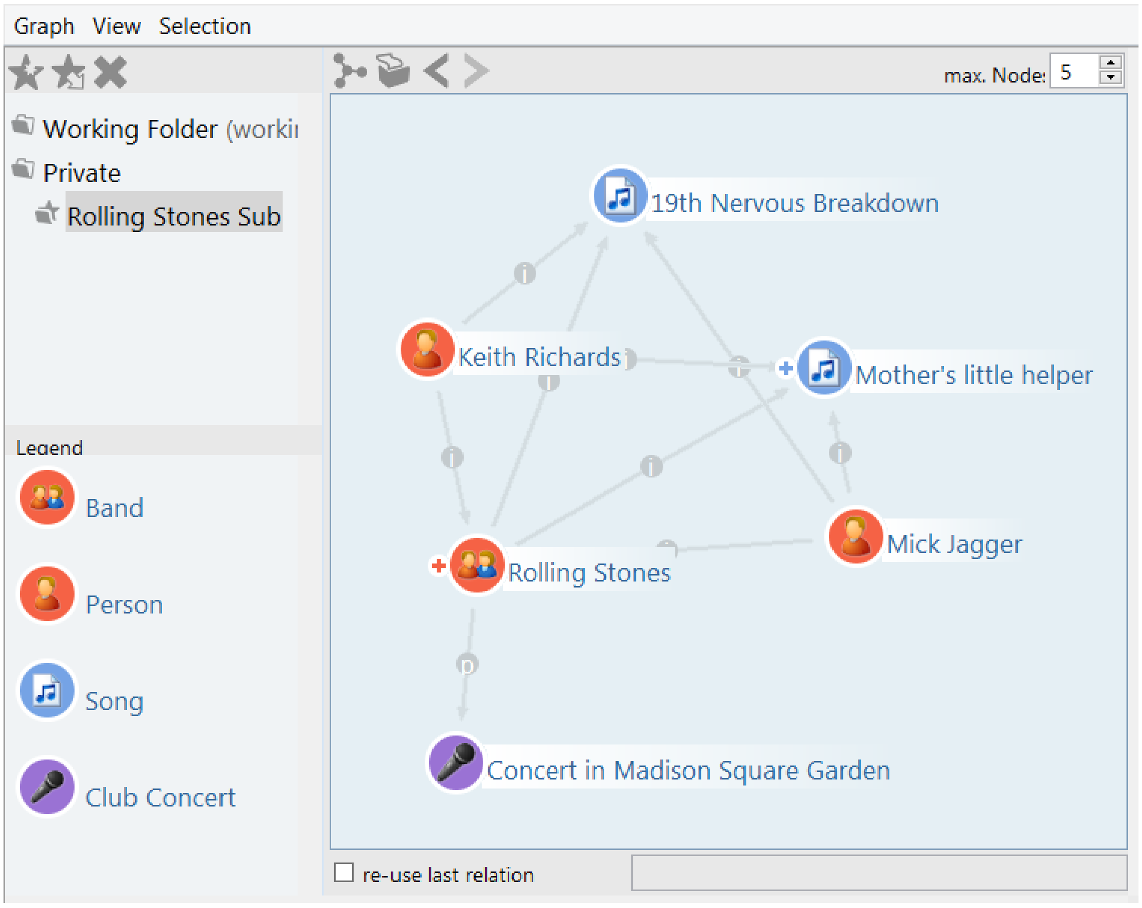
The menu *graph* contains more functions for the graph editor:

Bookmarks: parts of the knowledge network or "sub-networks" can be saved as bookmarks. The objects are saved in the same position as they are placed in the graph editor.

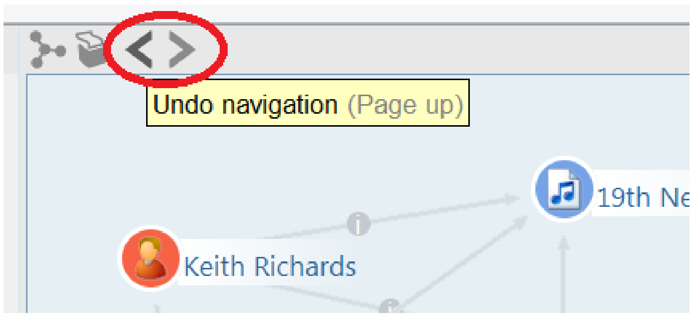


When a bookmark is created it may be given a name. All nodes contained in the bookmark are listed in the description of the bookmark.

Bookmarks, however, are not data backups: objects and relations which were deleted after a bookmark was saved are also no longer available when the bookmark is shown.



History: using the buttons "reverse navigation" and "restore navigation" elements of a (section of) a knowledge network may be hidden again in the order of sequence in which they were shown (and vice versa). Furthermore, these buttons reverse the auto layout. The buttons can be found in the header of the graph editor window or in the menu "graph".

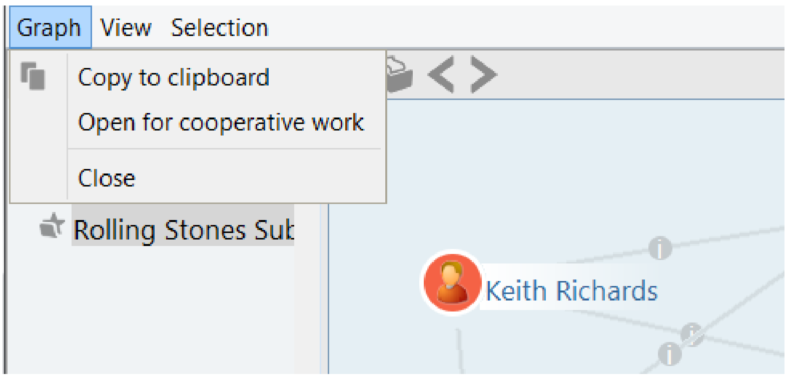


Layout: the layout function enables you to position nodes automatically when many nodes are not allowed to be positioned manually. When more nodes are displayed they will also be automatically positioned in the graph via the layout function.

Copy into the clipboard: this function creates a screenshot of the current contents of the graph editor. This image may then be inserted into a drawing or picture processing programme, for example.

Print: opens the dialogue window for printing or for generating a pdf file from the displayed graph.

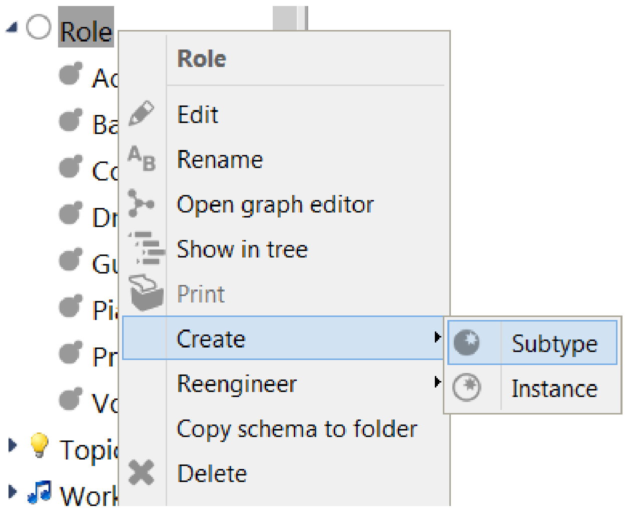
Cooperative work: this function enables other users to work on the graph mutually and simultaneously. All changes and selections of a user on the graph (layout, showing/hiding nodes, etc.) will then be shown to all other users synchronously.



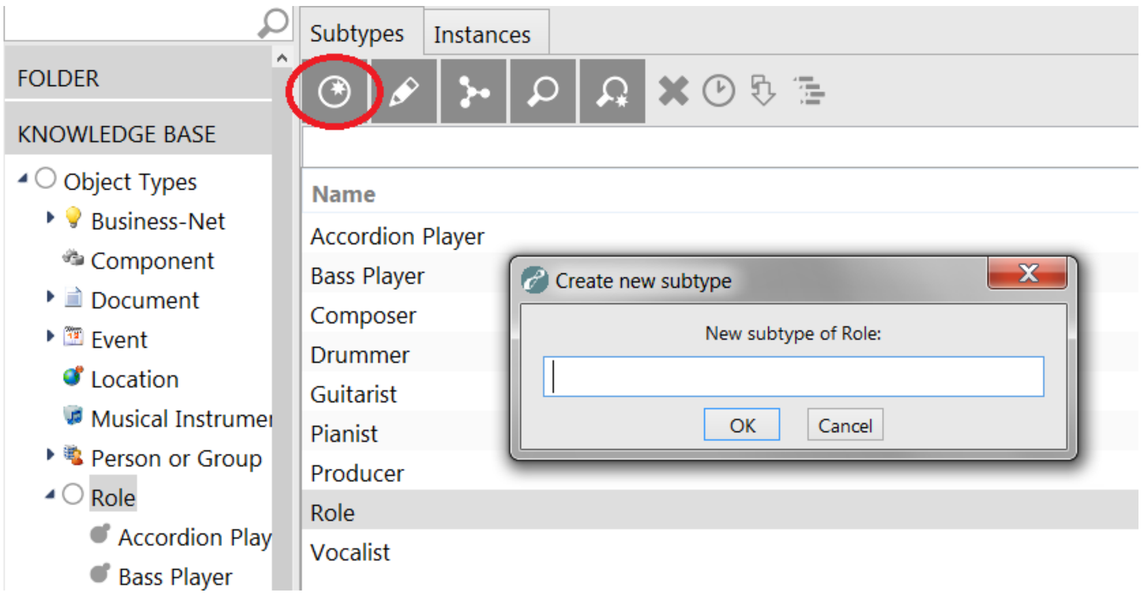
1.2 Definition of schema / model

1.2.1 Define types

The principle of the type hierarchy was already presented in Chapter 1.2. If new types are to be created this is always done as a subtype of a type which already exists. Creating subtypes can be carried out either via the context menu Create -> Subtype



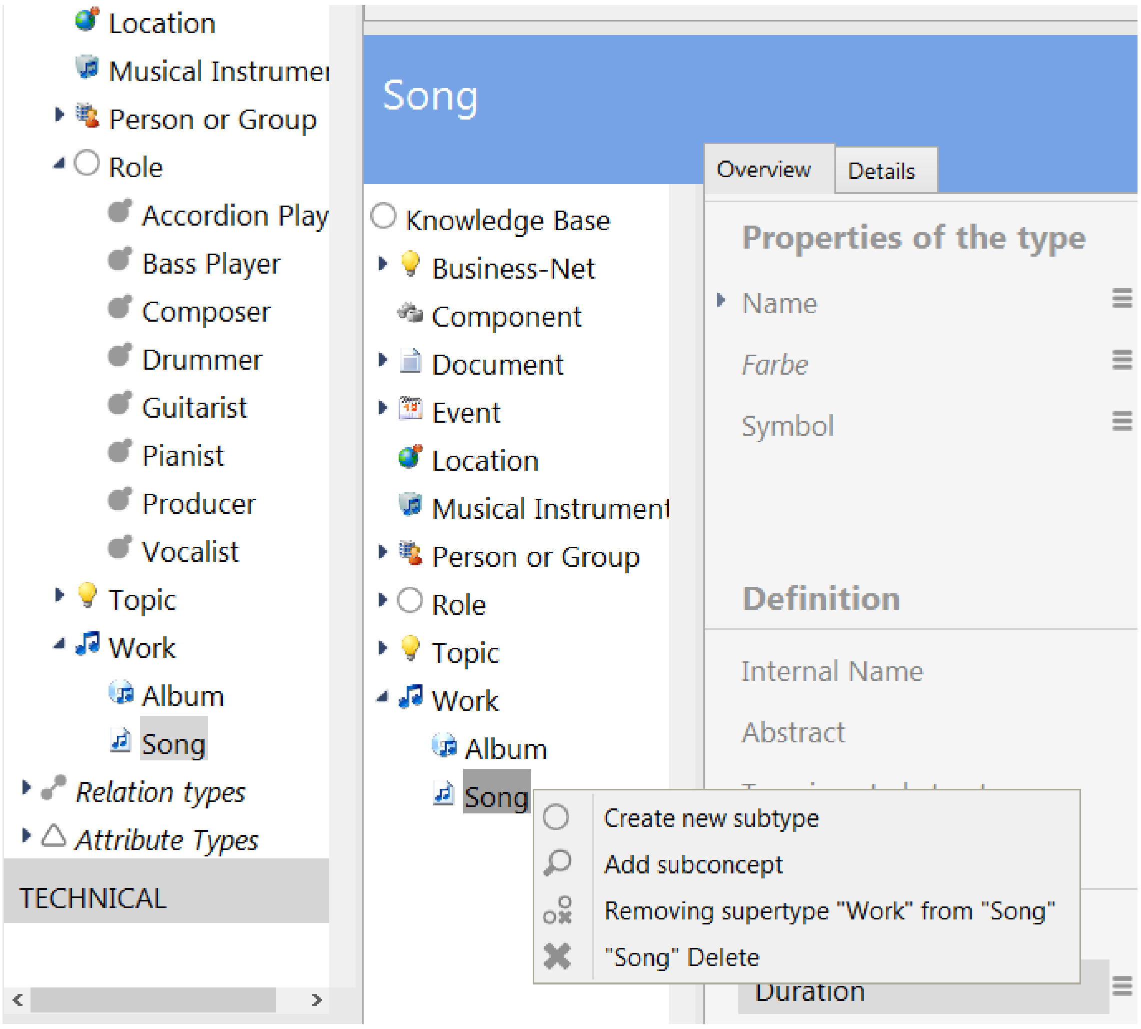
or in the main window using the tab "Subtypes" above the search field and the tab "new":



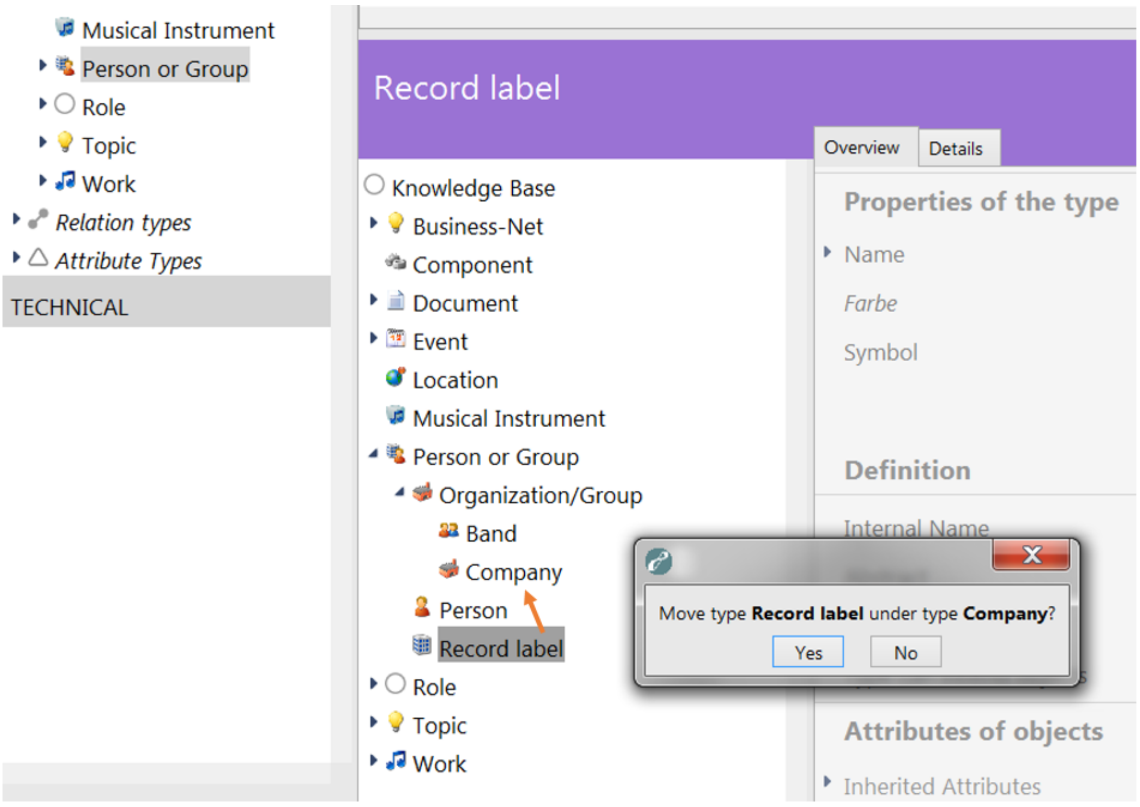
Changing the type hierarchy

In order to change the type hierarchy we have the tree of object types in the main window and the graph editor.

In the hierarchy tree of the object editor we will find the option "Removing supertype x from y" in the context menu.



Using this option we can remove the currently selected object type from its position in the hierarchy of the object types and with drag & drop we can move an object type to another branch of the hierarchy. If we hold down the Ctrl key when using the drag & drop function the object type will not be moved but additionally assigned to another object type. What still applies is: the hierarchy of the object type allows multiple assignments and inheritance.

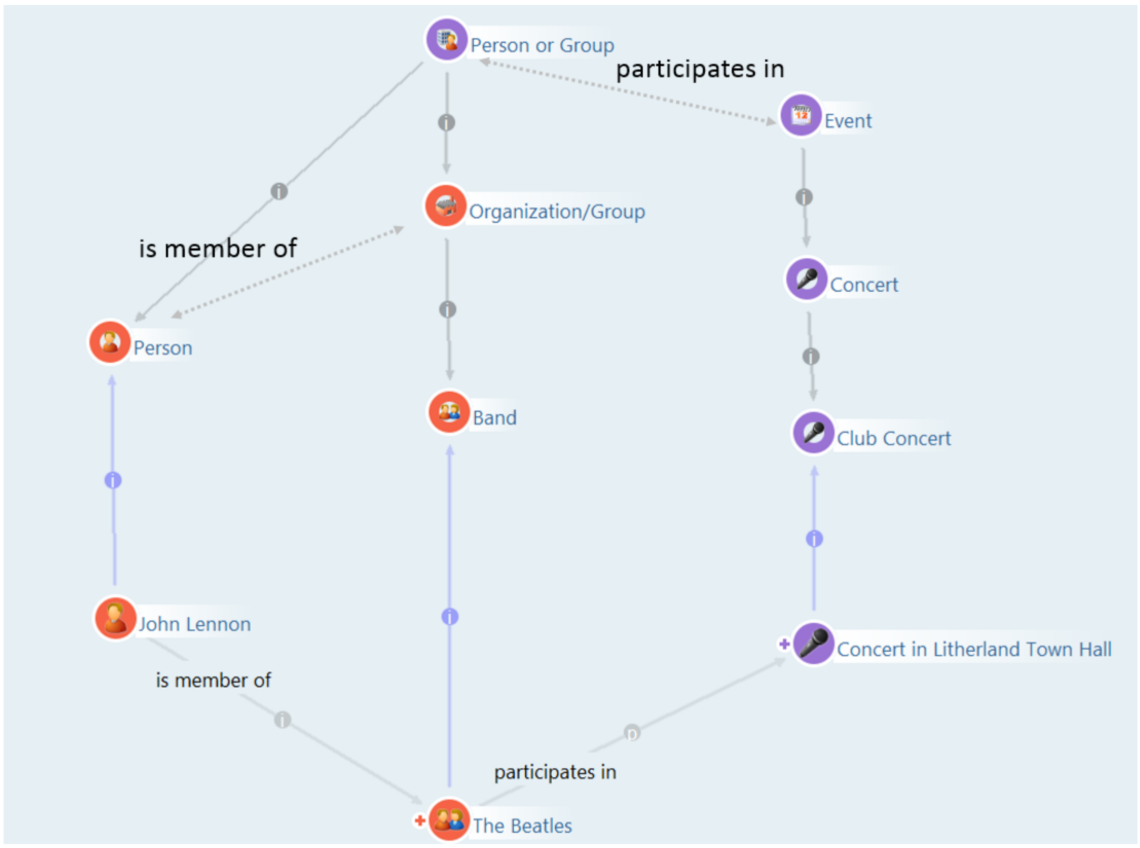


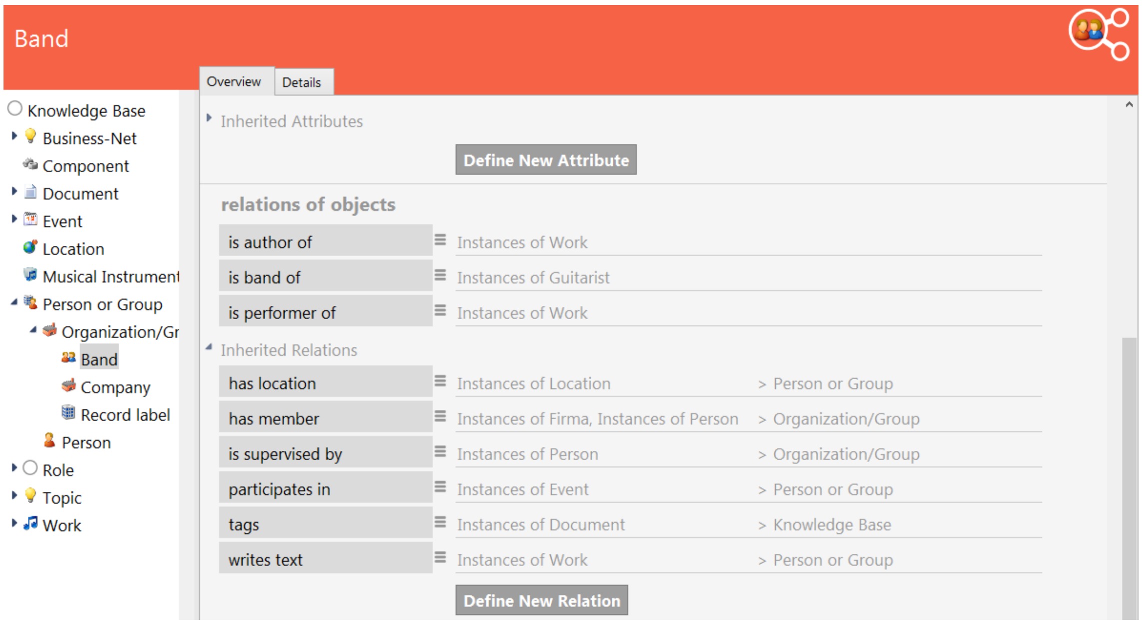
Move type **Record label** under type **Company**?

Configuring object types with properties

In the simplest case we define relations and attributes with an object type such as "band" or "person" and thus make them available for the specific objects of this type. (For example the year and location the band was established, date of birth and gender of people, location and date of events.)

If the object type for which the properties are defined has more subtypes the principle of inheritance will take effect: properties are now also available for the specific objects of the subtypes. Example: as a subtype of an organisation, a band inherits the possibility of having people as members. As a subtype of "person or band" the band inherits the possibility of taking part in events:





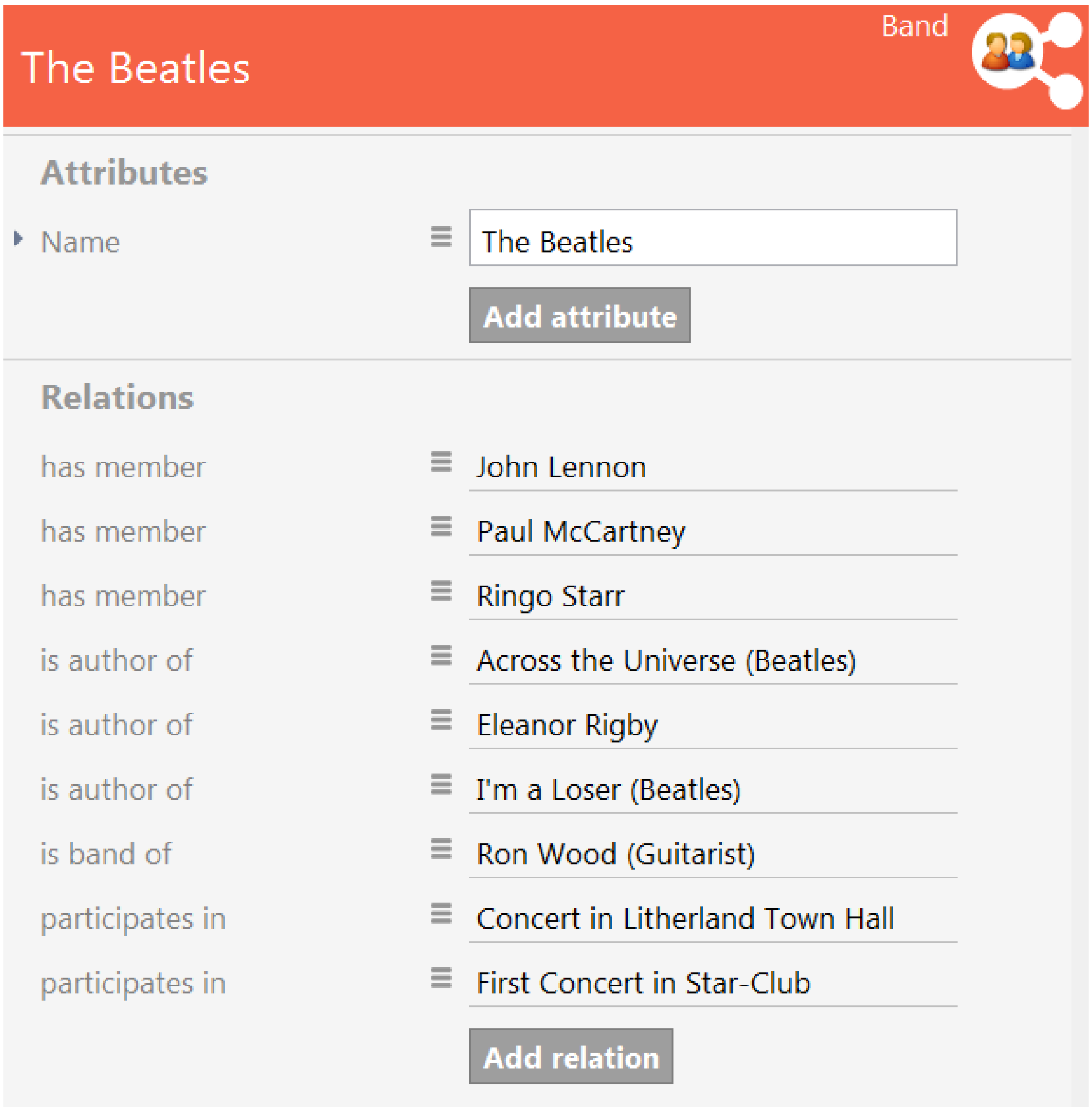
relations of objects

is author of	Instances of Work
is band of	Instances of Guitarist
is performer of	Instances of Work

Inherited Relations

has location	Instances of Location	> Person or Group
has member	Instances of Firma, Instances of Person	> Organization/Group
is supervised by	Instances of Person	> Organization/Group
participates in	Instances of Event	> Person or Group
tags	Instances of Document	> Knowledge Base
writes text	Instances of Work	> Person or Group

The editor for the object type "band" with directly defined and inherited relations there.


The Beatles

Band

Attributes

Name	The Beatles

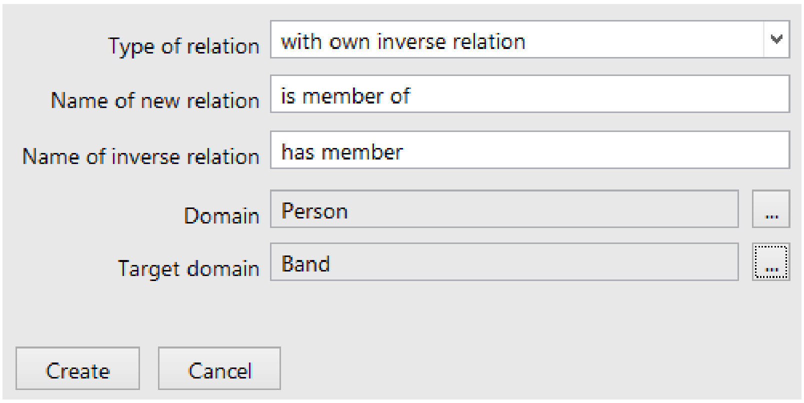
Relations

has member	John Lennon
has member	Paul McCartney
has member	Ringo Starr
is author of	Across the Universe (Beatles)
is author of	Eleanor Rigby
is author of	I'm a Loser (Beatles)
is band of	Ron Wood (Guitarist)
participates in	Concert in Litherland Town Hall
participates in	First Concert in Star-Club

With a specific object the inherited properties are available without further ado and the difference goes without notice.

Defining relations

When dealing with relations, the following basic principle governs at i-views: a relation cannot only be unidirectional. If we know of a relation for the specific person "John Lennon" to be "is a member of the band The Beatles" it then implies for the Beatles the contents "it has a member called John Lennon". These two directions cannot be separated. Therefore, i-views demands from us the types of source and target of the relations when creating new relation types - in our example that would be person and band as well as differing names: "is member of" and "has member".



Type of relation	with own inverse relation
Name of new relation	is member of
Name of inverse relation	has member
Domain	Person
Target domain	Band

Hence the relation is defined and can now be moved between objects using drag & drop.

Defining attributes

When defining new attribute types, i-views needs, above all, the technical data type as well as the name. The following technical data types are available:

Type of data	**What do the values look like?**	**Example (music network)**
Attribute	abstract attribute, without an attribute rating	
Selection	freely definable selection list	design of a music instrument (hollowbody, fretless, etc.)
Boolean	»yes« or »no«	music band still active?
Data file	random external data file which will be imported into the knowledge network as a »blob«	WAV file of a music title
Date	date dd.mm.yyyy (in the German language setting)	publication date of a recording medium
Date and time	date and time dd.mm.yyyy hh:mm:ss	start of an event, e.g. concert
Colour value	colour selection from a colour palette	
Flexible time	month, month + day, year, time, time stamp	approximate date when a member joined a band

Floating point number	numerical value with a random number of decimal places	price of an entrance ticket to an event
Integer	numerical value without decimal places	runtime of a music title in seconds
Geographical position	geographical coordinates in WGS84 format	location of an event
Band	without attribute rating, serves as a medium for meta attributes to be grouped	
Internet link	link on a URL	website of a band
Interval	date interval: interval of numbers, character string, time or date	period of time between the production of an album and its publication
Password	per attribute entity and password a clearly hashed value (Chaum-van Heijst-Pfitzmann) which is only used to validate the password	
Reference to [...]	reference to parts of the network configuration: search, diagram of a data source, scripts and files - is used for example in the REST configuration	
Character string	random sequence of alphanumeric characters	review text to a recording medium
Time	time hh:mm:ss	duration of an event

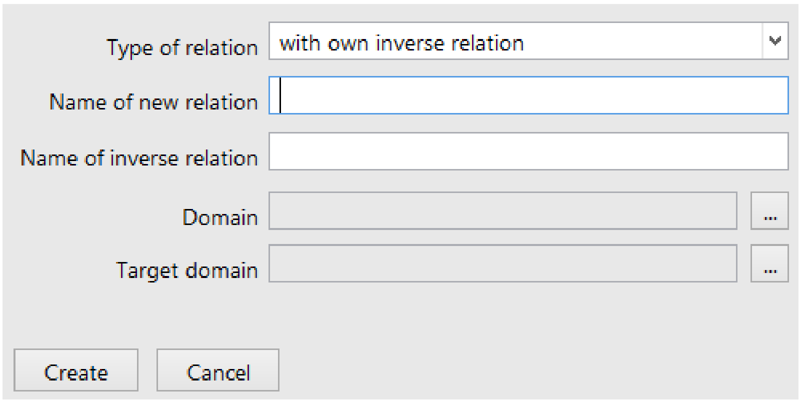
The intention of using these data types is not to define everything as character strings. Technical data types in a defined format later offer special feasibilities of inquiring and comparing. For example, numerical values may be compared to larger or smaller values within the structured queries and a proximity search can be defined for geographic coordinates, etc.

1.2.2 Relation types and attribute types

Relation types and attribute types (in brief property types) are always properties of specific objects.

1.2.2.1 Create a new relation type

Via the button "add relation" in the object editor the editor starts to create a new relation type.



Type of relation	with own inverse relation
Name of new relation	
Name of inverse relation	
Domain	
Target domain	

Editor for creating a new relation type (see also Chapter 2.1 Defining types)

Name of new relation: names for relation types may be chosen freely within i-views but should be selected under the premise of a comprehensible data model. The following convention may be of help for this: the name of the relation is phrased in such a manner that the structure [name of the source object] [relation name] [name of the target object] results in a comprehensible sentence:

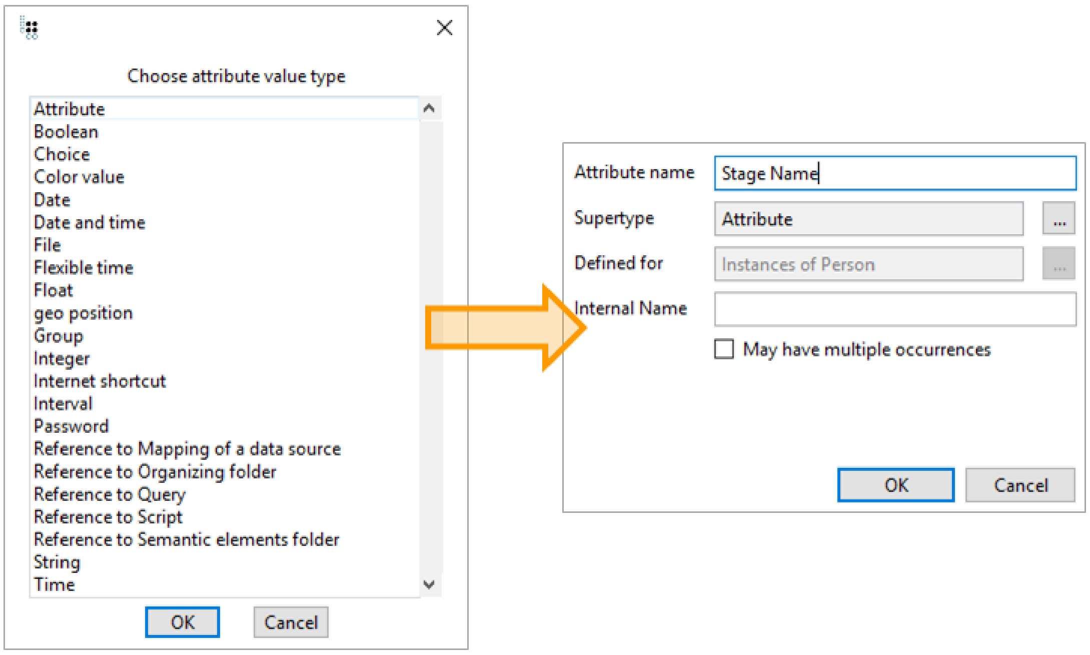
[John Lennon] [is a member of] [The Beatles]

Furthermore it is helpful when the opposite direction (inverse relation) takes on the word selection of the main direction: "has a member / is a member of".

Domain: here we define by which object types the relation has to be created: one object type forms the source of the relation and another object type the target. The tareget object type, in turn, forms the definition area of the inverse relation. To simplify matters, when creating you may only enter one object type at this stage. Afterwards, further object types may be defined in the editor for the relation type (see below).

1.2.2.2 Create a new attribute type

Via the button "define new attribute" in the object editor the editor starts to create a new attribute type:



Two-stage dialogue for creating a new attribute type

In the left-hand window the format of the attribute type is defined (date, floating point number, character string, etc.)

The following technical data types are available:

Type of data	**What do the values look like?**	**Example (music graph)**
Attribute	abstract attribute, without an attribute rating	
Boolean	»yes« or »no«	music band still active?
Choice	string values which can be selected from a drop-down menu	role; design of a music instrument (hollowbody, fretless, etc.)
Colour value	colour selection from a colour palette	
Date	date dd.mm.yyyy (in the German language setting)	publication date of a recording medium
Date and time	date and time dd.mm.yyyy hh:mm:ss	start of an event, e.g. concert

File	random external data file which will be imported into the Knowledge Graph as a »blob«	WAV file of a music title
Flexible time	month, month + day, year, time, time stamp	approximate date when a member joined a band
Float (floating point number)	numerical value with a random number of decimal places	price of an entrance ticket to an event
geo position (geographical position)	geographical coordinates in WGS84 format	location of an event
Group	without attribute rating, serves as a medium for meta attributes to be grouped	
Integer	numerical value without decimal places	runtime of a music title in seconds
Internet shortcut	link on a URL	website of a band
Interval	date interval: interval of numbers, character string, time or date	period of time between the production of an album and its publication
Password	per attribute entity and password a clearly hashed value (Chaum-van Heijst-Pfitzmann) which is only used to validate the password	
Reference to [...]	reference to parts of the Knowledge Graph configuration: search, diagram of a data source, scripts and files - is used for example in the REST configuration	
String (character string)	random sequence of alphanumeric characters	review text to a recording medium
Time	time hh:mm:ss	duration of an event

After selecting and confirming the attribute type it can be further specified with the name of the attribute in the subsequent dialogue.

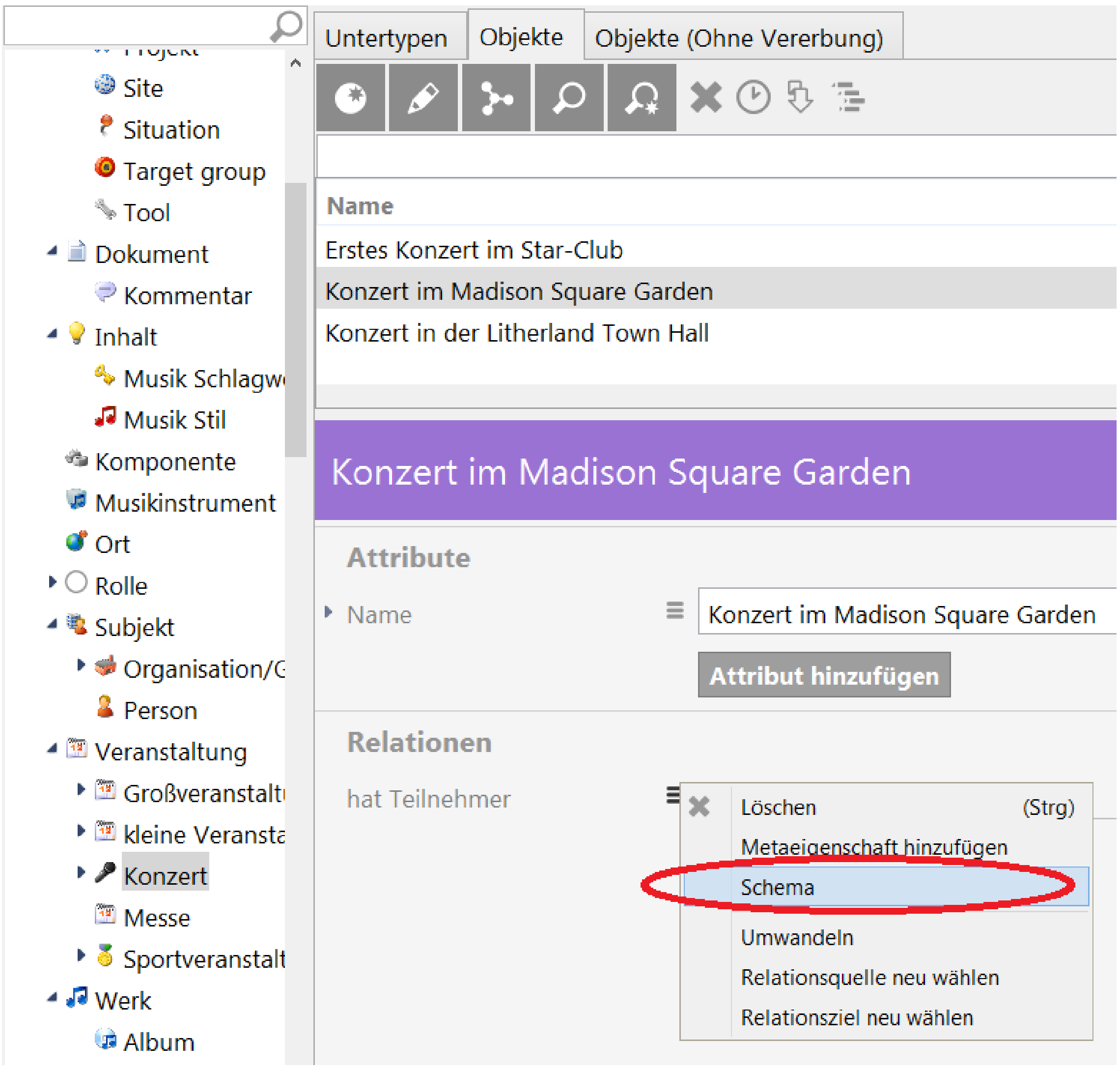
Supertype: here it is defined at what level in the hierarchy the attribute type should be placed.

May have multiple occurences: attributes may occur once or more than once, depending on the attribute type: a person only has one date of birth but may, for example, have several academic titles at the same time (e.g. doctor, professor and honorary consul).

1.2.2.3 Edit details

The dialogs for creating new attribute and relation types are limited views of the attribute and relation type editors. To edit details of relations and attributes, editors must receive and enhanced scope of functions.

You get to these two editors via the listing of relations and attributes on the “Schema” tab of the object editor:

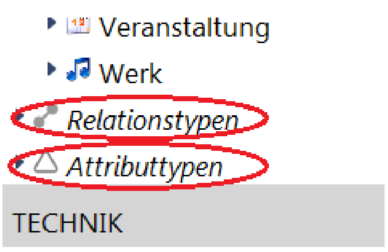


Erstes Konzert im Star-Club
Konzert im Madison Square Garden
Konzert in der Litherland Town Hall

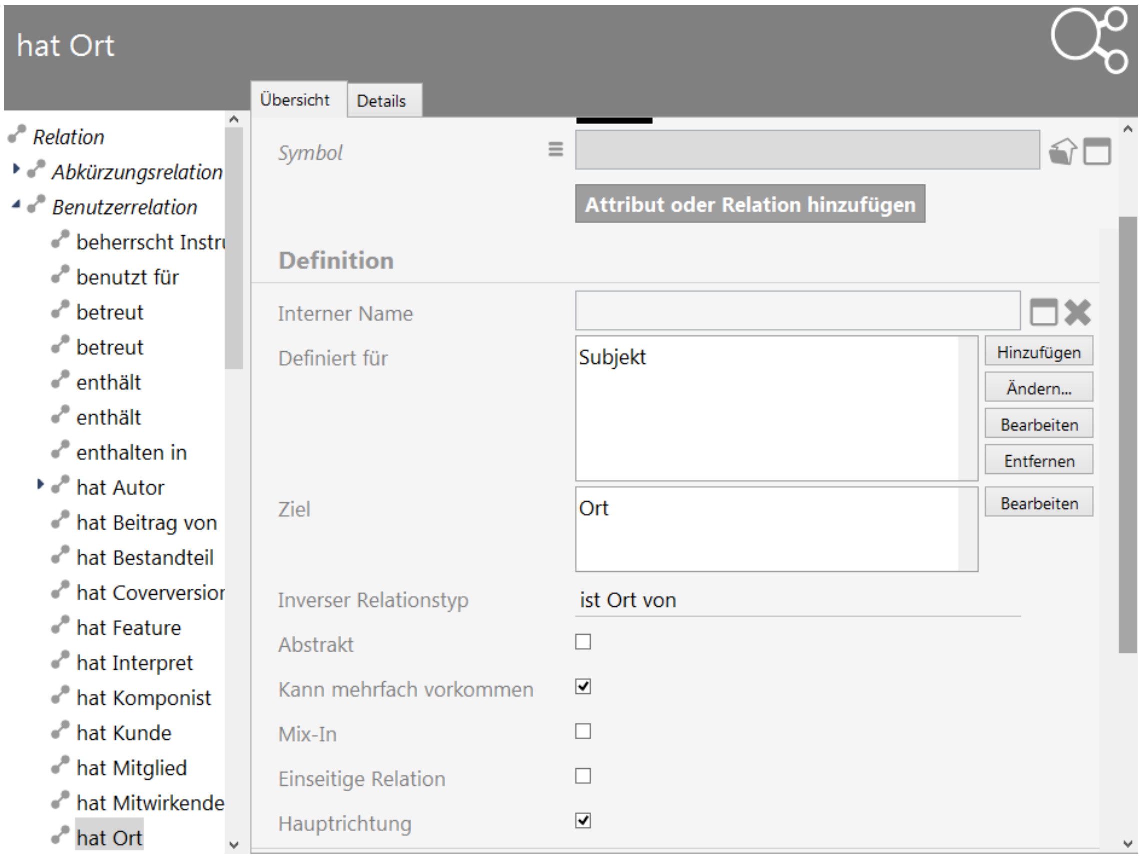
Konzert im Madison Square Garden

Name: Konzert im Madison Square Garden

Alternatively, you can use the hierarchy tree on the left side of the main window for access. The hierarchies for relation and attribute types are located underneath the object types. The editors are started by right-clicking on the relation or attribute to be edited in the context menu and choosing “Edit” .



Next, we will look at the details of the definition of properties by using the relation type editor as the example (the attribute type definition is a subset thereof):



Symbol	
Definition	
Interner Name	
Definiert für	Subjekt
Ziel	Ort
Inverser Relationstyp	ist Ort von
Abstrakt	☐
Kann mehrfach vorkommen	☑
Mix-In	☐
Einseitige Relation	☐
Hauptrichtung	☑

Defined for: Here we can subsequently check for which object types the relation can be created. Relations can be defined between several objects and thus have several sources and targets.

In this way, we can allow persons and bands to be authors of a song in the schema or assigned a location - even if they do not have a super-type in common.
We can use the “Add” button to add additional objects. We can use “Remove” to prevent this object type and all its objects from entering into this relation.

“Change” makes it possible to replace an object type. Already existing relations are then deleted by the system. If there are relations to be deleted, a confirmation prompt appears before the change is made.

Target: Here you can change retrospectively for which types of objects the relation can be used. To change the target object type you have to switch to the inverse relation type: The button for changing bears the label of the inverse relation type. After clicking on the button, the inverse relation appears in the editor and can be edited in the same way as the previous relation**.**

Abstract: If we want to define a relation which is only used for grouping but is not supposed to define concrete properties, we define it as “abstract.”

Example: If the relation “Writes song” is defined as abstract, this means: if we create songs and their relation to artists and bands, we can now enter specific information (who wrote the lyrics, who wrote the music). The unspecified relation “Writes song” cannot be created in the actual data but can only be used for queries.

May have multiple occurrences: One characteristic of relations is whether they may have several occurrences. For example: the relation “Has place of birth” can only occur once for each person whereas e.g. the relation “is member of” can occur several times for a person. Hence, logical matters can be modeled precisely. For example, musicians as persons can only have one place of birth but (at the same time) can also be members of several bands. Whether the relation can occur multiple times is specified independently for each direction of the relation: A person can only have one place of birth but the place can be the place of birth of several persons.

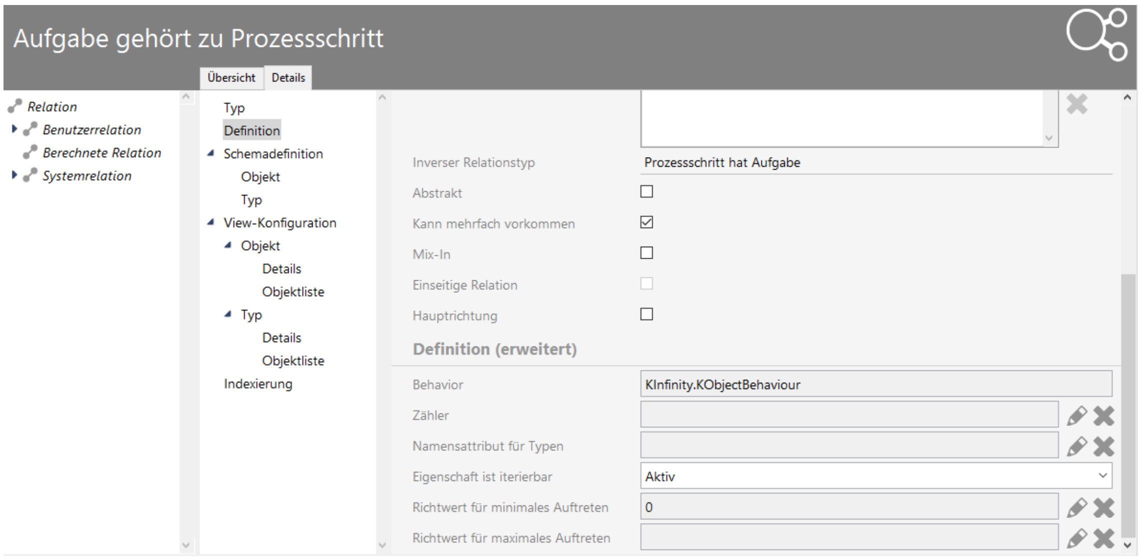
The option can only be deactivated if the relation does not occur several times in the actual data set. If it occurs several times, the system cannot decide automatically which of the relations is to be removed.

Mix-in: Mix-ins are described in the Extension chapter.

Main direction: Every relation has an opposite direction. In the core, the two directions are equivalent, but there are two places where it makes sense to determine a main direction:

- In the Graph editor: Here the relations always present themselves in the main direction in relation to the direction of the arrow and labeling; irrespective of the direction in which they were created.
- For single-sided relations (without inverse relation)

Additional setting options for relations and attributes are located in the “Definition” sub-item on the “Details” tab. The setting options under Definition are often used and that is why they are already available on the Overview tab. Under “Definition (advanced)” in contrast, there are setting options that are not required as frequently.


Aufgabe gehört zu Prozessschritt

Inverser Relationstyp	Prozessschritt hat Aufgabe
Abstrakt	☐
Kann mehrfach vorkommen	☑
Mix-In	☐
Einseitige Relation	☐
Hauptrichtung	☐

Definition (erweitert)

Behavior	KInfinity.KObjectBehaviour
Zähler	
Namensattribut für Typen	
Eigenschaft ist iterierbar	Aktiv
Richtwert für minimales Auftreten	0
Richtwert für maximales Auftreten	

Counter: If a number is entered in the counter, this is the number with which objects of this type are counted up. The JavaScript functions getCounter(), increaseCounter() and setCounter() can be used to access the counter.

Name attribute for objects: *(Note: can only be set on object types, not relation or attribute types)*
Typically many views in i-views only represent an object via its name (e.g. in object lists, hierarchies, in the Graph editor, the relation target search, etc.). Instead of the name you can use any other attribute of the objects here with which it can be represented. A prominent example for products: The article number.

Name attribute for types: This can be used also to select an alternative attribute for a more descriptive display for types.

Property can be iterated:
Selection options: Active / Write only / Inactive.
Default: Active.

Sometimes the maintenance of the index for iterating properties severely affects performance. This typically happens with meta properties such as “changed by” or “changed on” which do not necessarily have to be taken into account all the time. In such cases we recommend setting the properties to cannot be iterated by using the “Inactive” selection option. The purpose of “Write only” is to deny read access but still allow write access. This makes it possible to test for inadvertent side effects.

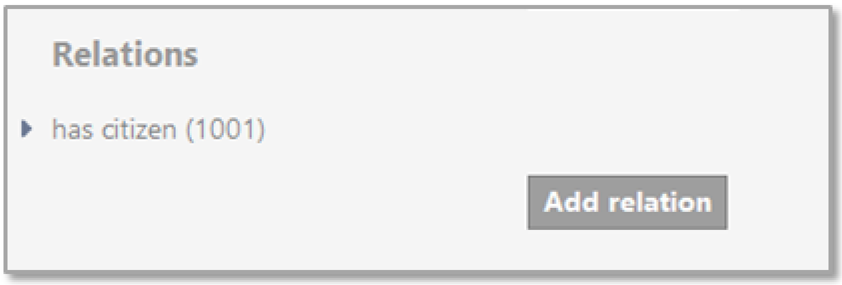
Reference value for minimum occurrence: This reference value relates to the user interface in Knowledge Builder and as of Version 5.3 it also affects the user interface in the web front-end and specifies the minimum number of times a property is supposed to occur on an object. If the number falls below the specified number, the property is displayed in red in the user interface but the object can continue to exist. An import ignores the reference value.

Reference value for maximum occurrence: As of Version 5.3, this reference value relates to the user interface in Knowledge Builder and the user interface in the web front-end. It specifies the maximum number of times the property should occur on an object. If the specified number is reached, no additional properties can be created. An import ignores the reference value.

1.2.2.4 Single-sided relations

Application of single-sided relations - basic principles

When an object is called up for import purposes or displaying in view configuration, all of its properties will be loaded (especially when not indexed sufficiently). This in turn means that besides of the attribute values, all existing relations will be loaded including their target objects as well, leading to an overhead which slows down performance.



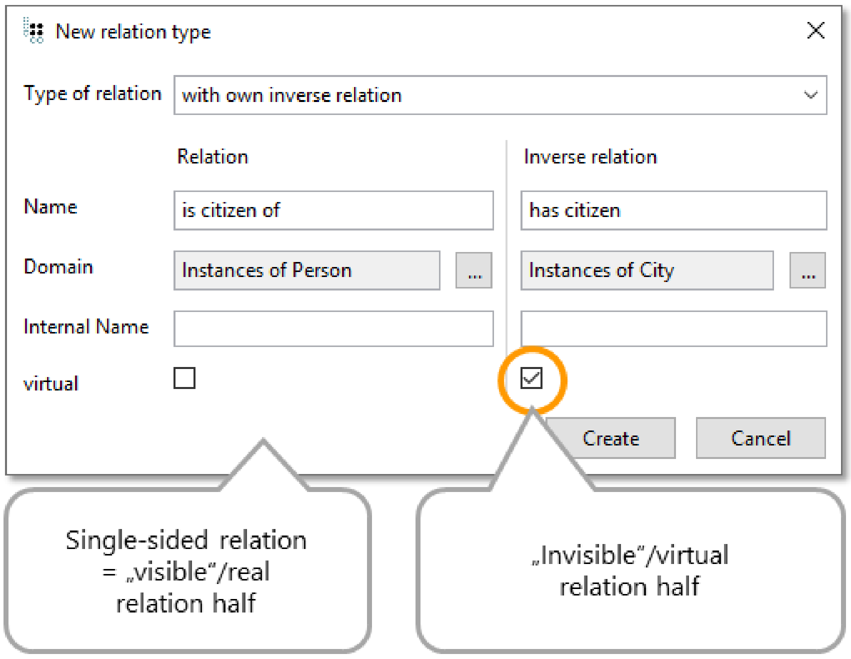
Especially for *catalog objects*, the loading of all properties can lead to long loading duration. A catalog object is an object which serves as central reference for other objects and therefore is interrelated with them.

Example: A semantic network has objects of the type "city" which are connected by relations to its citizens. When a detailed view of a city has to be loaded for indicating the number of citizens only (and not their names, addresses and hobbies etc.), single sided relations serve for this purpose.

In this case, the single-sided relations direct from the individual satellite objects towards the catalogue object. This results into the relation "is citizen of" being visible on the citizen side only, but the relation "has citizen" from the city towards the citizens will be suppressed. Nevertheless, the ’virtual’ relation "has citizen" can be used for structured queries and it can be found within the schema.

Defining single-sided relations

In order to define a single-sided relation, we must specify in the dialog which relation half (original or inverse orientation) has to be kept virtual, in other words "invisible". Here fore we choose the checkbox "virtual" on the affected half. The other relation half automatically becomes the real relation half which builds up the relationship between start domain and target domain.


New relation type
Create
Cancel
Single-sided relation = „visible"/real relation half
„Invisible"/virtual relation half

Type of relation: with own inverse relation

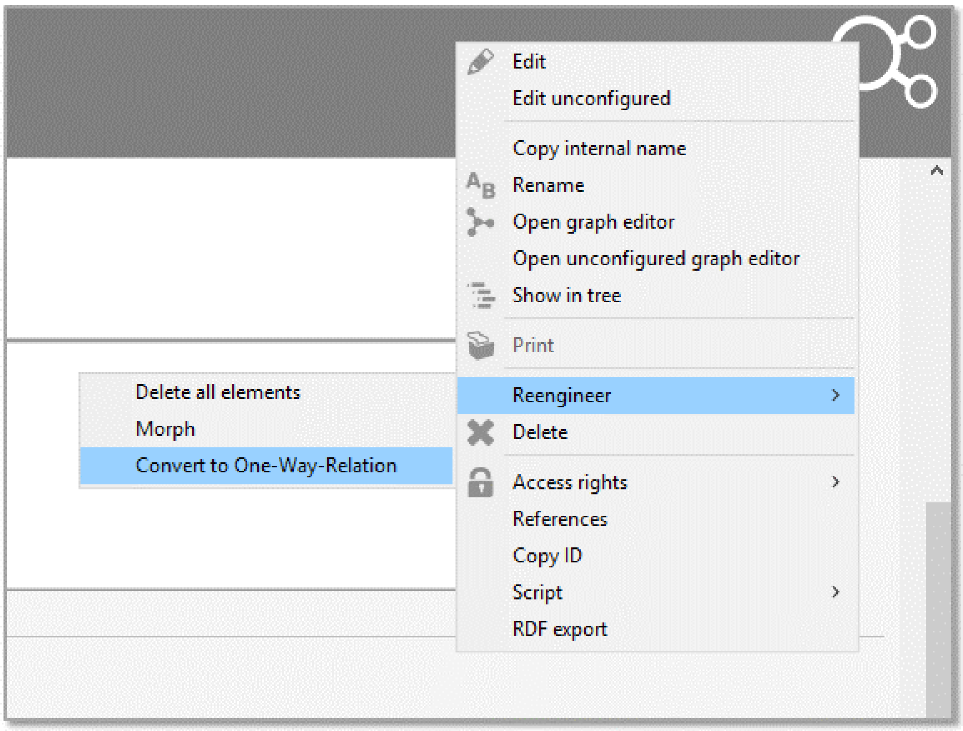
	Relation	Inverse relation
Name	is citizen of	has citizen
Domain	Instances of Person	Instances of City
Internal Name		
virtual	☐	☑

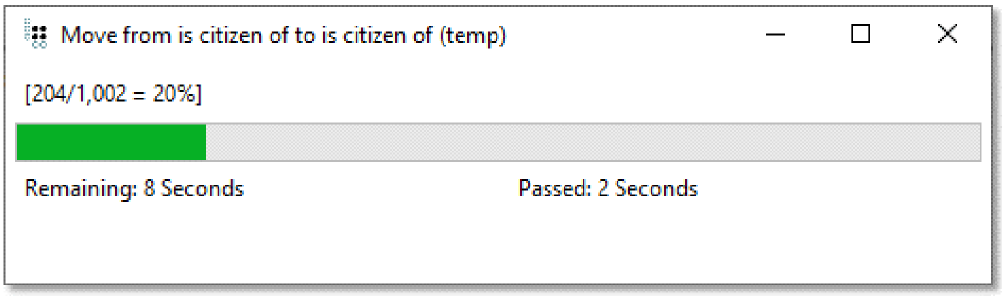
Supplementary declaration of a conventional relation as a single-sided relation

When a preliminary declared conventional relation type is going to be converted into a single-sided relation type, the instances of the virtual relation half will be deleted. This process can be inverted when redefining the relation form. Then the particular relation halves are going to be determined again.

The conversion to single-sided relations will show its effect as follows: For a catalog object, all the virtual relation halves including their relation targets are not going to be displayed anymore. But the virtual relation instances are still represented as an instance in the knowledge network and therefore can be called up in structured queries.

In the best case, when defining import mappings for large amounts of objects that relate to a catalog object, always use the real, single-sided relation type half. This can lead to performance improvement when importing.




Move from is citizen of to is citizen of (temp)

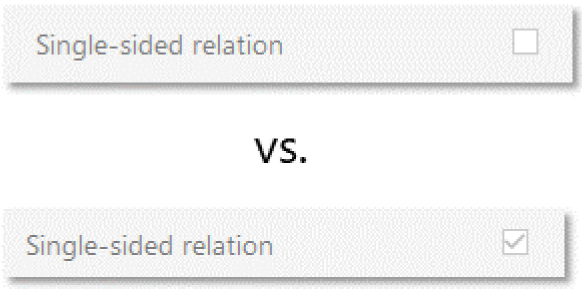
[204/1,002 = 20%]

Remaining: 8 Seconds

Passed: 2 Seconds

As a result, the checkbox "Single-sided relation" indicates that the respective relation half is used as a single-sided relation.

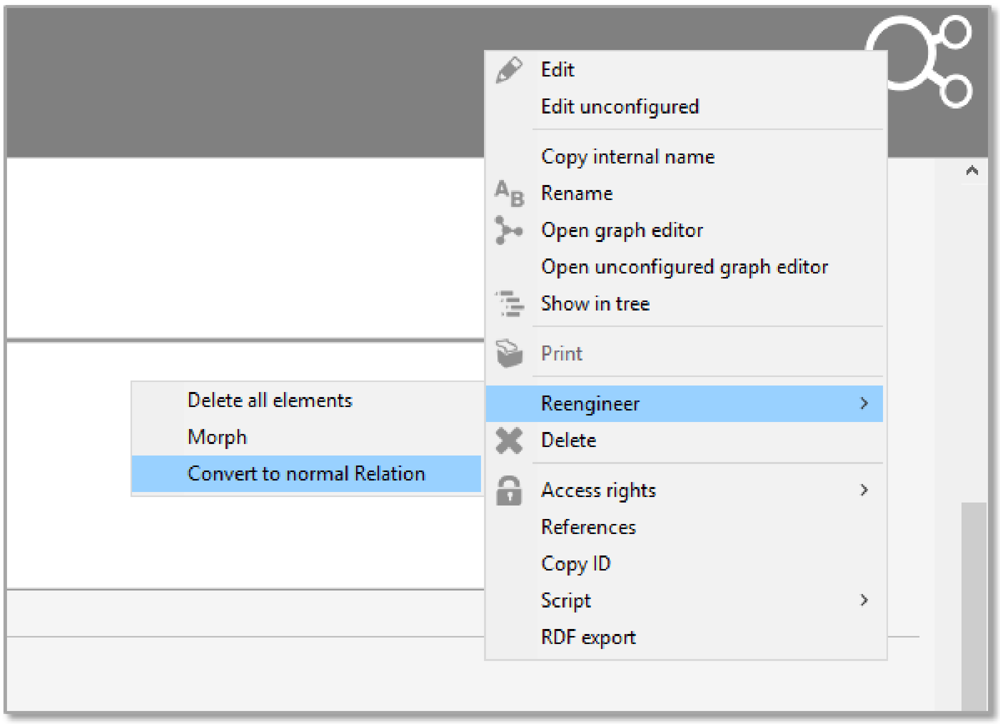
Hint: The checkbox of the Boolean attribute "Single-sided relation" only serves for indication purpose. A redefinition only can be executed via the context menu in the detail editor.



Hint: After conversion to single-sided relation, the performance for indicating **virtual** relations can be improved by means of indexing.

Supplementary conversion of a single-sided relation into a conventional relation

If we realize afterwards that a relation type actually should be declared as a conventional relation type, a correction can be made without further consequences. In the detail editor of the relation type, we therefore click onto the context menu and choose Reengineer > Convert to normal relation.



Immediately, the Knowledge-Builder changes all existing virtual and single-sided relations into normal relations.

Supplementary swapping of the orientation of a single-sided relation type

The supplementary change of orientation of the single-sided relation type is done analogous via the "Reengineer" command in the context menu of the detail editor. In order to do this, we change to the opposite relation type half which has to be converted from virtual to single-sided and choose Reengineer > Convert to one-way relation.

1.2.3 Model changes

In i-views you can make changes to the runtime of the model:

- implement new types
- make random changes to the type hierarchy (without creating tables and giving any thought to primary and secondary keys).

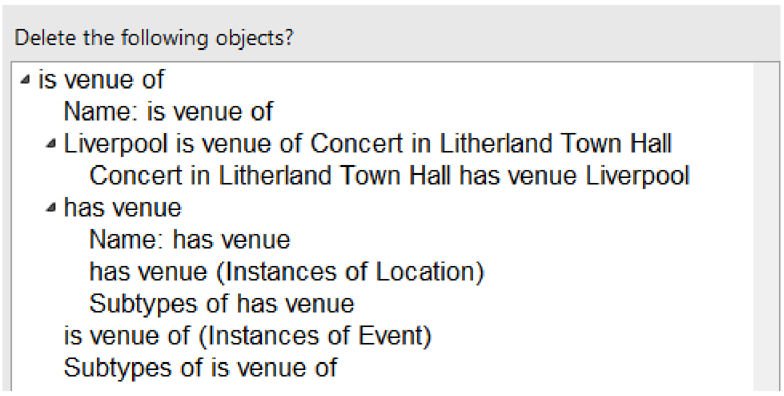
The system ensures consistency. When creating objects and properties the opposite direction of a relation is always included. Attribute values are checked as to whether they match the defined technical data type (for example, in a date field we cannot enter any random character string).

Consistency is also important when deleting: dependent elements always have to be deleted with them so that no remaining data of deleted elements stays in the network.

- Thus, when an object is deleted all its properties will be deleted along with it. If, for example, we delete the object "John Lennon" we also delete his date of birth and his biography text which we can have as a free text attribute for each person, etc. Likewise, his relation "is member of" to the Beatles and "is together with" to Yoko Ono. The objects "The Beatles" and "Yoko Ono" will not be deleted; they only lose their link to John Lennon.
- When deleting a relation the opposite direction is automatically deleted with it.

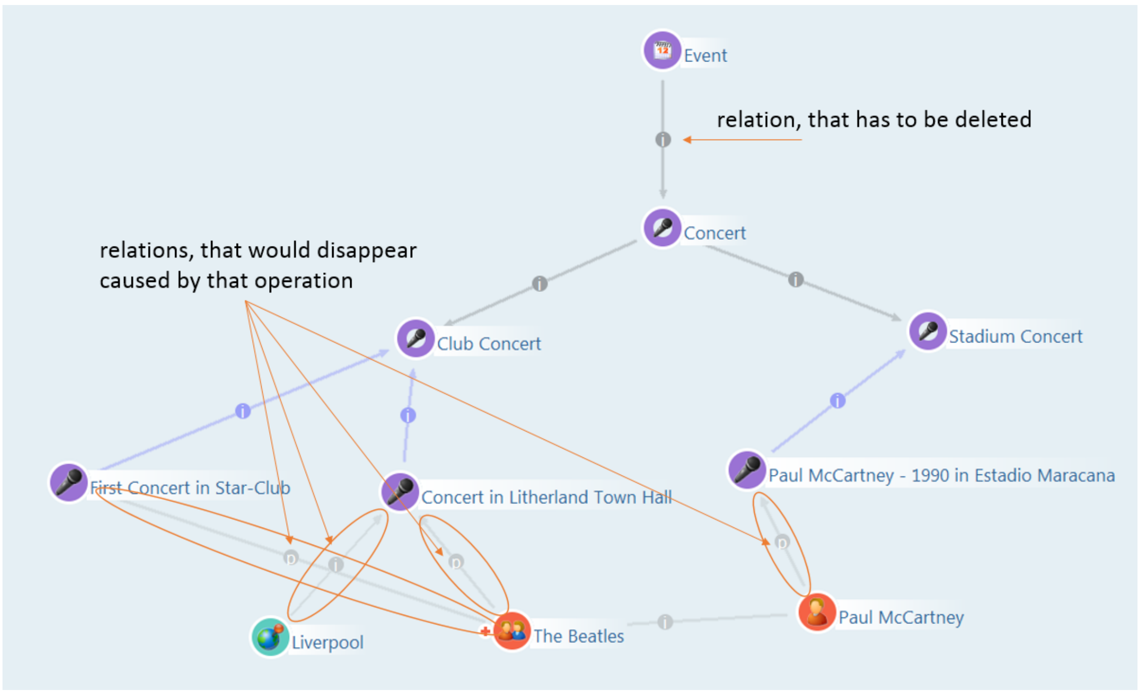
Since i-views always ensures that the objects and properties are in accordance with the model, deleting an object type or, where necessary, an operation has far-reaching consequences: when an object type is deleted, all its specific objects are also deleted - analogue to the relation and attribute types.

In this process, i-views always provides information on the consequences of an operation. If an object has to be deleted, i-views lists all properties which will thus be removed in the confirmation dialogue of the delete operation:


Delete the following objects?
- is venue of
  - Name: is venue of
  - Liverpool is venue of Concert in Litherland Town Hall
    - Concert in Litherland Town Hall has venue Liverpool
  - has venue
    - Name: has venue
    - has venue (Instances of Location)
    - Subtypes of has venue
  - is venue of (Instances of Event)
  - Subtypes of is venue of

i-views controls where, by the change, objects, relations or attributes become lost. The user is made aware of the consequences of the deletion.

Not only the deletion, but also conversion or change of the hierarchy type may have its consequences. For example, when objects have properties which no longer comply with the model after a change in type or change in the inheritance.


Event
relation, that has to be deleted
Concert
relations, that would disappear caused by that operation
Club Concert
Stadium Concert
First Concert in Star-Club
Concert in Litherland Town Hall
Paul McCartney - 1990 in Estadio Maracana
Liverpool
The Beatles
Paul McCartney

Let us assume that we delete the relation "is supertype of" between "event" and "concert" and thus remove the object type "concert" and all its subtypes from the inheritance hierarchy of event to add them to "work", for example. In this case, i-views draws our attention to the fact that the "has participants" relations of the specific concerts would be omitted. This relation is defined in "event" and would thus no longer apply to the concerts.

There are possibilities for preventing the omission of relations as a result of model changes. If an object type has to move within the type hierarchy, for example, the model of the affected relation has to be adapted prior to this.

For example, if "concert" is to be located under "work" within the hierarchy and no longer under "event". To this end, the relation "has participants" will be assigned to a second source: that can be either the object type concert itself or the new item "work". The relation will hence not be lost.

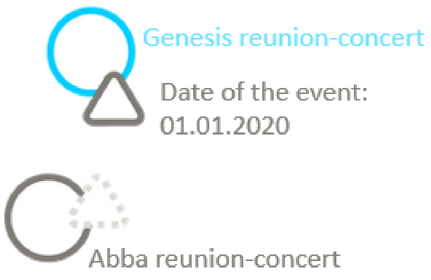
i-views pays particular attention to the type hierarchy. If we delete a type from the middle of the hierarchy or remove a super/sub relation type, i-views then closes the gap which has ensued and puts back the types which have lost their supertypes into the type hierarchy to the extent that they keep its properties as far as possible.

Special functions

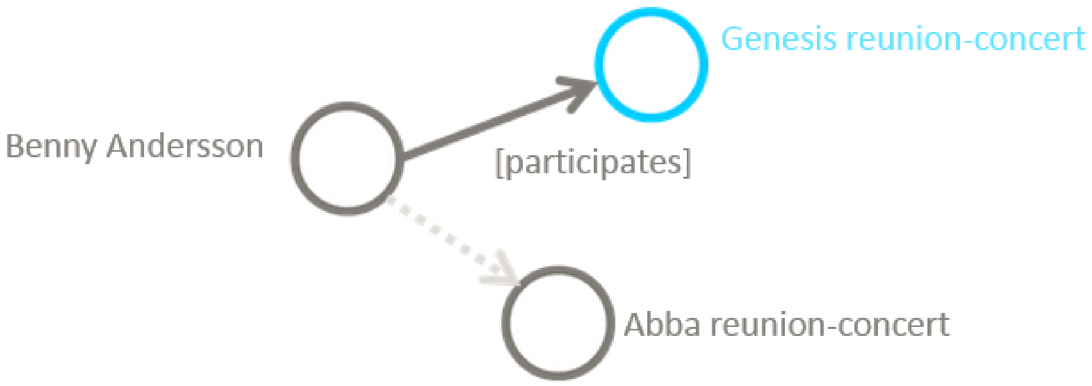
Changing type: objects already in the knowledge network may be moved to objects of another type. For example, if the object type "event" differentiates to "sports event" and "concert". If there are already objects of the type sports event or concert in the knowledge network, they may be selected from the list in the main window and quite simply moved to a new, more suitable object type using drag & drop.

Alternatively, we can find more information in the context menu under the item "edit".

Select type: using this operation we can assign a property to an object.



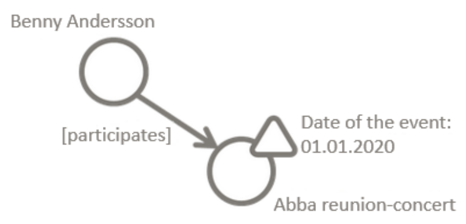
Reselect relation target: in relations this does not only apply to the source, but also the relation target.

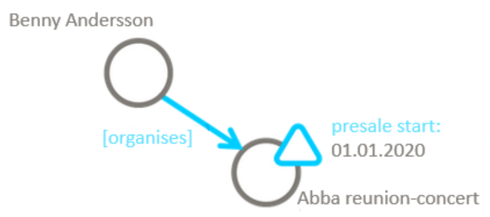


Convert subtypes to specific objects (and vice versa): the border between object types and specific objects is, in many cases, obvious but not always. Instead of setting up only one object type called "musical direction" as in the case of our sample project, we could have set up an entire type hierarchy of musical directions (we decided against this in this network because the musical directions classify so many different things such as bands, albums and songs and therefore they do not provide any good types). It may happen, however, that we change our minds in the middle of the modelling. For this reason, there is the possibility of changing subtypes into specific objects and specific objects into subtypes. Any relations which may already exist will be lost in the process if they do not match the new model.

Converting the relation: source and target of the relation will remain the same, only the relation type will be converted.

Converting the attribute: source/object will remain the same but it will be assigned to another attribute type:





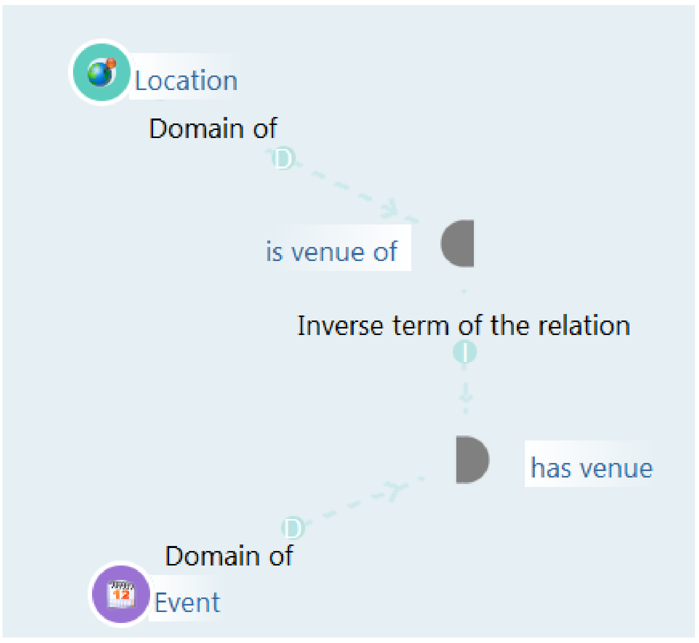
When converting the individual relations we are usually quicker when we delete these and replace them with another one. However, it may happen that meta properties are attached to the properties which we do not want to lose. On the other hand, the converting operations are also available for all properties of a type or a selection thereof. A prerequisite is, of course, that the new relation or attribute type is also defined for the source and target objects.

If changes are made to the model, consideration should always be given to the fact that restoring a previous condition may only be carried out by installing a backup. Analogue to the related databases there is no "reverse" function.

1.2.4 Representation of schema in the graph editor

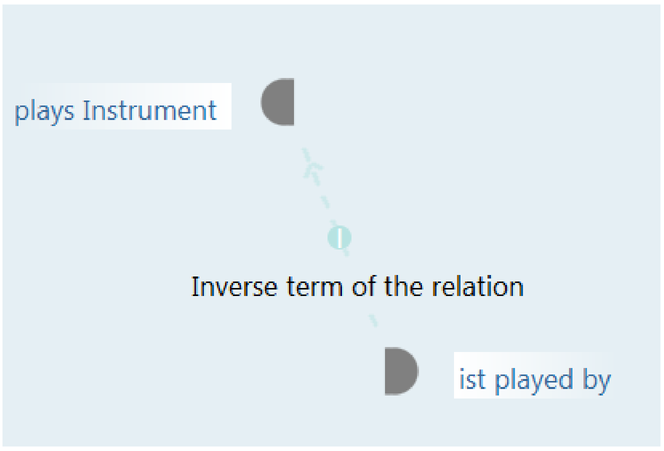
Until now we have mainly been dealing with linking of specific objects within the graph editor. Presenting such specific examples, discussing them with others and, where necessary, editing them is also the main function of the graph editor. We can, however, also present the model of the semantic network directly using the graph editor, e.g. the type of hierarchy of a network.

Types of objects will then be displayed as nodes with a coloured background and types of relations as a dotted line:

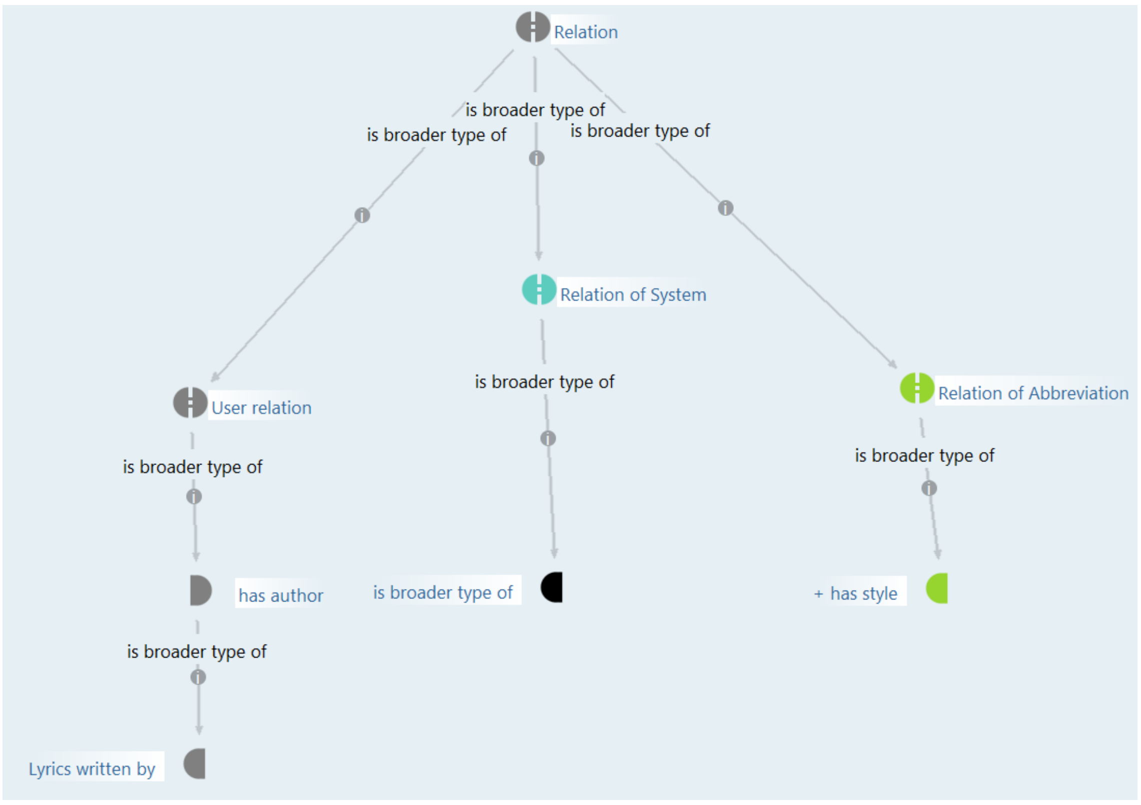


Relation types in the graph editor

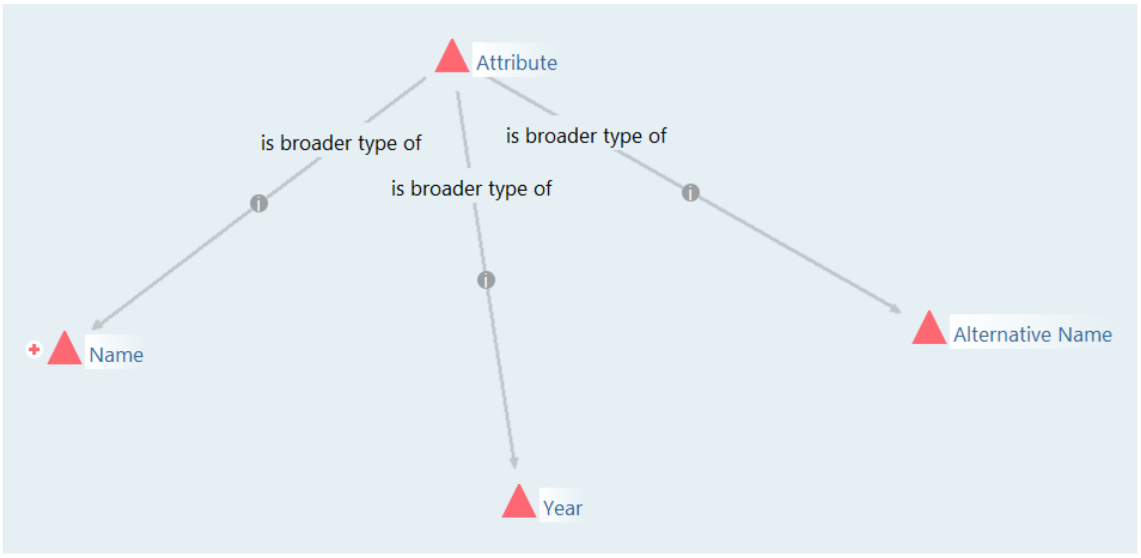
If until now we have been referring to relations in the graph editor, this concerned relation objects between specific objects of the knowledge network. Moreover, the general types of relations (hence the diagrams of the relations) may also be presented in the graph editor. A relation is depicted in the graph editor as two semi-circles which represent the two directions (main direction and inverse direction). Therefore, between these two nodes there is the relation "inverse type of relation":



The presentation of a type of relation and the hierarchy within the graph editor may be shown analogue to the object editor with all supertypes and subtypes:



Attribute types may also be depicted in the graph editor - they are shown as triangular nodes.



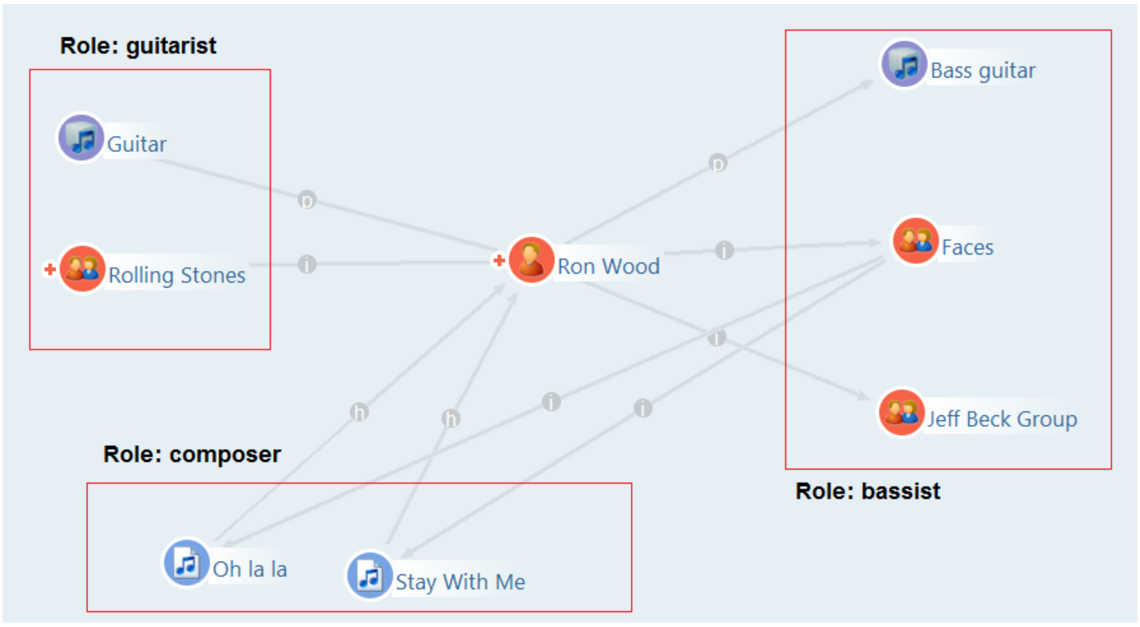
Analogue to the type of object hierarchy the hierarchy of the relations and attributes within the graph editor may be changed by deleting and dragging the supertype relation.

1.2.5 Metamodeling and advanced constructs

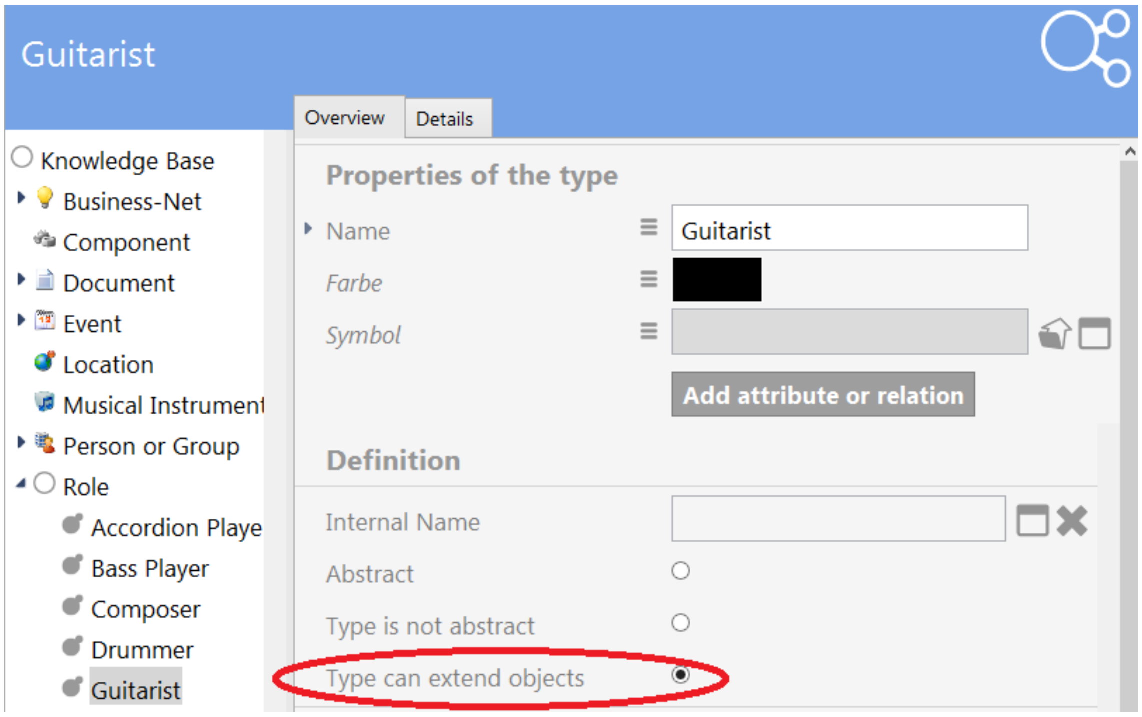
1.2.5.1 Enhancements

As a further means of modelling, i-views offers the possibility of enhancing objects.

For example, if a person performs the role of a guitarist in a band but plays another kind of instrument in another band. In addition, the person exercises the role of the composer.



The fact that one person can play different roles in a knowledge network may be regulated via a special form of a object type. This may not contain any objects, but enhance objects from another object type (e.g. in this case "person"). For this purpose, the object type "role" is implemented into the knowledge network, for example and the different roles created for persons as subtypes: guitarist, composer, singer, bassist, etc. In order that these "role object types" may enhance objects this function will be defined in the editor for the object type by checking the box "type can extend objects":



Guitarist

Overview | Details

Knowledge Base
- Business-Net
- Component
- Document
- Event
- Location
- Musical Instrument
- Person or Group
- Role
  - Accordion Playe
  - Bass Player
  - Composer
  - Drummer
  - Guitarist

Properties of the type

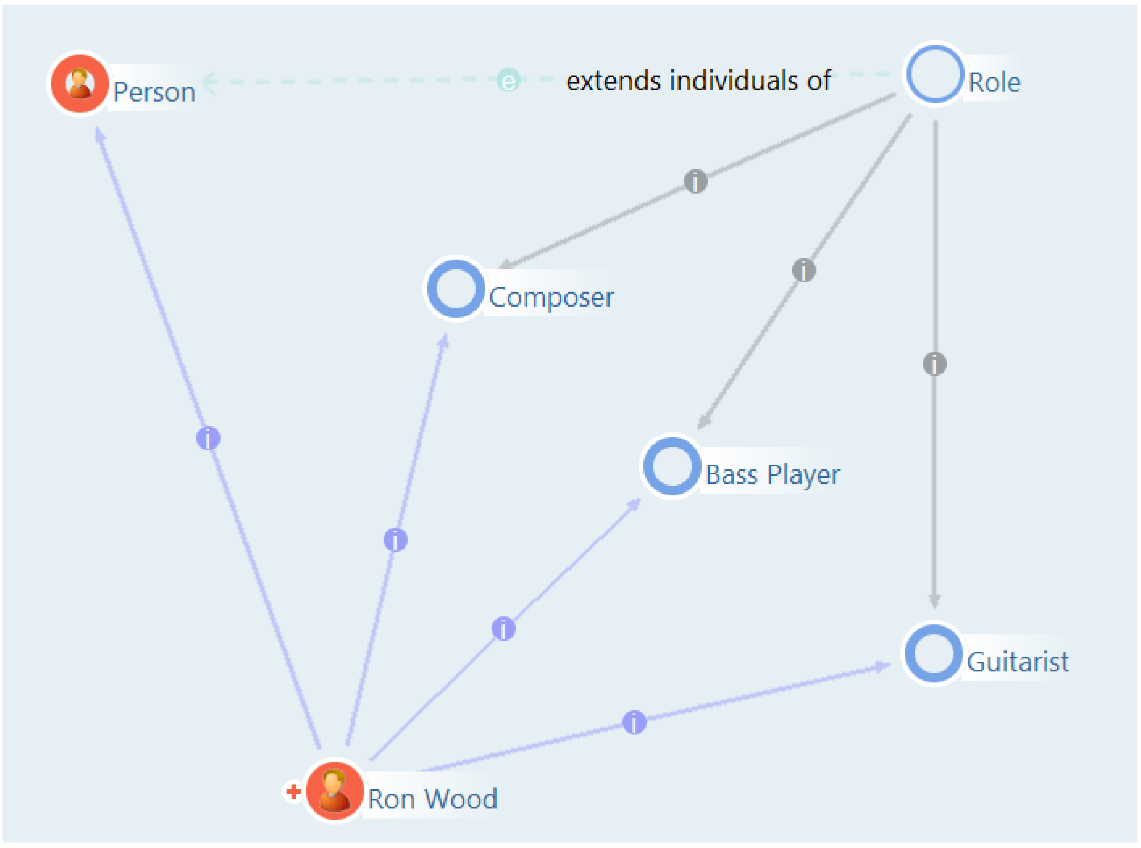
Name	Guitarist
Farbe	■
Symbol	

Add attribute or relation

Definition

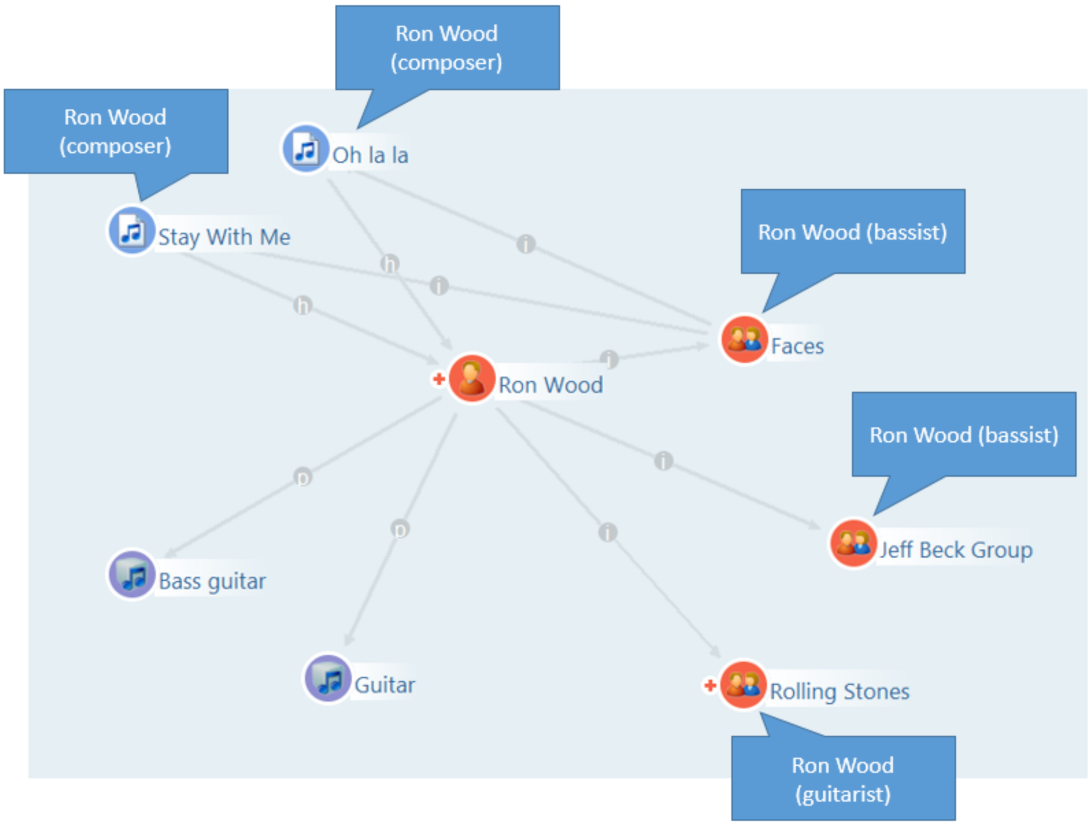
Internal Name	
Abstract	○
Type is not abstract	○
Type can extend objects	●

Enhancements are displayed in the graph editor as a blue dotted line:



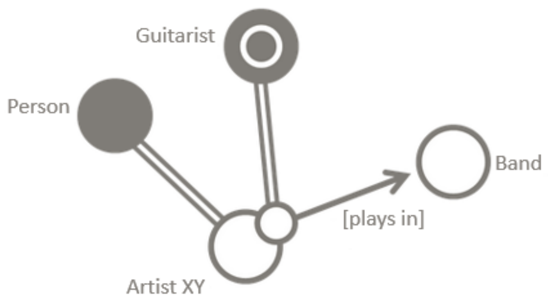
As a result of this enhancement we have achieved several things simultaneously:

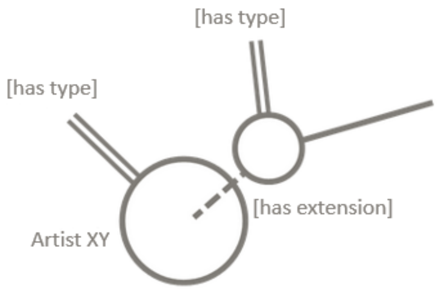
- We have formed sub objects for the persons (we can also imagine these as sections or - with persons - as roles). These sub objects may be viewed and queried individually. They are not independent, when the person is deleted the enhancement "guitarist" along with the relations to the bands or titles are gone.
- We have expressed a multi-digit content. We cannot express anything on separate relations between persons, instruments, title/band - in this case the assignment would no longer succeed.



For this purpose the relation "plays in the band" for the enhancement "guitarist" has to be defined. This effect that persons inherit an additional model via the enhancement may be helpful regardless of multi-digital contents.

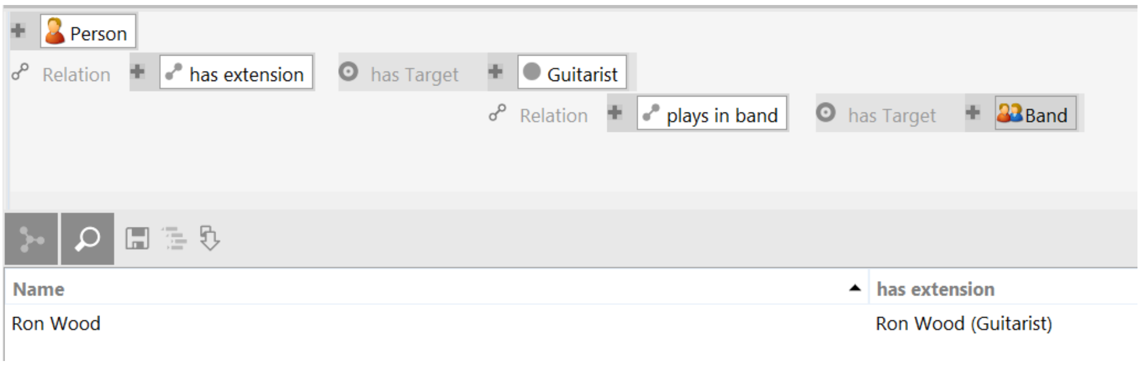
From a technical point of view, the enhancement is an independent object which is linked to the core individual by means of the system relation "has enhancement" or inverse "enhanced individual". Its type (system relation "has a type") forms the enhancement type.





When defining a new enhancement, two object types play a role: in our example we want to give persons an enhancement and we have to provide this information to your type "person". The enhancement itself again has an object type (usually even quite a lot of object types); in our case "guitarist". With the type "guitarist" (and with all others with which we want to enhance the persons) his specific objects will be dependent.

When querying enhancements in the structure search we have to traverse individual relations. From the specific person via the relation "has extension" via the enhancement object "Guitarist". From there you can reach the band via the relation "plays in band".



Name	has extension
Ron Wood	Ron Wood (Guitarist)

Mix-in

The essence of this example with the role "guitarist" is that the relation "plays in a band" is linked to the enhancement but not with the person. Hence, a consistent assignment is possible with several instruments and several bands.

If the option mix-in is selected the relation, on the other hand, is created with the core object (person) itself. The reason for this is that enhancements are sometimes not used to express more complex contents but to assign an object polyhierarchically to different types. This object inherits in this manner relations and attributes of several types.

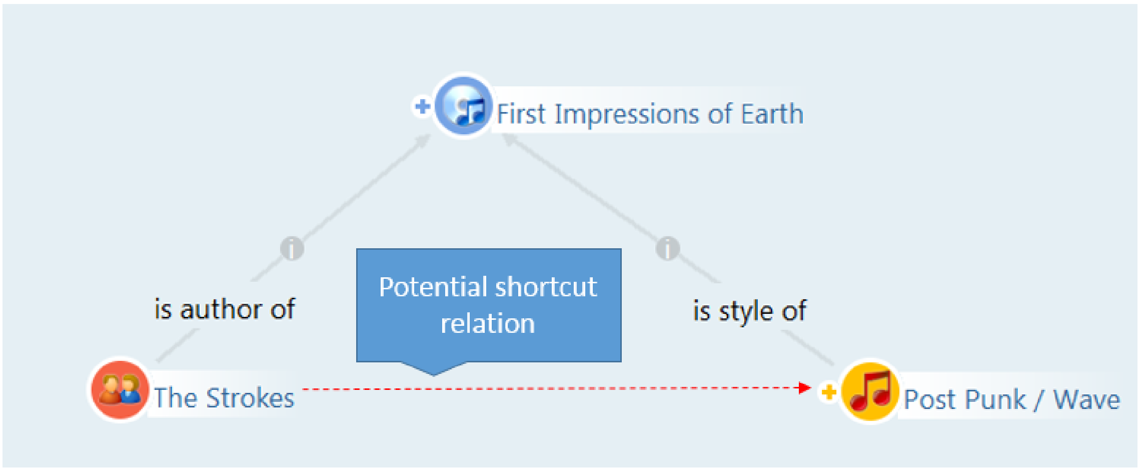
When we setup an extensive type hierarchy of events, for example, with the subdivision into large and small events, outdoor and indoor events, sports and cultural events, we can either characterise all combinations (large outdoor concert, small indoor football tournament, etc.) or create the different types of events as possible enhancements of the objects of the type "event". Then we can assign an event via its enhancements as a football tournament and, at the same time, as an outdoor event as well as a large event. Via the enhancement "football tournament" the relation "participating team" may then be inherited, via the enhancement "outdoor event", for example, still the property "floodlight available". When we have placed these properties in mix-in they may be queried like direct properties in the events.

If a mix-in enhancement is deleted it acts like a "normal" enhancement: there has to be at least one enhancement available which entails the mix-in property. When the last of these enhancements is deleted the relation or the attribute in the core object is also deleted.

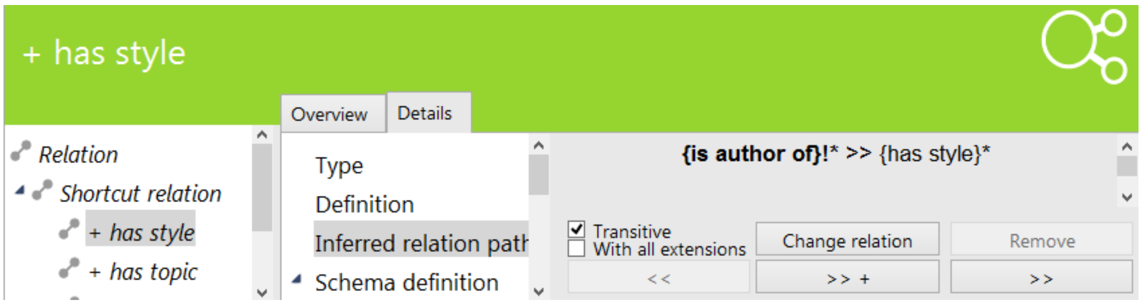
1.2.5.2 Shortcut relations

A special form of the relation is the shortcut relation. Hidden behind this is the possibility to shorten several relations already available and defined and which are present in the semantic graph database in a connected row by means of a suitable relation. In this manner the system can, to a certain extent, draw a direct conclusion from A to B from an object A in the semantic graph database which is connected to an object B via several nodes.

For example, a band publishes a recording media in a certain style of music, ergo this style of music can likewise be assigned to this band:



In the form editor the inferred relation path is defined via the relations "is author of" and "has style".



{is author of}!* >> {has style}*

[✓ Transitive]
With all extensions

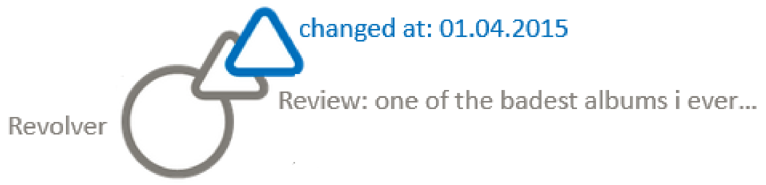
In the queries the shortcut relation can be used like any other relation as well.

In the current version of i-views it is recommended that several nodes and edges be queried via search modules as a result of the improved overview in the structured queries.

1.2.5.3 Meta properties

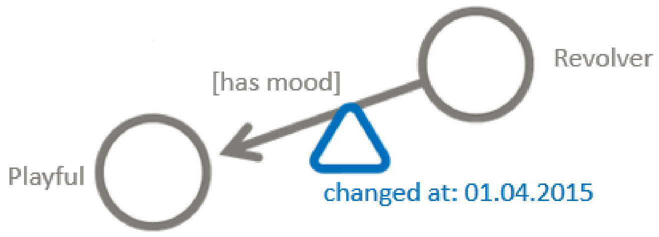
In Chapter 2.1 properties of a lesser complexity in object types for objects were defined. For example, users can add or edit contents to the music knowledge network which we are treating here as an example via a web application. It should, however, be noted which information was changed from whom and when. To do this, attributes and relations and, in turn, for attributes and relations are required in all combinations.

Attributes to attributes: for example, discussions and reviews are listed in the music knowledge network as text attributes for music albums. If it is to be noted when discussions and reviews were added or when they were last changed we can define a date attribute which is assigned to the discussion and review attributes:

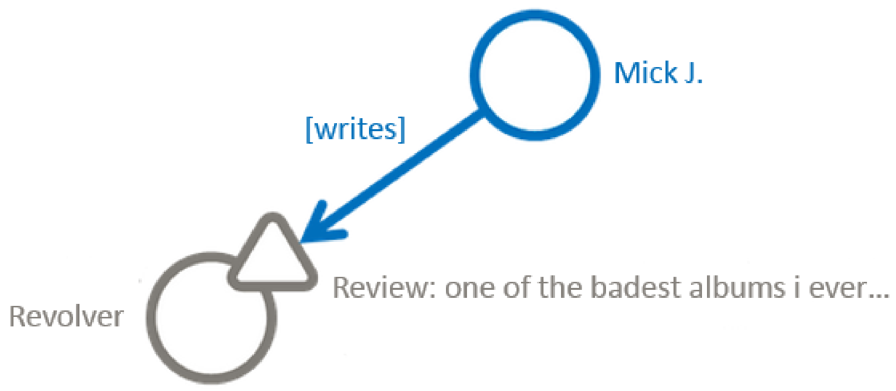


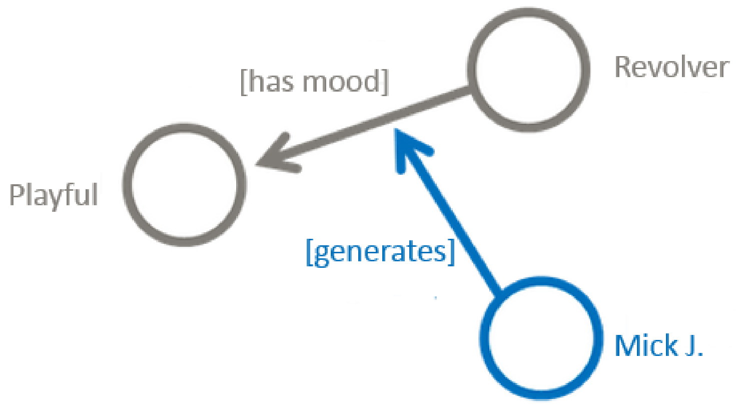
Attributes to relations:

This date attribute may also be located at a relation between albums and personal sentiments such as "moods" if the users are given the possibility of tagging:



Relations may be used on attributes and on relations. For example, those users should be documented who have created or changed an attribute (e.g. review of an album) or a relation between an album and a mood at certain times:





These examples together with the editing information form a clearly demarcated meta level. Properties of properties are, however, usable for complex "primary information":

If, for example, the assignment of bands or titles to the genres be weighted, a rating as "weight" may be given to the relation as an attribute.

An attribute of a relation may also be the sum of a transfer or the duration of participation or membership.

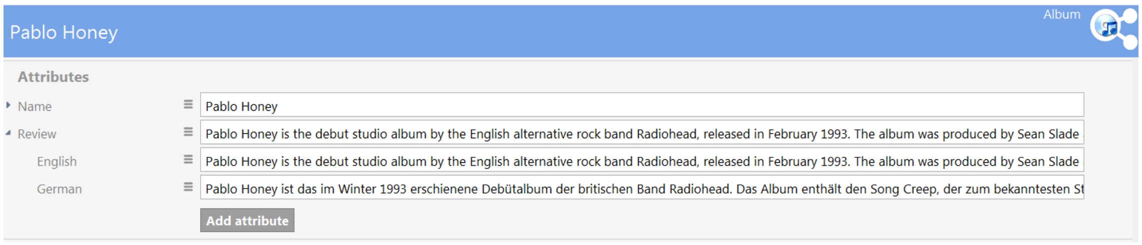
Relations to relations may also be expressed as "multi-digit contents". For example, the fact that a band performs at a festival (that is a relation) and in doing so takes a guest musician with them. He doesn’t always play with the band and hence doesn’t have a direct relation to it. Likewise, he cannot be generally assigned to the festival but is assigned to the performance relation.

Modelling of meta properties may, of course, also be realised by implementing additional objects. In the last example the fact that the band performed at a festival enabled an object of the type "performance" to be modelled. A significant difference is that in the meta model the primary information can simply be separated from the meta level: the graph editor does not show the meta information until it is requested and in queries, also in the definition of views the meta information can simply be left out. The second difference lies in the delete behaviour: objects are viable independently. Properties, even meta properties, are not on the other hand; when primary objects and their properties are deleted the meta properties are deleted with them.

Incidentally: properties can not only be defined for specific objects but also for the types themselves. A typical example of this is an extensive written definition with a object type, e.g. "what do we understand by a company?" That is why we are always asked whether we want to create them for concrete objects or subtypes when creating new properties.

1.2.5.4 Multilingualism

The attributes "character string", "data file attribute" and" selection" may be created multilingually. In the case of the character string attribute and data files, several character strings may then be entered for an attribute:



Pablo Honey

Album

Attributes

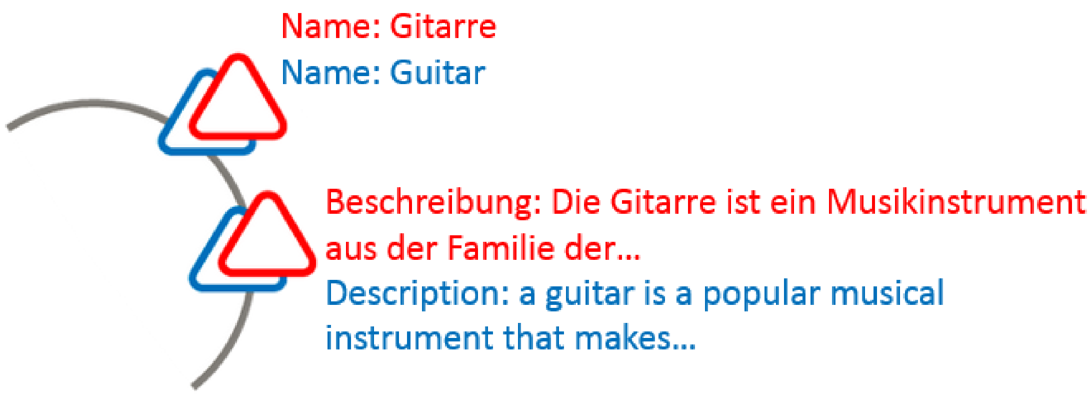
Name	Pablo Honey
Review	Pablo Honey is the debut studio album by the English alternative rock band Radiohead, released in February 1993. The album was produced by Sean Slade
English	Pablo Honey is the debut studio album by the English alternative rock band Radiohead, released in February 1993. The album was produced by Sean Slade
German	Pablo Honey ist das im Winter 1993 erschienene Debütalbum der britischen Band Radiohead. Das Album enthält den Song Creep, der zum bekanntesten St

With data file attributes several images (e.g. with labels in other languages) may be uploaded analogically. In the case of selection attributes all selection options are deposited in the attribute definition; here it doesn't matter in which language the selection for the specific object is made.

All other attributes are depicted in the same manner in all languages, e.g. Boolean attributes, integers or URLs.

If the image deviates in other languages attributes adapt their image automatically, depending on the language: for example, dates according to European spelling day|month|year are shown in US format month|day|year.

In i-views separate attributes are not simply deposited for values in other languages, instead they remain as a separate layer for an attribute with language variations. You don't have to bother about the management of different languages when developing an application, but only the desired language for the respective query:


Name: Gitarre
Name: Guitar
Beschreibung: Die Gitarre ist ein Musikinstrument aus der Familie der...
Description: a guitar is a popular musical instrument that makes...

In i-views preferred alternative languages can be defined: if there is no attribute value, e.g. a descriptive text in the queried language the missing text can be shown in other languages if they are available. The order of sequence of the alternative languages may also be defined.

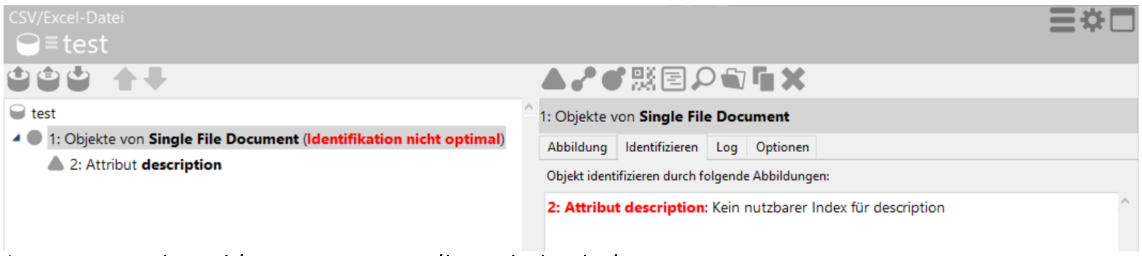
Multilingual settings are, for example, used in search.

1.2.6 Indexing

Indexing forms part of the internal data management of databases. Used correctly, the setting of indexes can improve performance significantly.

Background: In i-views, all semantic elements (types or objects) are generally stored in a cluster with their properties (attributes or relation halves). For certain transactions or uses, however, it can be better to only load part of the information. Instead of having to load the entire elements or clusters to read a few properties for queries, a corresponding index is used to refer exclusively to the required properties. Metaphorically, these indexes are both signposts and shortcuts to the required partial information.

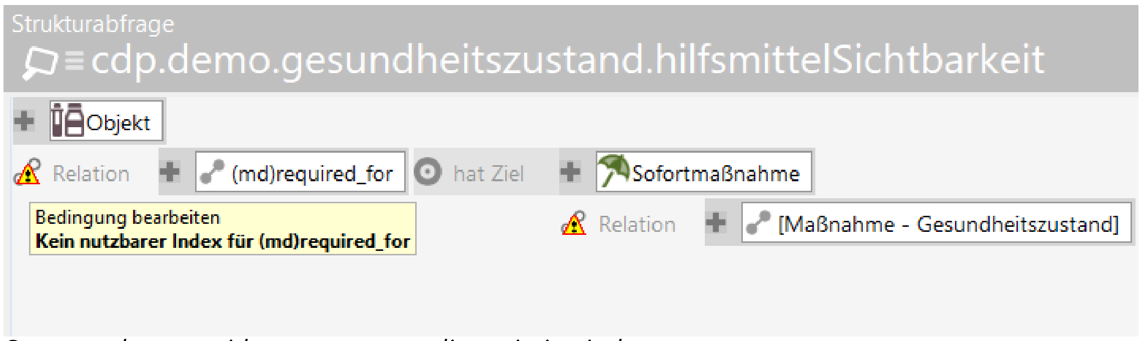
The requirement for indexing in structured queries or during import mapping becomes apparent through various notes: In import mapping, if an object is not identified using the primary name, as expected, but through a different attribute, the note appears: "No usable index for [...]."


CSV/Excel-Datei
test

test
1: Objekte von **Single File Document** (**Identifikation nicht optimal**)
2: Attribut **description**

1: Objekte von **Single File Document**
Abbildung | Identifizieren | Log | Optionen
Objekt identifizieren durch folgende Abbildungen:
2: Attribut description: Kein nutzbarer Index für description

Import mapping with message regarding missing index


Strukturabfrage
cdp.demo.gesundheitszustand.hilfsmittelSichtbarkeit

Objekt
Relation (md)required_for hat Ziel Sofortmaßnahme
Bedingung bearbeiten
Kein nutzbarer Index für (md)required_for
Relation [Maßnahme - Gesundheitszustand]

Structured query with message regarding missing index

Indexing can improve performance in particular when it comes to writing data (= importing).

Indexing is required for:

- Transaction:
  Read transactions: Search/structured query; view configuration
  Write transactions: Imports (import mapping)
- Checking rights

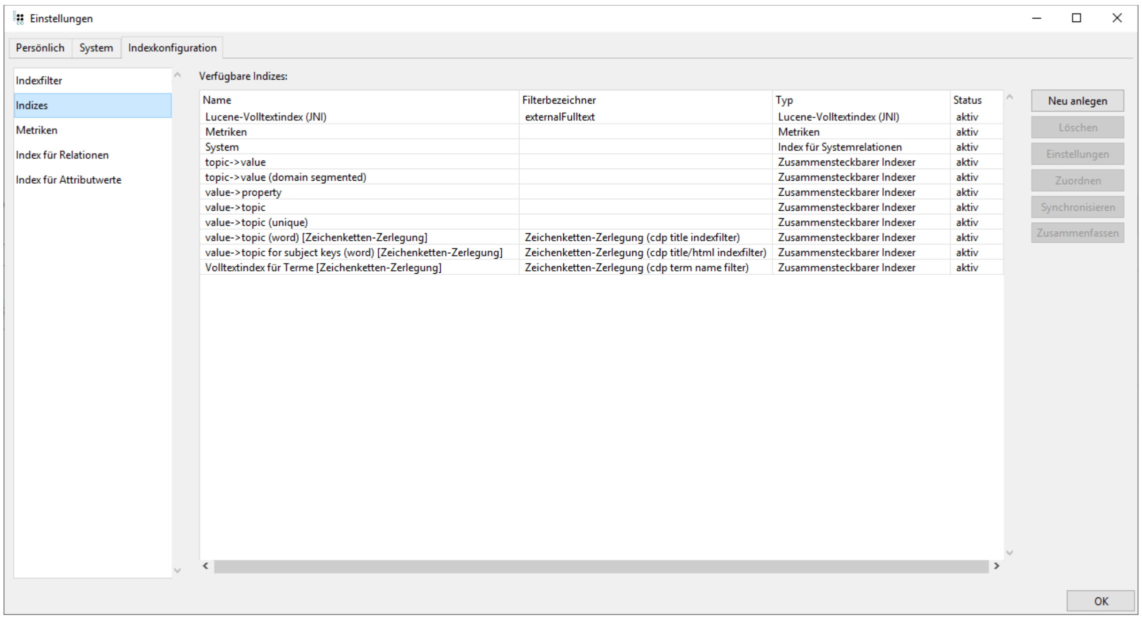
Depending on the intended use, suitable indexes must be selected for certain attributes or relations.

The indexes are **defined** in the Knowledge Builder settings. The **assignment** of the indexes can take place either in the settings of the KB or in the Detail editor of a type (Details > Indexing > Assign index).

1.2.6.1 Manage and apply available indexes

Available indexes (Settings > Index configuration)

All indexes created in the Knowledge Builder can be managed centrally in the settings.



Verfügbare Indizes:

Name	Filterbezeichner	Typ	Status
Lucene-Volltextindex (JNI)	externalFulltext	Lucene-Volltextindex (JNI)	aktiv
Metriken		Metriken	aktiv
System		Index für Systemrelationen	aktiv
topic->value		Zusammensteckbarer Indexer	aktiv
topic->value (domain segmented)		Zusammensteckbarer Indexer	aktiv
value->property		Zusammensteckbarer Indexer	aktiv
value->topic		Zusammensteckbarer Indexer	aktiv
value->topic (unique)		Zusammensteckbarer Indexer	aktiv
value->topic (word) [Zeichenketten-Zerlegung]	Zeichenketten-Zerlegung (cdp title indexfilter)	Zusammensteckbarer Indexer	aktiv
value->topic for subject keys (word) [Zeichenketten-Zerlegung]	Zeichenketten-Zerlegung (cdp title/html indexfilter)	Zusammensteckbarer Indexer	aktiv
Volltextindex für Terme [Zeichenketten-Zerlegung]	Zeichenketten-Zerlegung (cdp term name filter)	Zusammensteckbarer Indexer	aktiv

Category “Indexes”

This setting option can be used to manage the index structures. All available index types are listed under “Available indexes”. Each index type can be used for specific types of attributes or relations.

If an index is shown in grey, then the index is currently deactivated; if it is highlighted in red, then the index is currently not synchronous.

There are buttons to generate, delete, configure, assign and synchronize on the right-hand side.

Index	Use
Lucene full text index (JNI)	Full text query
Metrics	Performance improvement in structured queries

System	System relations (predefined, cannot be changed) This is used for “extends object” / “has extension” / “is super-type of” / “is subtype of” relations
topic -> value	To list attribute values/relation targets in object lists
topic -> value (domain segmented)	To list attribute values/relation targets in object lists
value -> topic topic -> value	For single-sided relations, results in a speed-up for weighted inverse single-sided relations
value -> topic	Attribute values for an object
value -> topic (unique)	Attribute values that may only occur once per attribute type for an object (write rights check for imports)
value -> topic (word)[string splitting]	CDP-specific: This is only used in *i-views content*
value -> topic for subject keys (word)[string splitting]	CDP-specific: This is only used in *i-views content*
Full text index for terms [string splitting]	CDP-specific: This is only used in *i-views content*

Category “Index for relations"/ “Index for attribute value”

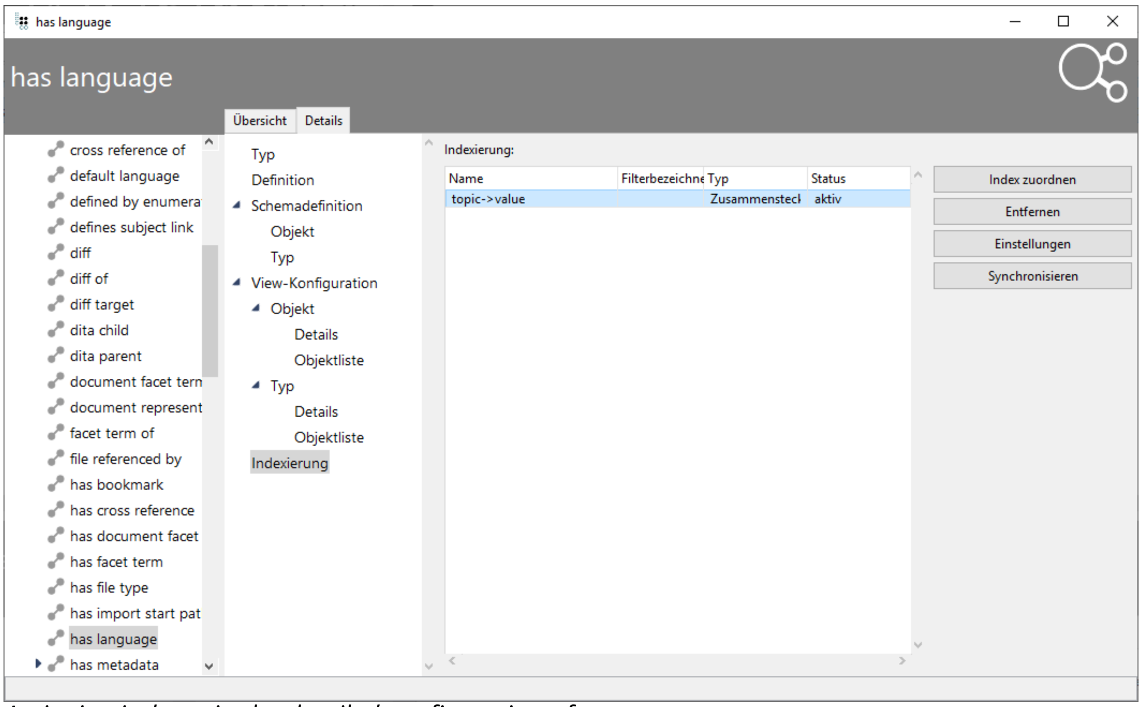
Indexes can be divided up using different aspects. First of all, a distinction can be made between forward and reverse indexes. In the case of the reverse indexes, it may make sense to refer to the property from target/value to resolve the metaconditions on the property. Ultimately, an index can optionally perform a segmentation by each type of source object in order resolve structured queries that are limited to objects of subordinate types more efficiently.

Some properties may not require an index depending on the specific application. (They can then be marked with “Ignore”. They are not examined further in this optimization step.)

- Relations can use a reverse index instead of a forward index on the inverses - and vice-versa.
- Attributes can also be indexed with modified/standardized values (e.g. full text with basic word forms). A corresponding operator can then be used for search for these.

Applicable indexes (detailed configuration)

The indexes that can be used for a relation type or attribute type can be assigned using the detailed configuration.



Name	Filterbezeichne	Typ	Status
topic->value		Zusammensteck	aktiv

Assigning indexes in the detailed configuration of type

Attribute types	Relation types
topic -> value	topic -> value
topic -> value (domain segmented)	topic -> value (domain segmented)
value -> property	value -> property
value -> topic	value -> topic
value -> topic (unique)	

1.2.6.2 Create a new index

In the settings of Knowledge Builder, a new index can be created under:
Settings > Index configuration > Indexes > Create new

The following selection is available at the start:

Index	Use
Lucene full text index (JNI)	Full text query

Redundant storage for relation attributes	To display meta properties of symmetric relations more quickly; this is used without additional filters
Pluggable indexer	Combined use of distributor and index modules for adapted indexing; specific configuration by means of index filters is possible

The following section describes the configuration of the pluggable indexers because these can be used most flexibly and cover almost all use cases.

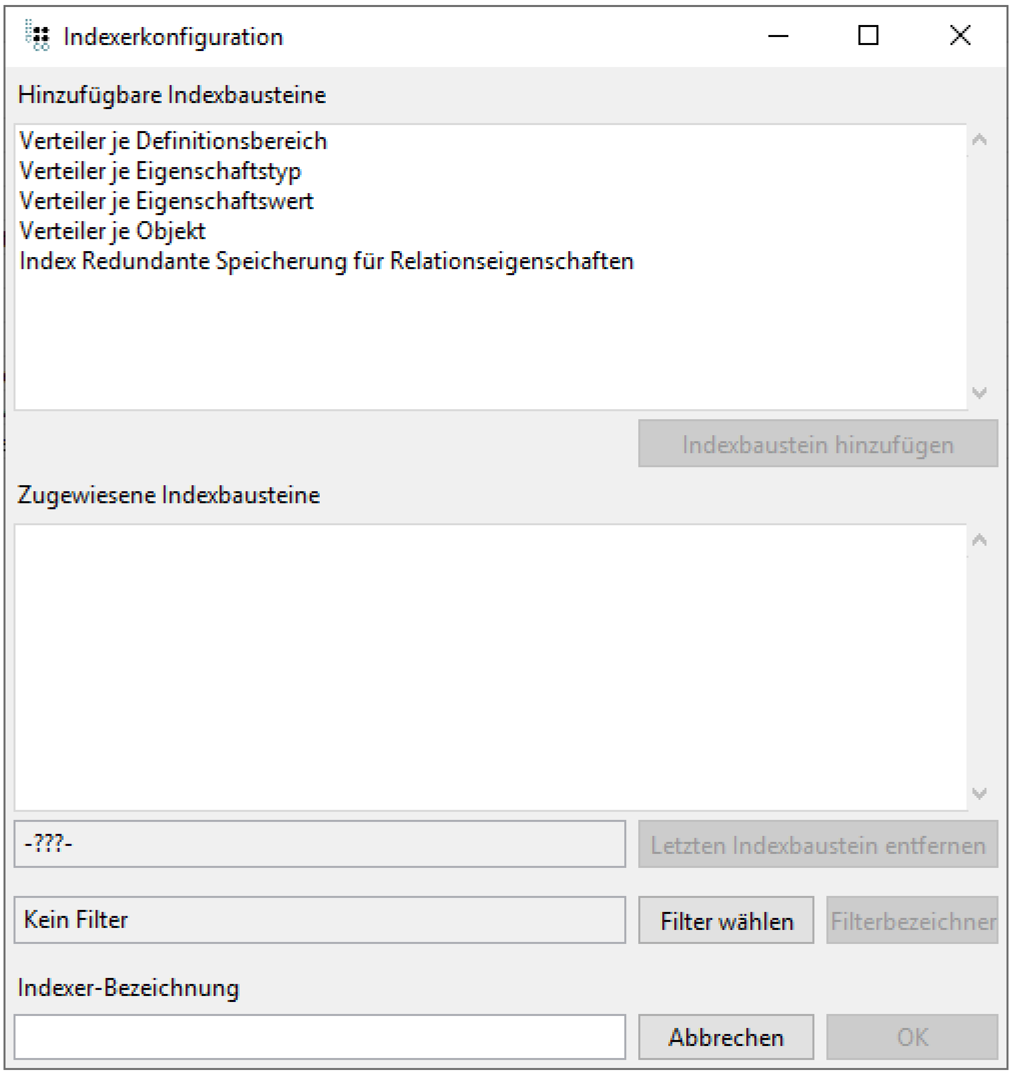
Addable index modules

Pluggable indexers enable the administrator to create an indexer from prefabricated modules in order to achieve the corresponding indexer behavior.

A pluggable indexer consists of distribution levels that are closed by an index level that regulates data storage. Hence, an indexer can index both attributes and relations.

If the indexer is assigned an optional index filter, the indexer behavior can be influenced further; only suitable property types can then be assigned to the indexer.

Since properties include attributes and relations, the following section refers to an attribute value or relation target as a value of the property.

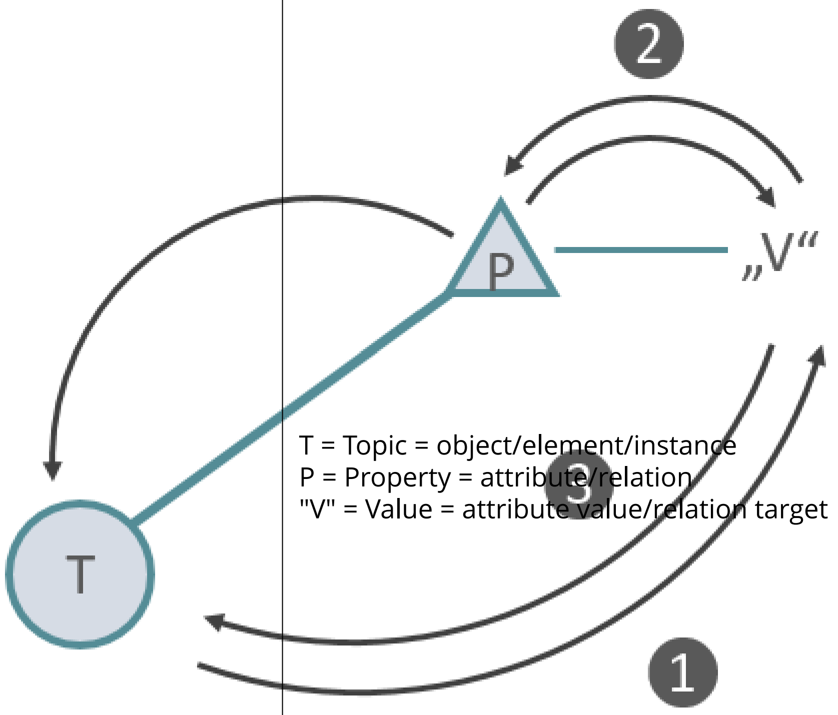


Verteiler je Definitionsbereich
Verteiler je Eigenschaftstyp
Verteiler je Eigenschaftswert
Verteiler je Objekt
Index Redundante Speicherung für Relationseigenschaften

-???-

Kein Filter

Pluggable indexer


2
P
„V“
T = Topic = object/element/instance
P = Property = attribute/relation
"V" = Value = attribute value/relation target
3
T
1

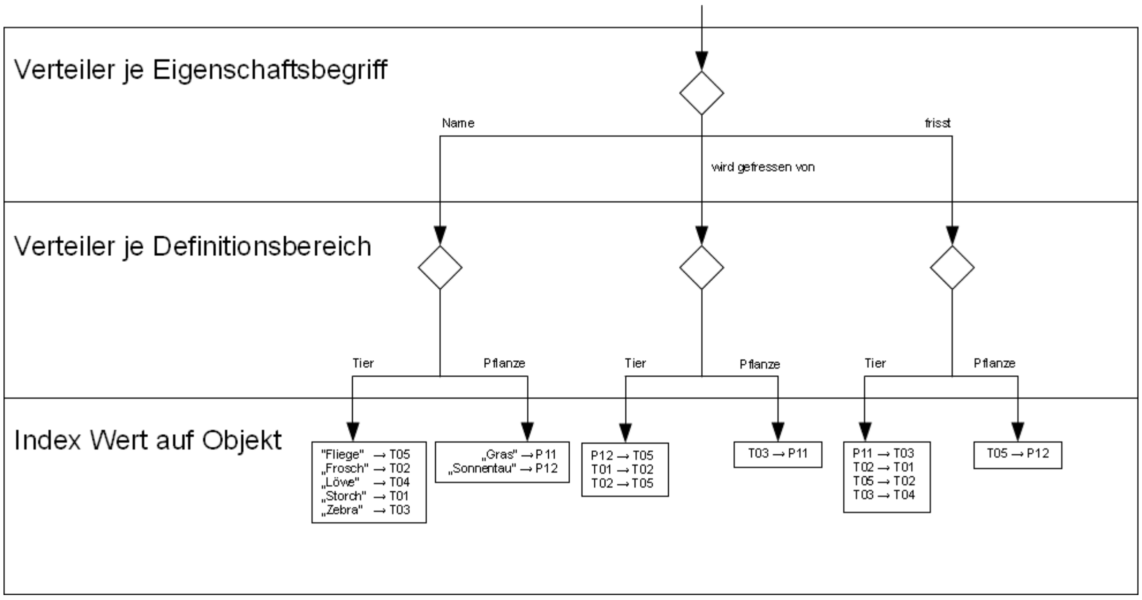
Distributor/index	Use
Distributor by domain (after that, all other distributors can be selected)	To search for a subset of object types that jointly use a property
Distributor for each property type (index can be selected afterwards:)	Distinction between attribute and relation
Index property on value/target	Attribute -> Attribute value, Relation -> Target object/target type To find relation targets in structured queries with a restriction on the meta property
Index object on value/target = topic -> value = topic -> value (domain segmented)	Object -> Attribute, Abject -> Target object of relation To list attribute values/relation targets in object lists

Index value/target on property = value -> property	Attribute value -> attribute Meta-relation target -> Attribute Relation target -> relation Meta-attribute (value) -> Relation For single-sided relations, results in a speed-up for weighted inverse one-way relations
Index value/target on property (uniqueness check)	Attribute value -> Attribute To search for meta properties
Index value to semantic element = value -> topic	Attribute value -> Attribute Relation target -> Relation To support structured queries on objects with specified values/targets on attributes/relations
Index value to semantic element (uniqueness check) = value -> topic (unique)	Attribute value -> Object (e.g.: email address)
Distributor for each property value	Together with “Index property”: For compact storage of many identical values/targets; same response as for “Index value/target on property”
Distributor for each object	For single-sided inverse relations
Index redundant storage for relation properties	(Might not be used in combination with pluggable indexes) Faster display of meta properties on relations when using symmetric relational properties

Filter	
Filter type	**Use**
Latitude	For indexing an attribute type of the value type “geographical position”
Longitude	For indexing an attribute type of the value type “geographical position”
Interval start value	For indexing an attribute type of the value type “interval”
Interval stop value	For indexing an attribute type of the value type “interval”
String filtering	.
Strings to words filter	.

1.2.6.3 Details about indexer blocks

A distinction is made between the breakdown indexer modules and the indexing indexer modules. A breakdown indexer module partitions the index according to different aspects. Following that, there is either another breakdown or an indexing index module that stores the index entries.



The figure shows an example of how a stackable indexer consisting of three modules (without value filter) groups the index entries. This index can now efficiently provide answers to questions such as

- Which animals start with S
- Which plants either other organisms
- Which animals eat zebras (T03)
- etc.

Questions such as

- Which organisms start with S
- Which organisms eat flies (T05)

could also be answered. To do so, an indexer configuration without “Distributor by domain” would suffice (and might be more efficient depending on the data situation).

1.2.6.3.1 Distributor by property type

The most important module, without which most indexing modules cannot be added. It generally appears in first place and partitions the entries according to their property type.

1.2.6.3.2 Distributor by domain

Enables partitioning according to the relevant terms of the property-carrying objects. The module is only useful for properties of individuals.

If a property can occur in multiple object types and a search only searches for a subset of these object types, this module accelerates the search through corresponding index access.

1.2.6.3.3 Distributor by object

This module can be used for indexing to summarize the relation targets on the source object. As the previous module, it is used for mapping older indexers and its K-Infinity 3.1 only makes sense for single-sided inverse relations.

1.2.6.3.4 Distributor by property value

Used to partition according to relation target or attribute value. In this case, only the property can still be indexed (see Index property).

1.2.6.3.5 Index value / target to object

This index module is used to store an attribute value on an object or a relation target on the source of a relation in the index. This type of indexing makes sense if expert queries for objects with specified values on indexed attributes (e.g. with specified target on indexed relations) are supposed to be supported.

1.2.6.3.6 Index object to value / target

The index module indexes in the exact opposite way as the “Index value to semantic element” and, for attributes, can be used to determine the column values of the indexed attributes for object lists. For relations, it can be used in the same way as the “Index value to semantic element” if either the inverse relation is indexed or the source object is already more restricted by the search than the target object.

If you want to support expert queries with the indexed relation in both directions (source-target and target-source), the relation can be indexed either with this value and the “Index value to semantic element” or the relation and its inverse relation can both be indexed with one of the two index types. Here, it can make a difference if the index module is combined with a “Distributor by domain” because use of this distributor module for an index on the inverse relation can be used for partitioning by means of the target domain.

1.2.6.3.7 Index value / target to property

This index module is used to store values on the attribute or target on a relation in the index. This type of indexing makes sense if searches for additional meta properties are supposed to be supported for the indexed attributes. To ensure this index can also be used in a search for the objects of the property (analogous to “Index value to semantic element”), the respective property must remain set to “Active” under “Property can be iterated” in the corresponding term editor.

1.2.6.3.8 Index property to value / target

This index module supports expert queries to search for targets of the relations. To do so, the meta properties of the relation are used for a highly restricted process. Simple source-target conditions are not, however, supported.

1.2.6.3.9 Index property

Together with the distributor for each property value, the same behavior can be achieved as for an index value / target to property. If there are a great many identical values or targets, this makes it possible to achieve more compact storage; otherwise, this combination has no advantages.

1.2.6.3.10 Index property value

This index only stores the attribute values or relation targets. Using it makes sense if a “Distributor for each object” is used upstream and few objects have many values/targets.

1.2.6.3.11 Index redundant storage for relation properties

This module can only be used by itself and is used to display the meta properties on relations more quickly if symmetric relational properties are used. No index structure is created at the technical level but the indexer can be addressed via the same configuration and programming interfaces.

1.2.6.3.12 Uniqueness check

The Index value to semantic element and Index value to property modules can be supplemented with a uniqueness check. The modules supplemented in this way are usually used for the consistency check of unique identifiers. They are available in the selection list for the addable index modules (e.g. Index value to semantic element (uniqueness check)).

If a new value is to be written and the same value is found in the index, this new value cannot be adopted. Values are recognized as identical if they are also grouped identically by all distributors of the index. If, for example, you want to perform a uniqueness check by domain only (this, for example, makes it possible for “modern” to coexist as an individual of verb and as an individual of adjective), the index must contain a Distributor by domain.

If a value filter is also configured, the uniqueness check is executed on the filtered values. This makes it possible, for example, to identify “arm” and “Arm” as identical.
Note: a value filter that splits strings (for full text) can be combined with the uniqueness check, but this is not usually sensible, because even a partial string can lead to duplicates after splitting, for example “The house” and “house and home.”

The Index value to semantic element cannot recognize duplicate values of this property as duplicates in an object if properties occur multiple times. It is therefore possible for two identical attributes with identical values to exist in the same object, but not in different objects. If you want to prevent this, you must deactivate multiple occurrences in the attribute term or instead use an Index value/target to property for the uniqueness check.

1.2.6.4 Details about value filter

1.2.6.4.1 Value decomposition

No atomic attribute value can be indexed for geocoordinates and interval attributes. Instead, longitude and latitude or interval start value and interval stop value are used to index one component of the value. For complete indexing, a corresponding indexer for the other component of the value must be configured respectively.

1.2.6.4.2 String manipulations

Full text filters for strings can be configured in the Admin tool. These can be used to configure which manipulation is possible on the strings, and how the strings should be split into individual words. Additional operators are then offered in expert queries, to which the respective filter label has been added, to allow a specific query to be executed using this filter.

Strings can be indexed in manipulated form by means of "string filtering," and when a query is executed, this results in all attribute values being interpreted as hits which the filter maps to the same string as the search input.

By means of "string splitting," several (manipulated) sub-strings (tokens) from a text can be indexed. The related index then allows expert queries that execute a search within the string by means of the operators "Contains words" and "Contains phrase."

1.2.6.5 Metrics

An attribute "Average number (calculated)" can be created on all property types. The value of the attribute specifies how many values of the corresponding property an object from the property domain has on average.

This information enables structured queries to better decide how they determine their result set. In addition, you can create an attribute "Average number (manual)" whose value overwrites this value. (This makes sense if the domain is abstract but the property in enquiries is supposed to be used only when it actually occurs.)

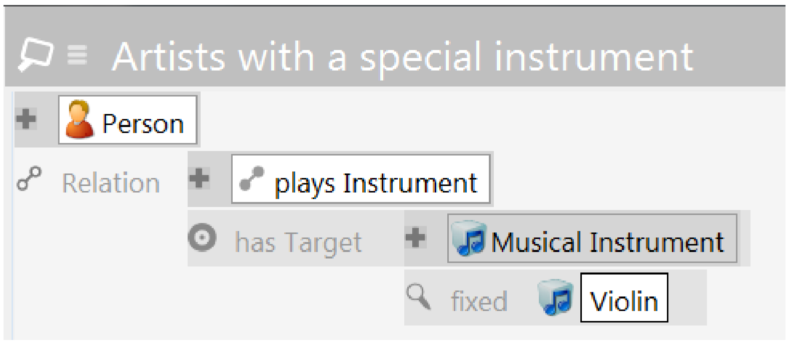
1.3 Searches / Queries

Querying of the semantic network has various subtasks for which we can configure different search modules: often we would like to process the user's entry in a search box (character strings). Usually we would like to pursue the links for the queries within the semantic network.

- Structured queries
- Simple/direct queries (simple search, full text search, trigram search, regular expressions, parameterised hit quality)
- Search pipeline

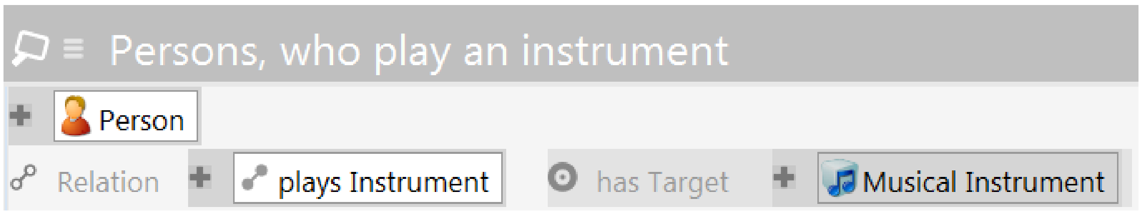
1.3.1 Structured queries

Using structured queries you can search for objects which fulfilled certain conditions. A simple example for a structured query is as follows: all persons who master a certain instrument should be filtered.

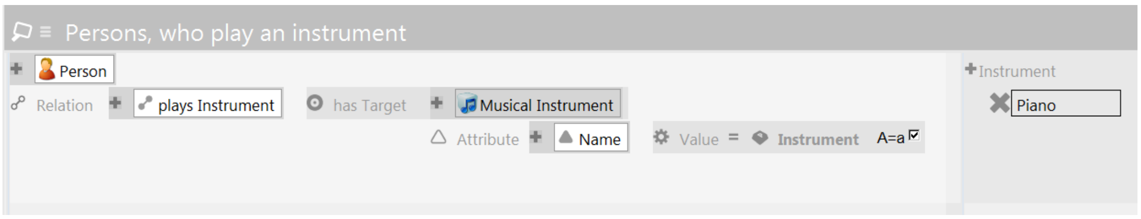


At first there is the type condition: objects of the type person are searched for. The second condition: the persons have to master an instrument. Third condition: this instrument has to be the violin.

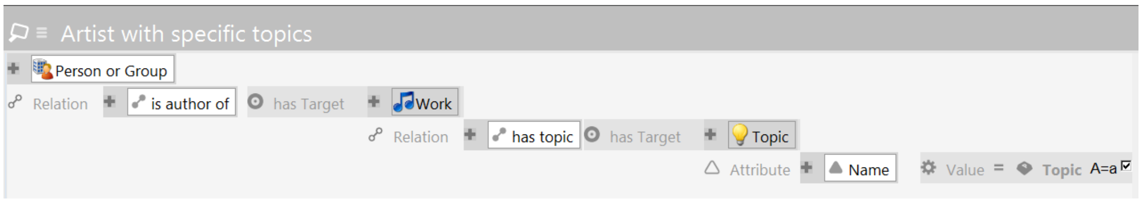
In the structured query the relation "plays an instrument", the type of the target of the relation and the value of the target "violin" form three different lines and thus also three search modes. The third condition that the instrument has to be a violin may also optionally be omitted. In the hit list you would then find all persons who play any random instrument.



Often conditions (in this case the instrument) should not be determined previously but be approved completely. Depending on the situation, an instrument may be given as a parameter in the application:



The conditions may thereby be randomly complex and the network traversed as far as possible:

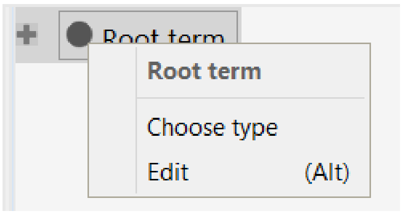


Slightly more complex example: persons or bands who deal with a certain issue in their songs (to be more exact in at least one). In this case you do not search for the name but the ID of the issue as the parameter - typical for searches, for example, which are queried via a REST service from the *application [Figure - "ID" instead of "name"]*

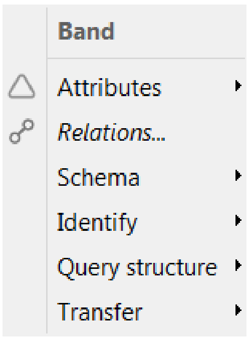
The type hierarchies are automatically included in the structured queries: The type condition "work" in the search box above includes its subtypes albums and songs. Even the relation hierarchy is included: if there is a differentiation below "is author of" (e.g. "writes text" or "writes music") the two sub-relations will be included in the search. The same applies for the attribute type hierarchy.

Interaction

If a new structured query is created, the topmost of all types is entered at first per default. In order to limit the query even more you can simply overwrite the name or select "Choose type" by clicking on the icon.



The button allows you to add more conditions to the structured query. Deleting conditions takes place at the beginning of each line where the type condition is listed (relation, attribute, target, etc.). When you click on the button the following menu will appear which may vary slightly depending on the context.



From all possible conditions, focus has, until now, been on the very first item in the menu. A complete explanation of all conditions and options of the structured queries can be found in the next chapters.

1.3.1.1 Use of structured queries

One of the main purposes of structured queries is to provide information on a certain context in applications. The structured query from the last section, for example, can enable end users in a music portal to generate a list of all artists or bands who cover subjects such as love, drugs, violence etc. in their songs.

To do so, the structured query is usually integrated into a **REST service** via the query s **registration key**. We include the subject in which the user is interested as a parameter in the query with the user s ID.

Example scenario: A user enters a search string to search for their topic. Hence, there is no ID but only a string that is to be used to identify the topic. However, the query result is supposed to show immediately which bands have written songs on the subject. For this purpose, a structured query can be integrated into a **search pipeline** as a component - after the query that processes the search string.

One of the reasons why structured queries are such a central tool for i-views is that the conditions for **rights and triggers** are defined with structured queries. Let s assume the only people allowed to leave comments in a music portal are artists and bands. In the rights system, you can thus specify that only artists and bands that have written at least one song on a topic may leave comments on this topic. Structured queries can also be used in exports to determine which objects are to be exported.

All these uses have one thing in common: we are only interested in qualitative, not weighted statements. This is the domain of structured queries in contrast to search pipelines.

Last but not least, structured queries are also important tools for us as knowledge engineers. We can use them to get an overview of the network and compile **reports** and **to-do lists**. Here are some examples of questions that can be answered using structured queries:

- Which topic is featured by many artists/bands?
- Do specific topics have to be removed because too many relations have amassed or conversely should rarely used topics be merged or closed?

For ease of use, it makes sense to be able to organize structured queries in **folders**.

One main purpose of the structured queries lies in the applications for a certain context to provide information. The structured query in the last paragraph may generate, for the user, a list of all artists or bands in a music portal who deal with the topics of love, drugs, violence, etc. in their songs.

For this purpose, a REST service is usually implemented via their registration key. The topic in which the user is interested just now is the one we enter into the query with his ID as a parameter.

The following is an example of this scenario: a user searches for a topic by entering a search string according to a topic. There is therefore no ID, but only a string on the basis of which the topic should be identified. Thereby, the search results should show which bands wrote songs on this topic. For this purpose, a structured query can be installed in a search pipeline as a component - behind the query which processes the search string.

Therefore, structured queries are, among other things, a central tool within i-views because the conditions for rights and trigger are also phrased using the structured queries: assuming only artists and bands are allowed to leave comments within a music portal. With the rights management it may be stated that only artists and bands who have written at least one song on a certain topic are allowed to leave their comments on this topic. Structured queries may also be used in exports in order to determine which objects are to be exported.

All these applications have one thing in common: we are only interested in qualitative, not weighted statements. That is the domain of the structured queries in comparison to the search pipelines.

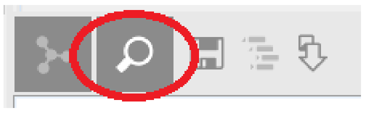
Last but not least, structured queries are also tools for us knowledge engineers. With them we can obtain an overview of the network and compile reports as well as to-do lists:

- To what topics there are how many artists/bands?
- Do certain topics have to be removed because they have collected too many relations or should, on the other hand, sparse topics be compiled or closed?

For this application it makes sense to be able to organise structured queries into folders.

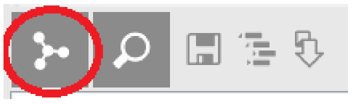
Implement

The structured queries are implemented on the results tab with the button search.



The search results can then be further processed (e.g. copied into a new folder) but they are not kept there permanently.

The path which the structured query has taken may only be viewed in the graph editor to backtrack it. To this end, one or more hits are selected and displayed using the button graph.



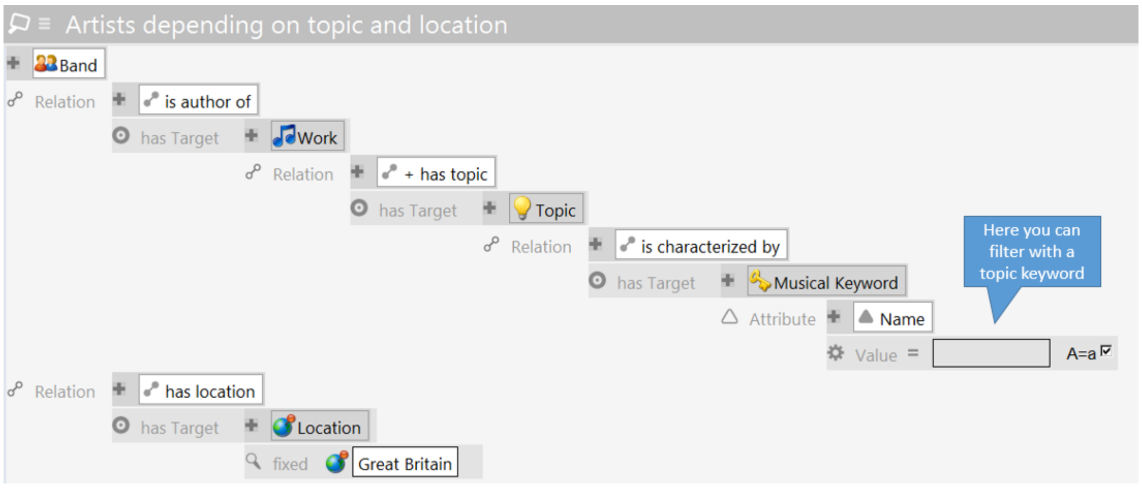
A structured query may be copied in order to create different versions, for example. Likewise there is the possibility of saving them in XML format, regardless of the network. The structured query may therefore be imported into another network. However, this is limited to versions of the same network, e.g. to backup copies, because the structured query references types of objects, relations and attributes via their internal IDs.

1.3.1.2 Structure of structured queries

Very indirect conditions can be expressed within structured queries: you may randomly traverse between the elements throughout the structure of the knowledge network. Artists and bands may be found who wrote songs on certain topics but which we cannot name specifically using their titles.

1.3.1.2.1 Serveral conditions

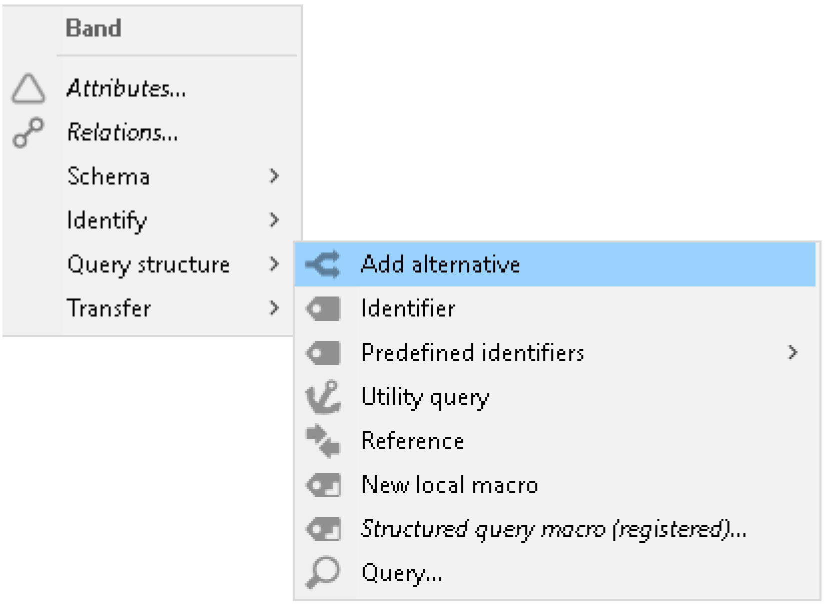
Condition chains may either be randomly deep or several parallel conditions may be expressed: additional conditions are added to any random condition element as a further branch:



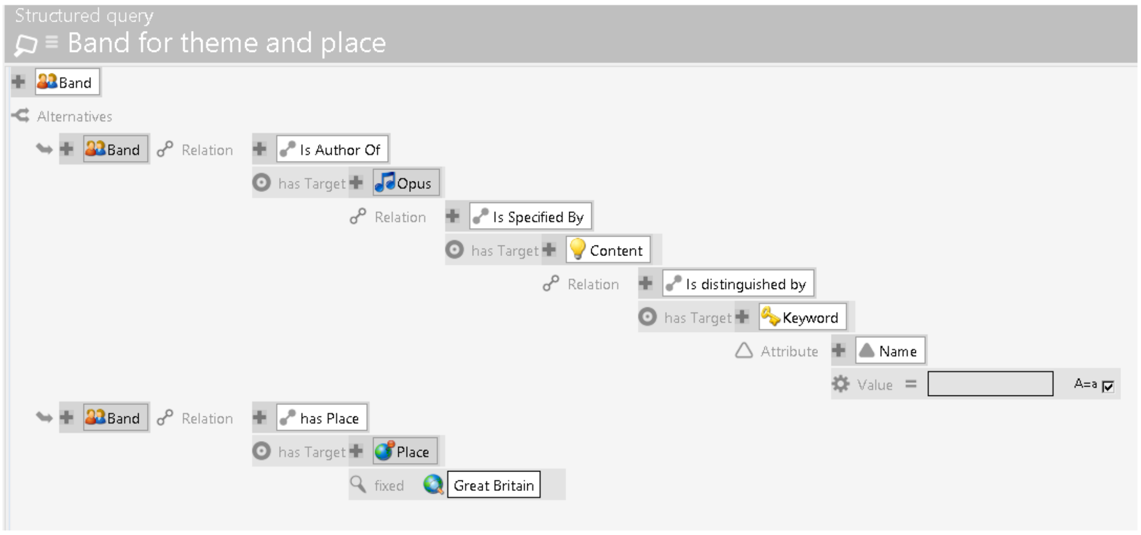
Several conditions: English bands with songs on a certain subject

1.3.1.2.2 Alternative Conditions

In the example mentioned above only artists or bands can be found who created songs on a defined subject and who come from England. If, instead, we want to find all artists and bands which fulfil one of the two conditions they will be expressed as 'alternative'. You can create an alternative via the menu of 'Band'.



By clicking the symbol you can add a further alternative condition for 'Band':



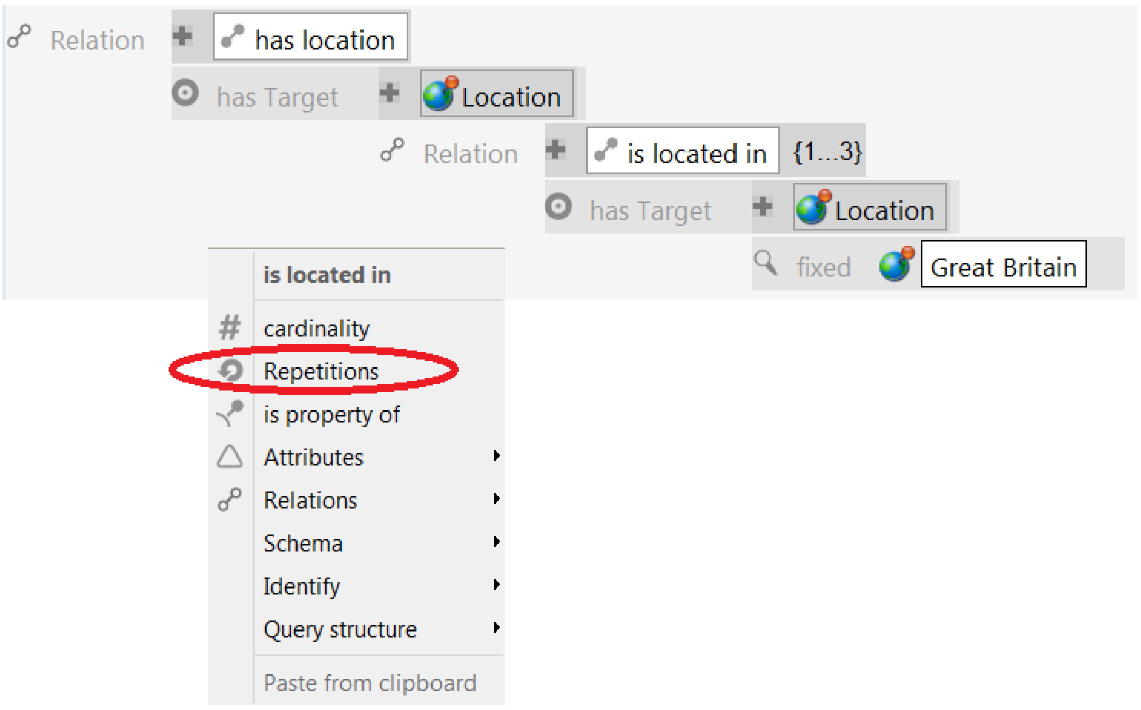
Alternative conditions - the band either has to be English or have songs on a certain subject

If there are further conditions outside the alternative bracket there are objects in the hit list which fulfil **one** of the alternatives and **all other** conditions.

1.3.1.2.3 Transitivity / Repetitions

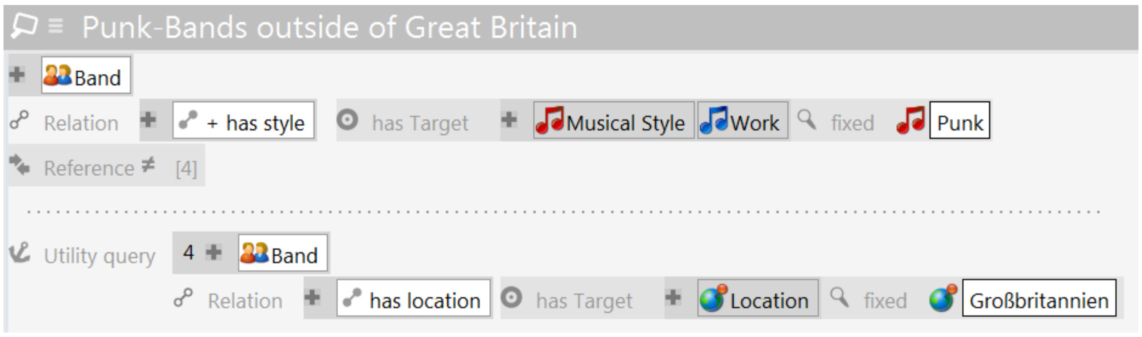
Let’s assume the bands are assigned to either cities or countries within the network. Of these, in turn, it is known which cities are in which countries. In order to document these contents in the search it was possible to very simply expand the condition string: we were able, for example, to search for bands which are assigned to a city which, on the other hand is in England. However, in this manner those bands will not be found which are directly assigned to England. In order to avoid this we can state in the relation "is located in" that it is optional and therefore does not have to be available.

Simultaneously, we can also include hierarchies which are several levels deep using the function "Repetitions". For example, is known from the band ZZ Top that they come from the city of Houston which is in Texas. In order to also retain the band as a result when bands from the USA are queried we can state in the relation "is located in" that this relation has to be followed up until repetitions are reached:



1.3.1.2.4 Negated conditions

Conditions can likewise be purposefully negated. For example, if punk bands are searched for, which do not come from England. To this end, the negative condition is setup as a so-called "Utility query".

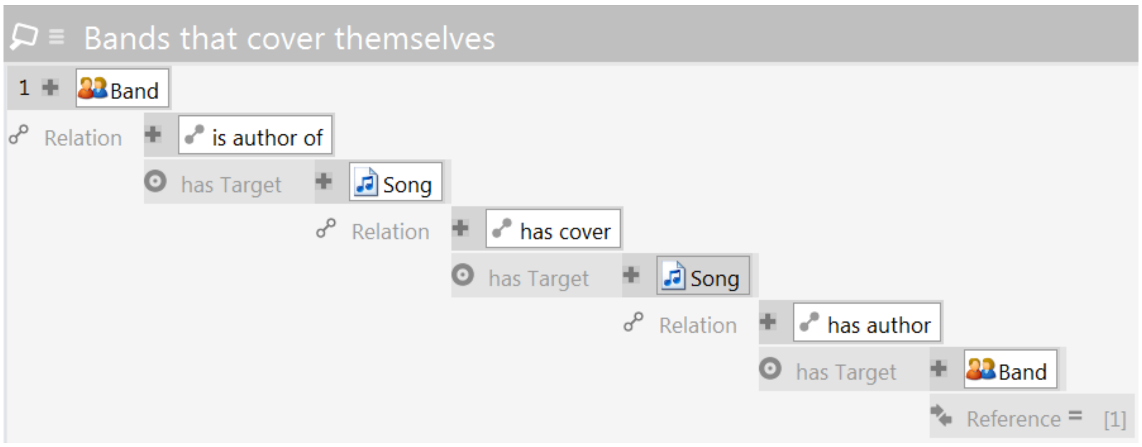


The utility query delivers bands from England - from the main search a reference can be established and thereby noted that the search results are not at all allowed to comply with the criteria of the search help - in this manner we remove the results of the search help from those of the main search and only obtain bands which do not come from England.

Interaction takes place as follows: the utility query is compiled in the type condition and can, after conclusion of the main search above, be linked with the menu item "reference". At this stage you can then select which type the reference should be (in this case negative).

1.3.1.2.5 Corresponds to condition

The reference allows references to be made to other conditions of the same query within a structured query:

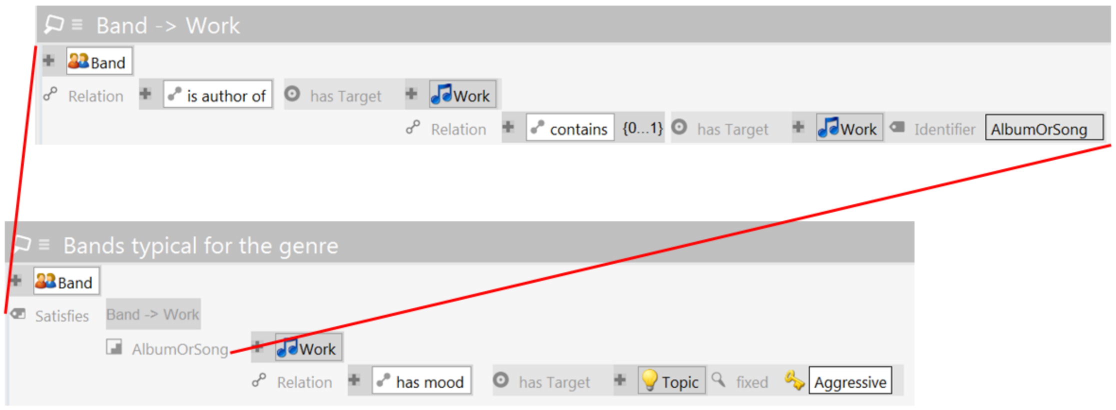

Bands that cover themselves
1 Band
Relation is author of
has Target Song
Relation has cover
has Target Song
Relation has author
has Target Band
Reference = [1]

Here the last condition references the first one, i.e. the band who writes the cover version also has to be the author of the original. Without a reference the search would read as follows: bands which have written songs which cover other songs which were written by any (random) bands. Incidentally, the result is, for example, the band "Radiohead" (they covered their own song "Like Spinning Plates").

1.3.1.2.6 Other options in building the structured queries

Search macros: other structured queries but also other searches can be integrated into structured queries as macros. In doing so, there is the possibility of outsourcing repeating, partial queries into your own macros and thus adapting the behaviour at a central location when changing the model. A macro can be integrated into each condition line.

An example from our music network: from bands to all their achievements, no matter whether they are albums, songs which are directly assigned to the band or songs on the albums of the bands. We need these partial queries more frequently, for example in a structured query which returns the bands to a certain mood. We start this query with a type condition - we are looking for bands - and integrate the pre-defined module as a condition for these bands:


Band -> Work
Band
Relation is author of
has Target Work
Relation contains {0...1} has Target Work Identifier AlbumOrSong

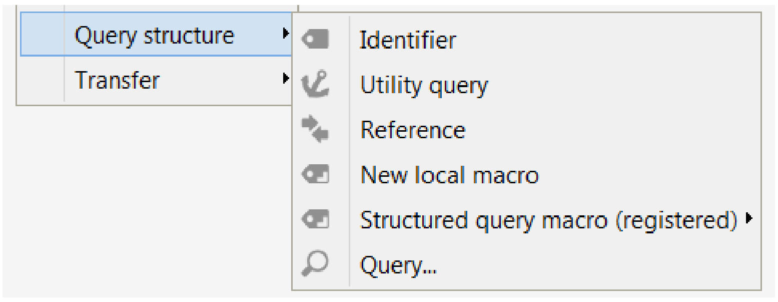
Bands typical for the genre
Band
Satisfies Band -> Work
AlbumOrSong Work
Relation has mood has Target Topic fixed Aggressive

The objects which return those which are integrated into the structured query as macros have, of course, to match the condition with which they are linked from the point of view of their type. With the aid of the **identifier** function, the query (from the "invoking" query) can still be continued with additional conditions. In our case the albums and songs from where the macro query originates are defined by the invoking query: namely albums and songs with the mood "aggressive". Integrating the search macro into the structured query is carried out through the menu "Query structure". Under *structured query macro (registered)* there is a selection list with all the registered macros.

Simple search: using the search mode "simple search" the results of a simple search or a search pipeline may serve as input for a structured query. Each respective simple search can be selected by means of the selection symbol. The input box contains the search entry for the simple search. Further conditions can enable a simple search to be filtered further, for example.

Cardinality condition: a search for attributes or relations without its own conditions may be carried out with a cardinality operator (characterised by a hash tag #). You may use the cardinality greater than or equal to, less than or equal to and equal. The normal equal operator of the relation or attribute condition corresponds to greater than or equal to 1.

We have thus covered everything we can find within the menu "Query structure":



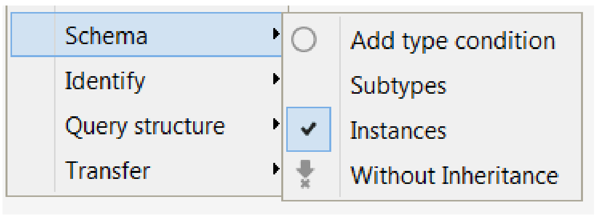
1.3.1.3 Details of the conditions

The type condition

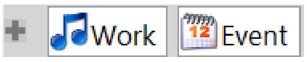
The beginning of the structured query determines which objects should appear as the results. To do this you click on the type icon for the first condition and select "Choose type" in the menu, the input mask then starts in which the name of the object can be entered.

Alternatively, you can simply overwrite the text behind the type icon with the name of the object.

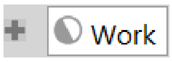
In the second step the relation condition is added. For example, a search is made for the place of origin of a band and "has location" is set as a relation condition. The target type of the relation is added automatically which, however, can also be changed (if, for example, the "has location" relation for countries, cities and regions applies but we only wish to have the cities).



There are further functions available for a type condition. In the item for this there is the item "Schema" in the general condition menu which we can reach via the button +: several types of conditions are defined consecutively what is interpreted by "or" in the query. For example, we search for works or events on a particular style of music as follows:


Work Event

We can just search for types of objects instead of specific objects or both at the same time by checking the boxes "Subtypes" and "Instances" in the menu "Schema".

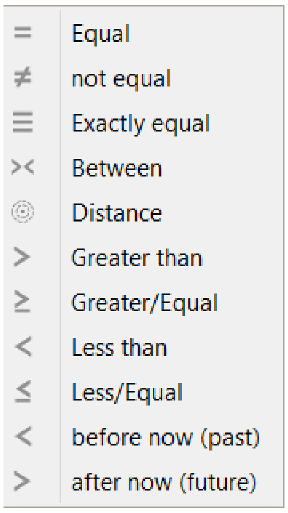

Work

This is what the condition looks like when a search is made for both specific works as well as subtypes of the work (albums and songs).

Without inheritance: normally, the inheritance starts automatically with all types of conditions of the structured query. If a search is made for events in which bands play a certain style of music, all subtypes of events are then incorporated into the search and then we are provided with indoor concerts, club concerts, festivals, etc. In the vast majority of cases this is exactly what is desired. For exceptions there is the possibility of switching off the inheritance and restrict the search to direct objects of the type event, i.e. by excluding the subtypes of objects.

Operators for the comparison of attribute values

Attributes may also play a role as conditions for structured queries. For example, if it does not suffice to only identify objects which show an exact predefined value or the value entered as a parameter. For instance, bands which were founded after 2005 or songs which are more or less 3 minutes long or songs which contain the word "planet" in their title. These require comparison operators. The type of comparison operators which i-views offers us depends on the technical data type of the attribute:



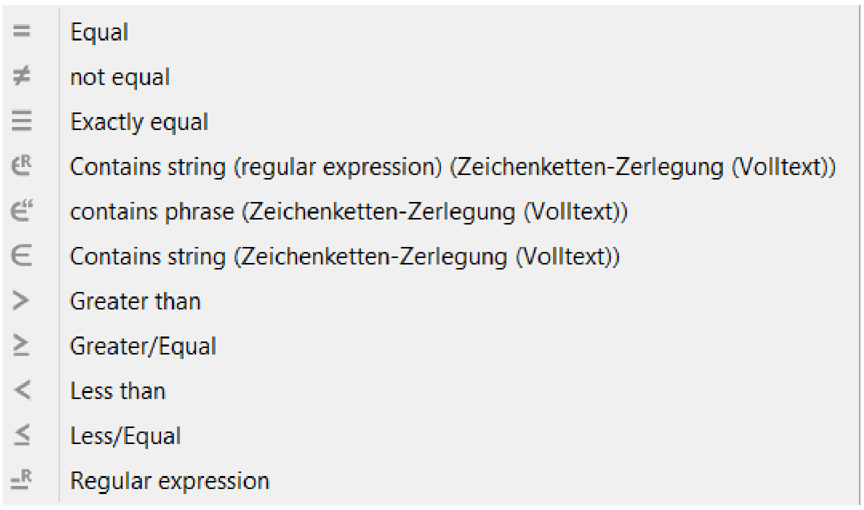
=	Equal
≠	not equal
≡	Exactly equal
><	Between
◎	Distance
>	Greater than
≥	Greater/Equal
<	Less than
≤	Less/Equal
<	before now (past)
>	after now (future)

Comparison operators for dates and quantities

The comparison operator *Exactly equal* constitutes a special case: the index filter is switched off and a search can be made after the special character * which is normally used as a wildcard.

The comparison operator *Between* requires spelling of the parameter value with a hyphen, e.g. "10.1.2005 - 20.1.2005".

The comparison operator *Distance* requires spelling of the parameter value with a tilde, e.g. "15.1.2005 ~ 5" - i.e. on 15.1.2005 plus/minus 5 days.



=	Equal
≠	not equal
≡	Exactly equal
∈ᴿ	Contains string (regular expression) (Zeichenketten-Zerlegung (Volltext))
∈"	contains phrase (Zeichenketten-Zerlegung (Volltext))
∈	Contains string (Zeichenketten-Zerlegung (Volltext))
>	Greater than
≥	Greater/Equal
<	Less than
≤	Less/Equal
=ᴿ	Regular expression

Comparison operators for character strings

Operator	**Beschreibung**
Abstand	
Enthält	
Enthält Phrase	
Enthält Zeichenkette	
Enthält Zeichenkette (Regulärer Ausdruck)	
Entspricht	
Exakt gleich	
Gleich	
Gleich (Geo)	
Gleich jetzt (Gegenwart)	
Grösser als	
Grösser/Gleich	
Ist enthalten in	
Kleiner als	
Kleiner/Gleich	
Nach jetzt (Zukunft)	
Regulärer Ausdruck	

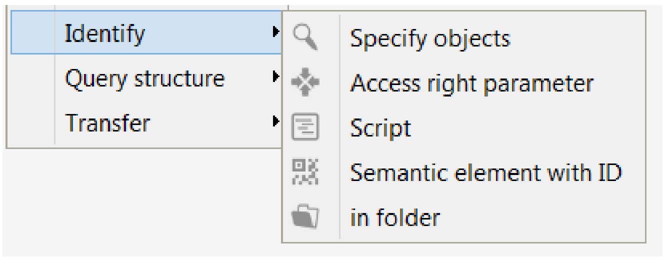
Überschneidet	
Überschneidet von oben	
Überschneidet von unten	
Ungleich	
Vor jetzt (Vergangenheit)	
Zwischen	

Comparative value results from the script: attribute value conditions may be removed from partial searches and replaced by a script and attribute condition. The results of the script are then used as a comparative value for the attribute value condition, e.g. if the comparison operators do not suffice for a specific query.

Identifying objects

The structured query provides several options for identifying objects within the knowledge network. To simplify matters, the previous examples often defined the objects. This type of manual determination may, in practice, be of help in testing structured queries or determining a (replaceable) default for a parameter entry.

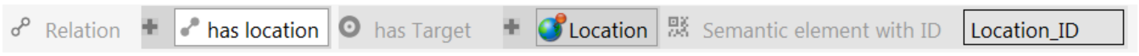
At this point we have already become familiar with the combination with the name attribute which can, of course, be any random attribute. In the menu item "Identify" we will find some more options for defining starting points for the structured query:



Access right parameter: the results of the query may be made dependent on the application context. This particularly applies in connection with the configuration of rights and triggers when, generally speaking, only "user" is usable.

Script: the objects to be entered at this point are defined by the results of the script.

Semantic element with ID: you may also determine an object via its internal ID. This condition is normally only used in connection with parameters and the use of the REST interface.



In folder: using the search mode "in folder" the contents of a collection of semantic objects can be entered into a structured query as input. The selection symbol will enable you to select a folder within the work folder hierarchy. The objects of a collection are filtered with respect to all other conditions (including conditions for terms).

1.3.1.4 Parameter conditions

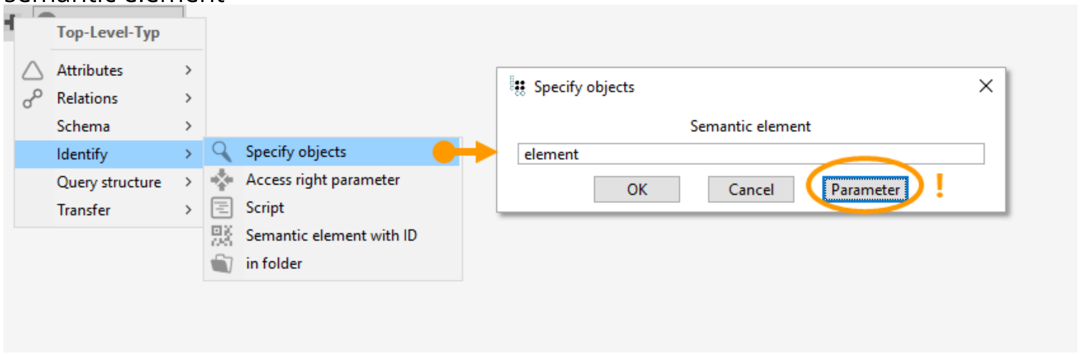
Parameters

In structured queries, input can be passed on by means of a parameter. This allows handing over query input within **JavaScript** code in forms of:

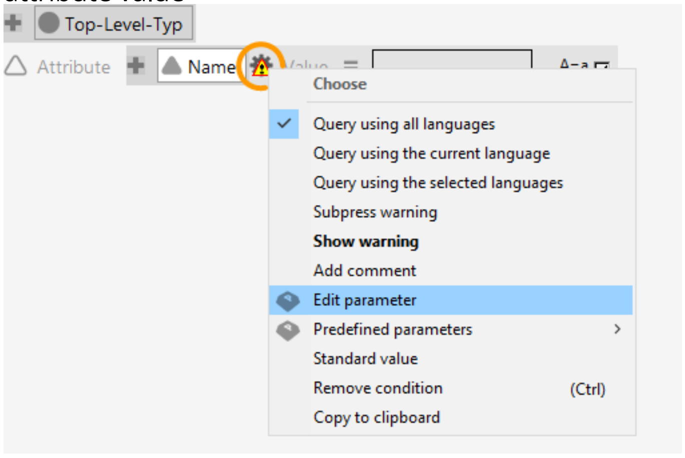
```
$k.Registry.query('<registryKeyOfQuery>').findElements({<parameterNameInQuery>: <input>})
```

The parametrized input can be in forms of:

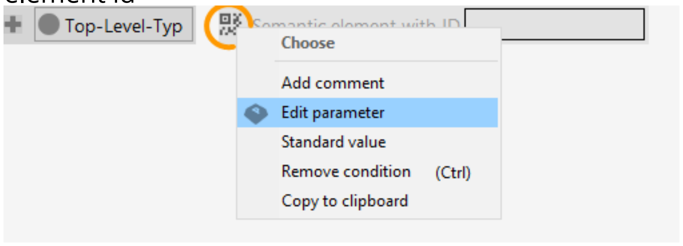
- semantic element



- attribute value



- element id



There are two possibilities to test structured queries using parameters:

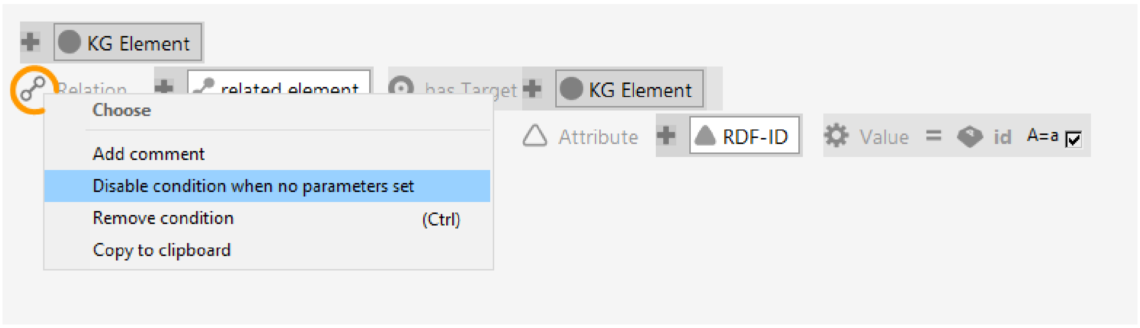
1. Using the test environment of the structured query
2. Invoking the structured query by script (executing or debugging)

In general, there are four conditions a parameter can have:

- Parameter is set
- Parameter is not set
- Parameter is deactivated
- (Parameter contains empty string)

Optional parameters

The structured query has a feature that allows using optional parameters: for a certain branch of the query, the context menu offers the condition "*Disable condition when no parameters set*".

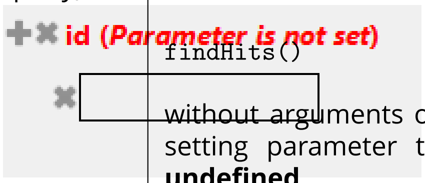


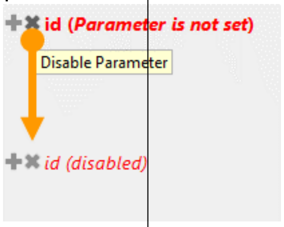
If the optional parameter condition has been set, it has the following effect: From this point on, the rest of the branch (to the right) will not be encountered as condition for the query result when the respective parameter has been **deactivated**.

Note: When the parameter is not set, the test environment will nevertheless throw an error despite the optional parameter condition. If testing of optional parameters is needed, the parameter needs to be **disabled** in order to test an unset parameter condition.

Important rules about setting parameters

Parameter condition	Setting in structured query	Setting in JavaScript	Result
Parameter is set	Parameter value has been entered	Variable containing parameter value is defined	Parameter condition is encountered in query result

Parameter is **not set**	No parameter value has been entered (just executing query) 	Handing over no parameter
		
		Using
		
		`findElements()`
		or
		`findHits()`
		
		without arguments or setting parameter to **undefined**.
			Error: "Parameter xy is missing"

Parameter is **disabled**	Clicking on x besides parameter  id (Parameter is not set) Disable Parameter id (disabled)	Setting parameter to **null**.	• With conventional parameter: **as if the parameter requirement would not exist** within the structured query • With optional parameter: the branch **from the optional condition until end of query branch** will be ignored
Parameter contains empty string	Entering " or "", rejecting search dialog if occurring	Variable for parameter set to **empty string** " or ""	Query branch will return no result; if no alternatives exist, the whole query might return no results

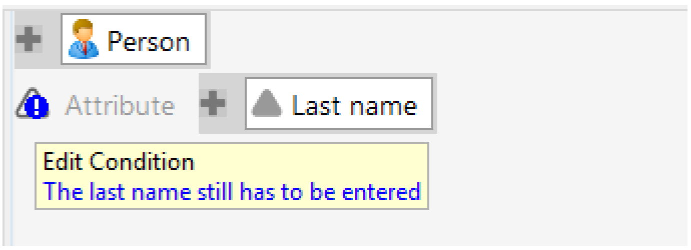
Caution: Risk of search results containing false positives.
For predictability and reliability of query results in scripts, make sure to avoid parameter values from being *null* inadvertently, since no errors are thrown system wise. Use control structures to catch unattended conditions of parameter input.

When an optional parameter is passed on to the structured query by means of a script in a *search view* or a *search result view*, the value type of the parameter also needs to be set to "**optional**". If the value type is set to "obligatory", the structured query will not deliver any search result when the script sets the parameter value to "null" (with the intention to deactivate the optional parameter).

1.3.1.5 Comments in structured queries

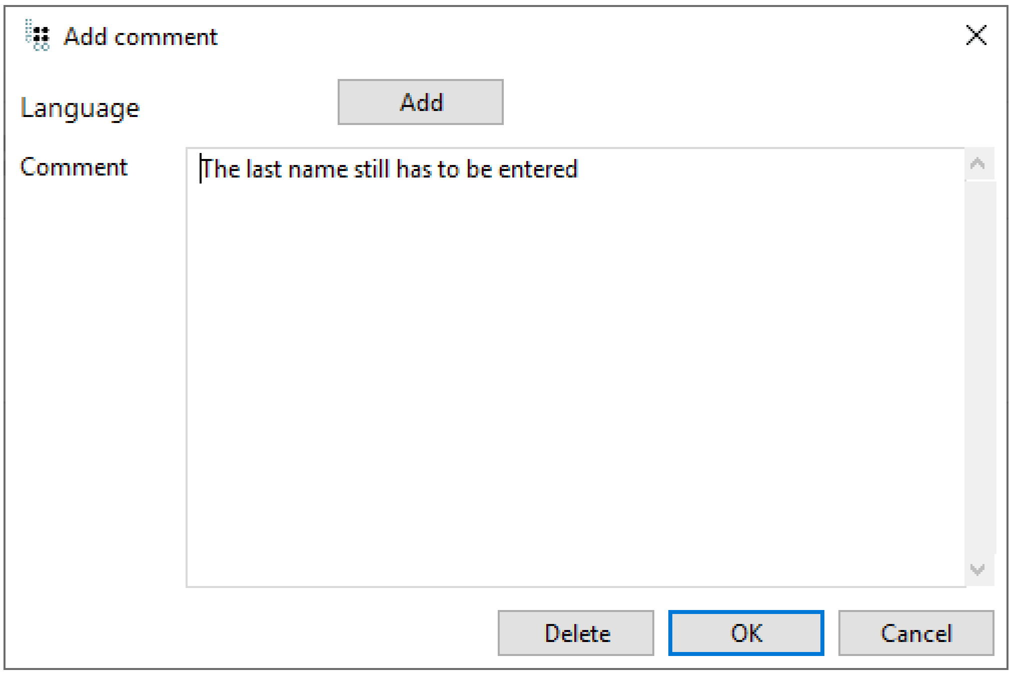
Adding comments

Every condition in a structured query can be commented. For adding a comment, choose the option "add comment" in the context menu. At the condition in the structured query, an existing comment causes a blue indicator flag which shows up a text in case of mouseover.



Edit Condition
The last name still has to be entered

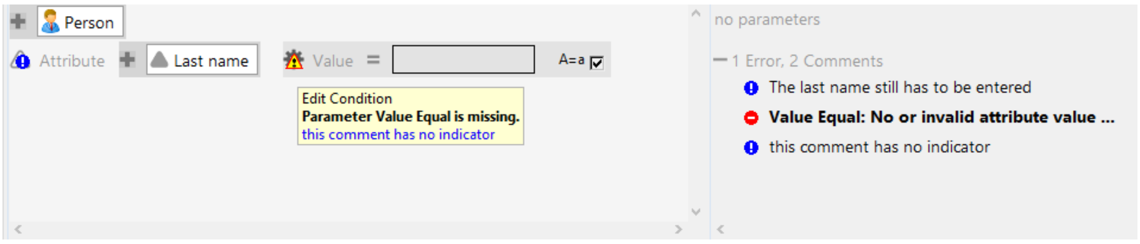
By means of the dialog "Edit comment", the corresponding comment can be changed or removed:



Language

Comment: The last name still has to be entered

The indicator flag for comments is not shown when the condition has a warning or a fault. In this case you only can see the yellow warning indicator or the red fault indicator. Additionally, all warnings, faults or comments will be listed in their order on the right side below the parameters editor.

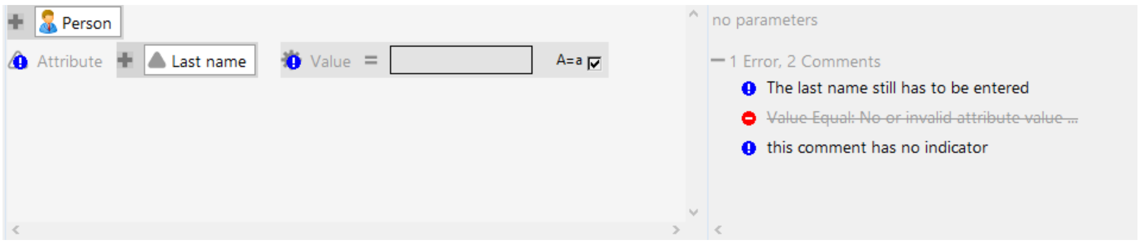


Edit Condition
Parameter Value Equal is missing.
this comment has no indicator

no parameters

1 Error, 2 Comments
- The last name still has to be entered
- Value Equal: No or invalid attribute value ...
- this comment has no indicator

Warnings and cautions can be suppressed in the indicator indication if you want to ignore them at this point (of course, this is not recommended). To do so, click on the indicator symbol in the listed view or choose the function "Suppress warnings" in the context menu of the condition. The indication can be reactivated on the same way or by choosing the context function "Show all warnings" of the root finder.



no parameters

1 Error, 2 Comments

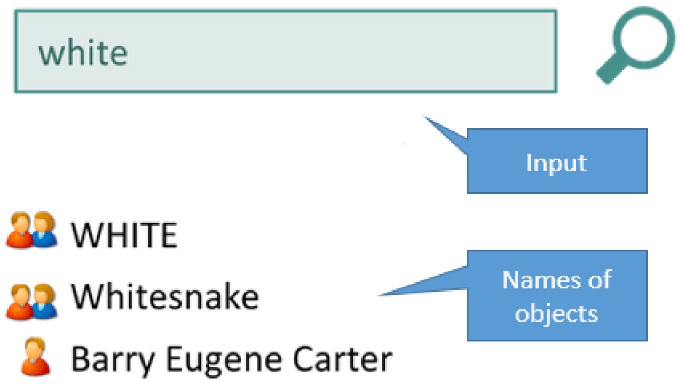
The last name still has to be entered

~~Value Equal: No or invalid attribute value ...~~

this comment has no indicator

1.3.2 Simple Search / Fulltext search

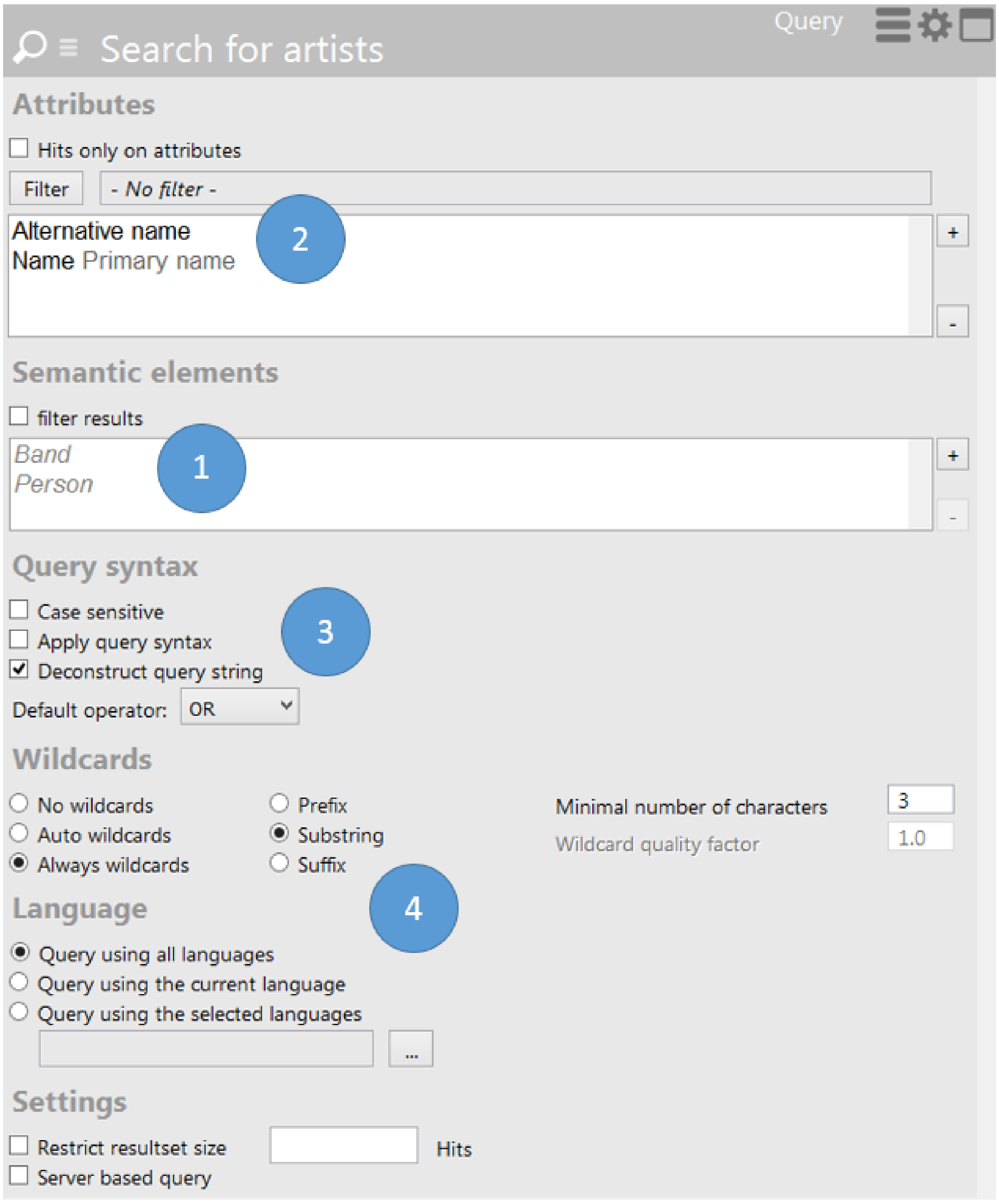
Processing the search queries of users may be carried out with or without interaction (e.g. with type-ahead suggestions). The starting point is, in any case, the character string entered. In configuring the simple search we can now define with which objects and in which attributes we search according to the user input and how far we differ from the character string entered. Here is an example:



How do we have to design and organise the search in order to receive the below feedback on objects from the entry "white"? In all cases we will have had to configure the query to show that we only want to have persons and bands as the results. How is it, however, if there are any deviations from the user input?

- When is the (completely unknown) Chinese experimental band called "WHITE" a hit? If we state that upper case and lower case doesn't matter
- When will we receive "Whitesnake" as a hit? If we understand the entry to be a substring and attach a wildcard
- When "Barry Eugene Carter"? If we not only search through the object names but include other attributes as well - his stage name is namely "Barry White".

These options can be found again in the search configuration as follows:



Search for artists

Query

Attributes

[ ] Hits only on attributes

Filter - *No filter* -

Alternative name
Name Primary name

Semantic elements

[ ] filter results

Band
Person

Query syntax

[ ] Case sensitive
[ ] Apply query syntax
[✓] Deconstruct query string

Default operator: OR

Wildcards

○ No wildcards	○ Prefix	Minimal number of characters	3
○ Auto wildcards	● Substring	Wildcard quality factor	1.0
● Always wildcards	○ Suffix		

Language

● Query using all languages
○ Query using the current language
○ Query using the selected languages

Settings

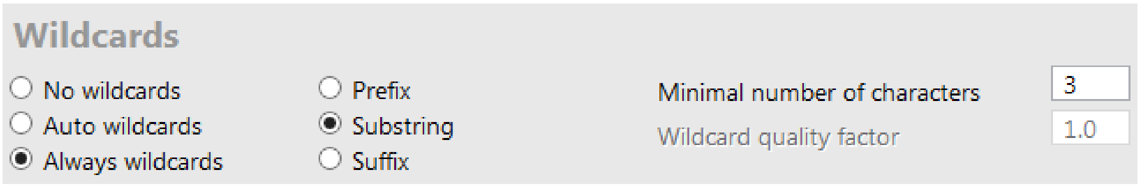
[ ] Restrict resultset size Hits
[ ] Server based query

Configuration of the simple search with (1) details as to which types of objects are to be browsed through, (2) in which attributes the search has to be made, (3) upper case and lower case and (4) placeholders.

1.3.2.1 Simple search - details of the options

Placeholder/wildcard

The entry is often incomplete or we want to retrieve the entry in longer attribute boxes. To do this, we can use placeholders in the simple search. The following settings for placeholders can be found in simple search:


Wildcards

( ) No wildcards
( ) Auto wildcards
(●) Always wildcards

( ) Prefix
(●) Substring
( ) Suffix

Minimal number of characters: 3
Wildcard quality factor: 1.0

- **Placeholder behind** (prefix) finds the [White Lies] for the entry "white"
- **Placeholder in front** (suffix) finds [Jack White]
- **Placeholder behind and in front** (substring) finds [The White Stripes]
- Caution! Placeholder in front is slow.

The option "Always wildcards" works as if we had actually attached an asterisk in front and/or behind. Behind automatic wildcards there is an escalation strategy: in the case of automatic placeholders, a search is made first with the exact user entry. If this does not deliver any results a search will be made with a placeholders, depending on which placeholders have been set. With the option prefix or substring there is once again a chronological order: in this case you look for the prefix first (by attaching a wildcard) and, if you still can't find anything, you make a search for a substring (by means of a prefix and attaching a wildcard).

If you are allowed to attach placeholders in your search you can state in the box minimal number of characters how many characters the search entry must show to actually add the placeholders. By entering 0 this condition is deactivated. This is particularly important if we set up a type ahead search.

With the weighting factor for wildcards you can adapt the hit quality to the extent that the use of placeholders will result in a lower quality. In this manner we can, if we want to give the hits a ranking, express the uncertainty contained in the placeholders with a lower ranking.

If the option "No wildcards" is selected the search entry will not be changed. The individual placeholder settings will then not be available.

The user can, of course, him/herself use placeholders in the search entry and these can be included in the search.

Apply query syntax: when the box for the option "Apply query syntax" has been checked a simplified form of the analysis of the search input is used in which, for example, the words "and" and "or" and "not" no longer have a steering effect. Nevertheless, in order to be able to define how the hits for the tokens should be compiled, the default operator can be switched to "#and" or "#or". What applies to all linking operators is the fact that they do not refer to values of individual attributes, but to the result objects (depending on whether "hits only for attributes" has been set). A hit for online AND system thus delivers semantic objects which have a matching attribute for both online and the system (which is not necessarily the same).

Filtering: simple searches, full-text searches and also some of the specialised searches may be filtered according to the types of objects. In the example described in the last paragraph we made sure that the search results only included persons and bands. Attributes which do not match a possible filtering are depicted in red bold print within the search configuration dialogue. In our case this could be an attribute "review", for example, which is only defined for albums.

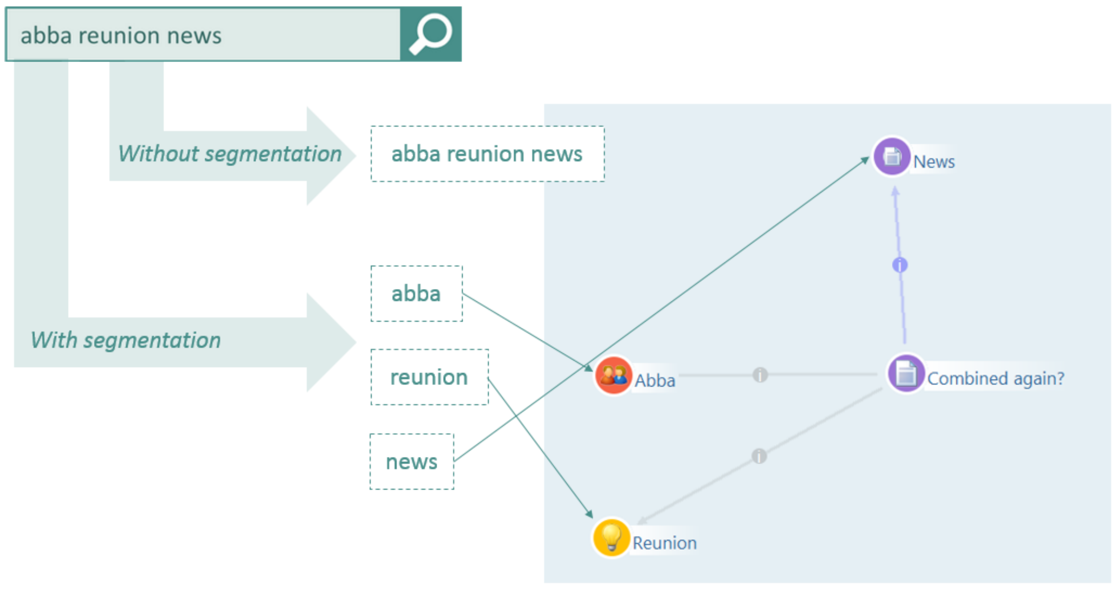
Translated attributes: in the case of translated attributes we can neither select a translation, nor have the language dynamically defined. Search for multilingual attributes, then in the active language or in all languages, depending on whether the option "in all languages" is checked.

Query output: a maximum query output may be defined by entering the maximum number in the "results" box. This checkbox will then limit the query output and the mechanism can be activated or deactivated. By entering the number in the output the checkbox will automatically be activated. Caution: if the number is exceeded no output will be shown!

Server-based search: generally speaking, each search can also be carried out as a server-based search. The prerequisite for this is that an associated job client is running. This option can be used when it can foreseen that very many users will make search queries. By outsourcing certain searches to external servers, the i-views server will be disburdened.

1.3.2.2 Multi word search inputs

In our examples for queries the users have, until now, only entered one search term. However, what would happen if the user entered "Abba Reunion News", for example, and thus would like to find a news article which is categorised by the keywords "Abba" and "reunion"? We have to disassemble this entry because none of our objects would match the entire string or at least not the article being searched for:



Our examples so far do not, however, fall short only due to multi word search inputs. We also often have search situations in which it does not make sense to regard the names or other character strings from the network, with which we compare the input, as blocks , e.g. because we would like to retrieve input in a longer text. In this case the wildcards will eventually no longer be an adequate means: if we also want to disassemble the input on the page of the object and the text attributes which have been searched through it would be better to use the full-text search.

1.3.2.3 Fulltext search and indexation

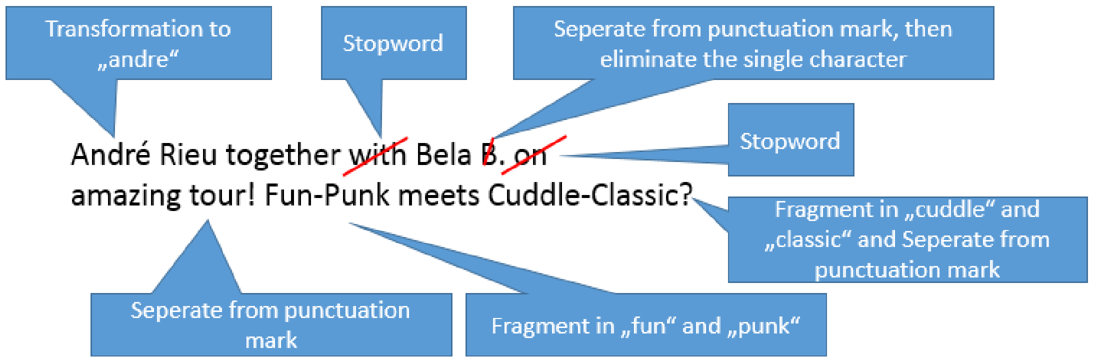
If we want to view or search through longer texts word by word, e.g. description attributes we recommended the use of full-text index. What does something like that look like?

Term	Occurence
aaliyah	Doc#155, Pos. 548644 / Doc#459, Pos. 934875 / Doc#935, Pos. 26526
abba	Doc#132, Pos. 43095 / Doc#459, Pos. 46795 / Doc#935, Pos. 534955 / Doc#343, Pos. 367773 / Doc#711, Pos. 92634
abbey	Doc#464, Pos. 95367 / Doc#2543, Pos. 65258 / Doc#634, Pos. 35241
abbreviation	Doc#436, Pos. 54362
abbreviator	Doc#463, Pos. 234652
abnormity	Doc#253, Pos.4652
abo	Doc#234, Pos. 32243 / Doc#332, Pos.23414

The full-text index records all terms/words which occur within a portfolio of texts so that i-views can quickly and easily look up where a particular word can be found in which texts (and in which part of the text).

"Texts", however, are not usually separate documents within i-views, but the character string attributes which have to be searched through. Their full-text indexing is a prerequisite for the fact that these attributes are offered in the search configuration.

Even full-text indexing concerns the deviations between the exact sequence of characters within the text and the text which is entered in the index and which can hence be retrieved accordingly. An example of this: a message from the German music scene:


Transformation to „andre“
Stopword
Seperate from punctuation mark, then eliminate the single character
Stopword
André Rieu together ~~with~~ Bela ~~B.~~ ~~on~~ amazing tour! Fun-Punk meets Cuddle-Classic?
Fragment in „cuddle“ and „classic“ and Seperate from punctuation mark
Seperate from punctuation mark
Fragment in „fun“ and „punk“

In this example we find a small part of the filter and word demarcation operations which are typically used for setting up a full-text index:

Word demarcation / tokenizing: often in punctuation such as exclamation marks are placed directly on the last word of the sentence without a space in between. In the full-text index, however, we want to include the entry {tour}, not {tour!} - hardly anyone will search for the latter. For this purpose, when setting up the full-text index we have to be able to specify that certain characters do not belong to the word. The decision is not always so easy: In a character string such as "Cuddle-Classic" which occurs in a text we have to decide whether we want to include it as an entry in the full-text index or as {cuddle} and {classic}. In the first instance our message will then only be found if an exact search is made for "Cuddle-Classic" or, for example, "*uddle-c*", in the second instance for all "classic" searches.

What we will probably keep together in spite of the occurrence of punctuation, i.e. exclude from tokenizing, are abbreviations: when AC/DC come to Germany o.i.t. (only in transit) it is probably better to have the abbreviation in the index instead of the individual letters.

Filter: by using filter operations we can both modify words when they are included in the full-text index and also completely suppress their inclusion. Known: **stop words**, at this point we can maintain a list. Moreover, we probably do not want **individual words** (Bela B.) to be in the index like this - the likelihood of confusion is too great. Using other filters we can restore **words to their basic forms** or define **replacement lists for individual characters** (e.g. in order to eliminate accents). Other filters, in turn, clear the text of XML tags.

We can set all this in the admin tool under "index configuration". We can then assign these configurations (in the knowledge builder or in the admin tool) to the character string attributes. The index configuration is organised in such a manner that filtering can take place before the word demarcation and after the word demarcation.

The full-text search does not affect the wildcard automatism of the other queries but the user may, of course, provide his input with wildcards.

1.3.3 Search pipeline

Search pipelines enable individual components to be combined to complex queries. Single components perform operations in the process, e.g.:

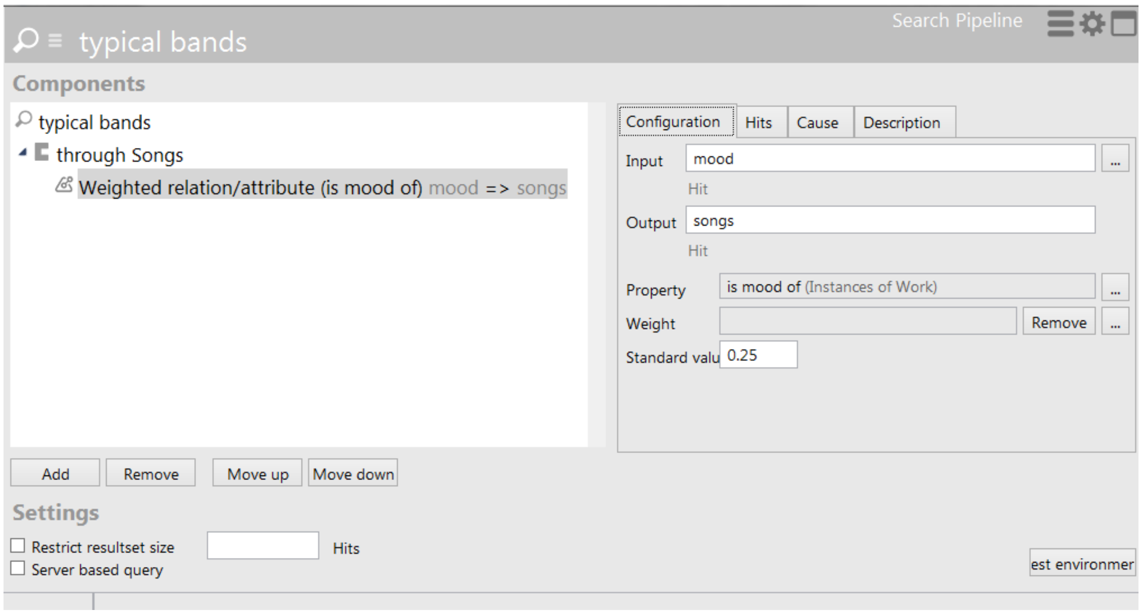
- traversing the network and thus determining the weighting
- performing structured queries and simple queries
- compiling hit lists

Every query step produces a query output (usually a number of objects). This query output may, in turn, be used as input for the following components in the pipeline.

Example

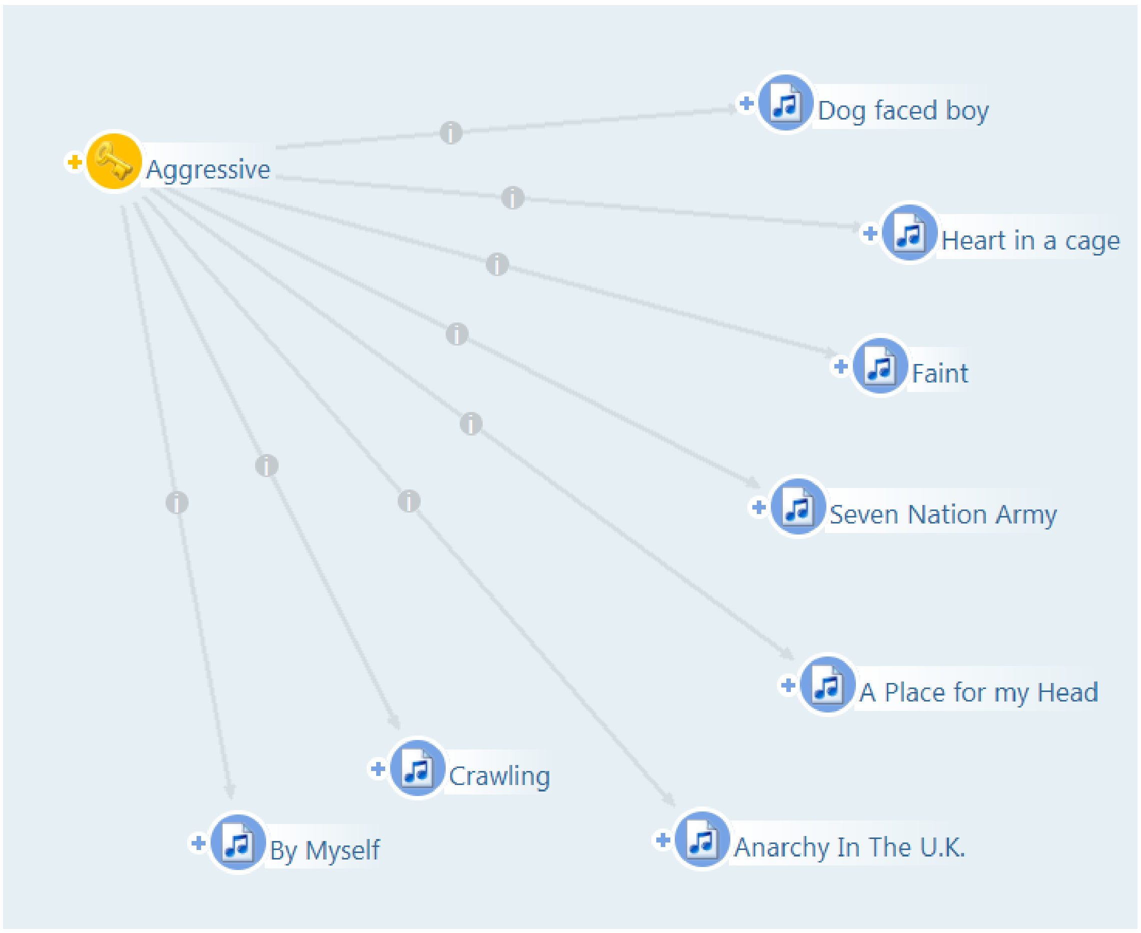
Let us assume that songs and artists from our musical network are characterised with tags named 'moods'. Based on a certain 'mood' we now want to find which bands best represent this mood.

Step 1 of our search pipeline goes from a starting mood (in this case "aggressive") via the relation *is mood of* to the songs which are assigned to the mood 'aggressive':

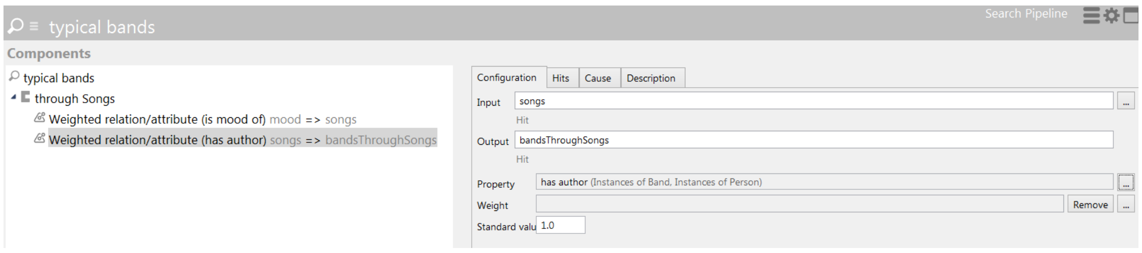


Input: mood (Hit)
Output: songs (Hit)
Property: is mood of (Instances of Work)
Weight: [Remove]
Standard valu: 0.25

☐ Restrict resultset size — Hits
☐ Server based query

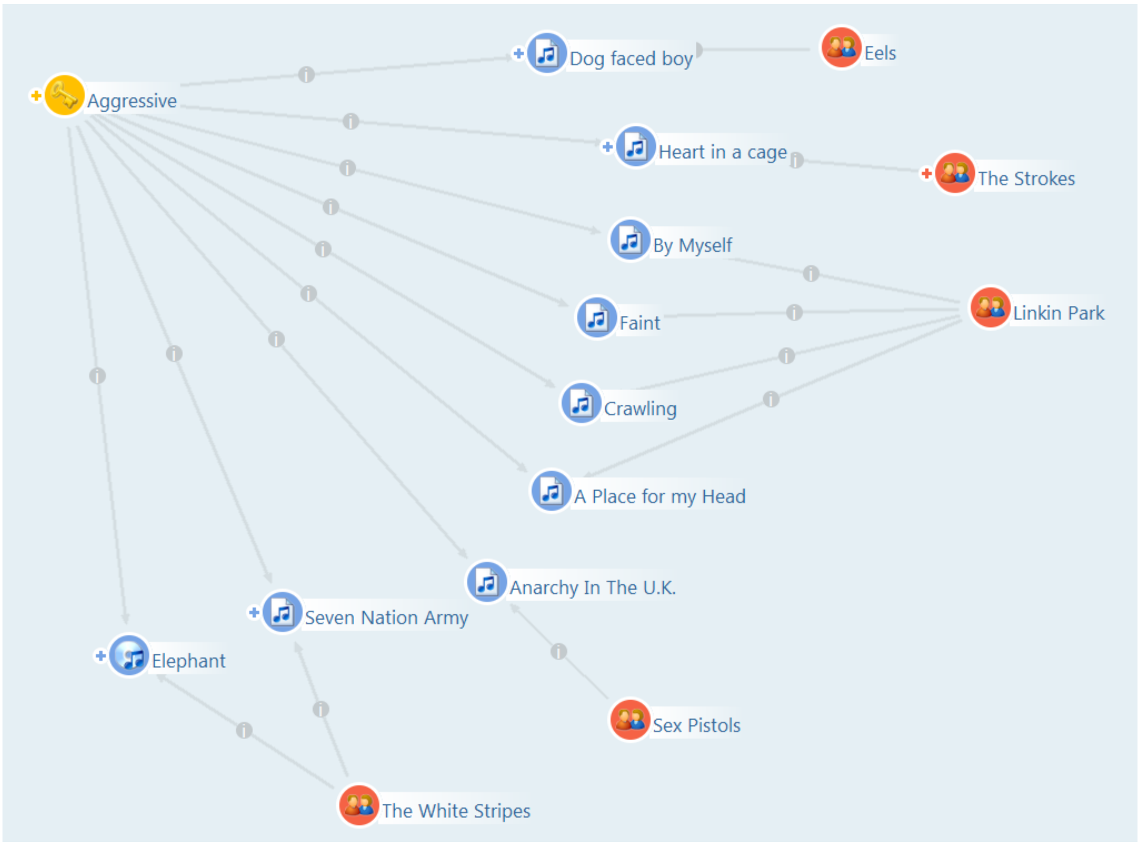


In the second step we go from the number of songs detected in the 'mood' searched for to the corresponding bands via the relation *has author*:



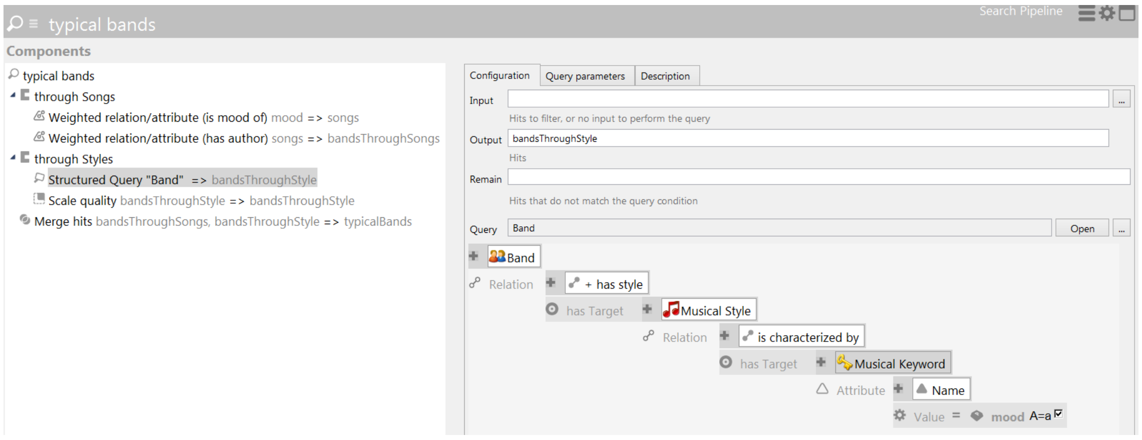
typical bands
through Songs
Weighted relation/attribute (is mood of) mood => songs
Weighted relation/attribute (has author) songs => bandsThroughSongs

Input: songs
Hit
Output: bandsThroughSongs
Hit
Property: has author (Instances of Band, Instances of Person)
Weight:
Standard valu 1.0



Now we would like to pursue a second path: from the starting point 'mood' "aggressive" to the musical directions which are characterised by aggressiveness.

Based on this number of relevant musical directions we have to go to bands which are assigned to this mood. We go down this alternative path in one step using a structured query:


typical bands

Components

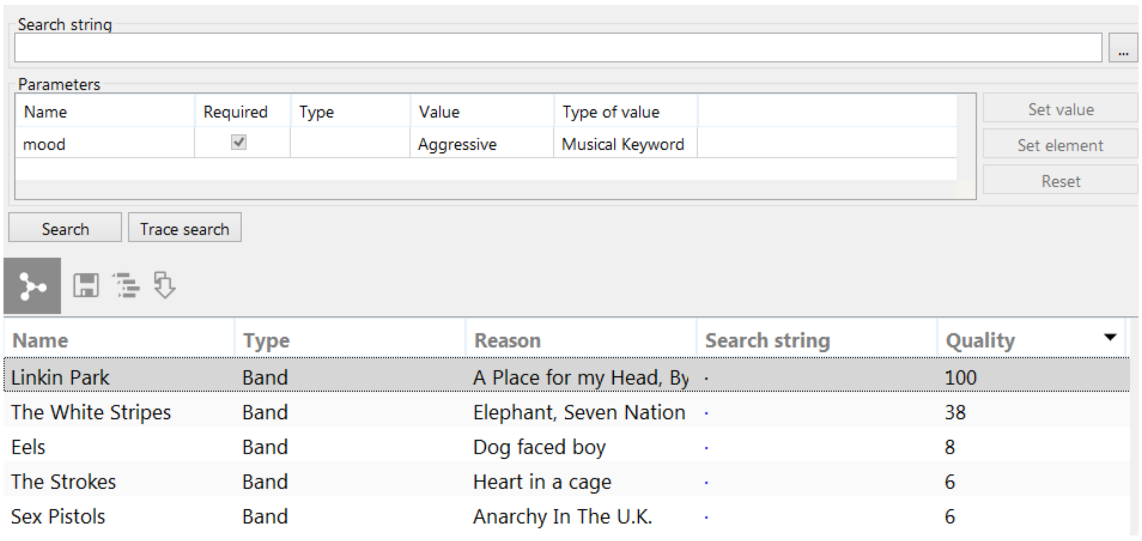
- typical bands
  - through Songs
    - Weighted relation/attribute (is mood of) mood => songs
    - Weighted relation/attribute (has author) songs => bandsThroughSongs
  - through Styles
    - Structured Query "Band" => bandsThroughStyle
    - Scale quality bandsThroughStyle => bandsThroughStyle
  - Merge hits bandsThroughSongs, bandsThroughStyle => typicalBands

Configuration | Query parameters | Description

Input: (empty) — Hits to filter, or no input to perform the query
Output: bandsThroughStyle — Hits
Remain: (empty) — Hits that do not match the query condition
Query: Band

```
Band
  Relation  + has style
    has Target  Musical Style
      Relation  is characterized by
        has Target  Musical Keyword
          Attribute  Name
            Value = mood A=a
```

From the last two steps we give the indicator "musical direction" a somewhat lower weighting and compile the outputs at the end:



Name	Required	Type	Value	Type of value
mood	✓		Aggressive	Musical Keyword

Name	Type	Reason	Search string	Quality
Linkin Park	Band	A Place for my Head, By	·	100
The White Stripes	Band	Elephant, Seven Nation	·	38
Eels	Band	Dog faced boy	·	8
The Strokes	Band	Heart in a cage	·	6
Sex Pistols	Band	Anarchy In The U.K.	·	6

The steps are processed in sequence: the input and output define which step will continue to work with which hit list. For instance, in this manner we would be able to begin again with 'mood' on our alternative path.

The principle of weightings

It was the goal to give the bands we obtained as outputs a ranking which shows how great their semantic "proximity" is to the mood aggressive. In particular, we influence ranking in this search at two positions: right at the end we weight bands higher in the summary which are found both via their musical direction and their songs. In this case this applies to Linkin Park and the Sex Pistols. The higher ranking of Linkin Park results from the fact that again and again different songs lead to Linkin Park with the mood aggressive. Since more aggressive songs from Linkin Park are in the database, Linkin Park should be 'rewarded' with a higher ranking.

1.3.3.1 Configuration of search pipelines

The individual components of a search pipeline are depicted in the main window in the box *components* in the order of sequence in which they are implemented.

Using the button *add* we can insert a new component at the end of the existing components.

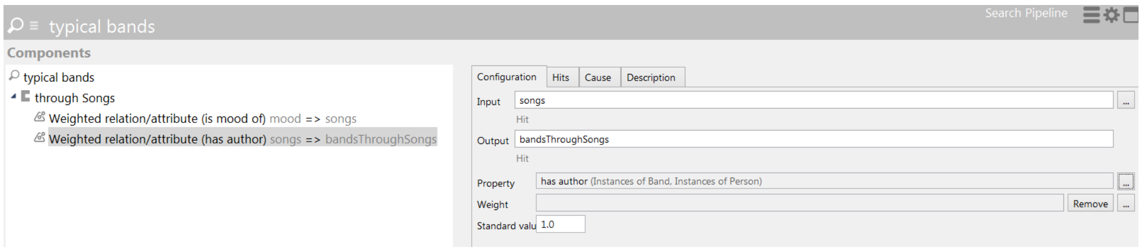
Grouping with blocks serves only to provide an overview, e.g. for the compilation of several components in a functional area of the search pipeline.

The order of sequence of the steps can be changed using the button *upwards* and *downwards* or with drag & drop.

Using the button *remove* the component selected will be removed, to include all possible sub components. The configuration for the component selected is displayed on the right-hand side of the main window.

Configuration of a component

A selected component may be configured on the right-hand side of the main window using the tab "configuration": most components need **input**. This usually comes from a previous step. In this way, the first components in our example pass on the output under the variable "songs" to the next component, this then goes from there to the bands and, in turn, gives the output to the next steps as "bandsThroughSongs":



typical bands — Search Pipeline

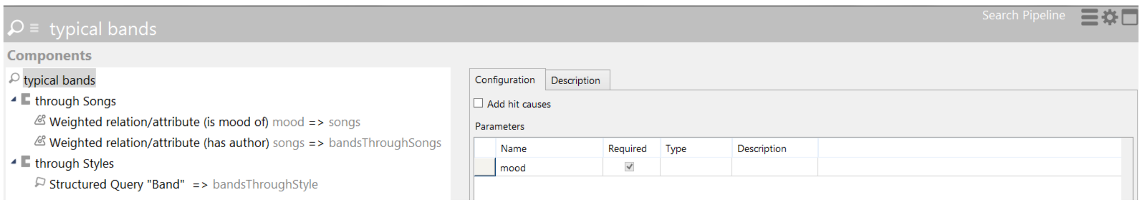
Components
- typical bands
- through Songs
  - Weighted relation/attribute (is mood of) mood => songs
  - Weighted relation/attribute (has author) songs => bandsThroughSongs

Configuration | Hits | Cause | Description

Input	songs
Output	bandsThroughSongs
Property	has author (Instances of Band, Instances of Person)
Weight	
Standard valu	1.0

Using the input and output variable we can also, in later steps, re-set to the initial output which we saw in the last paragraph.

We define the input parameters as global settings for the search. Under the name which we assign here we can then access these inputs in our search pipeline during each step. In our example the input parameter for identifying typical bands is the mood.



typical bands — Search Pipeline

Components
- typical bands
- through Songs
  - Weighted relation/attribute (is mood of) mood => songs
  - Weighted relation/attribute (has author) songs => bandsThroughSongs
- through Styles
  - Structured Query "Band" => bandsThroughStyle

Configuration | Description

☐ Add hit causes

Parameters

Name	Required	Type	Description
mood	✓		

Some components enable a deviation from the standard processing sequence:

Individual processing: elements of a quantity, e.g. hits from a search may be processed individually. This is practical if you want to assemble an individual environment of adjacent objects for search hits. In individual processing each element of the configured variable in the single hit is saved and implemented in the sub components.

Condition for set parameters: this component only carries out further sub components if predefined parameters have been set, whereby the value is insignificant. New sub components may be added by using the 'add' tab.

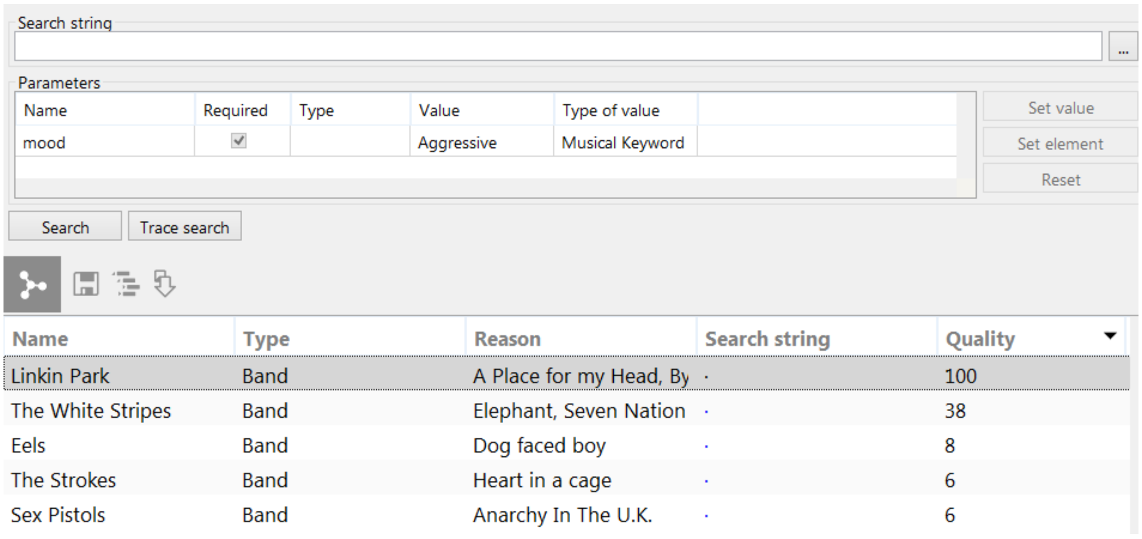
KPath condition: By using a KPath condition we can determine that the sub components may only then be implemented if a condition expressed in KPath is fulfilled. If the condition is not fulfilled the input will be adopted. KPath is described in the manual for KScript.

Output: we can stop the search at any stage and return the input. This component is also useful for testing the search pipeline.

The block components which we have also used in our example group a lot of individual steps. In order to maintain an overview in extensive configurations we can also change the name of the component using the tab "description" and add a comment as well. Neither the block components nor the description have any functional effects. Both of them only serve the ‘legibility’ of the search pipeline.

Test environment

Using the test environment in the menu we can analyse the functioning of the search. The upper section contains the search input and the lower section the output. The input may be a search text or an element from the knowledge network, depending on which required and optional input parameters we have globally defined in the search pipeline. If we wish to enter an element from the knowledge network as a starting point we select the corresponding parameter line and add an attribute value or a (knowledge network) element, depending on the type.

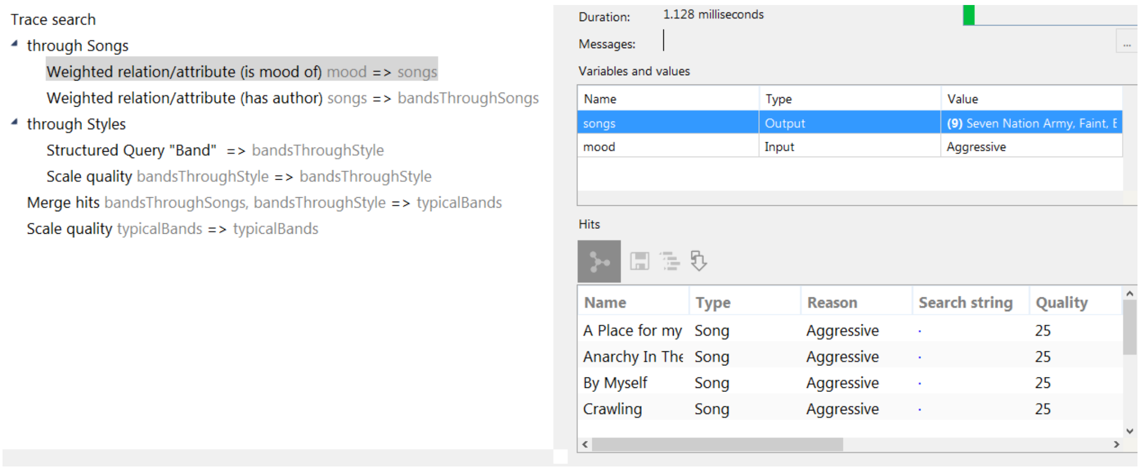

Search string

Parameters

Name	Required	Type	Value	Type of value
mood	✓		Aggressive	Musical Keyword

Name	Type	Reason	Search string	Quality
Linkin Park	Band	A Place for my Head, By	·	100
The White Stripes	Band	Elephant, Seven Nation	·	38
Eels	Band	Dog faced boy	·	8
The Strokes	Band	Heart in a cage	·	6
Sex Pistols	Band	Anarchy In The U.K.	·	6

On the tab *Trace search* a report of the search will be displayed. This primarily consists of the configuration of the output variables and the duration of the implementation of each component. The log begins with the pre-configured variables (search string) as well as active users.


Trace search
- through Songs
  - Weighted relation/attribute (is mood of) mood => songs
  - Weighted relation/attribute (has author) songs => bandsThroughSongs
- through Styles
  - Structured Query "Band" => bandsThroughStyle
  - Scale quality bandsThroughStyle => bandsThroughStyle
- Merge hits bandsThroughSongs, bandsThroughStyle => typicalBands
- Scale quality typicalBands => typicalBands

Duration: 1.128 milliseconds

Messages:

Variables and values

Name	Type	Value
songs	Output	(9) Seven Nation Army, Faint, E
mood	Input	Aggressive

Hits

Name	Type	Reason	Search string	Quality
A Place for my	Song	Aggressive	·	25
Anarchy In The	Song	Aggressive	·	25
By Myself	Song	Aggressive	·	25
Crawling	Song	Aggressive	·	25

Calculation possibilities

In the case of some components it is possible to summarise several quality values into one single quality value - e.g. in "summarise hits" but also when traversing the relations (see example above). For this purpose the following methods of calculation are available:

- addition / multiplication
- arithmetic average / median
- minimum / maximum
- ranking

The option "ranking" is then always suitable when we want to assemble an overall picture from individual references, e.g. if we want to calculate many paths, at least partially independent paths - at the end still with differing lengths - to an "overall proximity". Using the ranking calculation we ensure that all positive references (all independent paths) keep increasing their similarity without exceeding 100%.

In the search pipeline quality values are always specified as floating point numbers. The value 1 thereby corresponds to a quality of 100%.

1.3.3.2 The single components

Weighted relations and attributes

Starting with semantic objects, we can traverse the graph in this step and collect relation targets or attributes. To do so, we have to specify the type of relation or attribute.

Please note: Only collected targets are output, rather than the initial set. If this is to be displayed, we then have to summarize the input and output hits.

When traversing a relation, the weighting of hits can be influenced. Let s assume we want to semantically enhance the “initial mood” of our example search with “sub-moods”. But this indirection is to be reflected in a ranking: Connections to bands that run via sub-moods are not supposed to count as much as connections via an initial mood. For this purpose, we can assign a fixed value - e.g. 0.5 - for moving along the relation and then merge input quality, e.g. multiply it. In this case the sub-moods added in this step count only half as much as direct moods.

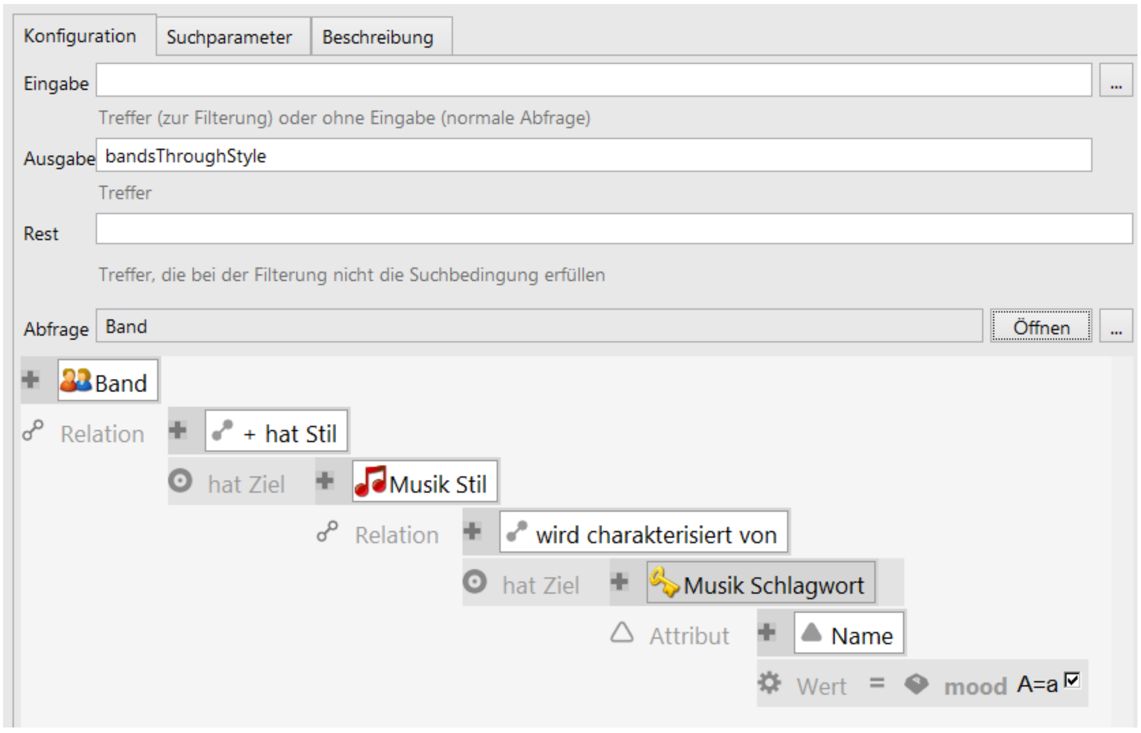
Instead of assigning a fixed weight for moving along the relation, we could also read the value from a meta-property of the basic type float of the selected relation. If the attribute is not available or no attribute has been configured, the default value is used. The value should be between 0 and 1. The hit generation can be configured in detail: For relations, you have the option to also generate a new hit for the source of the relation (rather than for the relation target).

If a relation has been selected as a property and hits are generated for relation targets, we can also transitively trace the relation. The quality value is reduced with each step until the value falls below the specified threshold. If an object has more relations than specified under maximum fan-out, these relations are not traced. The higher the damping factor, the more the quality value is reduced.

Structured query

We can use structured query components to either search for semantic objects/go from an existing set to other objects (as with the weighted relation) or filter a set.

If we search for objects, we forward our initial set of hits from a preceding step into the search via the parameter name. (In general: Within the expert query, variables of the search pipeline, e.g. search string, can be referenced via parameters.) In this case, the input stays blank.


Konfiguration | Suchparameter | Beschreibung

Eingabe:
Treffer (zur Filterung) oder ohne Eingabe (normale Abfrage)

Ausgabe: bandsThroughStyle
Treffer

Rest:
Treffer, die bei der Filterung nicht die Suchbedingung erfüllen

Abfrage: Band

```
Band
  Relation  + hat Stil
    hat Ziel  Musik Stil
      Relation  wird charakterisiert von
        hat Ziel  Musik Schlagwort
          Attribut  Name
            Wert  =  mood A=a ☑
```

For filtering, in contrast, we specify a set of objects as the input. The output contains all objects that meet the search condition. Objects that do not meet the search condition can optionally be stored in an additional variable (Rest).

We can either define the structured query ad hoc directly in the component or we can use an existing structured query.

Please note: If an existing search is selected, no copy is created. Any changes to the structured query that we make for search pipeline purposes also modify the query for all other uses.

Query

You can use the "Query" component to execute simple queries, full text queries and other search pipelines. Simple queries and full text queries receive a string here, e.g. the *search string*: This is a parameter that is available for processing user input in all search pipelines. The hit list of the called search fills the output of this component.

By integrating search pipelines into other search pipelines, we can factorize sub-steps that occur more frequently. Several parameters and entire sets of hits ("hit collections") can be transferred to other search pipelines. With integrated search pipelines we can also replace several parameters, that is, we can access of every sub-step output in the integrated search and vice versa. If we go to selected parameters, we can also rename them, for example, if we want to use a set of hits from the integrated search but have already used the name.

Alternatively, we can also apply only some of the parameters from the integrated search in order to avoid such conflicts.

Summarize hits

We can use this component to summarize different sets of hits (“hit collections”) from previous steps. The following methods are available for summarizing:

Join: All hits that occur in at least one of the sets are output as a result

Intersect: Only hits that occur in all sets are output as the result.

With joins and intersects, a semantic object can occur in several sets of hits (“hit collections”) and has to be computed as one total hit with a new hit quality. The aforementioned calculation options are also available here.

Difference: One of the sets of hits (“hit collections”) must also be defined as the initial set. The other sets are deducted from this set.

Symmetric difference: The result set consists of objects that are included in exactly one subset (= everything except for the intersection, when there are two sets).

Three different types of total hits can be generated. The selection is particularly relevant if partial hits include additional information.

- To generate uniform hits, remember the original hits as the cause: New hits are generated that contain the original hit as the cause.
- Extend original hits: The original hit is copied and receives a new quality value. If there are several hits for the same semantic object, a random hit is selected.
- Generate uniform hits: A new hit is generated. The properties of the original hit are lost.

Summarize partial hits

During individual processing you frequently have to general a total set from partial hits. The component “Summarize partial hits” enables you to do so. This summarizes all hits of one or more partial sets of hits (“hit collections”). The difference to Summarize hits is that summarizing only take parts at the end, not for every partial hit set. This is relevant in particular when calculating the quality because summarizing hits would return incorrect values, in particular for the median.

Script

A search pipeline can contain a script (JavaScript or KScript). This can access the variables of the search pipeline. Furthermore, a script can transfer several parameters to the search pipeline. The result of the script is used as the result of the component.

JavaScript API and KScript are described in separate manuals.

Copy quality from attribute value

For hits, we can copy the quality value from an attribute of the semantic object. If the object does not have exactly such an attribute, the default value is used. The value should be between 0 and 1.

Compute total quality from weighted qualities

To adapt the quality of a search hit, it can be useful to compute a total value from individual partial qualities. The qualities must be available as numeric values. These values are used to calculate a new total quality.

Compute overall quality of hits

You can use the individual quality values of a set of hits to compute a total quality.

Restrict quality

We can restrict sets of hits (“hit collections”) to hits whose quality value falls within specified limits (minimum or maximum). Normally, we want to filter out hits that fall below a certain quality threshold.

Restrict number of hits

If the total number of a set of hits is to be restricted, we can add the component “restrict number of hits”. We can use the option “Do not split hits of the same quality” to prevent a random selection in case of several hits of the same quality in order to comply with the total number. We then get more hits than specified.

If some very specific cases, we can also randomly select the hits, e.g. if we have a large number of hits with the same quality and want to generate a preview.

Scale quality

Die quality values of a set of hits can be scaled. A new set of hits with scaled quality values is calculated. The calculation takes place in two steps:

1. Die quality value of the hits are limited. The threshold values can either be specified or calculated. The calculation determines the minimum and maximum value of the hits. If thresholds are specified and a hit has a quality value that falls outside of the thresholds, the value is limited to the threshold value. If you want to remove such hits, you have to execute the restrict quality component first. Example: Mapping percentage values to school grades. 30% is average, over 90% is high score. The values can be scaled linearly from 30% to 90%.
2. Following that, the quality values are scaled linearly. Hits with the minimum/maximum input value receive the minimum/maximum scaled value.

Compute hit quality

You can use a KPath expression to generate a new hit with calculated quality for a hit. The KPath expression is calculated on the basis of the input.

1.3.3.3 KPath

KPath allows addressing of objects within the Knowledge Graph. The notation is similar to XPath but differs in some respects.

The individual elements of the expression normally are separated by a slash "/". If a KPath expression begins with "/", then the evaluation starts at the root type, else it starts at the current object (depends on the context of the evaluation). If an element does not correspond to one of the listed elements of the table, it will be interpreted as a name of a sub type. Simple names can be specified without quotation marks.

When specifying a language, it must be stated according its ISO 639-2 code ("eng" for English, "ger" for German, ...).

Examples:

- **@Name**
  Attribute "Name"

- **//book\Faust/∼author**
  Relation "author" of the book "Faust"

- **//$artifact$/book{eng}**
  Sub type "book" (English name) of the type "artifact" (internal name)

- **//book\*[∼author/target()/@Name = "Goethe"]**
  All books which had been written by Goethe

1.3.3.3.1 Names

In combination with **@**, **/**, **//**, \ and \\, following kinds of names can be used:

Name	**Description**
name	Name in standard language. Without quotation marks the name needs to be begin with a letter, an asterisk or with an underscore sign. Whitespaces or special characters which are used in other expressions are not allowed. The name must comply with following regular expression: [a-zA-Z_*][∧/(){}$%{}[],∼@§#+-’"s|∧&]* (For better reading, the escape character "\" has been left out)
"*name*" ’*name*’	If the name doesn’t meet the above-mentioned requirements, it needs to be surrounded by single quotes or double quotes. Here, the backslash sign "\" serves as escape character for possibly used apostrophes, e. g. ’Wendy\’s’.
name{lang}	Name in the specified language "lang"
$*name*$, $"*name*"$	Internal name
§*name*§, §"*name*"§	System name
#ID42_1013	ID of the object

Names are not replaceable by variables and must therefore be a part of the script.

1.3.3.3.2 Operators

Numeric values can be linked by the operators +, -, * or /.

When using "*", "-" and "/", at least one white space character must surround the operator on both sides each.

Parenthesis are supported, e. g. "(5 + 3) * 4" equals the value 32.

Example: Sum of all relations between Goethe and Schiller:

```
\\Goethe/~*/size() + \\Schiller/~*/size()
```

The operator "+" also can be used to append strings:

```
    //person\Goethe + " wrote " + //book\Faust
```

leads to:

```
    Goethe wrote Faust
```

By means of the unary operator "!", a Boolean expression can be negated, e. g.:

```
!1=2
```

For some operators, an alternative notation only consisting of alphabetical characters is possible, e. g. "eq" for equality. Applying this notation, at least one white space character needs to be used between operator and operand. The expressions are case-sensitive, so operators are only recognized if written in small letters.

Possible operators are (in descending precedence):

Operator	**Alternative notation**	**Meaning**
!	not	Negation (unary operator)
*		Multiplication
/		Division
+		Addition, linking (only character strings)
-		Subtraction
<	lt	Smaller than
>	gt	Greater than
<=	le	Smaller than or equal to
>=	ge	Greater than or equal to
=	eq	Equal to
!=	ne	Not equal to
^^	xor	Exclusive or (logical operator)

&&	and	And (logical operator)
||	or	Or (logical operator)

Due to KScript basing on XML, operators like '&&', '<' or '<=' need to be written using entities like '&lt' or '&amp' instead of the character signs '<' and '&' or alternative notation needs to be used.

Example for "and":

```
<Path path="var(left) &amp;&amp; var(right)"/>
<Path path="var(left) and var(right)"/>
```

Example for "smaller than":

```
<Path path="var(left) &lt; var(right)"/>
<Path path="var(left) lt var(right)"/>
```

1.3.3.3.3 Conditions

Conditions can be specified using the following notation:

path1[path2]path3

On all elements out of path1 for which the condition path2 applies, path3 will be executed. To express the condition path2, comparative operators can be used (see preceding section). Boolean expressions can be linked with Boolean operators.

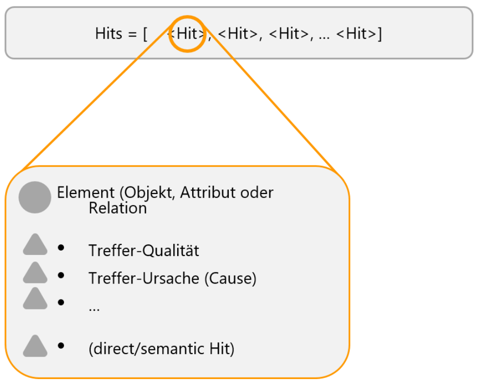
Example: Name of all books which had been written by Goethe:

```
    //book\*[~author/target()/@Name = "Goethe"]/@Name/value()
```

1.3.4 Model "hit"

The “Hit” type content model is available to ensure that search queries can be processed and transported both as quality and causes. A “Hit” can be seen as a container that summarizes the element including several properties and makes it temporarily available to the context. The contained properties can be, for example, calculated hit quality, hit cause, change log entry etc.

In search pipelines, the content models “Hit” and “Hits” are available. The “Hits” type is an array of several “Hit” elements:


Hits = [<Hit>, <Hit>, <Hit>, ... <Hit>]
Element (Objekt, Attribut oder Relation
Treffer-Qualität
Treffer-Ursache (Cause)
...
(direct/semantic Hit)

Meta-attributes of hits

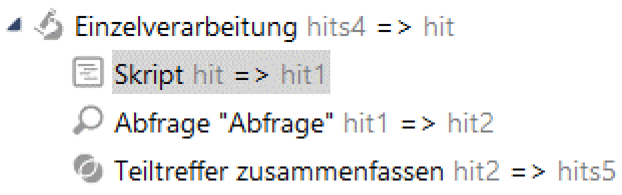
In addition to the semantic element, the following meta-attributes are transported in a hit:

- **Hit quality:** Can have a value between 0 and 1 by setting a quality in a search pipeline; the hits of a structured query receive the value 1 by default
- **Hit cause:** Refers to the input element that has led to the hit and its type
- **Hit cause (snippet):** Refers to the content or the search term that has led to the hit

For detailed information on the meta-attributes, refer to the JavaScript API.

Using hits in search pipelines

If a hit list is to be processed in a search pipeline by means of a simple query, individual processing is required because the hit list is in the form of an array: Queries can process an individual "hit" in the form of a string but not "hits" (= array). Converting a "hit" into string, in turn, can be done using a script that precedes the simple query.


Einzelverarbeitung hits4 => hit
Skript hit => hit1
Abfrage "Abfrage" hit1 => hit2
Teiltreffer zusammenfassen hit2 => hits5

Example script for converting a hit into a string:

```
function search(input, inputVariables, outputVariables) {  return input.element().name(); }
```

Using hits in tables

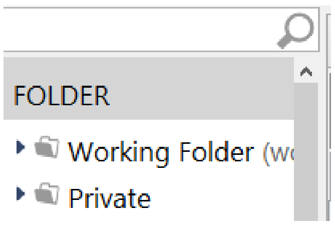
The "Use hits" option is available in the column element configuration of a table. This option determines whether the entire hit element (semantic element + meta-attributes) or only the semantic element is to be forwarded to display query results.

Processing hits in tables via a script

If the query results are to be processed further using a script, the "Use hits" option determines whether the query result is supposed to be treated as a hit: The script is forwarded either $k.SemanticElement or $k.Hit as a JavaScript object.

1.3.5 The search in the knowledge builder

With the exception of the structured queries which are created in the folders and also implemented there, all searches in the header of the knowledge builder are made available for internal usage.



For this purpose we have to drag & drop a pre-configured search only into the search box of the header of the knowledge builder. If this contains several searches to be selected from you can select the desired search from the pull-down menu by clicking on the magnifier icon. The search input box always contains the search mode which was last carried out.

We can remove the search using the global settings where we can also change the sequence of the various searches in the menu.

1.3.6 Special cases

1.3.6.1 Volltext search Lucene

The full-text search may also alternatively be carried out via the external indexer Lucene. The search configuration is then analogue to the standard full-text search, i.e. attributes may, in turn, be configured in the search which are also connected to the Lucene index; the search process is also analogue. In order to configure the Lucene indexer connection we hereby refer you to the corresponding chapter in the admin manual.

1.3.6.2 Search with regular expressions

Regular expressions are a powerful means of searching through databases for complex search expressions, depending on the task concerned.

Search with regular expressions	**hit**
The [CF]all	the call, the fall
Car.	cars
Car.*	cars, caravans, Carmen, etc.
[∧R]oom	doom, loom, etc. (but not room)

As search inputs, i-views supports the standard also known from the standard known from Perl which, for example, is described in the Wikipedia article for regular expression.

1.3.6.3 Search in folders

The search in folders is carried out in names of folders and their contents:

- folders whose name matches the search input
- fodlers which contain objects which match the search input
- expert searches which contains elements which match the search input
- scripts in which the search input appears
- rights and trigger definitions which contain elements which match the search input

Using the search input #obsolete, you can target your search for deleted objects (e.g. searching in rights and triggers). When configuring the search the number of folders to be searched through can be limited. Furthermore, the option "search for object names in folders" may be deactivated. This is helpful if you do not want to search for semantic objects in folders because in the case of extensive fodlers (e.g. saved search results) the search for object names may take a very long time.

1.3.6.4 Query for duplicates

After imports or due to other reasons such as quality assurance it can be necessary to check for duplicates semantic elements. To do so, a specially configured structured query can be used.

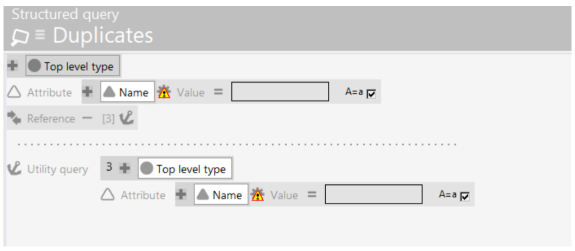
Note: Because the structured query shown in the following example refers to elements of the whole Knowledge Graph without further type restrictions (objects of top level type), executing the query can take an unusually amount of time. It therefore is adviced to restrict the query to the most exactly subtype as possible.

In principle, the structured query searches for **different** objects that have identical values for their identifying attributes (here: objects with identical names).

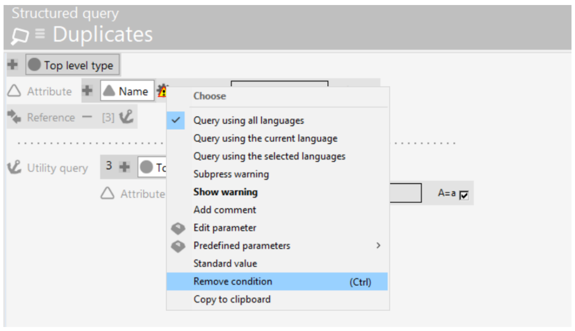
The query for duplicates can be configured as follows:

1. Create a query for objects of the subtype in question. Add the identifying attribute as condition (here: primary name).

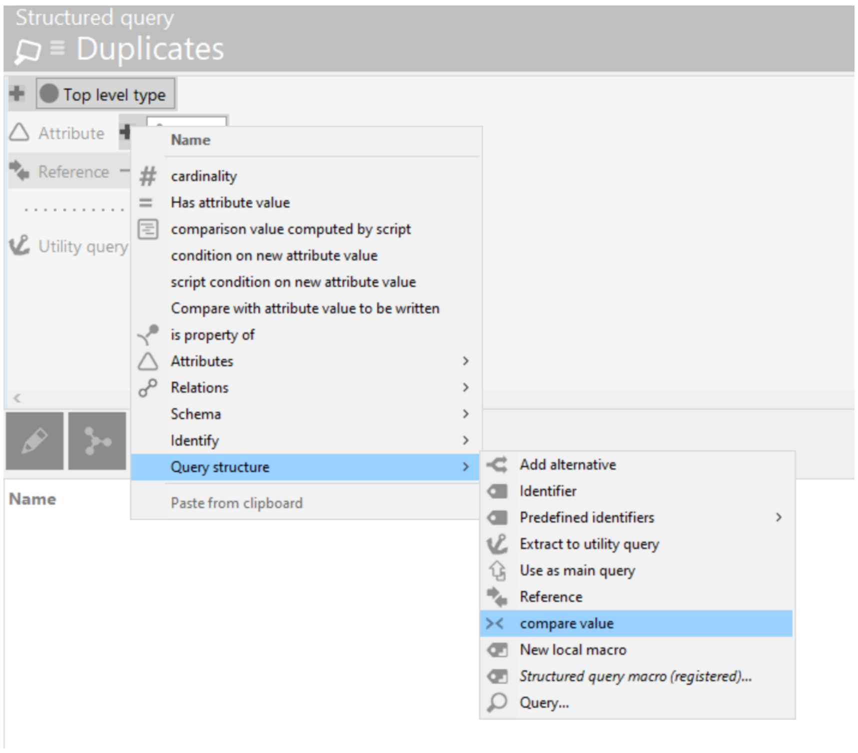
2. Depending on the object, create a utility query. Use a negative reference (comparison operator "is not") to make sure that only different objects will be found:



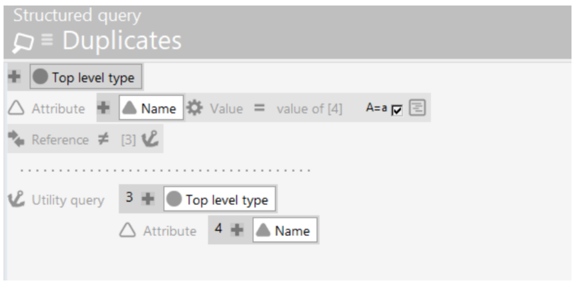
3. For comparing the attribute values against each other, the value fields need to be removed first:



4. After having removed the value fields, the context menu offers the option "compare values". Add this condition and select the identifying attribute of the utility query to compare against:



Result: Structured query for searching duplicates.

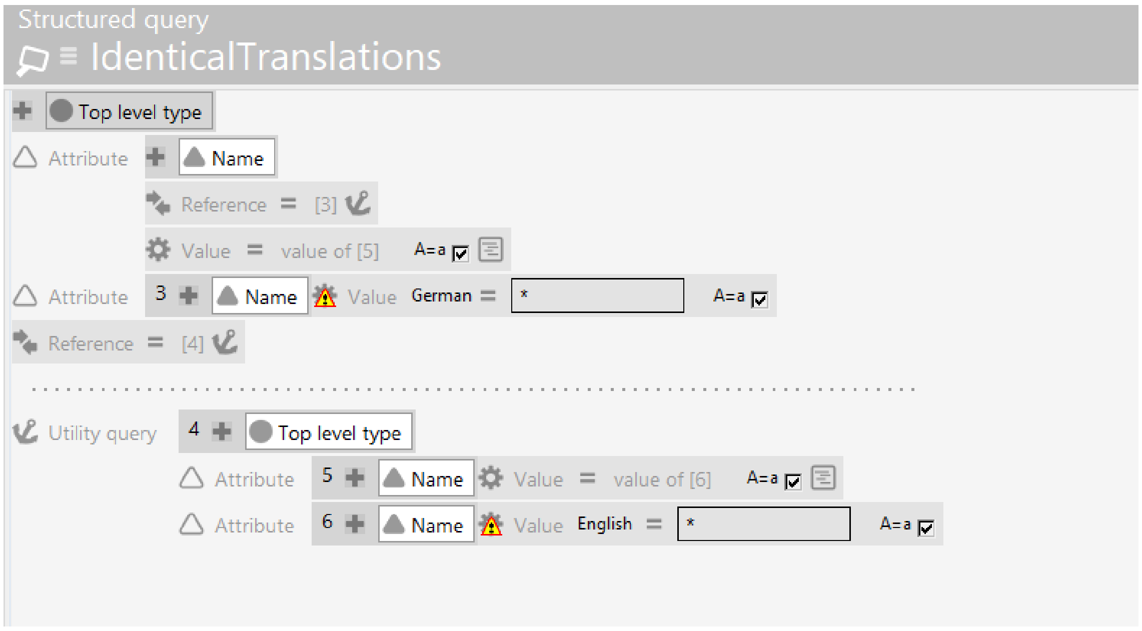


1.3.6.5 Query for identical translations

Similar to the query for duplicates, the query for objects with identical translations makes use of references and attribute value comparisons.

Note: Because the structured query shown in the following example refers to elements of the whole Knowledge Graph without further type restrictions (objects of top level type), executing the query can take an unusually amount of time. It therefore is adviced to restrict the query to the most exactly subtype as possible.

The difference between this query and the query for duplicates is that it is all about one and the same object this time, with the additional condition of identical attribute values in different translation layers of one and the same attribute:


Structured query
IdenticalTranslations

Top level type
Attribute Name
Reference = [3]
Value = value of [5] A=a
Attribute 3 Name Value German = * A=a
Reference = [4]

Utility query 4 Top level type
Attribute 5 Name Value = value of [6] A=a
Attribute 6 Name Value English = * A=a

1.4 Folder and registration

Along with the objects and their properties, we also build a variety of other elements in a typical project: we define, for example, queries and imports/exports, or write scripts for specific functions. Everything that we build and configure can be organized in folders.

The folders are shared with everyone else working on the project. If we do not wish to do so, we can file things in the private folder, for example for test purposes. This is only visible for the respective user.

A special form of the folder is the collection of semantic objects, in which we can file objects manually, for example for processing at a later date. To do so, we simply move them to the folders using Drag&Drop, and there are also operations to, for example, define result lists in folders.

In the moment we delete one of these objects within the Knowledge Graph, it is also deleted from the collection. If a semantic element is removed by clicking on "Remove from folder", it is only removed from the collection but still exists within the Knowledge Graph. If the actions "Delete" or "Delete selected elements" or "Delete all elements inside the folder" is used, the semantic element actually is deleted and from the Knowledge Graph and therefore is not accessible anymore within the collection.

Caution: The action "Remove from folder" has different functionalities, depending on the context of use: In the case of folders containing import mappings, the action "Remove from folder" actually means completely deleting the respective import mapping!

In the case of collections of semantic objects with more than 100 entries, for reasons of performance, no determination of the table configuration that best suits the content occurs. We can, however, request this by means of the context menu function “Determine configuration of the object list” when necessary.

Registration

Queries, scripts, etc. can call each other (a query can be integrated into another query or into a script, while, in turn, a script can be called from a search pipeline). There are registration keys for this purpose, with which we can equip queries, import/export mappings, scripts and even collections of semantic objects and organizing folders to ensure they provide other configurations with a functionality. The registration key must be distinct. Everything that has a registration key is automatically added to the “Registered objects” folder, or in the subfolder that corresponds to its type

Shift, copy, delete

Let us assume we have a folder called “Playlist functions” in our project. This might contain an export, some scripts and a structured query “similar songs”, which we would like to use in a REST service. The moment we give the structured query a registration key, it is added to the folder “Registered objects” (“Technical” section). This means the structured query “similar songs” appears in the folder “Registered object” under “Query”. It also remains there when we remove it from our project subfolder “Playlist functions”. If we remove the registration key, the query will automatically disappear from the registry.

The basic principle when deleting or removing: Queries, imports, scripts can be in one or several folders at the same time, and at least one folder must contain them. Only when we, for example, remove our query from the last folder will it actually be deleted. Only then does i-views also request a confirmation of the delete action. The same applies for removal of the registration key.

If we wish to delete the query in one step, regardless of the number of folders that contain it, we can only do this from the registry.

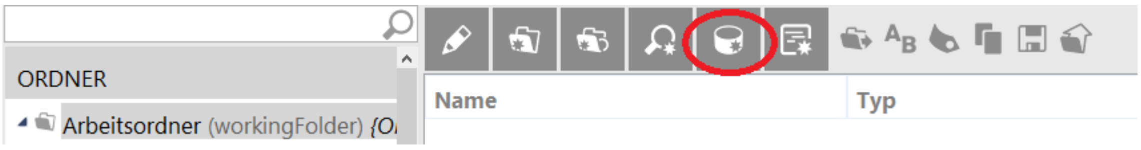
Folder settings

We can define quantitative limits for query results, folders and object lists (lists of the specific objects in the main window of the Knowledge Builder when an object type is selected on the left-hand side) in the folder settings. Automatic query up to the number of objects specifies up to which number of objects the contents of the folders or the object lists are shown without any further interaction by the user. If the limit set there is exceeded, the list initially remains empty, and the message “Query not executed” appears in the status bar. Executing a search without an input in the input line shows all objects. This, at least, until the second limit has been reached: Maximum number of query outputs, maximum number of outputs in object lists - in this instance high values - there is actually no result when these values are exceeded, queries must be restricted, e.g. in object lists in which we also have the beginning of the name in the input box.

1.5 Import and export

By mapping data sources we can import data to i-views from structured sources and export objects and their properties in structured form. The sources can be Excel/CSV tables, databases or XML structures.

The functions for import and export overlap to the most part and are therefore all available in a single editor. In order to access functions for import and export, it is first necessary to select a folder (e.g. the working folder). There the "New mapping of a data source" button can be used to select a data source for the import or export.



Alternatively, you can find the button on the "TECHNICAL" tab under "Registered objects" -> "Mappings of data sources".

The following interfaces and file formats are available for import and export:

- CSV/Excel file
- XML file
- MySQL interface
- ODBC interface
- Oracle interface
- PostgreSQL interface
- For the exchange of user IDs, a standard LDAP interface has been implemented.

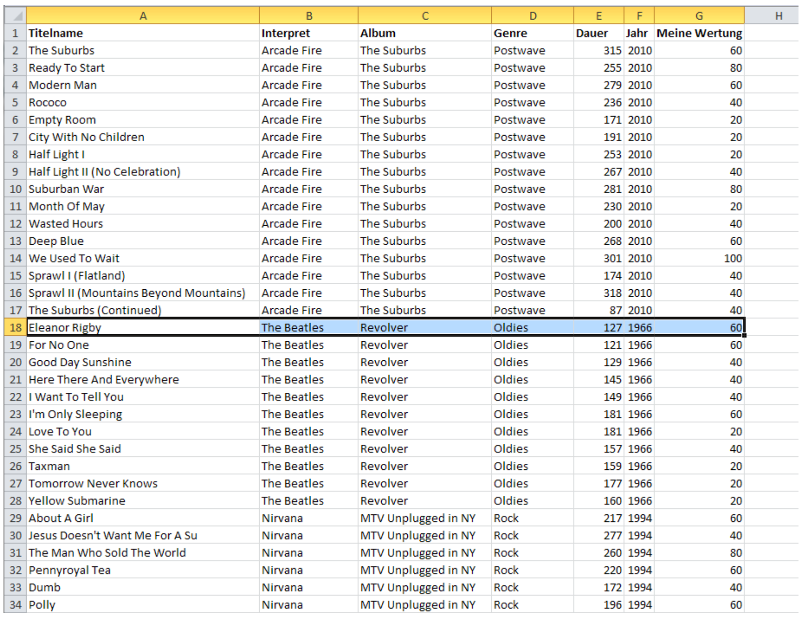
The following section uses a CSV file to describe how to create a table-oriented import/export. As all imports/exports apart from XML imports/exports are table-oriented and the individual data sources differ only in terms of their configuration, the example for the mapping of the CSV file can also be applied to the mapping of other databases and file formats.

1.5.1 Mapping of data sources

CSV files are the default exchange format for spreadsheet applications such as Excel. CSV files consist of individual rows of plain text in which columns are separated by a fixed, predefined character such as a semicolon.

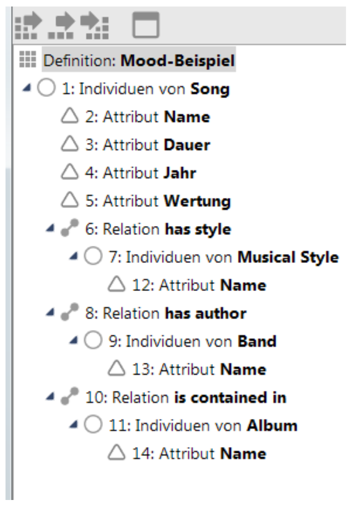
1.5.1.1 Principle of operation

Let s use a table with songs as a first example: When the table is imported, we would like to create a new, specific object of the type song for each line. The contents of columns B to G become attributes of the song, or relations to other objects:

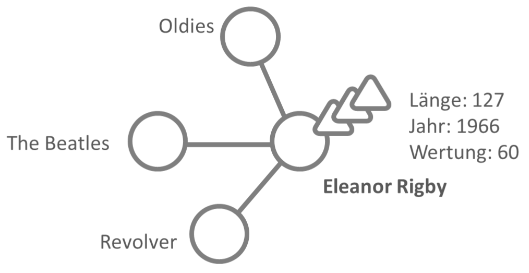


	A	B	C	D	E	F	G	H
1	Titelname	Interpret	Album	Genre	Dauer	Jahr	Meine Wertung	
2	The Suburbs	Arcade Fire	The Suburbs	Postwave	315	2010	60	
3	Ready To Start	Arcade Fire	The Suburbs	Postwave	255	2010	80	
4	Modern Man	Arcade Fire	The Suburbs	Postwave	279	2010	60	
5	Rococo	Arcade Fire	The Suburbs	Postwave	236	2010	40	
6	Empty Room	Arcade Fire	The Suburbs	Postwave	171	2010	20	
7	City With No Children	Arcade Fire	The Suburbs	Postwave	191	2010	20	
8	Half Light I	Arcade Fire	The Suburbs	Postwave	253	2010	20	
9	Half Light II (No Celebration)	Arcade Fire	The Suburbs	Postwave	267	2010	40	
10	Suburban War	Arcade Fire	The Suburbs	Postwave	281	2010	80	
11	Month Of May	Arcade Fire	The Suburbs	Postwave	230	2010	20	
12	Wasted Hours	Arcade Fire	The Suburbs	Postwave	200	2010	40	
13	Deep Blue	Arcade Fire	The Suburbs	Postwave	268	2010	60	
14	We Used To Wait	Arcade Fire	The Suburbs	Postwave	301	2010	100	
15	Sprawl I (Flatland)	Arcade Fire	The Suburbs	Postwave	174	2010	40	
16	Sprawl II (Mountains Beyond Mountains)	Arcade Fire	The Suburbs	Postwave	318	2010	40	
17	The Suburbs (Continued)	Arcade Fire	The Suburbs	Postwave	87	2010	40	
18	Eleanor Rigby	The Beatles	Revolver	Oldies	127	1966	60	
19	For No One	The Beatles	Revolver	Oldies	121	1966	60	
20	Good Day Sunshine	The Beatles	Revolver	Oldies	129	1966	40	
21	Here There And Everywhere	The Beatles	Revolver	Oldies	145	1966	40	
22	I Want To Tell You	The Beatles	Revolver	Oldies	149	1966	40	
23	I'm Only Sleeping	The Beatles	Revolver	Oldies	181	1966	60	
24	Love To You	The Beatles	Revolver	Oldies	181	1966	20	
25	She Said She Said	The Beatles	Revolver	Oldies	157	1966	40	
26	Taxman	The Beatles	Revolver	Oldies	159	1966	20	
27	Tomorrow Never Knows	The Beatles	Revolver	Oldies	177	1966	20	
28	Yellow Submarine	The Beatles	Revolver	Oldies	160	1966	20	
29	About A Girl	Nirvana	MTV Unplugged in NY	Rock	217	1994	60	
30	Jesus Doesn't Want Me For A Su	Nirvana	MTV Unplugged in NY	Rock	277	1994	40	
31	The Man Who Sold The World	Nirvana	MTV Unplugged in NY	Rock	260	1994	80	
32	Pennyroyal Tea	Nirvana	MTV Unplugged in NY	Rock	220	1994	60	
33	Dumb	Nirvana	MTV Unplugged in NY	Rock	172	1994	40	
34	Polly	Nirvana	MTV Unplugged in NY	Rock	196	1994	60	

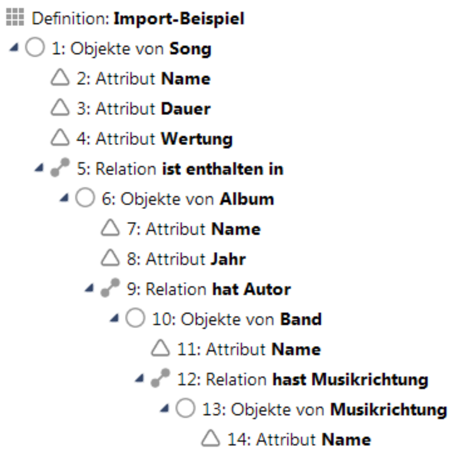
Using the song as a basis, we build up the structure of attributes, relations and target objects that should be created by the import (left-hand side). An object of type song is created this way for row 18, for example, with the following attributes and relations:



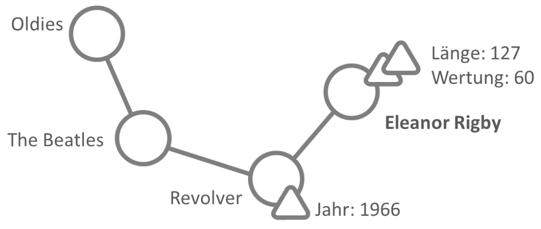
Definition: **Mood-Beispiel**
- 1: Individuen von **Song**
  - 2: Attribut **Name**
  - 3: Attribut **Dauer**
  - 4: Attribut **Jahr**
  - 5: Attribut **Wertung**
  - 6: Relation **has style**
    - 7: Individuen von **Musical Style**
      - 12: Attribut **Name**
  - 8: Relation **has author**
    - 9: Individuen von **Band**
      - 13: Attribut **Name**
  - 10: Relation **is contained in**
    - 11: Individuen von **Album**
      - 14: Attribut **Name**



We can, however, also decide to distribute the information from the table in a different way, for example allocate the year of release and artist to the album, and in turn the genre to the artist. A row still forms a context, however this does not mean it must belong to exactly one object:



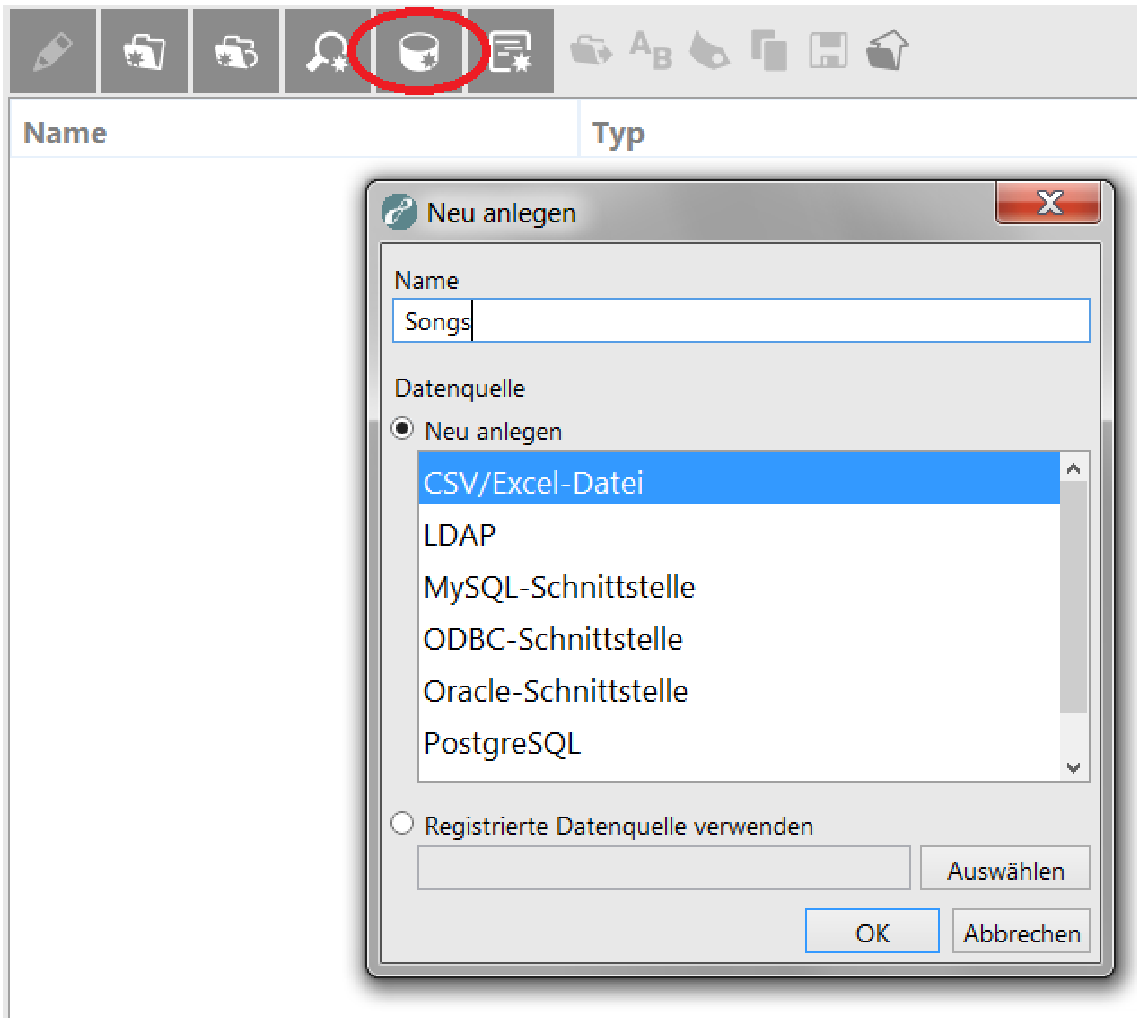
Definition: **Import-Beispiel**
- 1: Objekte von **Song**
  - 2: Attribut **Name**
  - 3: Attribut **Dauer**
  - 4: Attribut **Wertung**
  - 5: Relation **ist enthalten in**
    - 6: Objekte von **Album**
      - 7: Attribut **Name**
      - 8: Attribut **Jahr**
      - 9: Relation **hat Autor**
        - 10: Objekte von **Band**
          - 11: Attribut **Name**
          - 12: Relation **hast Musikrichtung**
            - 13: Objekte von **Musikrichtung**
              - 14: Attribut **Name**



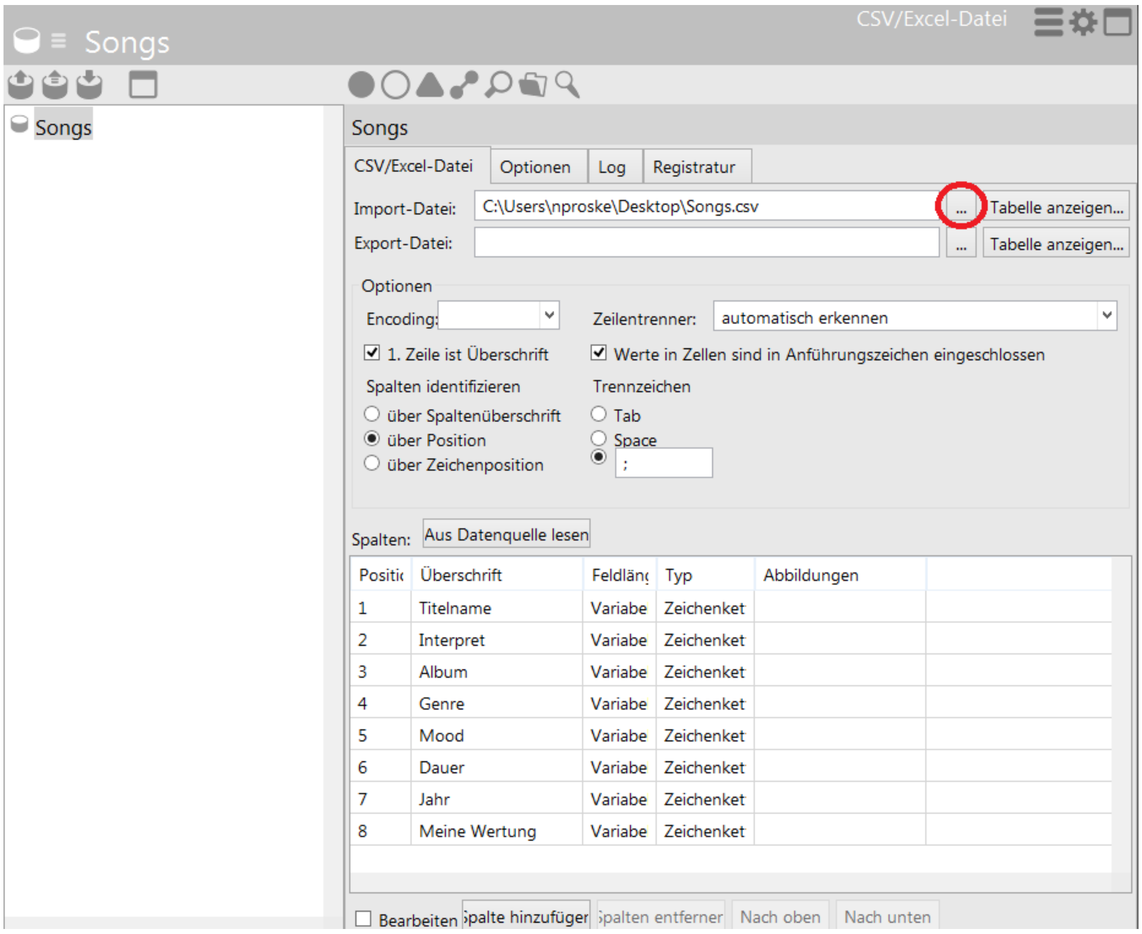
Everywhere that we build up new, specific objects and relation targets in our example, we must always specify at least one attribute for this object, in this case the respective name attribute that allows us to identify the corresponding object.

1.5.1.2 Data source - selection and options

Once we have selected the "*New mapping of a data source*" button, a dialog opens which we must use to specify the type of data source and the mapping name. If we have already registered the data source in the semantic graph database, then we will now find it in the selection menu at the bottom.



By pressing “OK” as confirmation, the editor for the import and export opens. We can specify the path of the file we wish to import under “Import file”. Alternatively, we can also select the file using the button to the right of it. As soon as the file has been selected, the column headings and their positions in the table are exported and shown in the field at the bottom right. The “*Read from data source*” button can read out the columns again in the event of any changes to the data source. The column “Mappings” shows us the respective attribute to which the respective column of the table is mapped later on.



Songs — CSV/Excel-Datei

Songs

Songs

CSV/Excel-Datei | Optionen | Log | Registratur

Import-Datei: C:\Users\nproske\Desktop\Songs.csv ... Tabelle anzeigen...

Export-Datei: ... Tabelle anzeigen...

Optionen

Encoding: Zeilentrenner: automatisch erkennen

☑ 1. Zeile ist Überschrift ☑ Werte in Zellen sind in Anführungszeichen eingeschlossen

Spalten identifizieren
- ○ über Spaltenüberschrift
- ● über Position
- ○ über Zeichenposition

Trennzeichen
- ○ Tab
- ○ Space
- ● ;

Spalten: Aus Datenquelle lesen

Positi	Überschrift	Feldläng	Typ	Abbildungen
1	Titelname	Variabe	Zeichenket	
2	Interpret	Variabe	Zeichenket	
3	Album	Variabe	Zeichenket	
4	Genre	Variabe	Zeichenket	
5	Mood	Variabe	Zeichenket	
6	Dauer	Variabe	Zeichenket	
7	Jahr	Variabe	Zeichenket	
8	Meine Wertung	Variabe	Zeichenket	

☐ Bearbeiten | Spalte hinzufügen | Spalten entfernen | Nach oben | Nach unten

The structure of our example table corresponds to the full default settings, so that there is nothing else to factor in under the menu item *Options*. CSV files can, however, exhibit structures that are very different, which must be factored in using the following setting options:

Encoding: The character encoding of the import file is defined here. This provides ascii, ISO-8859-1, ISO-8859-15, UCS-2, UTF-16, UTF-8 and Windows-1252 for selection. If nothing has been selected, the default setting that corresponds to the operating system in use is applied.

Line separator: In most cases, the setting "detect automatically", which is also selected by default, is sufficient. However, should the user establish that line breaks are not being identified correctly, then the corresponding, correct setting should be selected manually. This provides *CR* (*carriage return*), *LF* (*line feed*), *CR-LF* and *None* for selection. The standard used to encode the line break in a text file is *LF* for Unix, Linux, Android, Mac OS X, AmigaOS, BSD and others, *CR-LF* for Windows, DOS, OS/2, CP/M and TOS (Atari), and *CR* for Mac OS up to Version 9, Apple II and C64.

1st line is heading: It may the case that the first line does not include a heading, and the system must be notified of this by removing the checkmark set by default next to "*1st line is heading*".

Values in cells are surrounded by quotation marks is selected so that the quotation marks are not included in the import when this is not wanted.

Identify columns: Whether the columns are identified using their heading, the position or the character position must be specified, as otherwise the table cannot be captured correctly.

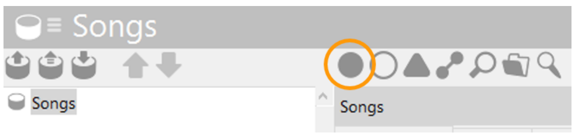
Separator: If a different separator than the default semicolon is used, this must also be specified when the column is not identified using the character position.

Moreover, the following rules apply: If a value in the table contains the separator or a line break, the value must be placed in double quotation marks. If the value contains one quotation mark, this must be doubled (»""«).

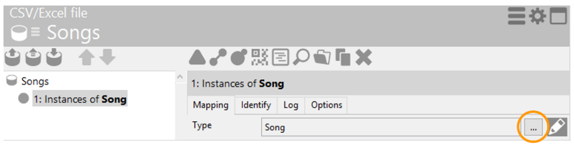
1.5.1.3 Definition of target structure and mappings

1.5.1.3.1 The object mapping

We will now start setting up the target structure that should be produced in the semantic graph database. In our example, we are starting with object mapping of the songs. In order to map a new object, we must press the "New object mapping" button.



The next step is to specify the type of object for import.



Type Song

There are further specific settings in the options tab of the object mapping.

With objects of all subtypes: If the checkbox is set to "With objects of all subtypes", the import also includes objects from all subtypes of "Song". Since this is usually desired, the checkmark is set here by default.

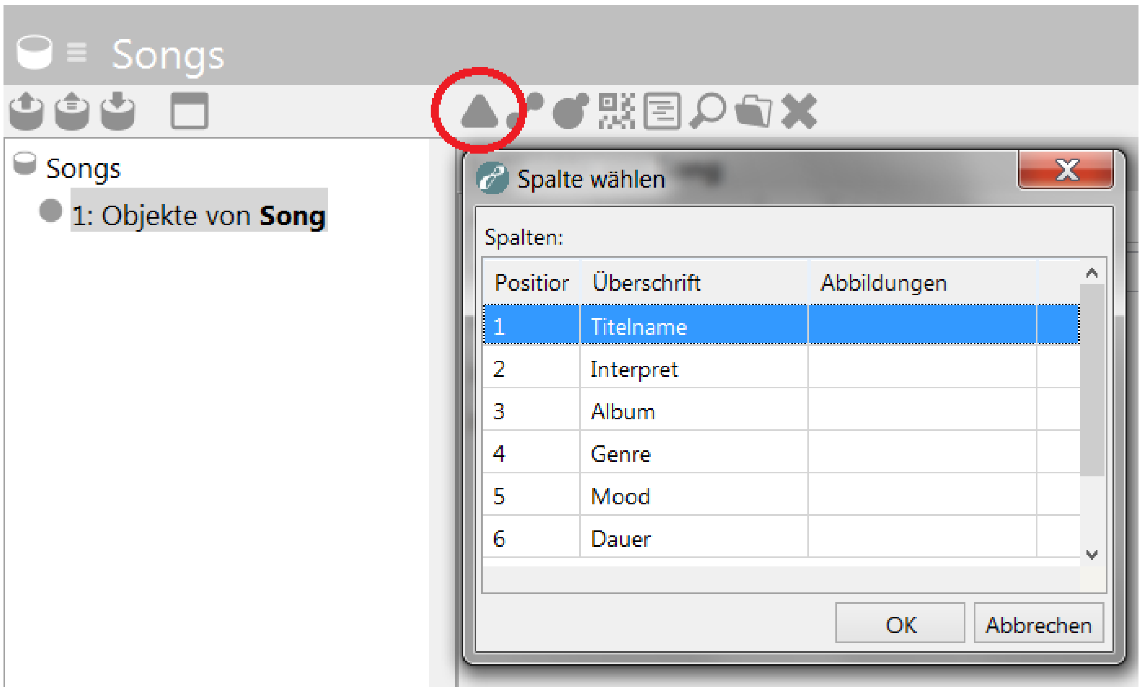
Exact type is specified by the following mapping: If the exact type to which the object is to be created is identified in the import source, this can be mapped here via the "New..." button. It must be a subtype of the type specified in the tab "Mapping".

Allow multiple objects: It is possible that the knowledge network already contains several objects with correspondent identifying properties (correspondent names). If the import mapping needs to be referred to these objects, an ambiguity conflict occurs. If you set the checkmark here, the import for all these objects is going to be performed disregarding the ambiguity.

If you do not set the checkmark, the import will not be carried out for the multiple occuring objects and instead the user will be informed that the importer cannot uniquely identify the object.

1.5.1.3.2 The attribute mapping / Identifying objects

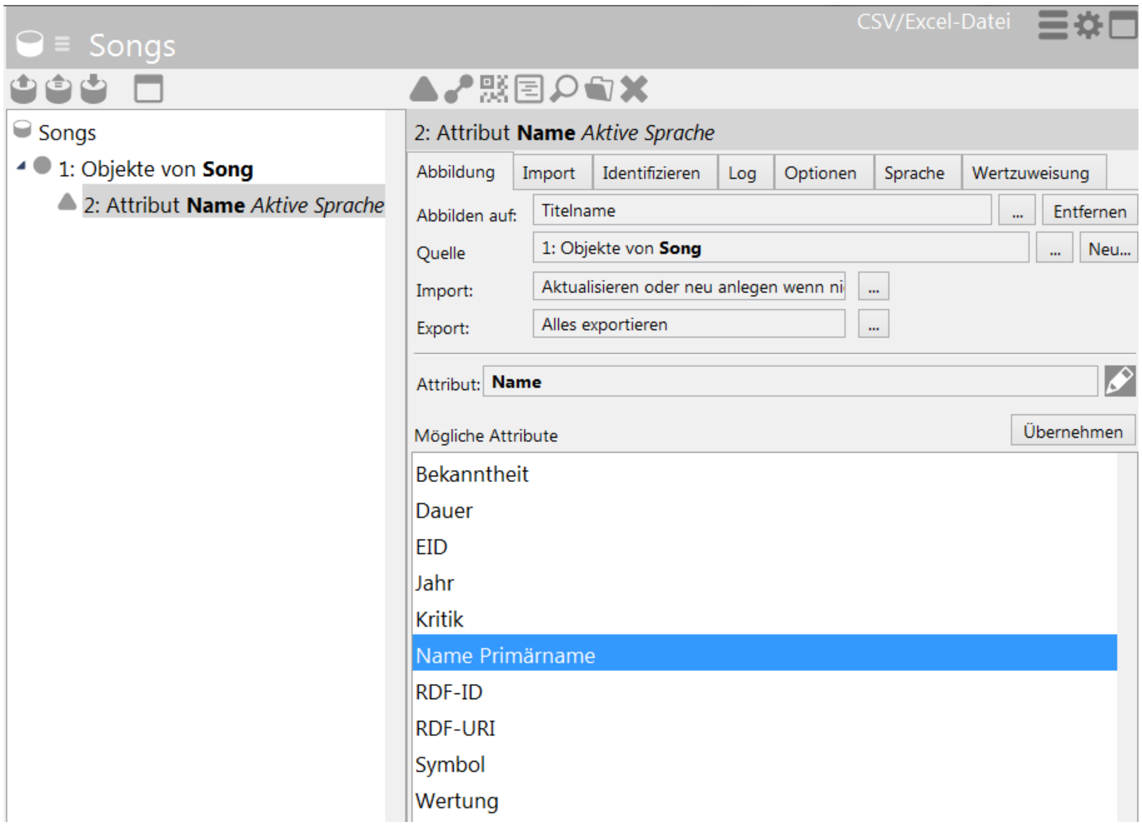
Now we want to link the information in the table to the object mapping of the songs. Attributes for individual songs are represented along with relations. In order to first create the track name for a song in the mapping, we add an attribute to the object mapping for song. Clicking on the "*New attribute mapping*" button opens a dialog, which must be used to select the relevant column from the table to be imported.



Spalten:

Position	Überschrift	Abbildungen
1	Titelname	
2	Interpret	
3	Album	
4	Genre	
5	Mood	
6	Dauer	

As this attribute is the first one we created for the object mapping of songs, it is then automatically mapped to the name of the object, as the name is usually the most commonly used attribute.



2: Attribut **Name** *Aktive Sprache*

Abbilden auf:	Titelname
Quelle	1: Objekte von **Song**
Import:	Aktualisieren oder neu anlegen wenn ni
Export:	Alles exportieren
Attribut:	**Name**

Mögliche Attribute

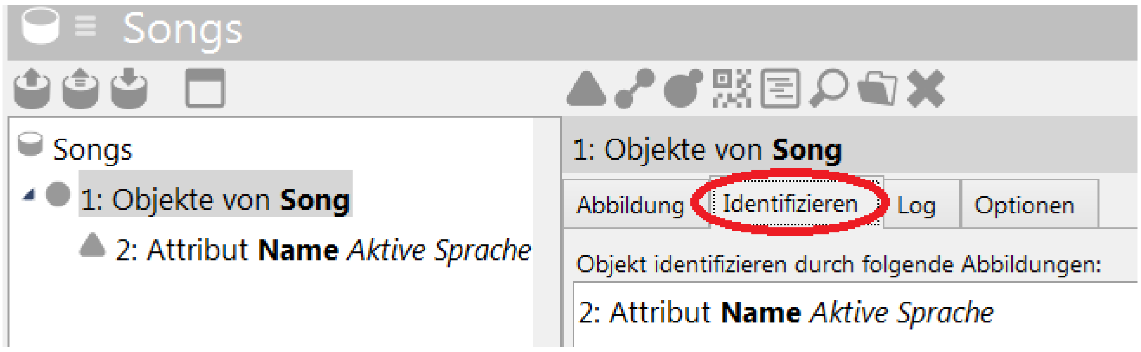
Bekanntheit
Dauer
EID
Jahr
Kritik
Name Primärname
RDF-ID
RDF-URI
Symbol
Wertung

Note: The first attribute created for an object is also used automatically in the **identification of the object**.

An object must be identified by at least one attribute - by its name or its ID, or by a combination of multiple attributes (as with the first and last name and date of birth of a person) - it should already exist so that it can be unambiguously found in the semantic graph database. This prevents unwanted duplicates from being created during import.

Note: Meta-Attributes at relations can also be imported. Here it is ensured that both the relation source and the relation target are specified and identified, otherwise the relation is ignored by the importer.

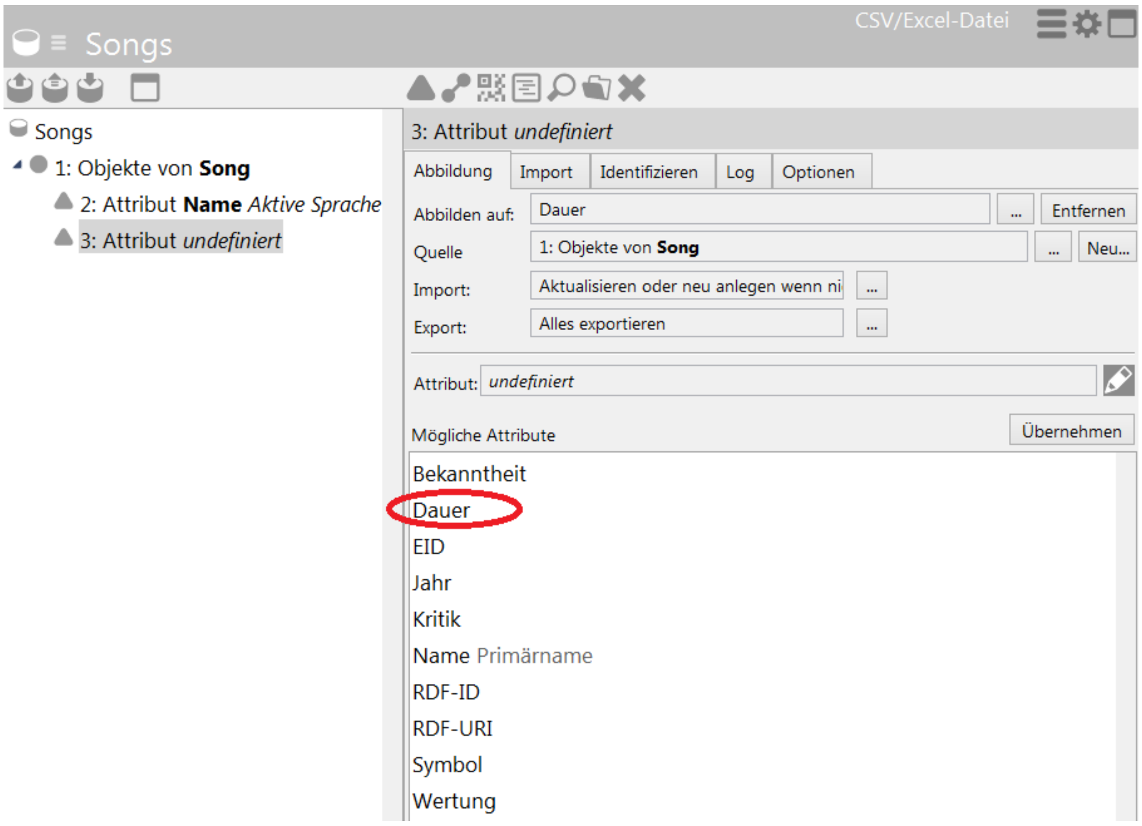
In the “*Identify*“ tab it is possible to subsequently change the attribute identifying the object, or to add multiple attributes. In addition, it is possible to specify whether the values should be matched in a case-sensitive fashion, and the query should return identical values (without index filter / wildcards). The latter is relevant if filters or wildcards are defined in the index that specify, for example, that a hyphen should be omitted from the index. The term would not be found with a hyphen if the search took place only via the index; in this case, a checkmark would be needed here so the search only finds the exactly identical value.



Objekt identifizieren durch folgende Abbildungen:

2: Attribut **Name** *Aktive Sprache*

Now we can add further attributes to object mapping that do not need to contribute towards identification, e.g. the length of a song - and this is once again done via the “New attribute mapping” button. (Please note: first the object mapping “objects of song” must be selected again.) Now we select the “Length” column from the table to be imported. This time we have to manually select the attribute to be mapped to the “Length” column. The field on the bottom right contains the list of all possible attributes defined in the schema that are available to us for objects of the “song” type, among them also the “length” attribute.



CSV/Excel-Datei

Songs

- Songs
  - 1: Objekte von **Song**
    - 2: Attribut **Name** *Aktive Sprache*
    - 3: Attribut *undefiniert*

3: Attribut *undefiniert*

Abbilden auf:	Dauer	Entfernen
Quelle	1: Objekte von **Song**	Neu...
Import:	Aktualisieren oder neu anlegen wenn ni	
Export:	Alles exportieren	

Attribut: *undefiniert*

Mögliche Attribute

Bekanntheit
Dauer
EID
Jahr
Kritik
Name Primärname
RDF-ID
RDF-URI
Symbol
Wertung

Mapping of translations

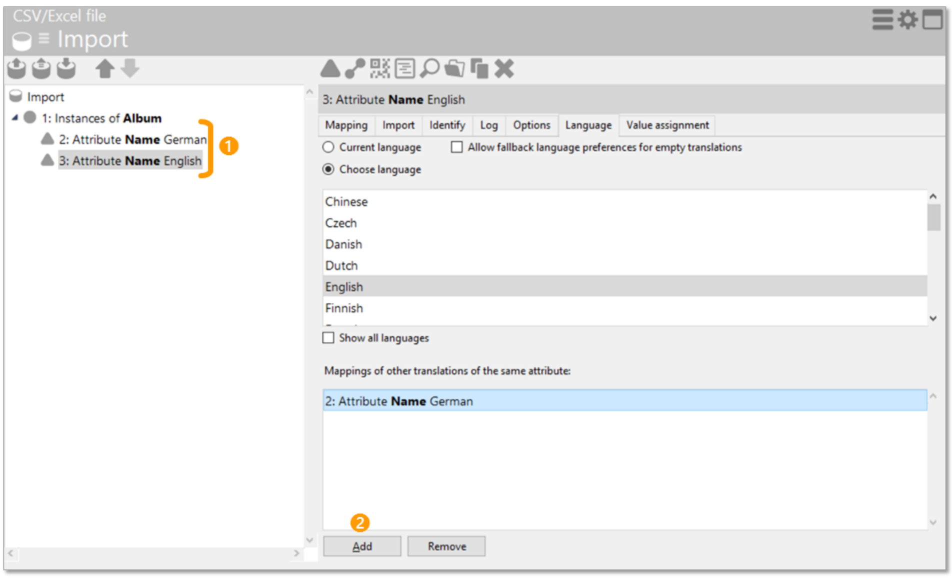
For string attritbutes with translations, e.g. the primary name of objects, we can define in which language the value needs to be imported.

If an attribute mapping is created for a translated attribute, the import language automatically is set to the "Current language". The current language equals the language in which the Knowledge Builder has been started (which at the same time is the language of the user interface).

If the import needs to be done in another language than the current language, this can be specified by selecting the tab "Language" and then by selecting a language of the list, which then becomes the chosen language for the attribute mapping.

In case of an import source containing several translations of one and the same attribute (within the same line), these values can be imported within one import mapping simultaneously.

The simultaneous import of translations for an attribute is done as follows:



Import
1: Instances of **Album**
2: Attribute **Name** German
3: Attribute **Name** English

3: Attribute **Name** English

( ) Current language [ ] Allow fallback language preferences for empty translations
(●) Choose language

Chinese
Czech
Danish
Dutch
English
Finnish

[ ] Show all languages

Mappings of other translations of the same attribute:

2: Attribute **Name** German

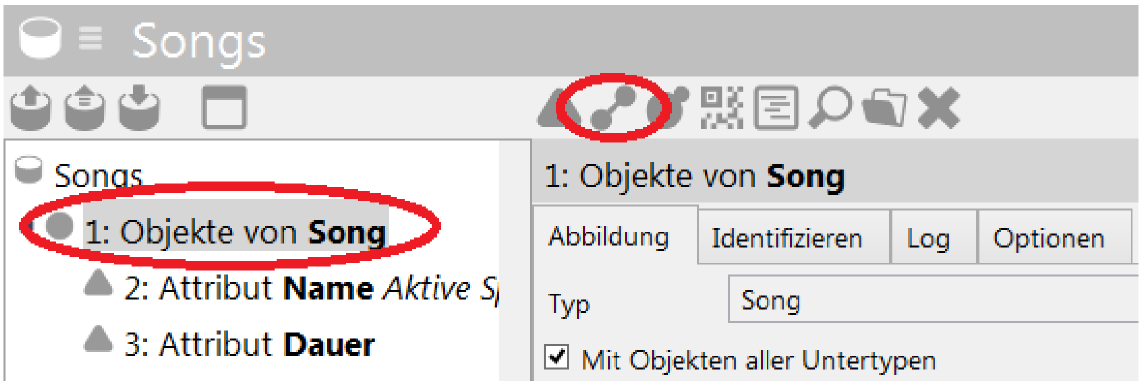
For each language, create a separate attribute mapping for the same attribute, but specify a different import language

In the "Language" tab for one of the attribute mappings, add the relevant attribute mappings of the other languages to the field "Mappings of other translations of the same attribute"

This prevents from separate attributes being created for each translation and ensures that corresponding translations are imported altogether at the same attribute.

1.5.1.3.3 The relation mapping

Next, we want to map the album on which the song is located. Since albums are concreate objects in the semantic graph database, we need the relation that connects the song and the album to do this. To map a relation, we first select the object for which the relation is defined and then click on the button “*Map new relation*.”



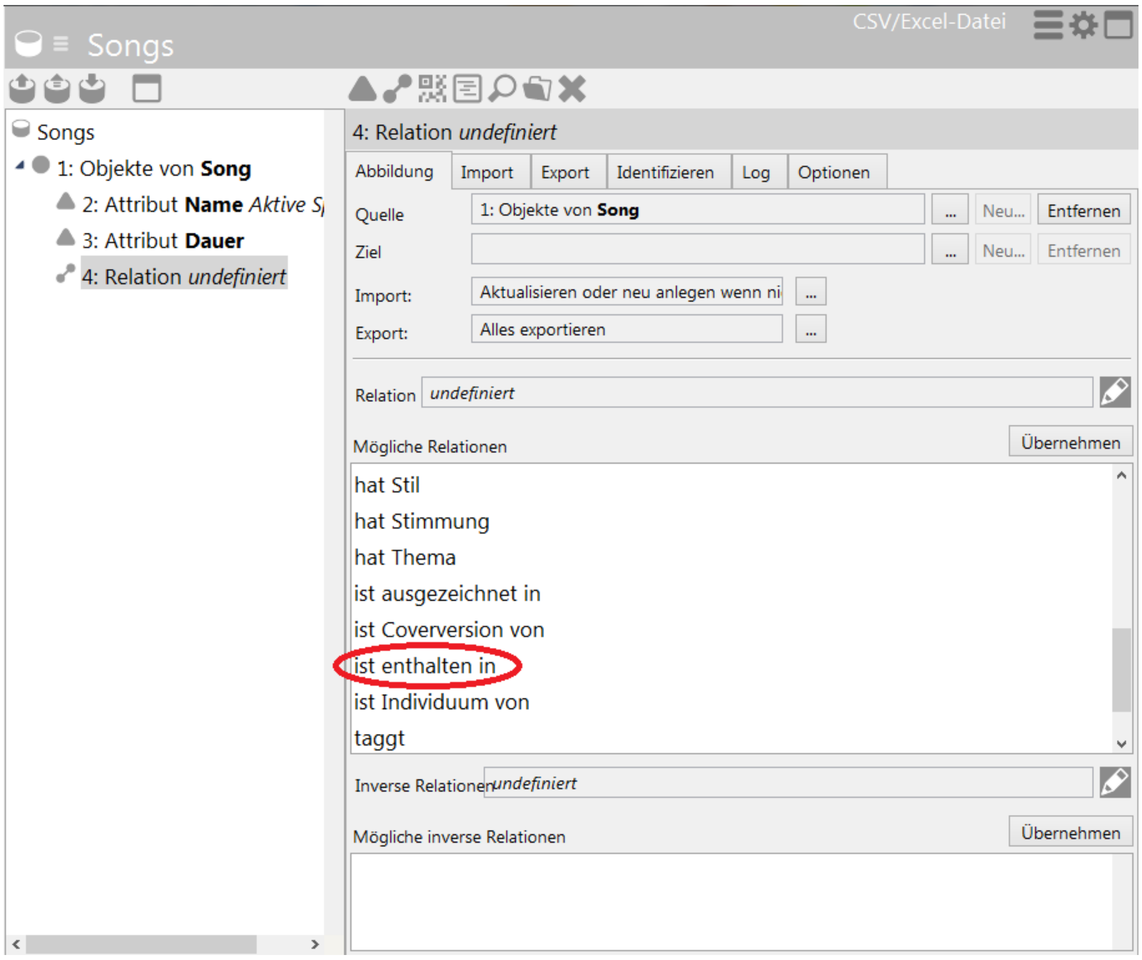
Songs
1: Objekte von **Song**
2: Attribut **Name** *Aktive S*
3: Attribut **Dauer**

1: Objekte von **Song**

Typ Song

[✓] Mit Objekten aller Untertypen

Following that, just like for attributes, we get a list of all possible relations; and the required relation “*is included in*” is naturally included.


Songs

CSV/Excel-Datei

Songs
1: Objekte von **Song**
2: Attribut **Name** *Aktive S*
3: Attribut **Dauer**
4: Relation *undefiniert*

4: Relation *undefiniert*

Abbildung | Import | Export | Identifizieren | Log | Optionen

Quelle: 1: Objekte von **Song**
Ziel:
Import: Aktualisieren oder neu anlegen wenn ni
Export: Alles exportieren

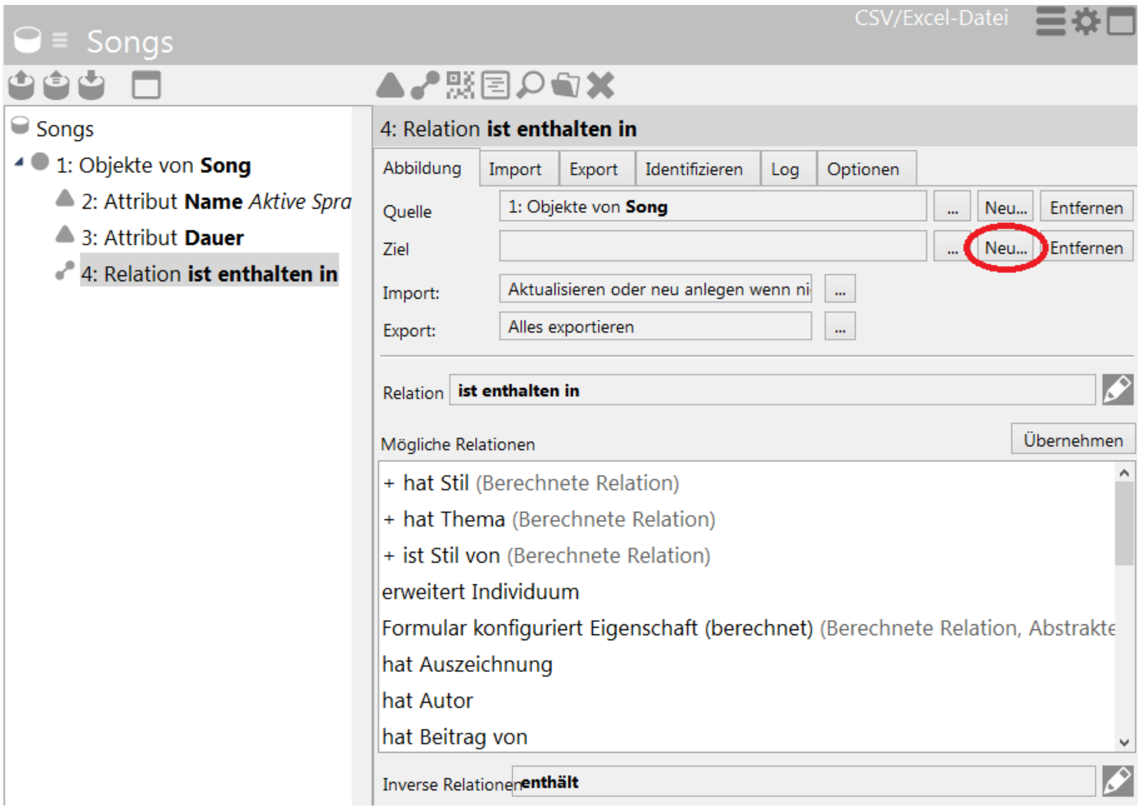
Relation: *undefiniert*

Mögliche Relationen
- hat Stil
- hat Stimmung
- hat Thema
- ist ausgezeichnet in
- ist Coverversion von
- ist enthalten in
- ist Individuum von
- taggt

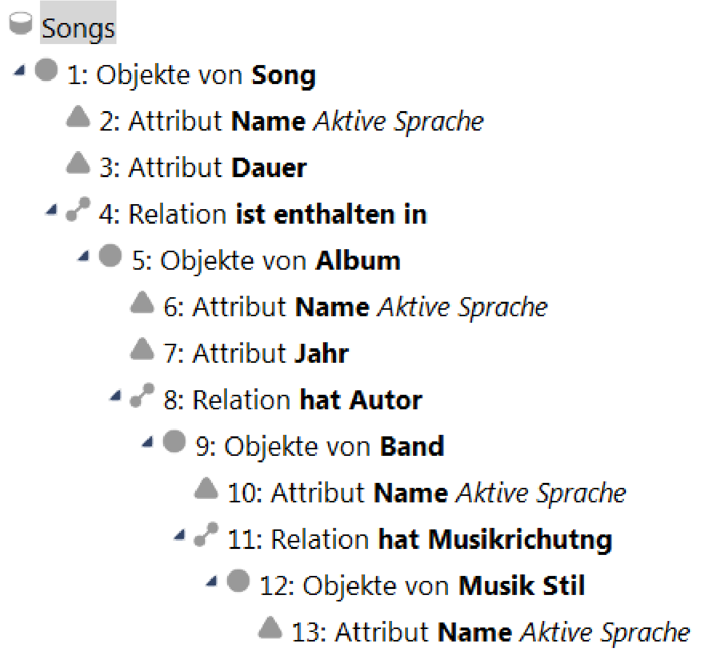
Inverse Relationen: *undefiniert*

Mögliche inverse Relationen

In the next step, we now have to define where in this table the target objects come from. A new object mapping is required for the target; this is created using the “New” button. If the type of the target object is uniquely identified in the schema, it is copied automatically. If not, a list of possible object types appears.



For new object mappings, we then once again have to select the attribute that identifies the target object etc. This creates the target structure of the import.

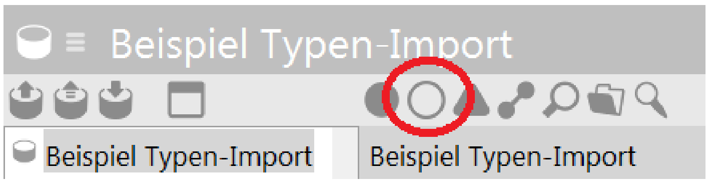


Songs
- 1: Objekte von **Song**
  - 2: Attribut **Name** *Aktive Sprache*
  - 3: Attribut **Dauer**
  - 4: Relation **ist enthalten in**
    - 5: Objekte von **Album**
      - 6: Attribut **Name** *Aktive Sprache*
      - 7: Attribut **Jahr**
      - 8: Relation **hat Autor**
        - 9: Objekte von **Band**
          - 10: Attribut **Name** *Aktive Sprache*
          - 11: Relation **hat Musikrichutng**
            - 12: Objekte von **Musik Stil**
              - 13: Attribut **Name** *Aktive Sprache*

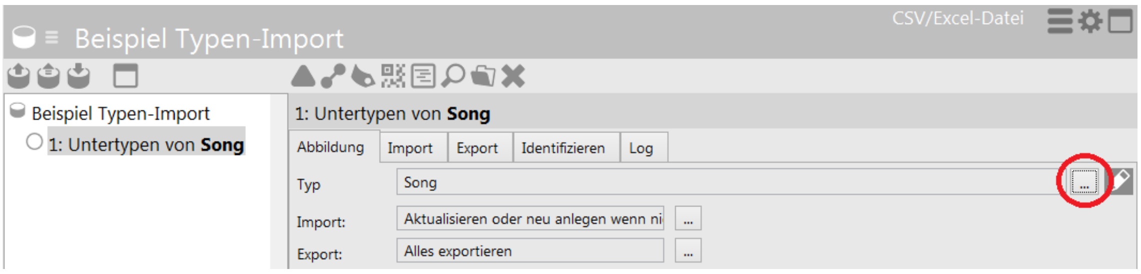
1.5.1.3.4 The type mapping

Types can also be imported and exported. Let s assume we want to import the genres of songs as types.

To map a new type, we choose the “New type mapping” button.



Following this, we have to specify the super-type of the new types to be created, in our example, the super-type would be “Song:”



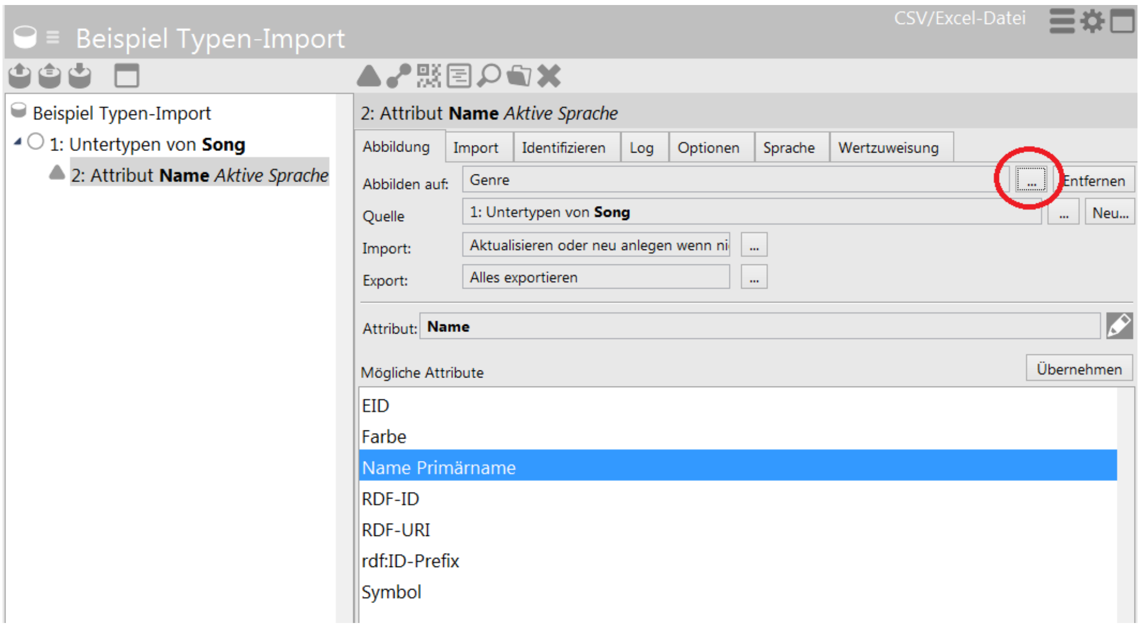
1: Untertypen von **Song**

Typ: Song

Import: Aktualisieren oder neu anlegen wenn ni

Export: Alles exportieren

Following that, we have to specify from which column of the imported table the name of our new types is to be taken:



2: Attribut **Name** *Aktive Sprache*

Abbilden auf: Genre

Quelle: 1: Untertypen von **Song**

Import: Aktualisieren oder neu anlegen wenn ni

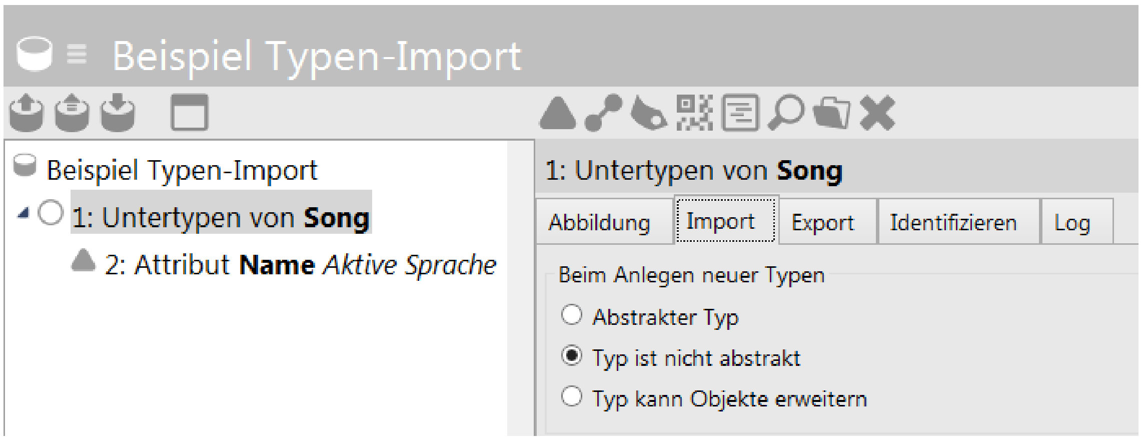
Export: Alles exportieren

Attribut: **Name**

Mögliche Attribute

EID
Farbe
Name Primärname
RDF-ID
RDF-URI
rdf:ID-Prefix
Symbol

Following that, we still have to specify on the “Import” tab that our new types are not supposed to be abstract:



Beispiel Typen-Import

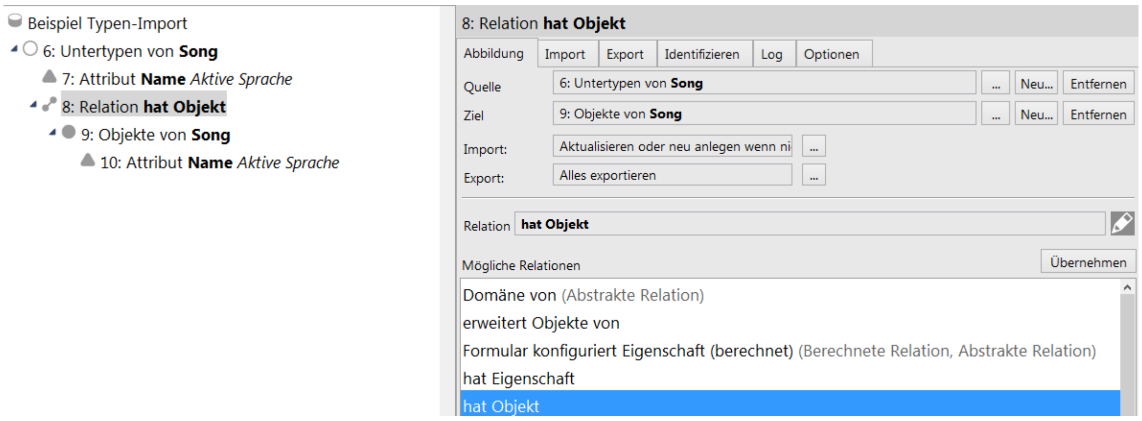
- Beispiel Typen-Import
  - 1: Untertypen von **Song**
    - 2: Attribut **Name** *Aktive Sprache*

1: Untertypen von **Song**

Abbildung | Import | Export | Identifizieren | Log

Beim Anlegen neuer Typen
- ( ) Abstrakter Typ
- (●) Typ ist nicht abstrakt
- ( ) Typ kann Objekte erweitern

If we now want to assign the corresponding songs to their new types, we have to use the system relation “has object.” In older versions of i-views this relation is called “has individual.” As the target we chose all objects of song (incl. subtypes), which are defined via the Name attributes in accordance with the Song title column.



- Beispiel Typen-Import
  - 6: Untertypen von **Song**
    - 7: Attribut **Name** *Aktive Sprache*
    - 8: Relation **hat Objekt**
      - 9: Objekte von **Song**
        - 10: Attribut **Name** *Aktive Sprache*

8: Relation **hat Objekt**

Abbildung | Import | Export | Identifizieren | Log | Optionen

Quelle: 6: Untertypen von **Song**
Ziel: 9: Objekte von **Song**
Import: Aktualisieren oder neu anlegen wenn ni
Export: Alles exportieren

Relation: **hat Objekt**

Mögliche Relationen

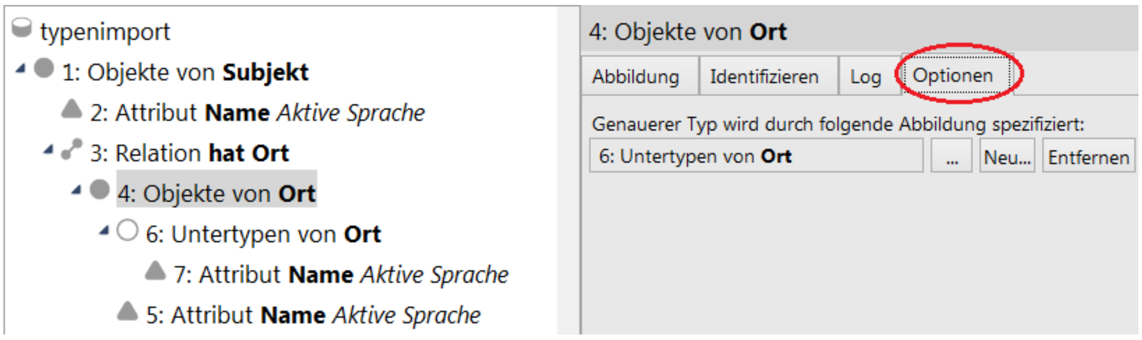
- Domäne von (Abstrakte Relation)
- erweitert Objekte von
- Formular konfiguriert Eigenschaft (berechnet) (Berechnete Relation, Abstrakte Relation)
- hat Eigenschaft
- hat Objekt

If we now import this mapping, we get the desired result. The songs that already exist in the semantic graph database are taken into account by the import setting “Update or create if not found” and moved under their respective type so that no object is created twice (see chapter Import behavior settings). A quick reminder: A specific object cannot belong to several types at once.

There is another special case. If we have a table in which different types occur in one column, we can also map this in our import settings.

Person/Band	**Herkunft**	**Typ des Ortes**
Paul McCarntey	Liverpool	Stadt
The Beatles	Großbritannien	Land

To do this, we count the mappings of objects to which we want to assign subtypes (in this case “objects of location”) and then select the corresponding super-type on the “Options” tab.


typenimport
1: Objekte von **Subjekt**
2: Attribut **Name** *Aktive Sprache*
3: Relation **hat Ort**
4: Objekte von **Ort**
6: Untertypen von **Ort**
7: Attribut **Name** *Aktive Sprache*
5: Attribut **Name** *Aktive Sprache*

4: Objekte von **Ort**

Genauerer Typ wird durch folgende Abbildung spezifiziert:
6: Untertypen von **Ort**

It is also important not to forget to specify on the “Import” tab that the type is not supposed to be abstract so that concrete objects can be created.

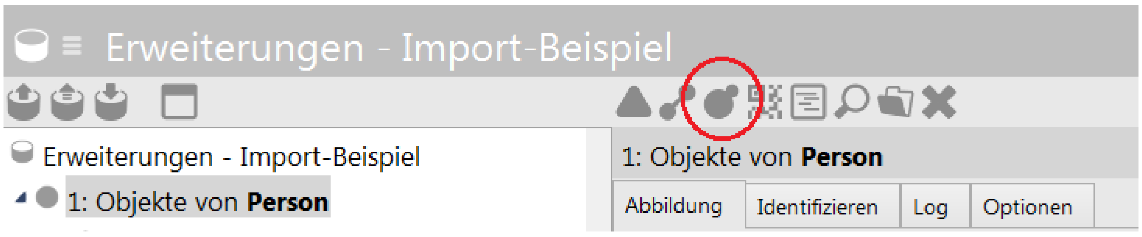
Careful: Assuming Liverpool already exists in the semantic network but is assigned to the type “Location” because it did not have subtypes such as “City” and “Country” at that time. In this case, Liverpool is **not** created anew under the type City. Reason: The objects of the Location type are only identified via the name attribute and not via the subtype.

1.5.1.3.5 Mapping of extensions

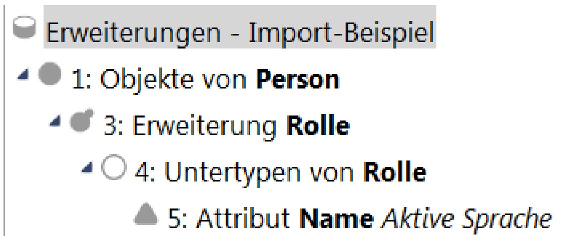
Extensions can also be imported and exported. Let s assume we have a table that shows the role of a band member in a band:

Person	**Band**	**Rolle**
Ron Wood	Faces	Gitarrist
Ron Wood	Jeff Beck Group	Bassist
Ron Wood	Rolling Stones	Gitarrist

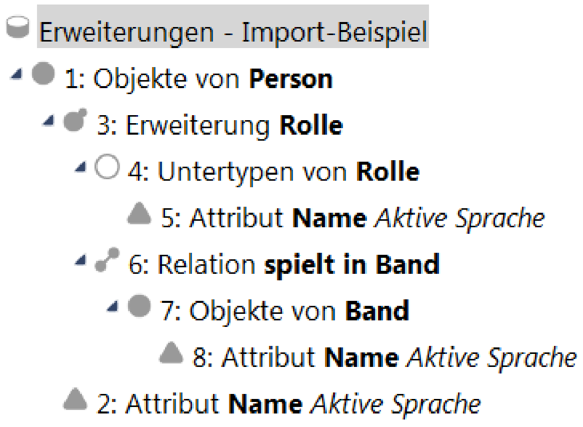
Ron Wood is a guitarist with the Faces and the Rolling Stones, but a bassist with the Jeff Beck Group. In order to map this, we must select the object for which an extension was defined in the schema and then press the “New extension mapping” button.


Erweiterungen - Import-Beispiel
Erweiterungen - Import-Beispiel
1: Objekte von **Person**
1: Objekte von **Person**

Like an object mapping, an extension mapping queries the corresponding type. In the schema of the music network, the “Role” type is an abstract type. So it is necessary to define in the mapping that the role is to be mapped to subtypes of the “Role” type (see Type mapping chapter).


Erweiterungen - Import-Beispiel
1: Objekte von **Person**
3: Erweiterung **Rolle**
4: Untertypen von **Rolle**
5: Attribut **Name** *Aktive Sprache*

As with objects and types, the relation can be mapped to the extension (or to the subtypes of an extension).



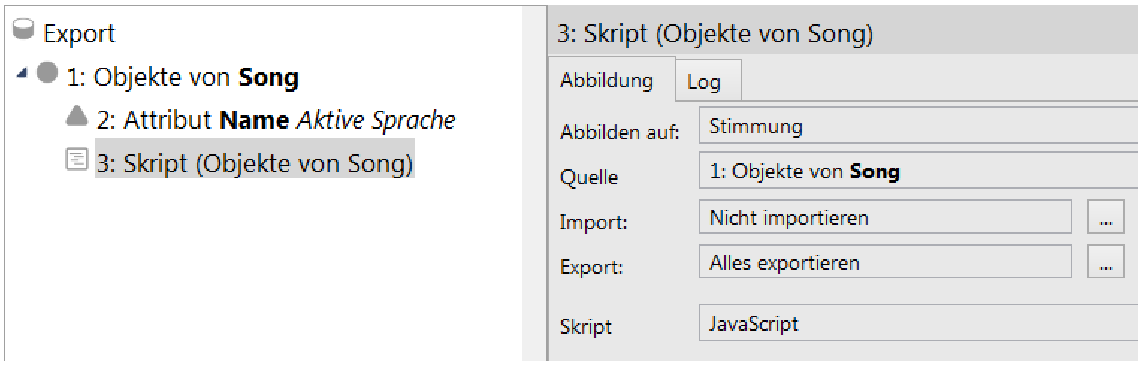
Erweiterungen - Import-Beispiel
1: Objekte von **Person**
3: Erweiterung **Rolle**
4: Untertypen von **Rolle**
5: Attribut **Name** *Aktive Sprache*
6: Relation **spielt in Band**
7: Objekte von **Band**
8: Attribut **Name** *Aktive Sprache*
2: Attribut **Name** *Aktive Sprache*

1.5.1.3.6 The script mapping

The script mapping can only be used upon export. The script can be written in either JavaScript or KScript.

The script mapping is, for example, used when we wish to combine three attributes from the semantic graph database to form an ID. However, this may make the export slower. (In the case of an import, this could be mapped using a virtual property more easily. The use of virtual properties is explained the chapter Table Columns.)

The following case is another example of the use of a script in the case of an export. It shows how several properties can be written into a cell with a separator. In this case, we wish to generate a table which lists the song names in the first column and all moods for the songs separated by commas:



Export
1: Objekte von **Song**
2: Attribut **Name** *Aktive Sprache*
3: Skript (Objekte von Song)

3: Skript (Objekte von Song)

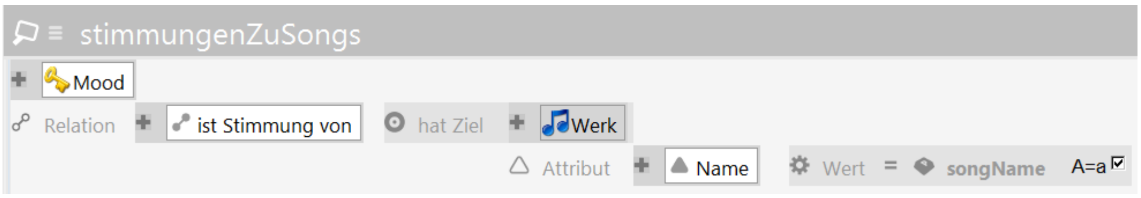
Abbilden auf:	Stimmung
Quelle	1: Objekte von **Song**
Import:	Nicht importieren
Export:	Alles exportieren
Skript	JavaScript

To generate the second column, we require the following script:

```
function exportValueOf(element)
{

    var mood = "";
    var relTargets = $k.Registry.query(“moodsForSongs").findElements({songName: element.attributeV
    if(relTargets && relTargets.length > 0){
        for(var i = 0; i < (relTargets.length-1); i++){
            mood += relTargets[i].attributeValue(“objectName") + ", ";
        }
        mood += relTargets[relTargets.length-1].attributeValue(“objectName");
    }
    return mood;
}
```

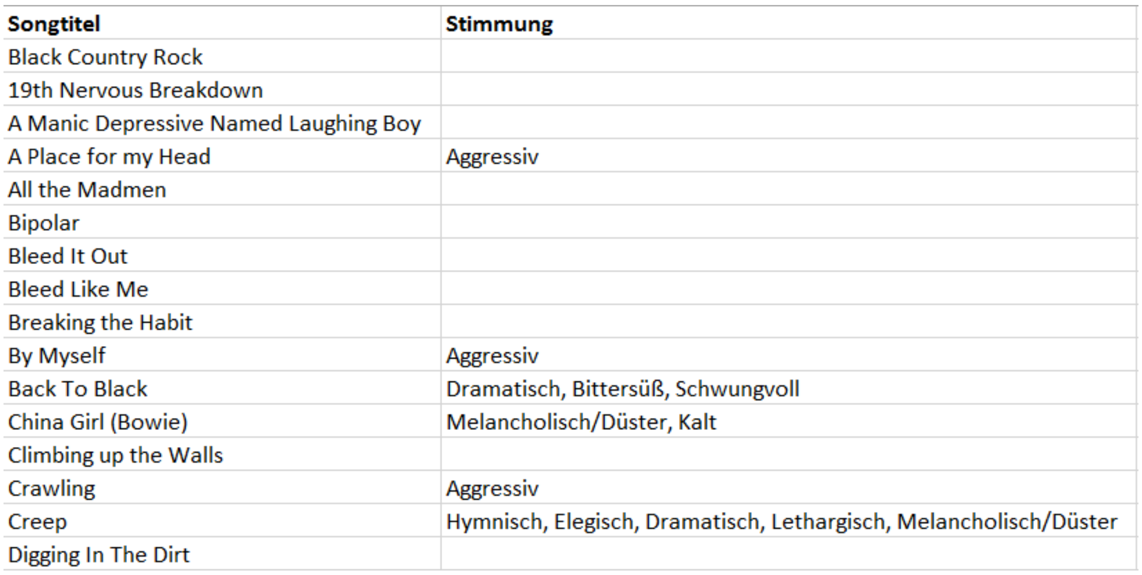
The script contains the following structured query (registration key: “mood ForSongs"):



The expression “findElements” allows us to access a parameter (in this case “songName”) within the query. The “objectName” is the internal name of the name attribute in this semantic model.

Within the if-instruction we state that when an element has several relation targets, these should be shown separated by a comma. After the last relation target that runs through the loop, there should no longer be a comma. Even when an element only has one relation target, this is shown without a comma accordingly.

The result is a list of songs with all their moods, which appear separated by a comma in the second column in the table:

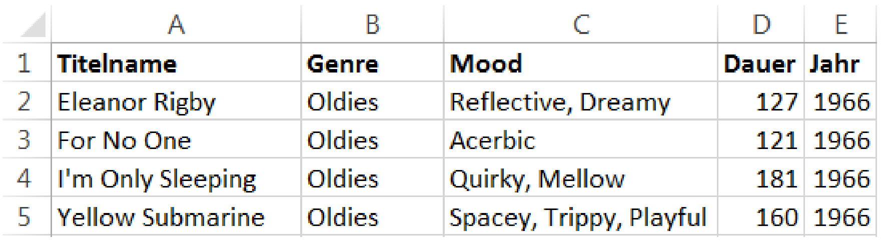


Songtitel	Stimmung
Black Country Rock	
19th Nervous Breakdown	
A Manic Depressive Named Laughing Boy	
A Place for my Head	Aggressiv
All the Madmen	
Bipolar	
Bleed It Out	
Bleed Like Me	
Breaking the Habit	
By Myself	Aggressiv
Back To Black	Dramatisch, Bittersüß, Schwungvoll
China Girl (Bowie)	Melancholisch/Düster, Kalt
Climbing up the Walls	
Crawling	Aggressiv
Creep	Hymnisch, Elegisch, Dramatisch, Lethargisch, Melancholisch/Düster
Digging In The Dirt	

1.5.1.4 Mapping of several values for an object type at an object

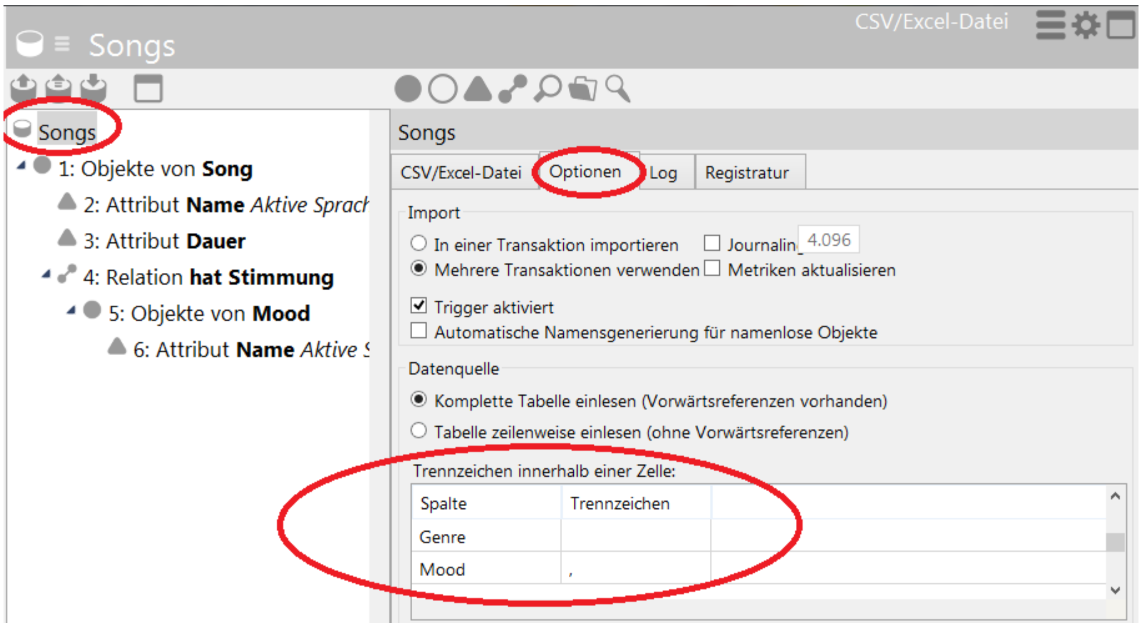
If several values are specified for an object type when there is an object (in our example, there are several “Moods” for each song), then there are three possible ways the table will look. For two of the three possible ways, the import must be modified, which is described in the following.

Option 1 - Values separated by separators: The individual values are found in a cell and are separated by a separator (e.g. a comma).



	A	B	C	D	E
1	**Titelname**	**Genre**	**Mood**	**Dauer**	**Jahr**
2	Eleanor Rigby	Oldies	Reflective, Dreamy	127	1966
3	For No One	Oldies	Acerbic	121	1966
4	I'm Only Sleeping	Oldies	Quirky, Mellow	181	1966
5	Yellow Submarine	Oldies	Spacey, Trippy, Playful	160	1966

In this case, we go to the mapping of the data source, where the general settings are found, and to the "*Options*" tab found there. The setting used to specify separators within a cell is found here in the lower section. We now only have to locate the corresponding column of the table to be imported ("Mood") and enter the separator used (",") in the column "*Separator*".



Songs

- Songs
  - 1: Objekte von **Song**
    - 2: Attribut **Name** *Aktive Sprac*
    - 3: Attribut **Dauer**
    - 4: Relation **hat Stimmung**
      - 5: Objekte von **Mood**
        - 6: Attribut **Name** *Aktive*

Songs

CSV/Excel-Datei | Optionen | Log | Registratur

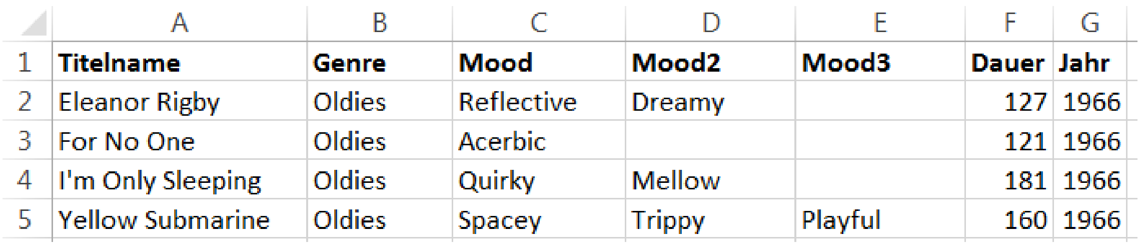
Import
- ( ) In einer Transaktion importieren
- [ ] Journalin_ 4.096
- (•) Mehrere Transaktionen verwenden
- [ ] Metriken aktualisieren
- [✓] Trigger aktiviert
- [ ] Automatische Namensgenerierung für namenlose Objekte

Datenquelle
- (•) Komplette Tabelle einlesen (Vorwärtsreferenzen vorhanden)
- ( ) Tabelle zeilenweise einlesen (ohne Vorwärtsreferenzen)

Trennzeichen innerhalb einer Zelle:

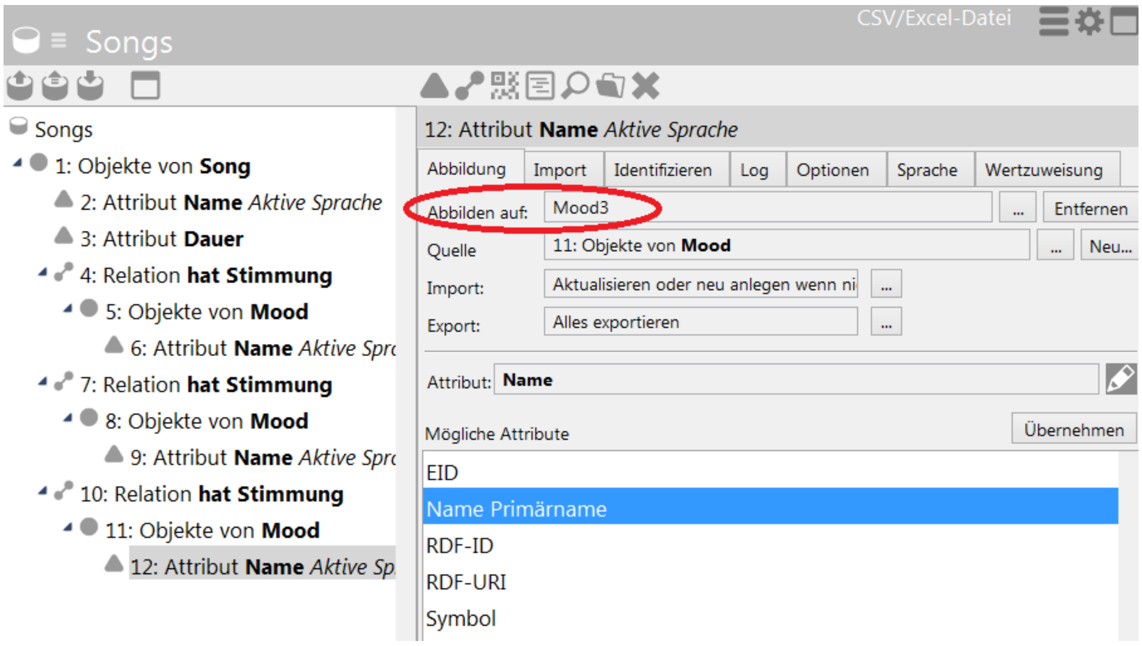
Spalte	Trennzeichen
Genre	
Mood	,

Option 2 - Several columns: The individual values are located in their own respective column, whereby not every field must be filled in. As many columns are required as the maximum number of moods there are per song.



	A	B	C	D	E	F	G
1	**Titelname**	**Genre**	**Mood**	**Mood2**	**Mood3**	**Dauer**	**Jahr**
2	Eleanor Rigby	Oldies	Reflective	Dreamy		127	1966
3	For No One	Oldies	Acerbic			121	1966
4	I'm Only Sleeping	Oldies	Quirky	Mellow		181	1966
5	Yellow Submarine	Oldies	Spacey	Trippy	Playful	160	1966

In this case, the corresponding relation must be created the same number of times as there are columns. In this case, the first relation must, accordingly, be mapped to "Mood1", the second relation to "Mood2" and the third relation to "Mood3".



Songs
- 1: Objekte von **Song**
  - 2: Attribut **Name** *Aktive Sprache*
  - 3: Attribut **Dauer**
  - 4: Relation **hat Stimmung**
    - 5: Objekte von **Mood**
      - 6: Attribut **Name** *Aktive Spr*
  - 7: Relation **hat Stimmung**
    - 8: Objekte von **Mood**
      - 9: Attribut **Name** *Aktive Spr*
  - 10: Relation **hat Stimmung**
    - 11: Objekte von **Mood**
      - 12: Attribut **Name** *Aktive Sp*

12: Attribut **Name** *Aktive Sprache*

Abbilden auf: Mood3

Quelle: 11: Objekte von **Mood**

Import: Aktualisieren oder neu anlegen wenn ni

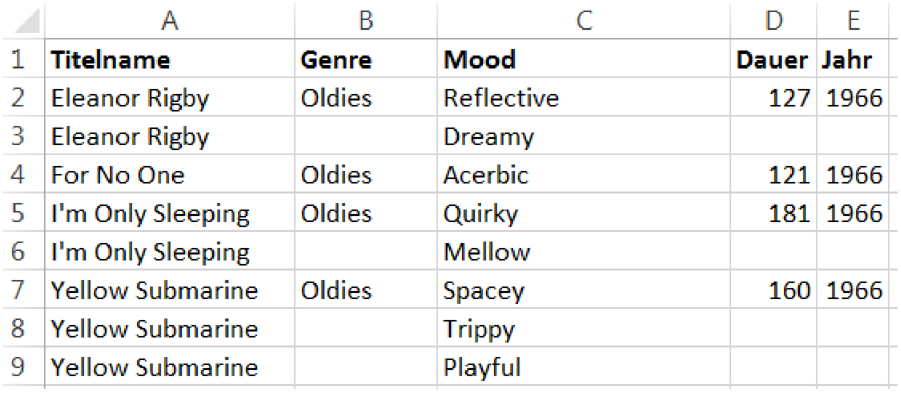
Export: Alles exportieren

Attribut: **Name**

Mögliche Attribute

- EID
- Name Primärname
- RDF-ID
- RDF-URI
- Symbol

Option 3 - Several rows: The individual values are located in their own respective row. Please note: In this case, it is essential that the attributes that are required for identification of the object (in this case the track name) appear in every row, as otherwise the rows would be interpreted as their own respective object without a name, making a correct import impossible.



	A	B	C	D	E
1	**Titelname**	**Genre**	**Mood**	**Dauer**	**Jahr**
2	Eleanor Rigby	Oldies	Reflective	127	1966
3	Eleanor Rigby		Dreamy		
4	For No One	Oldies	Acerbic	121	1966
5	I'm Only Sleeping	Oldies	Quirky	181	1966
6	I'm Only Sleeping		Mellow		
7	Yellow Submarine	Oldies	Spacey	160	1966
8	Yellow Submarine		Trippy		
9	Yellow Submarine		Playful		

In this case, no special import settings are required, as the system identifies the object using the identifying attribute and creates the relations correctly.

1.5.1.5 Settings of the import behaviour

During the import process, a check is always performed to determine whether an attribute already exists. “Identify“ infers concrete objects from attributes. When we refer below to “existing attributes“, these are attributes whose value precisely matches the value in the column to which they are mapped. When we refer to existing objects, we mean concrete objects that have been identified through an existing attribute.

Example: If our network already contains a song called “Eleanor Rigby“, the name attribute (mapped to the “track name“ column in our import table) is an existing attribute, so the song is an existing song as long as the song is identified only via the name attribute.

The settings for import behavior allow us to control how the import should react to existing and new semantic elements. The following table shows a brief description of the individual settings, while the sub-chapters of this chapter contain detailed and descriptive explanations.

Setting	Brief description
Update	Existing elements are overwritten (updated), no new elements are created.
Update or create if not found	Existing elements are overwritten; if none exist, they are created.
Delete all with same value (only available for properties)	All attribute values that match the imported value are deleted for the respective objects.
Delete all with same type	All attribute values of the selected type are deleted for the relevant objects, regardless of the values match or not.
Delete	Is used to delete that exact element.
Create	Creates a new property/object irrespective of whether the attribute value or the object already exists.
Create if type not found (only available for attributes)	An attribute of the required type is only created if none of this type exists.
Create if value not found (only available for attributes)	An attribute with this value is only created, if none with this value exists.
Do not import	No import.
Synchronize	In order to synchronize the contents for import with the contents in the database, this action creates all elements that do not yet exist, updates all elements that have changed, and delete all elements that no longer exist.

During an import, we have to decide individually for every mapped object, every mapped relation and every mapped attribute which import settings we want to use.

Note: Unlike in other editors of the Knowledge Builder, a setting is neither “inherited” by the subordinate mapping elements, nor is the import setting for an object “inherited” by its attributes.

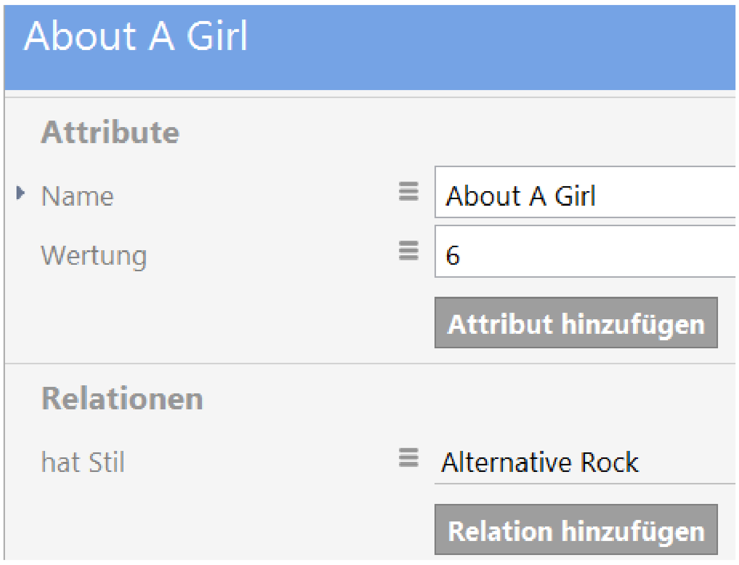
1.5.1.5.1 Update

If this setting is applied to an **attribute**, it ensures that the value from the table overwrites the attribute value of exactly one existing attribute. No new attributes are created with this setting. If the object has more than one attribute value of the selected type, no value is imported.

If you use the “Update” setting for an identifying attribute while using the “Update or create if not available” setting for a corresponding object, the error message “Attribute not found” appears, if the identifying object is not available in i-views.

If “Update” is applied to an **object**, this setting ensures that all properties of the object can be added or changed by the import. New objects are not created.

Example: Let s assume we keep a database of our favorite songs. We have just received a list with songs that contain new information. We want to get this information into our database but prevent songs that are not our favorite songs from being imported. We use the “Update” setting to do this.



About A Girl

Attribute

Name: About A Girl

Wertung: 6

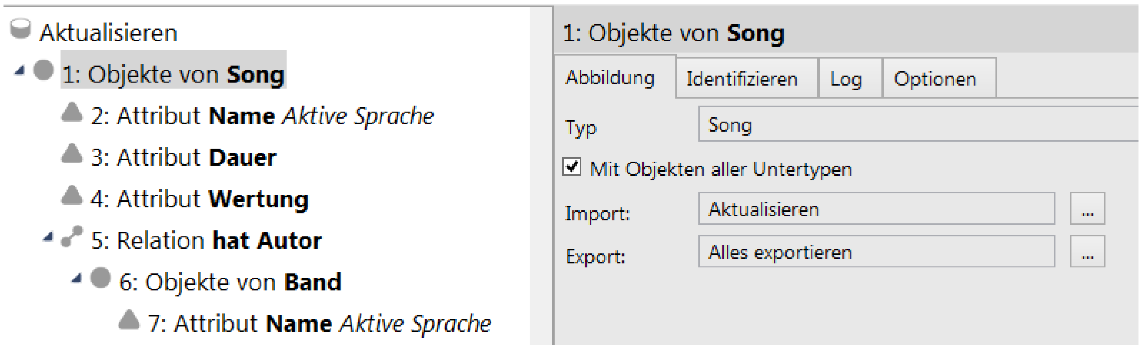
Relationen

hat Stil: Alternative Rock

The song "About A Girl" is already available in the Knowledge Builder.

Song	Dauer	Wertung	Autor
About A Girl	168	5	Nirvana

The import table contains information on the length, rating and creator of the song.



Aktualisieren
- 1: Objekte von **Song**
  - 2: Attribut **Name** *Aktive Sprache*
  - 3: Attribut **Dauer**
  - 4: Attribut **Wertung**
  - 5: Relation **hat Autor**
    - 6: Objekte von **Band**
      - 7: Attribut **Name** *Aktive Sprache*

1: Objekte von **Song**

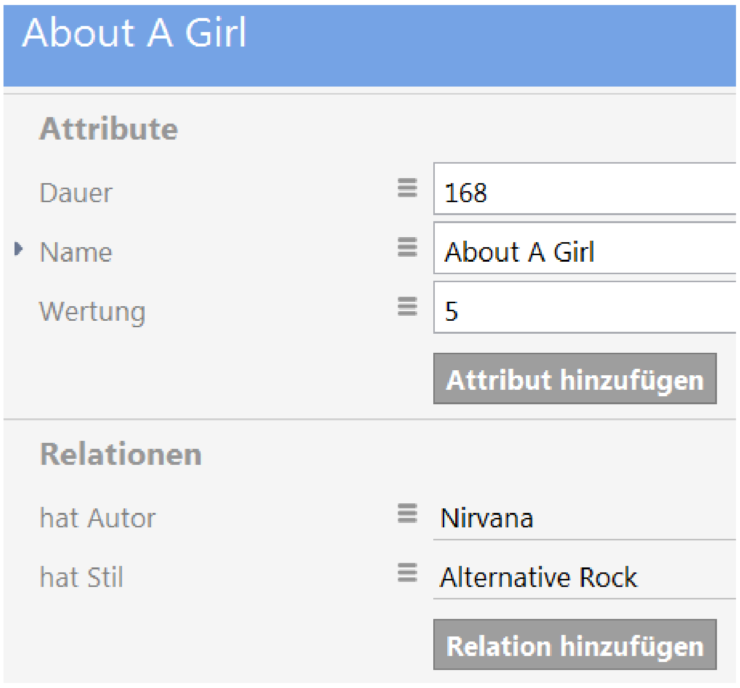
Typ: Song

☑ Mit Objekten aller Untertypen

Import: Aktualisieren

Export: Alles exportieren

For Song objects we specify that they are supposed to be updated. All attributes, relations and relational targets receive the import setting “Update or create if not available yet.”



About A Girl

Attribute

Dauer	168
Name	About A Girl
Wertung	5

Relationen

hat Autor	Nirvana
hat Stil	Alternative Rock

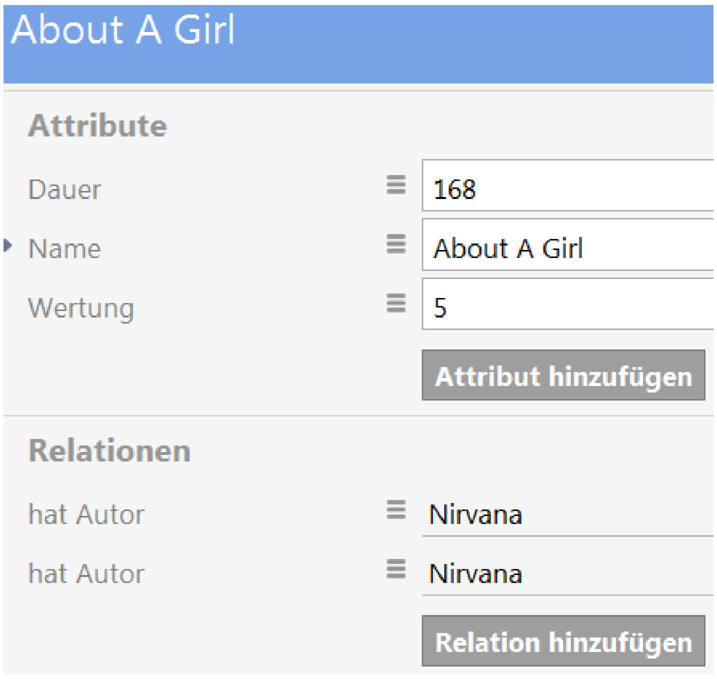
The result: The song has been updated and has received new attributes and relations. Already existing properties have been updated (value).

1.5.1.5.2 Update or create if not found

This import setting is required in most cases and is therefore set as the default setting. If elements already exist they will be updated. If elements do not exist yet they are created in the database.

1.5.1.5.3 Delete all with same value

This import setting is only available for properties (relations and attributes) and is only used when the import setting “Delete” cannot be used for deleting. “Delete” does not function for deleting when a relation or an attribute occurs on an object several times with the same value. If an attempt is made nonetheless, an error message appears. For example, the song “About A Girl” may have been linked to the band “Nirvana” using the relation “has author” by mistake.



About A Girl

Attribute

Dauer	168
Name	About A Girl
Wertung	5

Relationen

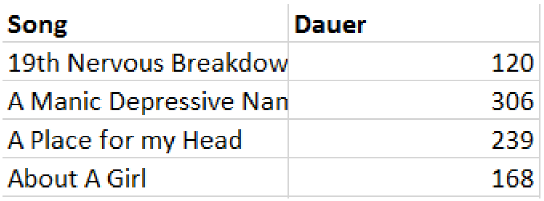
hat Autor	Nirvana
hat Autor	Nirvana

In cases like this, the import setting “Delete” does not have an affect, because due to multiple occurrences, it does not know which relations it is supposed to delete. In this case, “Delete all with the same value” must be used.

1.5.1.5.4 Delete all of same kind

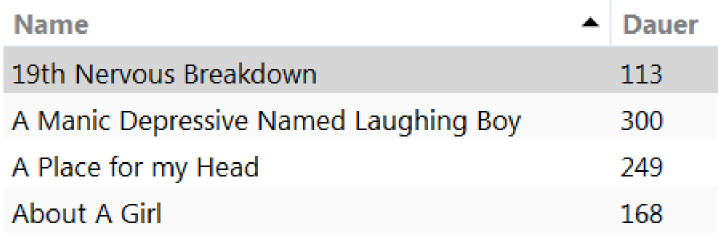
This import setting is used if all attributes, objects or relations of a type are supposed to be deleted, irrespective of existing values. In contrast to this, the settings “Delete” and “Delete all with identical value” take the existing values into account. Only the elements of those objects that occur in the import table are deleted.

Example: We have an import table with songs and the duration of the songs. We see that the duration differs in many cases and decide to delete the duration for these songs to make sure we do not have any incorrect information.



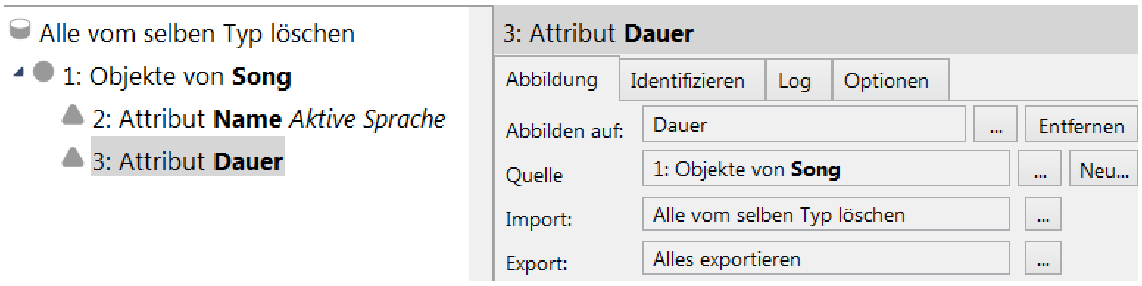
Song	Dauer
19th Nervous Breakdow	120
A Manic Depressive Nan	306
A Place for my Head	239
About A Girl	168

For most songs, the duration in the import table differs...



Name	Dauer
19th Nervous Breakdown	113
A Manic Depressive Named Laughing Boy	300
A Place for my Head	249
About A Girl	168

... from the duration of the songs in the database.



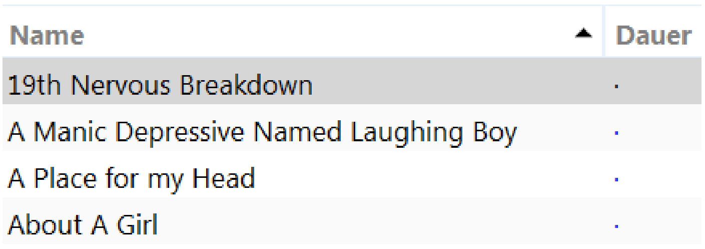
Alle vom selben Typ löschen
- 1: Objekte von **Song**
  - 2: Attribut **Name** *Aktive Sprache*
  - 3: Attribut **Dauer**

3: Attribut **Dauer**

Abbildung | Identifizieren | Log | Optionen

Abbilden auf:	Dauer
Quelle	1: Objekte von **Song**
Import:	Alle vom selben Typ löschen
Export:	Alles exportieren

For the attribute “Duration" we use the import setting “Delete all of the same type.”



Name	Dauer
19th Nervous Breakdown	·
A Manic Depressive Named Laughing Boy	·
A Place for my Head	·
About A Girl	·

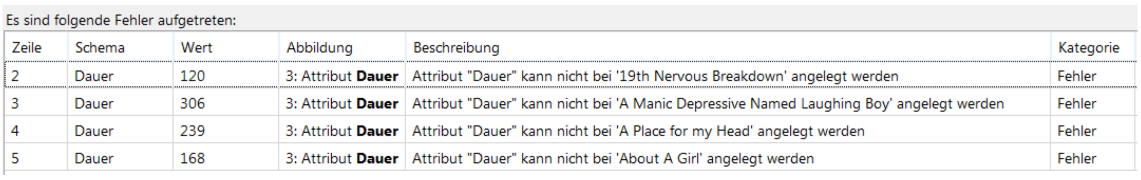
After the import, all attribute values of the attribute type duration have been deleted for these 4 songs.

1.5.1.5.5 Delete

The import setting “Delete” is used to delete exactly the one object/ exactly the one relation/exactly the one attribute value. If none or several objects/relations/attribute values match the elements for import, an error message about this appears and the elements concerned is not deleted.

1.5.1.5.6 Create new

This import setting creates a new property/a new object irrespective of whether the attribute value or the object already exists. Sole exception: If a property may only occur once (observe the setting “May have multiple occurrences” for the attribute definition), then the new attribute is not created and an error message appears noting this.



Es sind folgende Fehler aufgetreten:

Zeile	Schema	Wert	Abbildung	Beschreibung	Kategorie
2	Dauer	120	3: Attribut **Dauer**	Attribut "Dauer" kann nicht bei '19th Nervous Breakdown' angelegt werden	Fehler
3	Dauer	306	3: Attribut **Dauer**	Attribut "Dauer" kann nicht bei 'A Manic Depressive Named Laughing Boy' angelegt werden	Fehler
4	Dauer	239	3: Attribut **Dauer**	Attribut "Dauer" kann nicht bei 'A Place for my Head' angelegt werden	Fehler
5	Dauer	168	3: Attribut **Dauer**	Attribut "Dauer" kann nicht bei 'About A Girl' angelegt werden	Fehler

1.5.1.5.7 Create if type not found

This import setting is only available for attributes. A new attribute value is only created when the corresponding attribute does not yet have a value. The values do not have to be the same; what matters is that one value or another exists, or does not exist, for the corresponding attribute type. The simultaneous import of several attribute values to one attribute type is not possible, as in this case it is not possible to decide which of the attribute values should be used.

Example: Assuming that we have an import table that contains the musicians with their alias names. A number of musicians also have several alias names. In this case, we cannot use the setting “Create type if not found,” because then all musicians with several alias names would not be given one.

1.5.1.5.8 Create if value not found

This import setting is only available for attributes. A new attribute value is only created if the object does not yet have this value for the corresponding attribute.

Example: Let’s take again the import table that includes musicians wih their alias names. Here we can use the setting "Create value if not found", because then the musicians with several alias names can get all these alias names.

1.5.1.5.9 Do not import

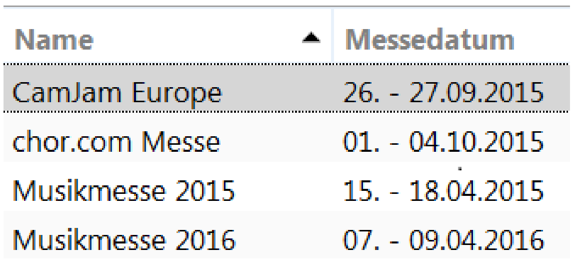
The import setting “Do not import” allows us to specify that an object or a property should not be imported. This is useful when a mapping has already been defined and we want to use it again, however do not want to import specific objects and properties again.

1.5.1.5.10 Synchronize

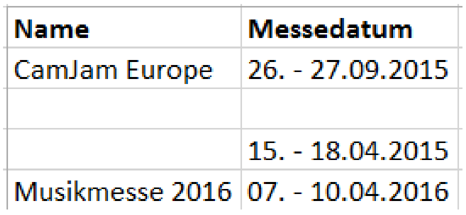
The import setting “Synchronize” should be used with caution, because it is the only import setting that not only affects the objects and properties in i-views that have values that match those in the import table, but also extends to all elements of the same type in i-views. When an import table is synchronized with i-views, in principle this means that after the import, the result should look exactly the same as it does in the table.

If objects of one type are synchronized, all objects of this type that are not in the import table are deleted. The objects that exist are updated and the objects that are not in i-views are created as new objects.

Example: We would like to synchronize the music fairs in i-views (at the left) with a table with the fairs and their date (at the right):

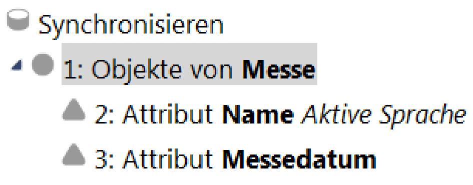


Name ▲	Messedatum
CamJam Europe	26. - 27.09.2015
chor.com Messe	01. - 04.10.2015
Musikmesse 2015	15. - 18.04.2015
Musikmesse 2016	07. - 09.04.2016

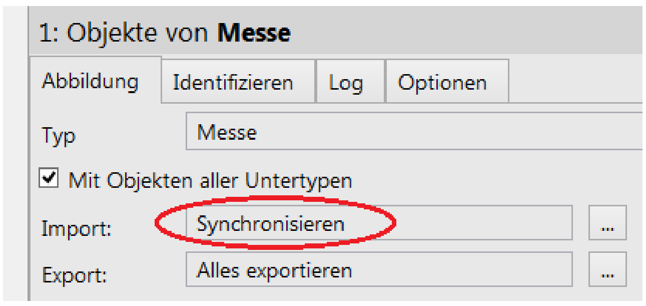


Name	Messedatum
CamJam Europe	26. - 27.09.2015
	15. - 18.04.2015
Musikmesse 2016	07. - 10.04.2016

For objects of the “Fair” type, we select the import setting “Synchronize;” for the individual attributes *Name* and *Date of fair* the import setting “Update or create if not found” is used:



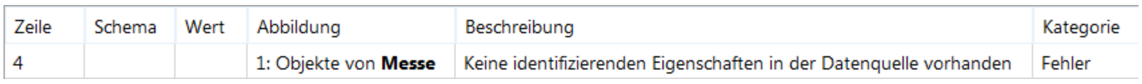
Synchronisieren
1: Objekte von **Messe**
2: Attribut **Name** *Aktive Sprache*
3: Attribut **Messedatum**



1: Objekte von **Messe**

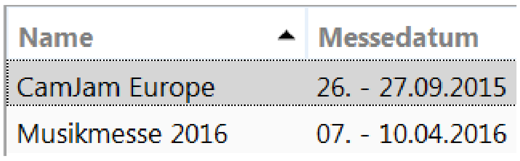
Typ: Messe
☑ Mit Objekten aller Untertypen
Import: Synchronisieren
Export: Alles exportieren

The attribute name is the identifiable attribute of fair. There is no name for the object Music fair 2015 in the import table. If we import the table this way, an error message is output:



Zeile	Schema	Wert	Abbildung	Beschreibung	Kategorie
4			1: Objekte von **Messe**	Keine identifizierenden Eigenschaften in der Datenquelle vorhanden	Fehler

After the import, we now see that the import caused two objects to be omitted that did not have a counterpart in the import table. The date was updated for Music fair 2016:



Name	Messedatum
CamJam Europe	26. - 27.09.2015
Musikmesse 2016	07. - 10.04.2016

When **attributes** are synchronized, the following applies: When an existing attribute is not given a value by an import, it is deleted for the corresponding object of the import table. If the existing attribute has a different value to the import table, it is updated, even when it is allowed to occur several times. If the attribute does not yet exist, a new one is created.

When **relations** are synchronized, and they are not given a value, they are deleted for the corresponding object. If the existing relation has a different value to the import table, it is updated. If the target object does not yet exist in the database, a new one is created, provided that a corresponding import setting has been assigned to the target object. If the target object cannot be created as a new one, because, for example, the import setting "Update" was assigned, an error message appears notifying us that the target object was not found and will not be created.

1.5.1.6 Table columns

When it comes to mapping database queries, the columns that are available for import are specified by the database tables and/or the Select statement. When mapping files, it is possible adopt the columns with the "Read from data source" button from the file. But you can also specify them manually. In that case you can choose whether to create a standard column or a virtual property.

If you want to export from the semantic graph database you have to enter the columns manually. You can export only standard columns, not virtual columns.

Virtual table column / virtual property
Virtual columns are additional columns that allow you to use regular expressions to transform the contents we find in a column of the table to be imported. Example: Let s assume that "a.d." is appended to all the years in our import table. We can correct this by creating a virtual column that adopts only the first 4 characters from the year column.

We can also define virtual properties during export.

We simply write the regular expressions into the column header (into the name of the column). During the process, partial strings enclosed in pointy brackets <...> are replaced according to the following rules, with *n, n1, n2, ...* representing the contents of other table columns with the column number n.

Expression	Description	Example	Input	Output
<n**p**>	Print output of content of column n	Hits: <1p>	1 (integer) none (string)	Hits: 1 Hits: none

<n**s**>	Output of string in column n	Hello <1s>!	'Peter'	Hello Peter!
<n**u**>	Output of string in column n in upper case	Hello <1u>!	'Peter'	Hello PETER!
<n**l**>	Output of string in column n in lower case	Hello <1l>!	'Peter'	Hello peter!
<n**c**start-stop>	Partial string from position start to stop from column n	<1c3-6> <1c3> <1c3->	Columns	olum umn lumns
<n**m**regex>	Test whether the content of column n matches the regex regular expression. The following expressions are only evaluated if the regular expression applies.	<1m0[0-9]>hi <1m$>test	01 123 (blank) 123	hi (blank) test (blank)
<n**x**regex>	Test whether the content of column n matches the regex regular expression. The following expressions are only evaluated if the regular expression does **not** apply.	<1x0[0-9]>hello	01 123	(blank) hello
<n**e**regex>	Selects all hits for regex from the contents of column n. Individual hits are separated by commas in the result.	<1eL+> <1e\d\d\d\d>	HELLO WORLD 02.10.2001	LL,L 2001
<n**r**regex>	Removes all hits for regex from the contents of column n	<1rL>	HELLO WORLD	HEO WORD
<n**g**regex>	Transmits the contents of all groups of the regular expression	<1g\+(\d+)\->	+42-13	42
<n**f**format>	Formats numbers, date and time specifications from column n according to the format format specification	<1f#,0.00> <1fd/m/y> <1fdd/mmm>	3.1412 1234.5 1 May 1935 1 May 1935	3.14 1234.50 1/5/1935 01/May

Table columns can also be referenced independently from their column number by using specially defined identifiers. The advantage in this case is that the allocation is not lost if the column order is changed in the import table.

The identifier for the relevant column of the import table is entered in the column with the heading *Identifier* in the column definition table. These columns are referenced by creating a virtual table column that is given the identifier as its table column heading (see example 2).

Ex- pre sio	Description	Ex ar pl	In- put	Out- put
<**$**name**$**regex>	Reference to a column by means of a unique column identifier *name* and subsequent transformation by means of the *regex* regular expression. The **$** characters are a functional component of the identifier syntax.	<$Name$U>	Company #1	COMPANY #1

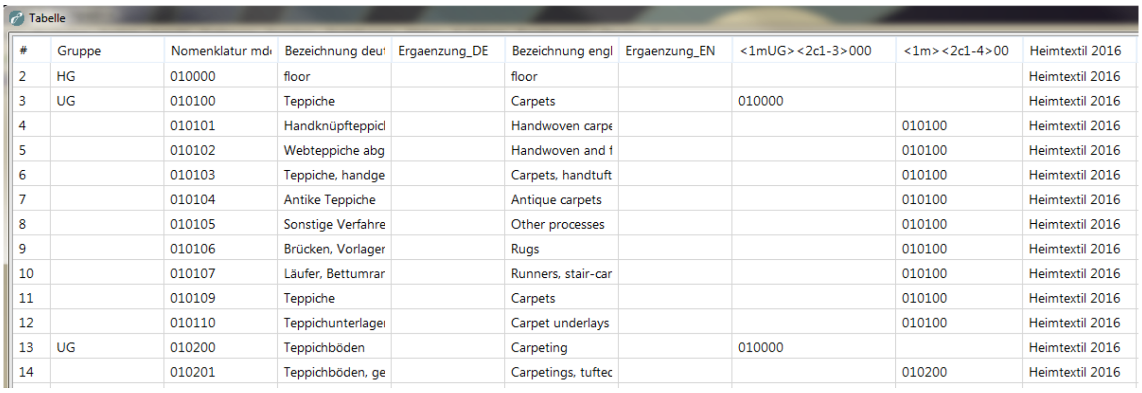
Example 1: Use of regular expressions (reference via column number)

Let s assume we have an import table containing concrete objects without a name. However, we want these objects to be modeled as separate objects in our data model. Example: for a load point, column 88 contains its main value, which is torque. So we enter the expression *load point <88s>* as the definition of our virtual column that will represent the name of this load point. The resulting name for a load point with a torque of 850 would therefore be “load point 850”.

We can also use the virtual property to create a username consisting of the first 4 letters of the first name and the last name. If the person is named Maximilian Mustermann and we define the virtual column with the relevant expression <1c1-4><2c1-4>, the result is “MaxiMust”.

The virtual property can also be used to create an initial password for a user during import. The expression could be *Pass4<2s>*. The resulting password for Maximilian Mustermann would be “Pass4Mustermann”.

A rather extensive example shows how the virtual property can be used to assign objects to the correct direct top-level group:



Tabelle

#	Gruppe	Nomenklatur md	Bezeichnung deu	Ergaenzung_DE	Bezeichnung engl	Ergaenzung_EN	<1mUG><2c1-3>000	<1m><2c1-4>00	Heimtextil 2016
2	HG	010000	floor		floor				Heimtextil 2016
3	UG	010100	Teppiche		Carpets		010000		Heimtextil 2016
4		010101	Handknüpfteppicl		Handwoven carpe			010100	Heimtextil 2016
5		010102	Webteppiche abg		Handwoven and f			010100	Heimtextil 2016
6		010103	Teppiche, handge		Carpets, handtuft			010100	Heimtextil 2016
7		010104	Antike Teppiche		Antique carpets			010100	Heimtextil 2016
8		010105	Sonstige Verfahre		Other processes			010100	Heimtextil 2016
9		010106	Brücken, Vorlager		Rugs			010100	Heimtextil 2016
10		010107	Läufer, Bettumrar		Runners, stair-car			010100	Heimtextil 2016
11		010109	Teppiche		Carpets			010100	Heimtextil 2016
12		010110	Teppichunterlage		Carpet underlays			010100	Heimtextil 2016
13	UG	010200	Teppichböden		Carpeting		010000		Heimtextil 2016
14		010201	Teppichböden, ge		Carpetings, tuftec			010200	Heimtextil 2016

The three right columns are virtual columns.

<1mUG>: The number of the top-level group of the object is only written to the first of the

virtual columns if the term “UG” (for Untergruppe (sub-group)) occurs in the first column for the object.

<2c1-3>000: The number to be written to the column consists of the first three characters of the second column and three zeros.

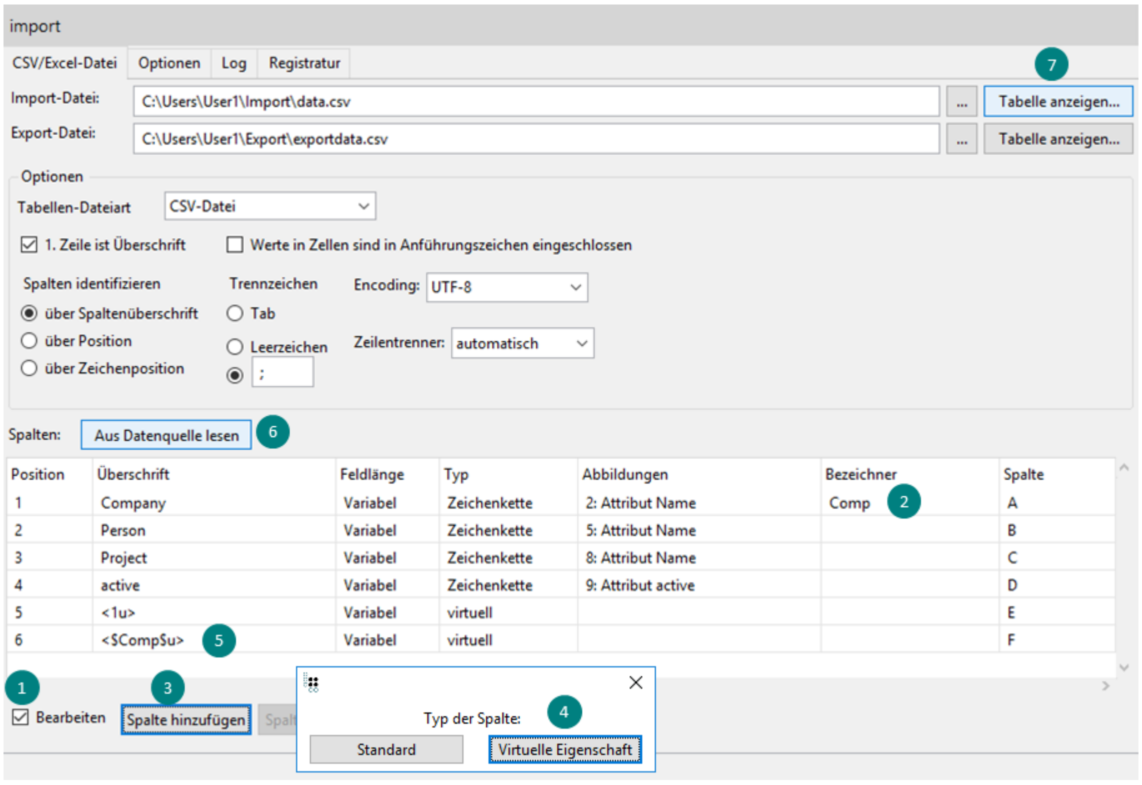
<1m>: Only if the first column for the object is empty, i.e. contains no value, is the number of the top-level group of the object written to the column.

<2c1-4>00: The number to be written to the column consists of the first four characters of the second column.

Heimtextil 2016: This expression (the German term for home textiles) is written to the column for all objects.

Example 2: Use of individual identifiers (in combination with regular expressions)

In the following example, the contents for the **Company** column are transformed into upper-case letters by means of virtual columns: Column 5 uses one reference per column number, column 6 uses one reference per column identifier.



Import-Datei: C:\Users\User1\Import\data.csv

Export-Datei: C:\Users\User1\Export\exportdata.csv

Tabellen-Dateiart: CSV-Datei

[✓] 1. Zeile ist Überschrift

[ ] Werte in Zellen sind in Anführungszeichen eingeschlossen

Spalten identifizieren: [✓ über Spaltenüberschrift] über Position über Zeichenposition

Trennzeichen: Tab Leerzeichen [✓ ;]

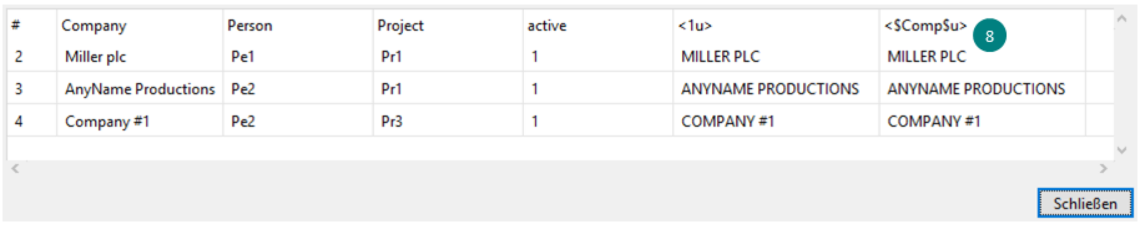
Encoding: UTF-8

Zeilentrenner: automatisch

Position	Überschrift	Feldlänge	Typ	Abbildungen	Bezeichner	Spalte
1	Company	Variabel	Zeichenkette	2: Attribut Name	Comp	A
2	Person	Variabel	Zeichenkette	5: Attribut Name		B
3	Project	Variabel	Zeichenkette	8: Attribut Name		C
4	active	Variabel	Zeichenkette	9: Attribut active		D
5	<1u>	Variabel	virtuell			E
6	<$Comp$u>	Variabel	virtuell			F

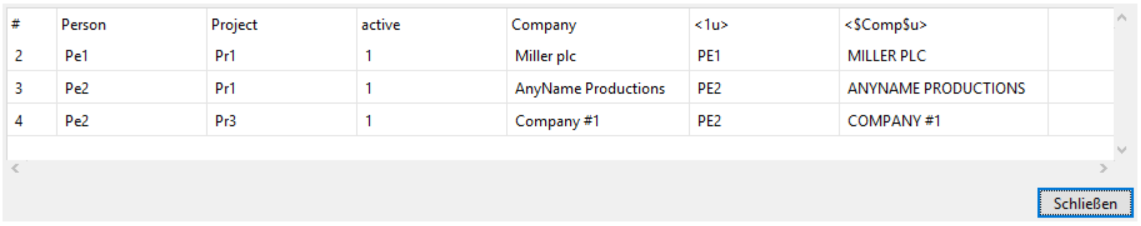
Typ der Spalte:

Click on the preview to view the transformed column entries:



#	Company	Person	Project	active	<1u>	<$Comp$u>
2	Miller plc	Pe1	Pr1	1	MILLER PLC	MILLER PLC
3	AnyName Productions	Pe2	Pr1	1	ANYNAME PRODUCTIONS	ANYNAME PRODUCTIONS
4	Company #1	Pe2	Pr3	1	COMPANY #1	COMPANY #1

The following figure shows the effect of swapped columns in an import table: If only regular expressions (<1u>) are used, the wrong column is transformed; if an identifier with a downstream regular expression (<$Comp$u>) is used, the content remains the same.



#	Person	Project	active	Company	<1u>	<$Comp$u>
2	Pe1	Pr1	1	Miller plc	PE1	MILLER PLC
3	Pe2	Pr1	1	AnyName Productions	PE2	ANYNAME PRODUCTIONS
4	Pe2	Pr3	1	Company #1	PE2	COMPANY #1

1.5.1.7 Configuration of further table oriented data sources

Databases

The database, user and password must be specified in the mapping for a PostgreSQL, Oracle or ODBC interface.

Database specification

The database specification consists of the name of the host, the port, and the name of the database. The syntax is:

Database system	Database specification
PostgreSQL	hostname:port_database
Oracle	//hostname:[port][/databaseService]
ODBC	Name of the configured data source
MySQL	Separate configuration of database and host name

Configure user name and password

The user name and password are specified as stored in the database. Under the Table option it is possible to specify the table to be imported. However, for import there is also the option of going to the “Query” option and formulating a query that specifies which data are to be imported.

Encoding

In case of PostgreSQL mapping, it is possible to specify the encoding on the “Encoding” tab.

Special requirements of the Oracle interface

The function for direct import from an Oracle database requires that certain runtime libraries are installed on the computer performing the import.

What is required directly is the “Oracle Call Interface” (OCI), and it is required in a version that, according to Oracle, matches the database server to be addressed. That means that the OCI in version 11 must be installed on the importing computer in order to address an Oracle 11i database. The easiest way to install the OCI is to install the “Oracle Database Instant Client”. The “Basic” package version is sufficient. The client can be obtained from the company operating the server, or from Oracle after registering at http://www.oracle.com/technology/tech/oci/index.html.

After the installation, it must be ensured that the library can be found by the importing client, either by placing it in the same directory or by defining environment variables that match the relevant operating system (documented for the OCI).

Depending on the operating system on which the import will be executed, further libraries are necessary, and these are not always installed.

- MS Windows: next to the required "oci.dll", two further libraries are required: advapi32.dll (extended Windows 32 Base-API) and mscvr71.dll (Microsoft C Runtime Library)

Apart from the XML import/export, all imports/exports are table-based and differ only in terms of the configuration of the source.

1.5.1.8 Mapping of an XML file

The principle of XML files is to make the different details for a record explicit by means of tags (<>) (and not by means of table columns). Accordingly, tags are also the basis for display when XML structures are imported to i-views.

Example: Let s assume that our list of songs is available as an XML file:

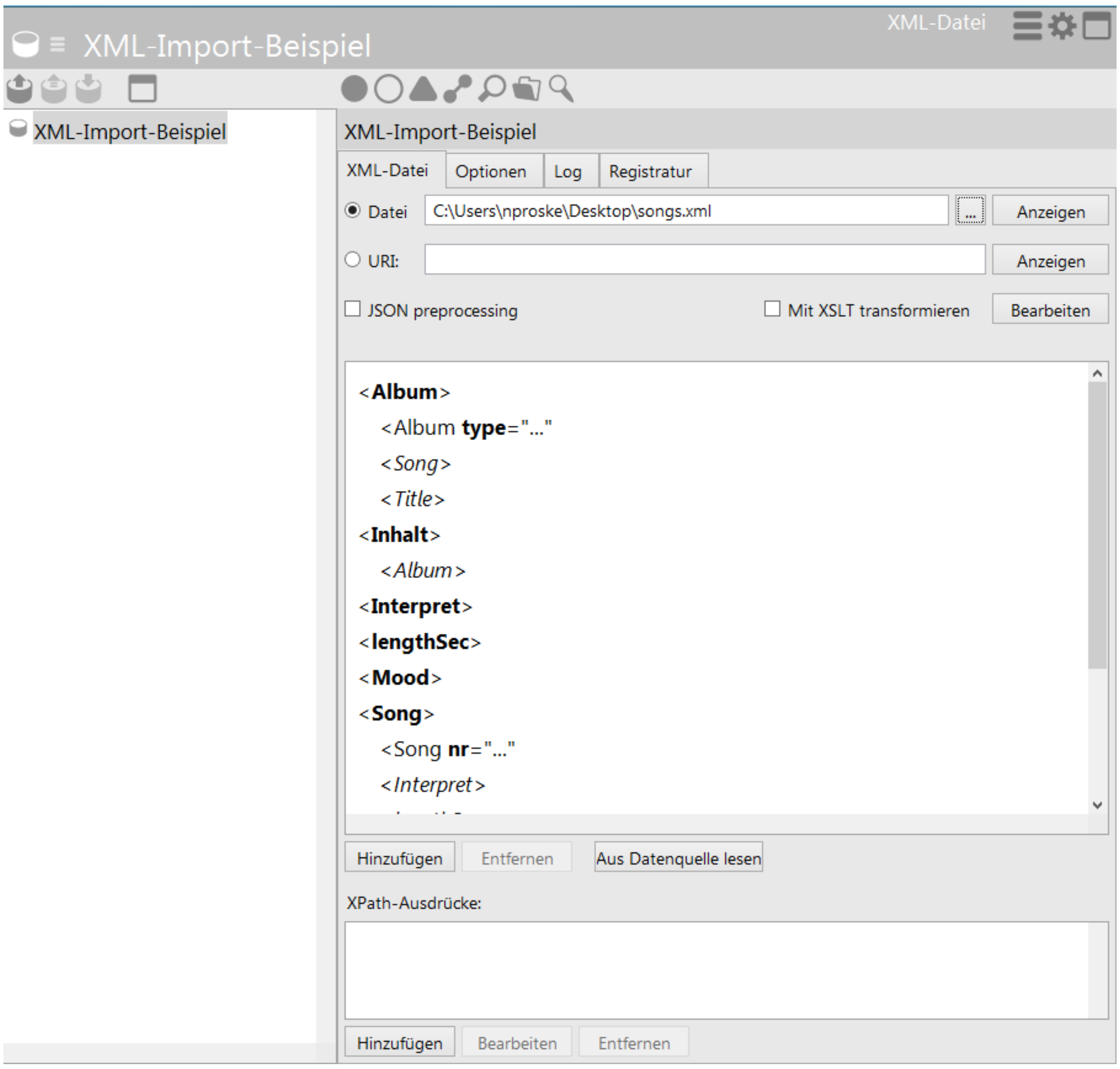
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Contents>
    <Album type="Oldie">
        <Title>Revolver</Title>
        <Song nr="1">
            <Title>Eleanor Rigby</Title>
            <lengthSec>127</lengthSec>
            <Artist>The Beatles</Artist>
            <Topic>Mental illness</Topic>
            <Mood>Dreamy</Mood>
            <Mood>Reflective</Mood>
        </Song>
        [...]
    </Album>
    [...]
</Contents>
```

If we want to import this XML file, we choose the "XML file" data source when selecting the type, which causes the editor for the import and export of XML files to open. Even the specification of the file location is different than in the editor for CSV files. We can now choose between a local file path and specification of a URI.

JSON preprocessing makes it possible to convert a JSON file to XML before the actual import.

You can choose **Transform with XSTL** if you want to convert the XML data from the selected XML file to different XML data before the import, for example in order to change the structure or further separate individual values. Use the "Edit" button to open the XML file, where you can then define the changes by means of XSLT.

Once the file has been selected, use the "Read from data source" button to read out the XML structure, which is then displayed in the right-hand window.



(●) Datei C:\Users\nproske\Desktop\songs.xml

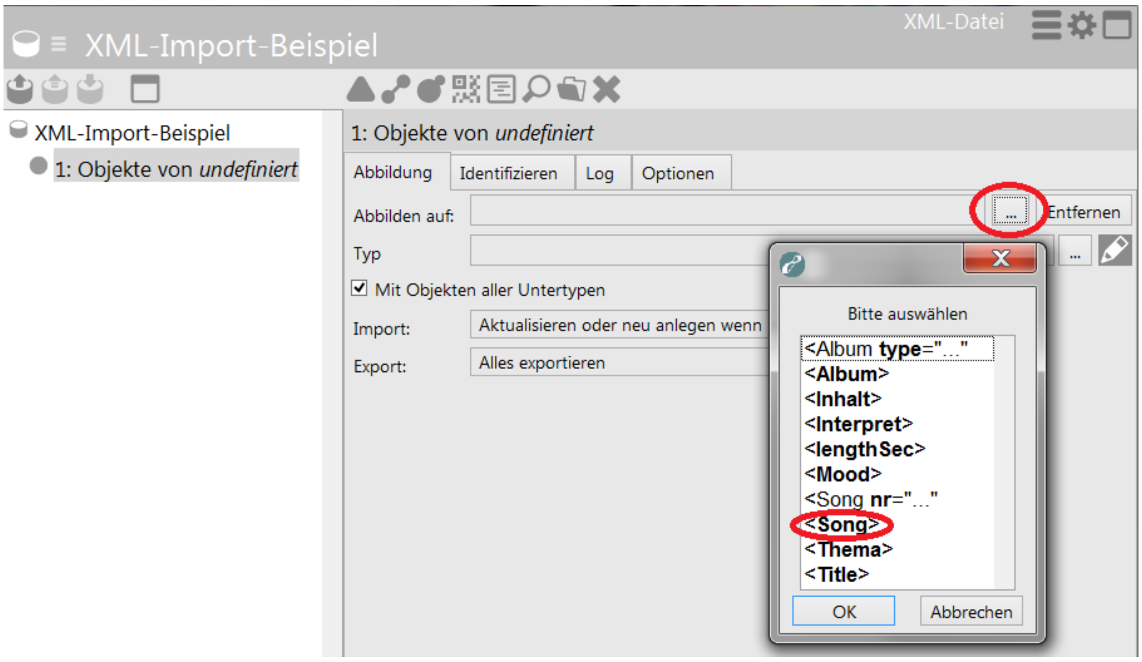
( ) URI:

☐ JSON preprocessing ☐ Mit XSLT transformieren

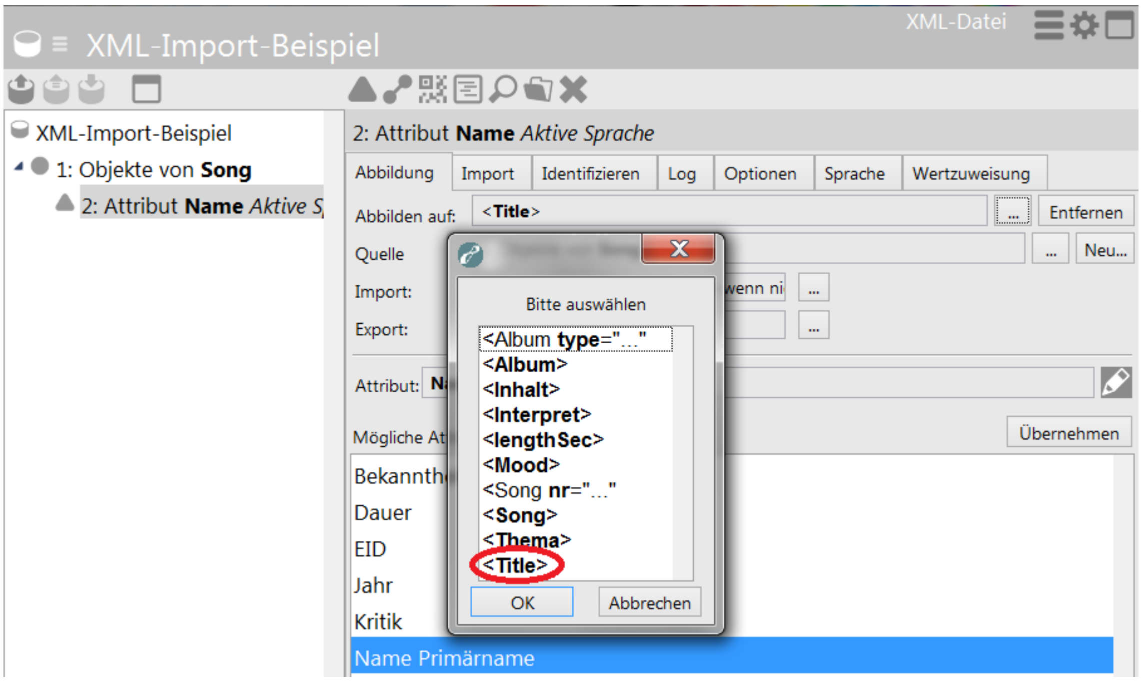
```
<Album>
    <Album type="..."
    <Song>
    <Title>
<Inhalt>
    <Album>
<Interpret>
<lengthSec>
<Mood>
<Song>
    <Song nr="..."
    <Interpret>
```

XPath-Ausdrücke:

We want to import the individual songs on our list. So we create a new object mapping and use the “Map to” button to select the <*Song*> tag. In contrast to a CSV import, where only the attribute values have an equivalent in the CSV table and where an individual row represents an object, which means that only the attribute values need to be mapped, semantic objects are here mapped by the XML structure. Therefore a corresponding tag of the XML file must be specified for each of the objects to be mapped.



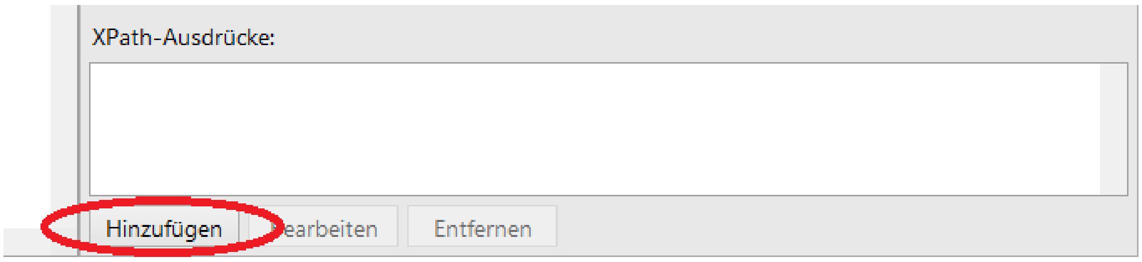
As our example shows, the tags are not always unambiguous without context: <Title> is used for both album titles and song titles. The object type only becomes clear in combination with the surrounding tag. Often the context of the XML structure and the context of the mapping hierarchy are synchronous: As we have already specified that the objects should be mapped to the <Song> tag, the XML structure makes clear which <Title> tag we actually mean when we map <Title> with the name attribute of songs. Where the mapping hierarchy and the tag structure are not parallel, we can use XPath to form strings in the XML import in addition to the tags occurring in the XML file.



As with the CSV import, it is necessary to use the “Identify” tab to specify for object mapping which attribute values should be used to identify the object in the semantic graph database. The first created attribute for an object is once again used automatically as the identifying attribute.

Options with XPath expressions

Let s assume we only want to import songs from albums with the “Oldie” music style. In our XML document, the information for the music style is specified directly in the album tag under *type="..."*. That means we have to use the editor to define an XPath expression describing the path in the XML document that contains only songs from oldie albums. The right-hand lower section of the editor contains a field for adding XPath expressions.



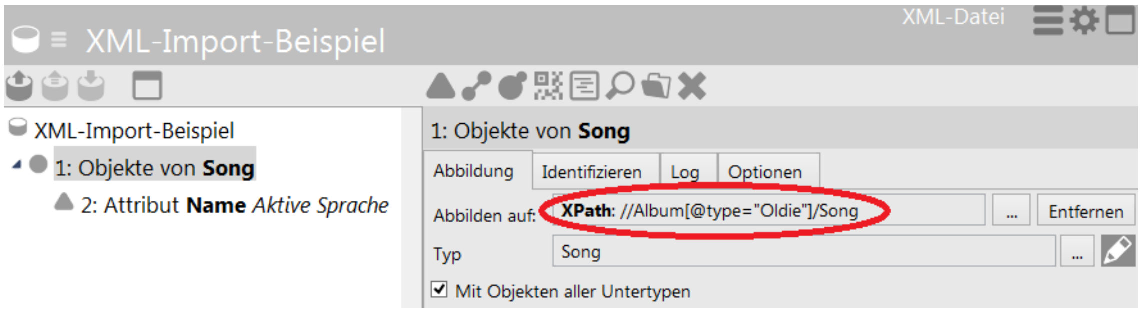
The correct XPath expression is:

```
//Album[@type="Oldie"]/Song
```

Explanation in detail:

//Album	Selects all albums; their position in the document is irrelevant.
Al-bum[@type="Oldie"]	Selects all albums of the “Oldie” type
Album/Song	Selects all songs that are sub-elements of albums.

We can now use this expression to define an equivalent for the object mapping of songs.

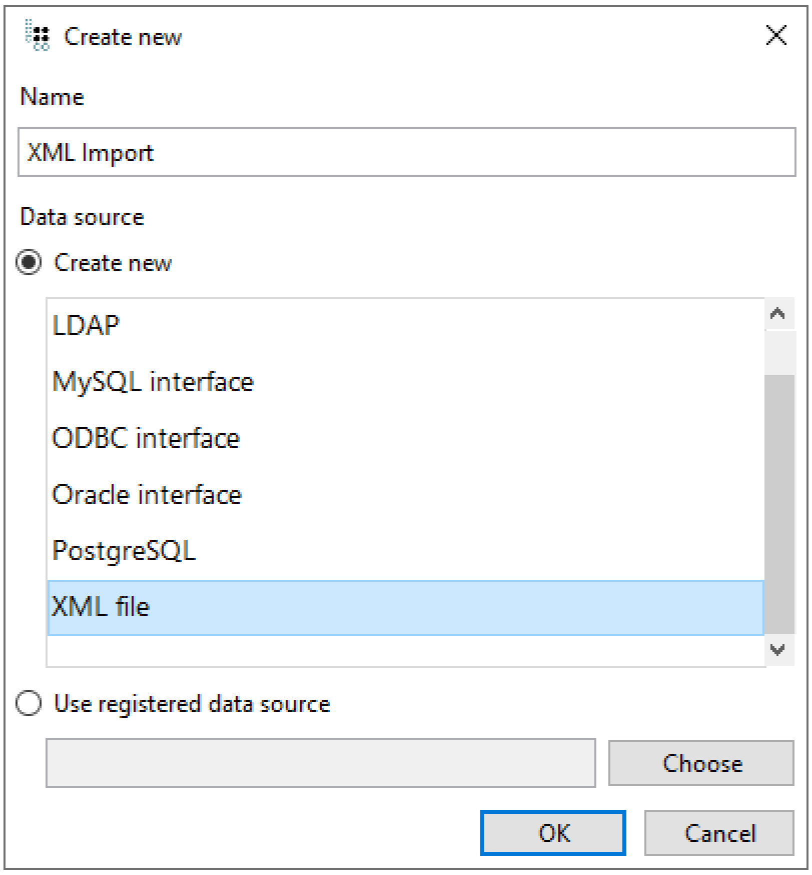


Abbilden auf: **XPath**: //Album[@type="Oldie"]/Song
Typ: Song
☑ Mit Objekten aller Untertypen

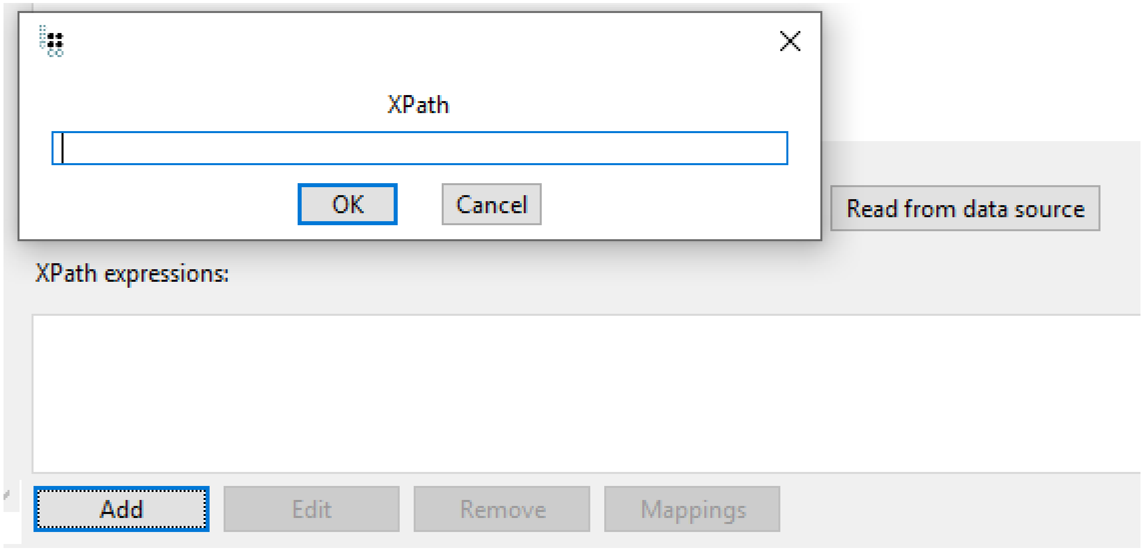
XPath also offers many other useful selection functions. We can, for example, select elements by their position in the document, use comparative operators, and specify alternative paths.

Alternative: XML import mapping for XML-based RDF files

If the schema in the semantic network is too specific for the existing RDF file *or* if the RDF file is too specific *or* the rdf schema is missing so that it cannot be imported by the import mechanism correctly, we can use the XML import mapping for specified import.



In most cases, we will need to use XPath expressions for dedicated value assignment. Pay attention that for the XML import mapping, an interactive step-by-step import is not available.



Examples for XPath expressions

Note: For Xpath expressions, the namespace (built up on to the qualifier) is not considered by the system for import mapping.

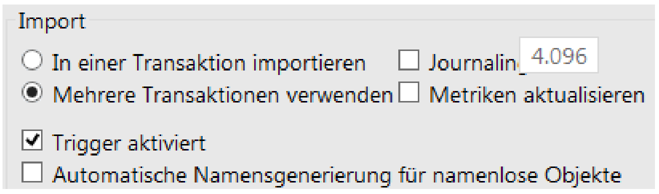
Input RDF-XML	XPath	Meaning
	//	Top-level of the RDF
	../	One level above
	../../xyz	Two levels above, from there the node below called “xyz”
<rdf:label>	/label/	Tag “label”
<rdf:prefLabel xml:lang="en"> Example </rdf:prefLabel>	prefLabel[@lang=”en”]	Node with attribute and certain attribute value. Output = "Example".

	ancestor::termEntry/attribute::id	Superordinate node on any level with name ("termEntry") and attribute ("id")
	/myparent/mychild[text()]	Text between certain tags

1.5.1.9 Further options, log and registry

1.5.1.9.1 Further options at the import

In the "Options" tab, the following functions are available for selection irrespective of the data source:


Import

( ) In einer Transaktion importieren
(●) Mehrere Transaktionen verwenden
☐ Journaling 4.096
☐ Metriken aktualisieren
☑ Trigger aktiviert
☐ Automatische Namensgenerierung für namenlose Objekte

Import in one transaction: This is slower than an import with several transactions and should only be used if a conflict would occur during an import with several transactions because many people are working in Knowledge Builder at the same time or because you want to import data where it matters that individual pieces of data are not viewed separately from each other. Example 1: Every hour, an import is executed with the machine load status. The combined load values must not exceed a certain value as that could result in a power failure. To ensure this rule can be taken into account (e.g. by means of a script), all values must be viewed jointly and then imported. Example 2: An import is executed with persons of which no more than one person may have a master key because only one master key exists. The import must also be performed in one transaction here because several transactions could result in missing the error that the attribute for the master key has been set for two persons.

Use several transactions: Default setting for fast import.

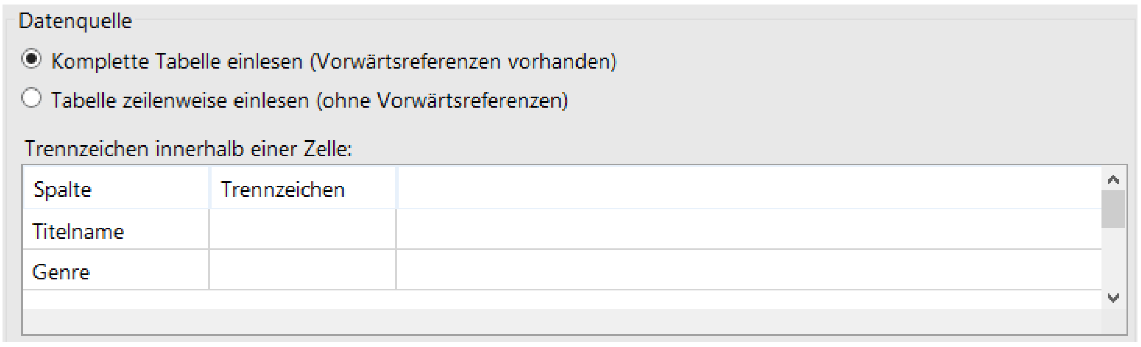
Journaling: Journaling should be used if very large amounts of data are deleted or modified in one import. The changes or deletions for these entries are only to be made to the index after 4,096 entries (the figure is variable). This speeds up the import because the index does not have to be used for every single change/deletion. Instead, these changes are copied to the index after a maximum of 4,096 changes.

Update metrics: Metrics are supposed to be updated if the import significantly affects the number of object types or property types, that is, if a large number of objects or properties of a type are added to the semantic graph database. If the metrics were not updated, this could negatively affect the performance of searches in which the corresponding types play a role.

Trigger activated: You can use this checkmark to determine if the trigger is supposed to be activated or not during import. If you wish to apply one trigger but not another one, you have to define two different mappings with the corresponding semantic elements. For information on triggers, refer to the Trigger chapter.

Automatic name generation for nameless objects: Enables the automatic name generation for nameless objects.

If there is a table-oriented source, we can make the following settings:


Datenquelle

[✓ Komplette Tabelle einlesen (Vorwärtsreferenzen vorhanden)]

Tabelle zeilenweise einlesen (ohne Vorwärtsreferenzen)

Trennzeichen innerhalb einer Zelle:

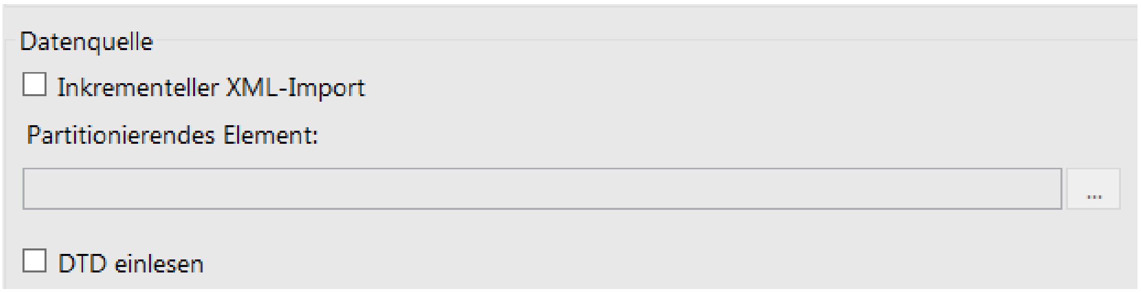
Spalte	Trennzeichen	
Titelname		
Genre		

Import entire table: Even though it can take longer to import the entire table at once, it makes sense to select this option if there are forward references, i.e. if relations are to be drawn between the objects to be imported. In this case, both objects must already be available, which is not the case if the table is imported one row at a time. Furthermore, the progress display is more precise than for importing one row at a time.

Import table row by row: A table should always be imported one row at time when the table contains no source reference since this procedure speeds up the import.

Separators within a cell: Refer to the chapter Mapping several values for an object type for an object.

If we have an XML-based data source, the following functions are available:


Datenquelle

☐ Inkrementeller XML-Import

Partitionierendes Element:

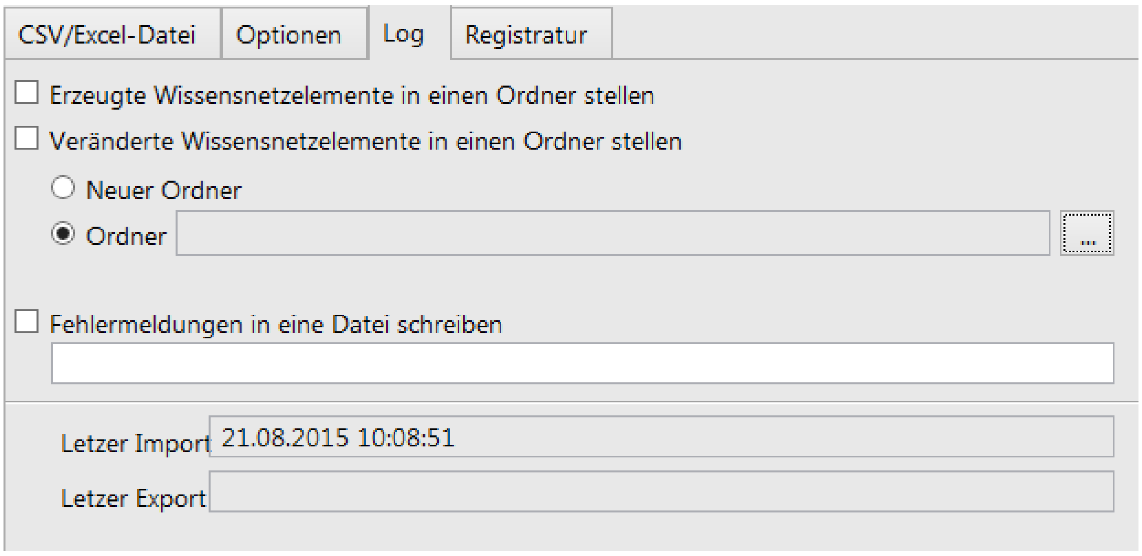
☐ DTD einlesen

Incremental XML import: The XML import is performed step-by-step**.** These steps are specified by the partitioning element.

Import DTD: Imports the document type definition (DTD)**.**

1.5.1.9.2 Log

The functions in the “Log” tab allow changes that are made upon import to be tracked.



☐ Erzeugte Wissensnetzelemente in einen Ordner stellen

☐ Veränderte Wissensnetzelemente in einen Ordner stellen

Neuer Ordner [✓ Ordner]

☐ Fehlermeldungen in eine Datei schreiben

Letzer Import 21.08.2015 10:08:51

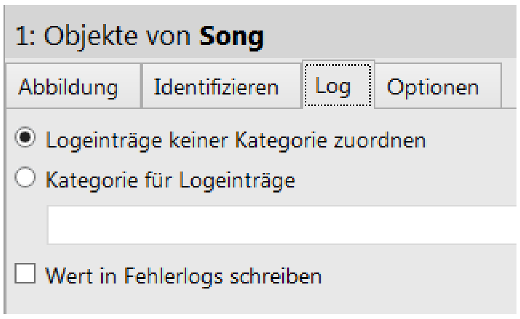
Letzer Export

Place generated semantic elements in a folder: If new objects, types or properties are generated by the import, they can be placed in a folder in the semantic graph database.

Place changed semantic elements in a folder: All properties or objects with properties that were changed by the import can be placed in a folder.

Write error messages to a file: Errors can occur during import (for example, there may have been an identifying attribute for several objects, which is why the object could not be identified uniquely). These errors are displayed in a window following import by default, and the option of saving the error log is provided. If this is to occur automatically, then a checkmark can be placed in the box and a file can be specified here.

Last import / Last export: The date and time of the last import performed and the last export performed are displayed here.


1: Objekte von **Song**

(●) Logeinträge keiner Kategorie zuordnen
( ) Kategorie für Logeinträge
☐ Wert in Fehlerlogs schreiben

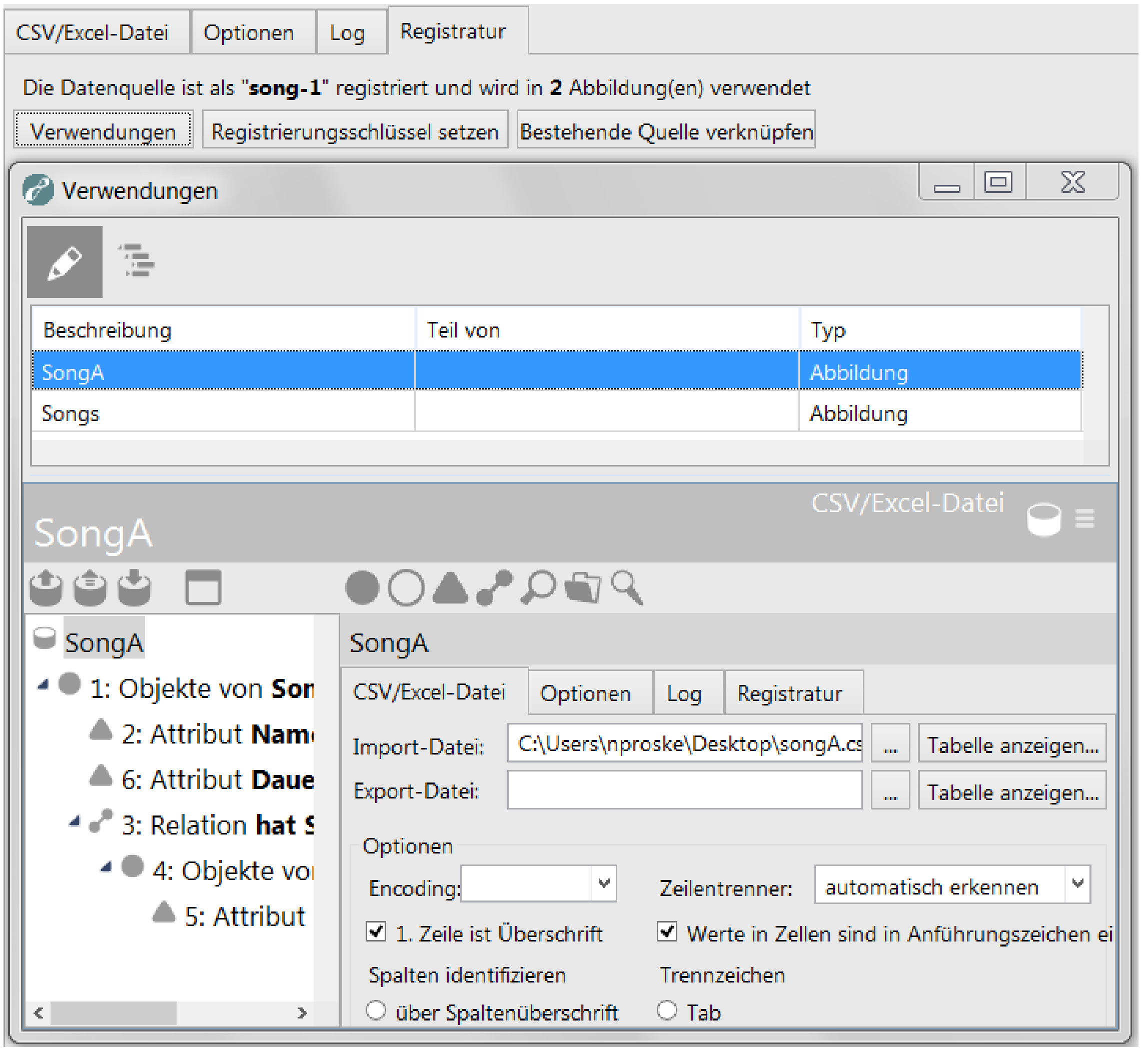
The “Log” tab is also available in the case of the individual mapping objects. When necessary, a category can be entered for log entries here. Moreover, it is possible to define that the value of the corresponding object/corresponding property should be written into the error log. This is not activated by default, in order to avoid revealing sensitive data (e.g. passwords).

1.5.1.9.3 Registry

The function “Set registration key” can be found under the “Registry” tab, and can be used to register the data source for other imports and exports.

The function “Link existing source” allows a registered source to be used again.

“References” shows other places where a data source is being used:



CSV/Excel-Datei | Optionen | Log | Registratur

Die Datenquelle ist als "**song-1**" registriert und wird in **2** Abbildung(en) verwendet

Verwendungen | Registrierungsschlüssel setzen | Bestehende Quelle verknüpfen

Verwendungen

Beschreibung	Teil von	Typ
SongA		Abbildung
Songs		Abbildung

SongA — CSV/Excel-Datei

- SongA
  - 1: Objekte von Son
    - 2: Attribut Nam
    - 6: Attribut Daue
    - 3: Relation hat S
      - 4: Objekte vo
        - 5: Attribut

SongA

CSV/Excel-Datei | Optionen | Log | Registratur

Import-Datei: C:\Users\nproske\Desktop\songA.cs ... Tabelle anzeigen...

Export-Datei: ... Tabelle anzeigen...

Optionen

Encoding:

Zeilentrenner: automatisch erkennen

☑ 1. Zeile ist Überschrift

☑ Werte in Zellen sind in Anführungszeichen ei

Spalten identifizieren

○ über Spaltenüberschrift

Trennzeichen

○ Tab

1.5.2 Attribute types and formats

One frequent job of attribute mapping is to import specific data from concrete objects, for example from persons: Telephone number, date of birth etc.

For the import of attributes for which i-views uses a specific format (e.g. date), the entries of the column to be imported must be provided in a form that is supported by i-views. For example, a string in the form abcde... cannot be imported to an attribute field of the date type; in this case, no value is imported for the corresponding object.

The following table lists the formats that i-views supports during the import of attributes. A table value yes or 1 is, for example, imported correctly as a Boolean attribute value (for a correspondingly defined attribute), while a value such as on or similar is not.

Attribute	Supported value formats
Selection	The mapping of import to attribute values can be configured with the “Value allocation” tab.

Boolean	The mapping of import to attribute values can be configured with the “Value allocation” tab.
File	It is possible to import files (e.g. images). For this to happen, either the absolute path to the file must be specified, or the files to be imported must be in the same directory (or a subdirectory that needs to be specified) as the import file.
Date	• <day> <monthName> <year>, e. g. 5 April 1982, 5-APR-1982 • <monthName> <day> <year>, e. g. April 5, 1982 • <monthNumber> <day> <year>, e. g. 4/5/1982 The separator between <day>, <monthName> and <year> can be a space, a comma or a hyphen, for example (but other characters are also possible). Valid month names are: • January , February , March , April , May , June , July , August , September , October , November , December • ’Jan’, ’Feb’, Mar’, ’Apr’, May’, ’Jun’, ’Jul’, ’Aug’, ’Sep’, Oct’, ’Nov’, Dec’. Please note: Two-digit years are expanded to 20xy (so 4/5/82 becomes 4/5/2082). If mapping is set to “Freely definable format”, the following tokens can be used: YYYY and YY (year), MM and M (month number), MMMM (name of month), MMM (abbreviated name of month), DD and D (day)
Date and time	For date and time see the corresponding attributes. The date must come before the time. If the time is omitted, 0:00 is used.
Color	Import not possible.
Fixed point figure	Import possible.
Integer	• Integers of any size • Floats (separated by a point), e.g. 1.82. The figures are rounded during import.
Internet link	Any URL possible.
Time	<hour>: <minute>: <second> <am/pm>, e.g. 8:23 pm (becomes 20:23:00) <minute>, <second> and <am/pm> can be omitted. If mapping is set to “Freely defined format” , the following tokens can be used: hh and h (hour), mm and m (minute), ss and s (second), mmm (millisecond)
String	Any string. No decoding is performed.

Boolean attributes and selection attributes

Selection or Boolean attributes can only assume values from a specified set; for selection attributes this is a specified list, and for Boolean attributes this is the value pair yes/no in the form of a clickable field. When importing these attributes, you can specify how the values from the import table are translated to attribute values of the semantic graph database. One option is to adopt the values as they are listed in the table; if they do not correspond to any possible attribute values defined in the semantic graph database, they are not imported. The other option is to specify value allocations between table values and attribute values, which are then imported.

1.5.3 Configuration of the export

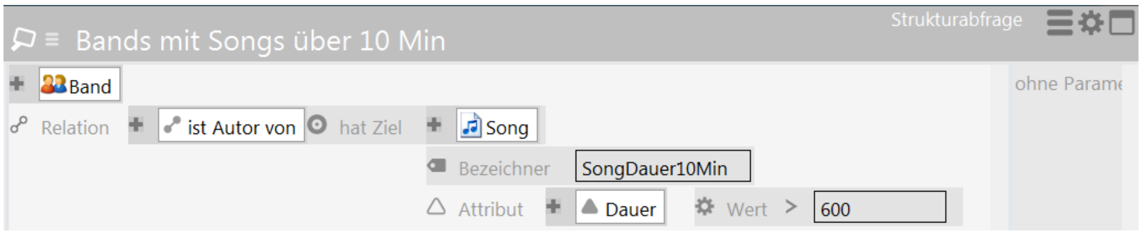
The export of data from a semantic graph database int a table is prepared in the same editor and in the same way as the import.

1. A new mapping is created in a table mapping folder in the main window.
2. In the table mapping editor, the file to be generated is specified.

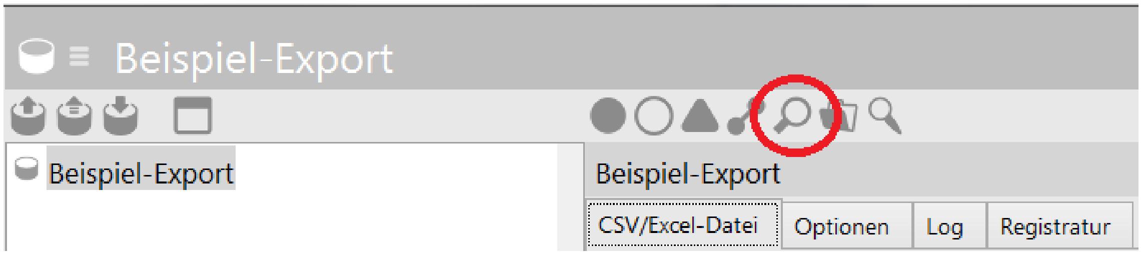
The difference to the import is that the columns are not imported from the table now but have to be created in the table mapping editor. Since the import and export editor are one and the same, you first have to select whether a new column to be created is a *standard* column or a *virtual property*. However, virtual properties cannot be used for export.

Exporting structured queries

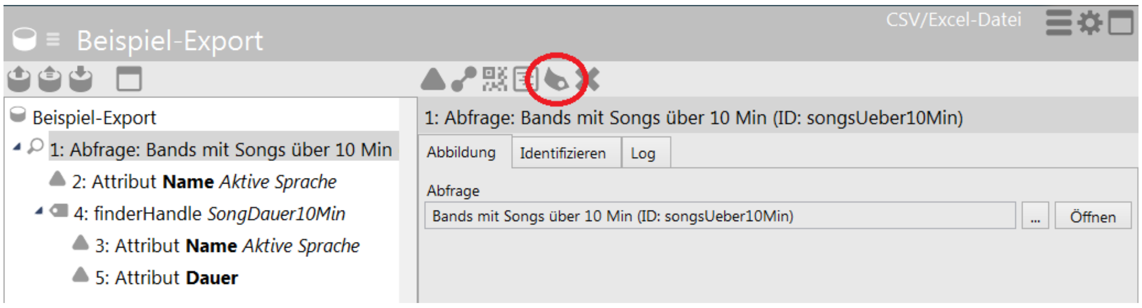
It is possible to export the result of a structured query. This procedure makes sense if only certain objects that have been restricted by a search are supposed to be exported. Let s assume, for example, we want to export all bands that have written songs that are more the 10 min long. To do this, we first have to define a structured query that collects the desired objects.



We then access this structured query from the configuration of the export. To do this, we select the mapping of a query rather than an object mapping in the mapping configuration header. The structured query can only be accessed with a registration key.



This has the effect that only the results of the structured query are exported. For these objects, we can now create properties that are to be included in the export: e.g. year the band was founded, members and songs. However, we might not want to export all of the songs of the bands we have thus compiled but only those songs that also match the search criterion, which is songs longer than 10 min in our example. To do this, we can assign identifiers to the individual search conditions in the structured query. These identifiers in turn can be addressed in the export definition.



Beispiel-Export

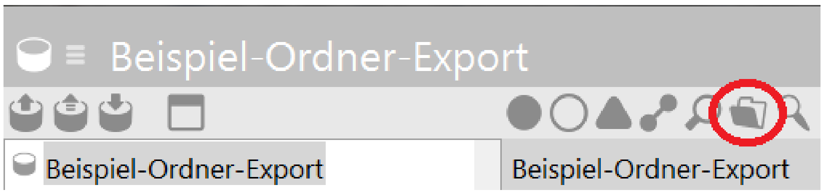
Beispiel-Export
1: Abfrage: Bands mit Songs über 10 Min
2: Attribut **Name** *Aktive Sprache*
4: finderHandle *SongDauer10Min*
3: Attribut **Name** *Aktive Sprache*
5: Attribut **Dauer**

1: Abfrage: Bands mit Songs über 10 Min (ID: songsUeber10Min)

Abfrage: Bands mit Songs über 10 Min (ID: songsUeber10Min)

Exporting collections of semantic objects

Collections of semantic objects can also be exported. These also need a registration key, which you can set under TECHNICAL -> Organizing folder.



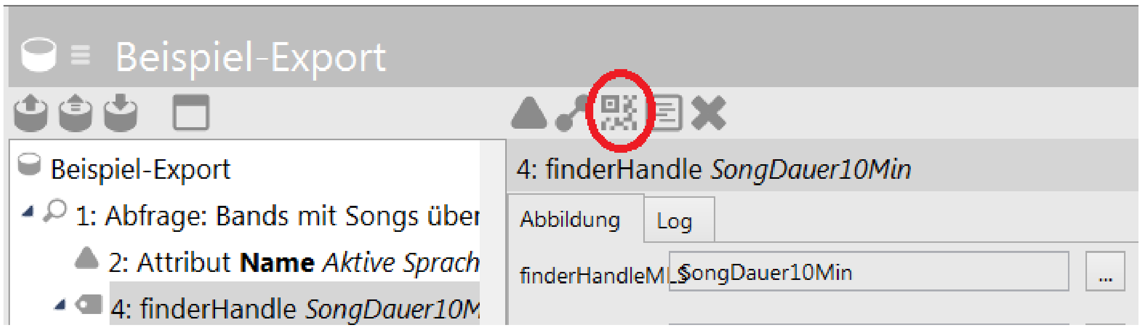
Beispiel-Ordner-Export

Beispiel-Ordner-Export

Beispiel-Ordner-Export

Exporting the frame ID

The mapping of the frame ID enables us to export the ID of a semantic element assigned in the semantic graph database. To do this, we simply select the object, type or property for which we need the ID and then choose the “New mapping of Frame ID” button:



Beispiel-Export

Beispiel-Export
1: Abfrage: Bands mit Songs über
2: Attribut **Name** *Aktive Sprach*
4: finderHandle *SongDauer10M*

4: finderHandle *SongDauer10Min*

finderHandleML: SongDauer10Min

We can also decide if we want to output the ID in string format (ID123_456) or as a 64-bit integer.

Export via script

Finally, we have one additional powerful tool for the export: script mapping. For further information on this subject, refer to the “Script mapping” chapter.

Export actions for database exports

Mapping the properties of an object for an export into a database takes place exactly like mapping for an import and all other types of mapping. The only difference is that the export action has to be specified for the export. This specifies which type of query is to be executed in the database. Three export actions are available:

The following actions are available in the selection dialog that opens:

- **Create new data records in table**: New data records are added to the database table. This action corresponds to an INSERT.
- **Update existing data records**: The data records are identified via an ID in the table. They are only overwritten if the value has changed. If there is no suitable data record, a new one is added. This action corresponds to an UPDATE.
- **Overwrite table content during export**: All data records are first deleted and then written again. This action corresponds to an DELETE on the entire table followed by an INSERT.

1.5.4 RDF-import/export

RDF is a standard format for semantic data models. We can use the RDF import and export to exchange data between the semantic graph database and other applications, and also to transport data from one i-views semantic network to another.

During an RDF export, the entire semantic network is exported into an RDF file. RDF import, in contrast, is interactive and selective. That is, we can specify at schema level as well as for individual objects and properties what is supposed to be imported and what not.

Reconciliation from RDF with the existing objects in the semantic network

If the RDF data originates from the same schema as the network into which it is imported, e.g. from a backup copy, the RDF import automatically assigns objects and object types by means of their ID. Just like for table and XML imports, we can use the import settings to determine, e.g. whether existing objects are to be updated by the import or if new ones are supposed to be created.

If the data originates from another source, the default setting of the import is into a separate subnet. We can also integrate this external information into our stock by means of manual assignments using the "Map to" function in the Mapping interface.

1.5.4.1 Basic principles

In this section we have a look on the basic principles of RDF and the special cases to be obeyed for import mapping. For further information about the RDF standard, see: w3c.org/rdf. In general, the i-views Knowledge-Builder supports XML-RDF.

For identifying the content within the RDF file and the Knowledge-Builder as well, the RDF-URI is used. It comprises the base URI (= base URL) and the RDF-ID:

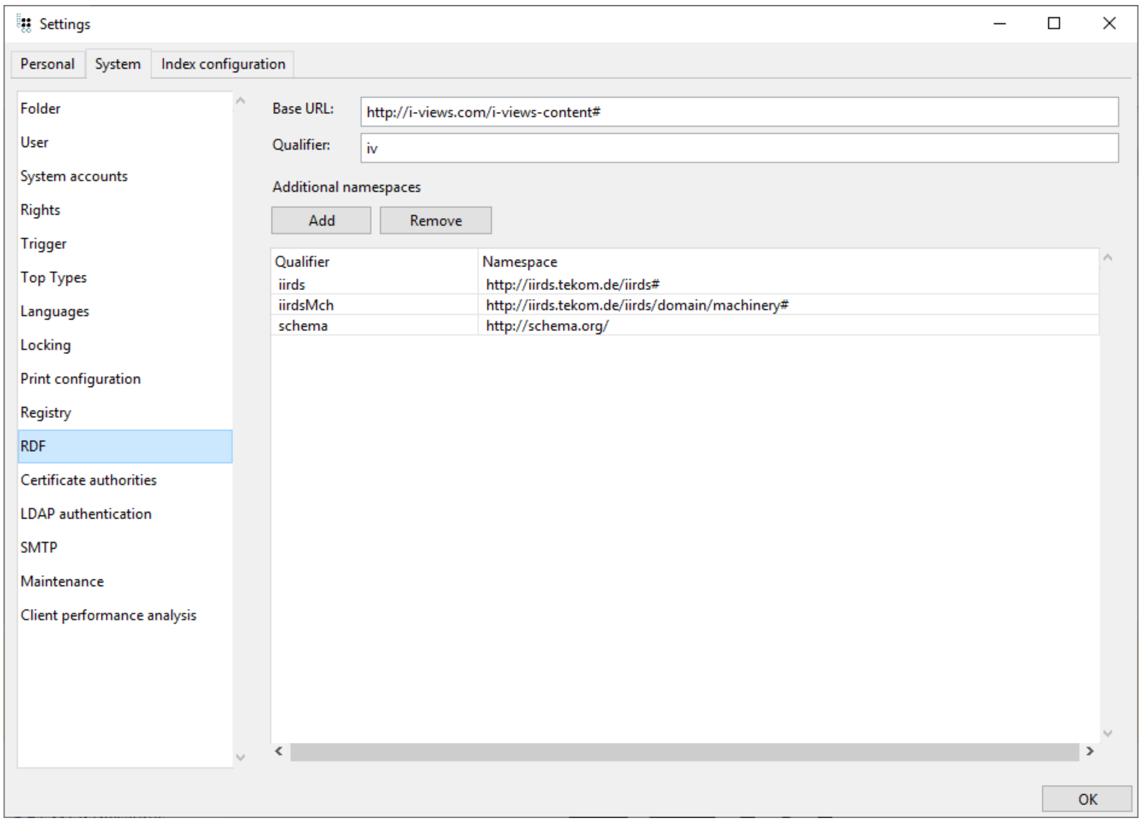
```
[RDF-URI] = [Base URI] + [RDF-ID]
```

The **Base URI** syntax in RDF is constructed by the "xml:base" prefix, like in "xml:base=http://example.org/". The base "base" is only a namespace for individual domains; the qualifier "xml" is for readability reasons in terms of XML transportation, which is irrelevant for import.

A relative URI in RDF is built up by the syntax "rdf:about". Attribute values are most likely text between tags: <rdf:prefLabel xml:lang="en">**Example**</rdf:prefLabel>, surrounded by the translation layer identified by "xml:lang". Relations will be formed by RDF-entries like "rdf:resource". IDs will only be created via import in the Knowledge-Builder if they are literally written in the RDF file. The RDF-ID is no absolute identifying characteristic! It is not recommended to set RDF-IDs manually in the RDF file, since duplicate values can lead to data being mislocated.

Global settings

The Knowledge-Builder Base URL is defined in the settings menu and it is valid for both import and export:



Base URL: http://i-views.com/i-views-content#
Qualifier: iv

Qualifier	Namespace
iirds	http://iirds.tekom.de/iirds#
iirdsMch	http://iirds.tekom.de/iirds/domain/machinery#
schema	http://schema.org/

“Additional namespaces” is for export only.

Note: Always use a local copy of the network for trying out RDF-Import. If all settings led to a successful import, then make the import on the real instance.

Possible issues

- In most cases when importing external RDF (RDF which didn t have been created by the Knowledge-Builder itself out of the same knowledge network), the namespaces possibly won t fit. This results into lots of types within separate main types being created in the network.
  Therefore we can prepare the import as described at the end of this section.

- In RDF, the definition and assignment of properties can lead to creation of many objects in the network which normally should be formed into an attribute value of some certain object instead.
  Therefore a manual correction of the type assignment in the mapping or an alternative

XML import using XPath expressions (Xpath 1.0) might be needed.

- Don t choose the option “Identify objects with global URI also by local ID” if the base URL in the RDF differs from the Knowledge-Builder base URL. Furthermore, some RDF-ID in the RDF file could be identical by accident with some existing ID in the network, resulting into the object in the network being overwritten!
  Always use the RDF-URI for identification.

- If your RDF file doesn t contain a base URL, the file path of the RDF will be used as the base URL instead.
  This can be checked by opening the import dialog first. We then can add the RDF-URI or RDF-URI-Alias accordingly and then check the assignment again by opening the import dialog once again.

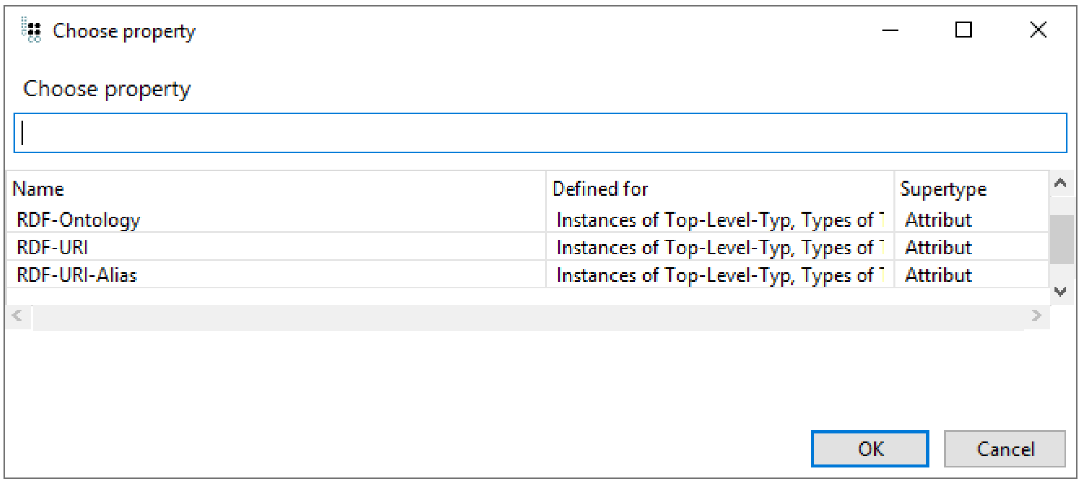
Preparation before import

Imports can be prepared regarding type assignment in the case that the RDF files contains foreign base URLs. Because RDF imports can lead to schema changes, it is always recommended to try the RDF import on a local copy of the knowledge-network before. To do so, we continue as follows:

1. Before importing, first create the scheme manually (object types, attribute types and relation types).

2. For the types, add the RDF properties by clicking on “Add attribute or relation”

   :

Name	Defined for	Supertype
RDF-Ontology	Instances of Top-Level-Typ, Types of	Attribut
RDF-URI	Instances of Top-Level-Typ, Types of	Attribut
RDF-URI-Alias	Instances of Top-Level-Typ, Types of	Attribut

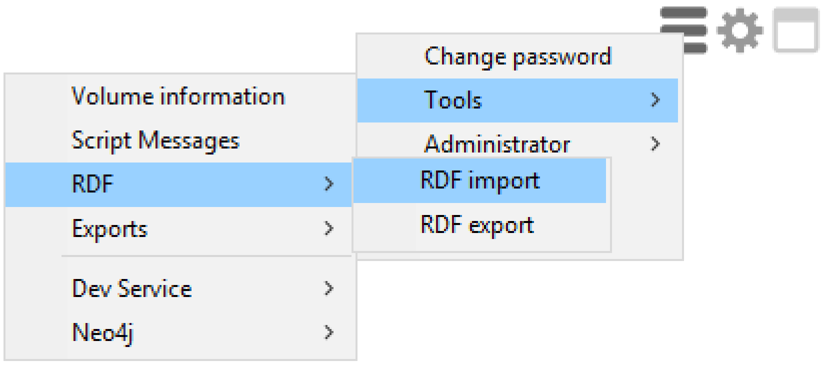
   RDF-Ontology: Attribute for RDF base URI “xml:base”
   RDF-URI: Attribute for “rdf:about”
   RDF-Alias: Further attribute, if the element is being fed by several RDF with different URIs

3. Open import dialog and check import mapping.

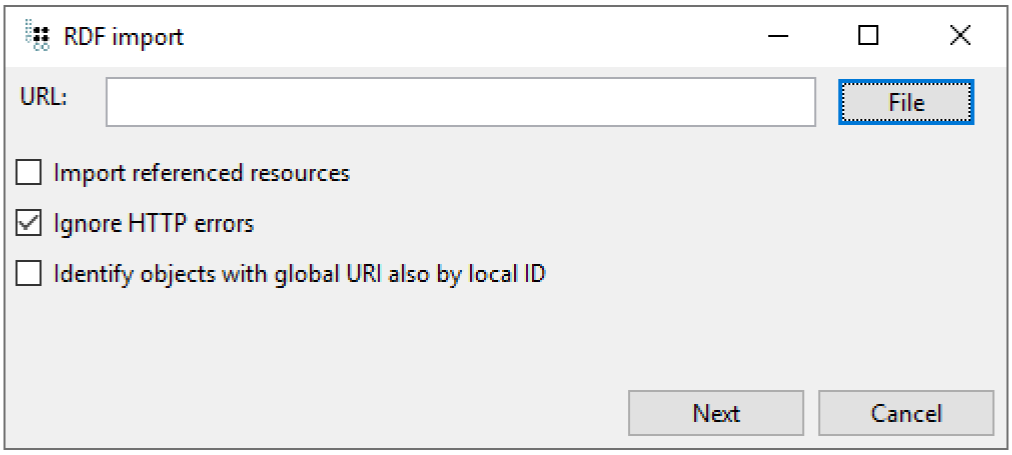
4. If the adjustments lead to the intended mapping, start the import and check the result.

1.5.4.2 RDF import

For accessing the RDF import mechanism, go to the global actions settings and choose Tools > RDF > RDF-Import.



A dialog opens for choosing the import file:


RDF import
URL:
File
Import referenced resources
Ignore HTTP errors
Identify objects with global URI also by local ID
Next
Cancel

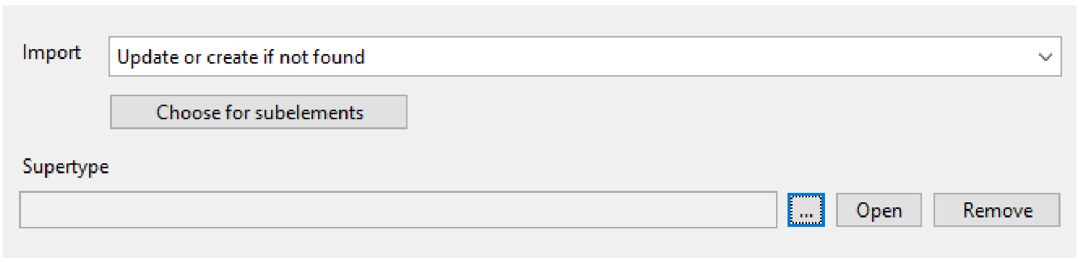
Options:

- Import referenced resources:
  If this option is chosen, all referenced resources specified in the RDF file are going to be imported additionally.
  Note: Be aware that the referenced resource itself can contain further references, leading to much more data being imported than initially intended.
- Ignore HTTP errors:
  The Knowledge-Builder will return error messages if the RDF namespace label is missing after the URL; only the namespace http-URL at the top will be considered.
- Identify objects with global URI also by local ID:
  This option only makes sense if the rdf to be imported is originated from the same knowledge network for which it is intended to be imported. Importing RDF with only considering the ID can lead to data being overwritten when the RDF is from another domain and the IDs match accidently. This option does not make sense when the RDF base URL differs from the knowledge network base URL.

Note: When importing RDF, for every unknown namespace a separate main type will be created in the knowledge network. The assignment of RDF content to dedicated knowledge network types depends on how the information is represented in the RDF file.

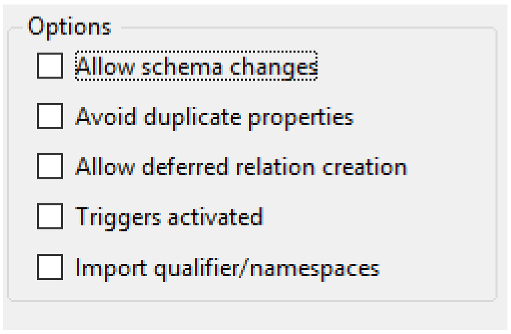
Setting the import options

1. Manual supertype mapping:
   Per default, the RDF-URL (RDF ontology) will be used as supertype assignment.
   For every type within the RDF, you can choose the supertype mapping in the semantic network manually if a different type assignment is needed:

   
   Import: Update or create if not found

   In order to be sure about which supertypes will be created by the import, you can check this in the “Schema changes” tab. By clicking on “Show in tree”, you can quickly jump to the location of a type in the hierarchy structure. The “Legend” tab explains the import mapping symbols.

2. Import options:

   
   Options
   - [ ] Allow schema changes
   - [ ] Avoid duplicate properties
   - [ ] Allow deferred relation creation
   - [ ] Triggers activated
   - [ ] Import qualifier/namespaces

   a. Allow schema changes: Since you don t want a file to change the schema, it is recommended to disable this option

   b. Avoid duplicate properties: Because in RDF properties cannot be assigned with an ID, a unique identification of properties and their values within the knowledge network is not possible when importing RDF without krdf. When you want to import a foreign RDF without krdf, it is recommended to enable this option.

   When transferring RDF between knowledge networks, knowledge network specific attributes can be identified by means of the enhanced krdf syntax. This includes properties for view configuration, REST configuration, attribute values, relation targets, meta-properties on relations etc. In this second case, it might be needed to disable the option. Pay attention that krdf adds the internal frame IDs for instances and properties whereas external IDs have no impact on identification of such content. The Knowledge-Builder automatically creates unique frame IDs when new elements are created within the knowledge network - either by an import or by the user.

c. Allow deferred relation creation: When importing data from public resources, the attribute “reference to URL” can be created as a substitute reference for (temporarily unavailable) dependencies. The attribute then can be used for re-identification in deferred imports. This might be useful when empty parts including URL without type definition exist within the RDF file.

d. Triggers activated: Normally, triggers are not activated during RDF import. If you nevertheless wish triggers being activated, enable this option.

e. Import qualifier/namespace: This option only makes sense when re-importing RDF that has been previously created out of the same network. If you import a RDF with a foreign namespace, skip this option.

3. Log options:
   a. Create folder with imported objects: This option allows you to inspect the imported objects within a folder that will be created in the working folder.

   b. With relation targets: When the RDF file contains new objects with relations to targets that already exist in the knowledge network, the relation targets will be included in the folder of imported objects.

4. Transaction options:
   a. Import in a single transaction: This is the most common import method.
   b. Use multiple transactions for import: This option is recommended when the RDF file contains a huge amount of content or when the connection to the external resource might be weak or unstable. When an error occurs, the amount of content affected by a rollback will be less due to the increased import steps in terms of transactions.
5. If you checked all settings, start the import and check the result.

Alternative: XML import mapping

If the schema in the semantic network is too specific for the existing RDF file *or* if the RDF file is too specific *or* the rdf schema is missing so that it cannot be imported by the import mechanism correctly, we can use the XML import mapping for specified import.

In most cases, we will need to use XPath expressions for dedicated value assignment. Pay attention that for the XML import mapping, an interactive step-by-step import is not available.

Examples for XPath expressions

Note: For Xpath expressions, the namespace (built up on to the qualifier) is not considered by the system for import mapping.

Input RDF-XML	XPath	Meaning

	//	Top-level of the RDF
	../	One level above
	../../xyz	Two levels above, from there the node below called “xyz”
<rdf:label>	/label/	Tag “label”
<rdf:prefLabel xml:lang="en"> Example </rdf:prefLabel>	prefLabel[@lang=”en”]	Node with at-tribute and cer-tain at-tribute value. Out-put = "Example".

	ancestor::termEntry/attribute::id	Superordinate node on any level with name (“termEntry”) and attribute (“id”)
	/myparent/mychild[text()]	Text between certain tags

Further RDF import/export possibilities

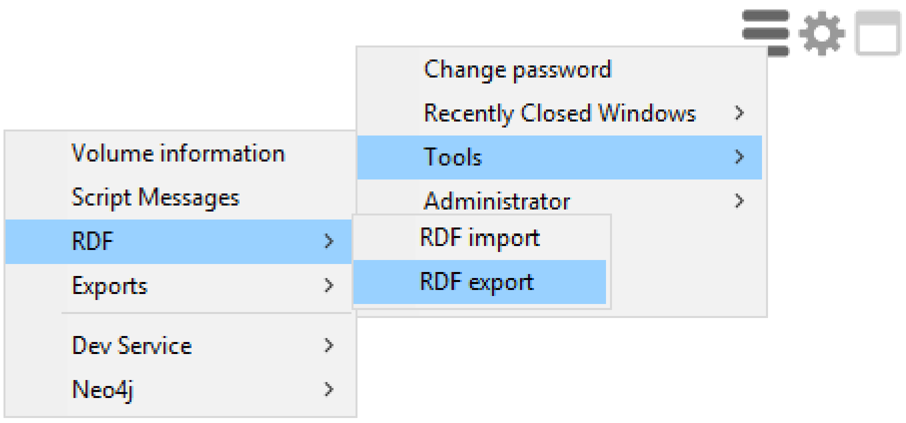
RDF files also can be imported or exported via the REST interface by means of a JavaScript mapping. In this case, only global options for import are available as specified in the JavaScript API documentation:
http://documentation.i-views.com/5.3/javascript-api/$k.RDFImporter.html.

Exceptions: Within i-views content, URIs are generated automatically for the semantic elements when being created in the knowledge network.

1.5.4.3 RDF export

Exporting the whole semantic network as RDF

On the global actions menu, select Tools > RDF > RDF export.



Exporting parts of the network

It is possible to export just a part of the knowledge network, for example:

- Listed elements from an objects list
- Elements from within a semantic elements folder
- Elements from within a graph editor bookmark

If you wish to export listed elements **without collecting** them in a folder:

- Select the list elements to be exported. Open the context menu by means of a right click. Then choose “RDF export”.

For **collecting** the elements to be exported, there are severely possibilities:

1. Create a semantic network elements folder and add the elements:
   a. In your private or working folder, create a semantic network elements folder.
   b. Go to the objects list of your choice an add the elements to the folder by dragging & dropping them.
   -or-
   Select the elements in the object list and open the context menu by means of a right click. Then choose “Store selected elements in folder”.
2. Right-click on the semantic network elements folder and choose “RDF export”.

If you wish to put **all selected list elements** into a semantic network elements folder:

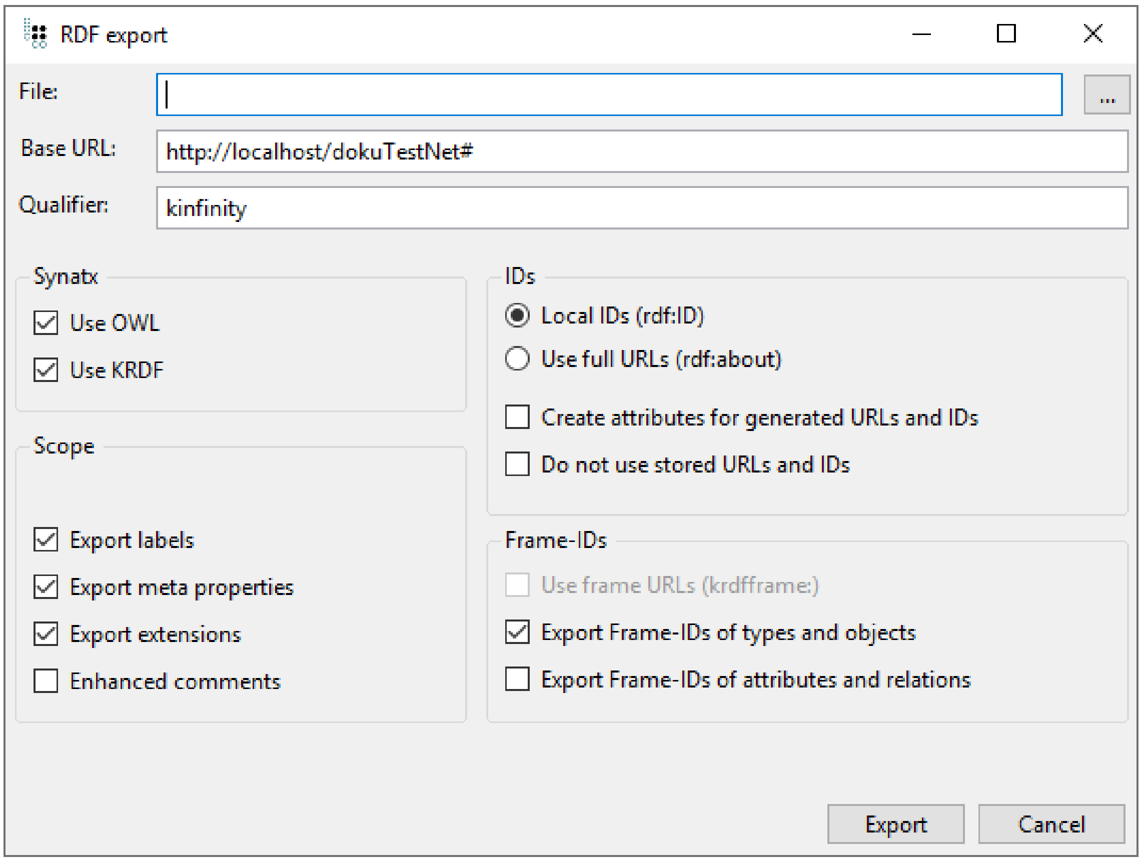
1. Open the context menu by means of a right click. Then choose “Copy semantic elements to new folder”.
2. Right-click on the semantic network elements folder and choose “RDF export”.

If you have already created a **bookmark of a graph editor view**, you simply can export them: Right-click on the bookmark and choose “RDF export”.

Note that only the content of the selection (of the folder or bookmark) will be exported. In terms of an object, this will be the cluster containing the attributes and the relation halves directly attached to the contained object only.

Note: When no base URL is specified in the global settings of the Knowledge-Builder, the path name of the RDF export file will be used as base URL instead.

RDF Exporting settings



File:

Base URL: http://localhost/dokuTestNet#

Qualifier: kinfinity

Synatx

- [x] Use OWL
- [x] Use KRDF

Scope

- [x] Export labels
- [x] Export meta properties
- [x] Export extensions
- [ ] Enhanced comments

IDs

● Local IDs (rdf:ID)
○ Use full URLs (rdf:about)

- [ ] Create attributes for generated URLs and IDs
- [ ] Do not use stored URLs and IDs

Frame-IDs

- [ ] Use frame URLs (krdfframe:)
- [x] Export Frame-IDs of types and objects
- [ ] Export Frame-IDs of attributes and relations

Syntax

- **Use OWL:** Since OWL (web ontology language) allows more options than the conventional RDF syntax, this option is always recommended except the case that the RDF is going to be reused for another system which does not accept OWL.
- **Use KRDF:** The KRDF syntax is i-views specific. It allows more enhanced constructions or representations compared to RDF or OWL like the following:
  o Instances that have several supertypes
  o Domains that consist of an intersection of supertypes
  o Frame IDs of semantic knowledge network elements

Scope

Note: The scope options comprise only schema (types) of the whole export

- **Export labels:** If activated, labels will not be exported as an attribute but in forms of a label literal with the syntax <label xml:lang="eng">.
- **Export meta properties:** In terms of official RDF specification, meta properties are out of scope. Nevertheless, meta properties can be regarded as a construct with statements about statements, as described in the reification rules of the RDF specification. Therefore, this option is useful when re-importing into an i-views semantic knowledge network is intended.

- **Export extensions:** This option allows the export of extensions of semantic objects.
- **Enhanced comments:** When enabled, XML comments with real name will be created. The exported RDF file will contain comments for dividing up into sections for objects, related objects and referenced schema hereafter, including statements about the relationships from each individual object to the related object.

IDs

- **Local IDs (rdf:ID):** This option only makes sense when re-importing the resulting rdf into the same knowledge network or into a highly similar knowledge network with the same namespace and correct IDs. If the target network accidently has existing elements with same ID, the elements might be overwritten without further recognition.
- **Use full URLs (rdf:about):** This option should be preferred, since the full RDF URL contains the namespace and thus ensures correct mapping when reimporting the RDF, provided the base URL of both RDF file and settings being identical.
- **Create attributes for generated URLs and IDs:**
- **Do not use stored URLs and IDs:**

Frame-IDs

- **Use frame URLs (krdfframe):** This option is only available in combination when used with full URLs instead of IDs. It provides internal URLs built up by frame IDs of the semantic knowledge network elements additionally.
- **Export Frame-IDs of types and objects:** Exporting frame IDs only is useful in the case if duplicating parts of the existing network is intended. Since frame-IDs change in various cases and differ highly due to their randomized creation (2^{29} possible values), they cannot be used for another knowledge network.

  Frame-IDs keep the same when:
  o Changing the type of an instance
  o Downloading a network
  o Updating a network

  Frame-IDs change when:
  o Changing relations into single-sided relations
  o Another instance of knowledge network is used
  o Creating objects, even if they will be given identical properties

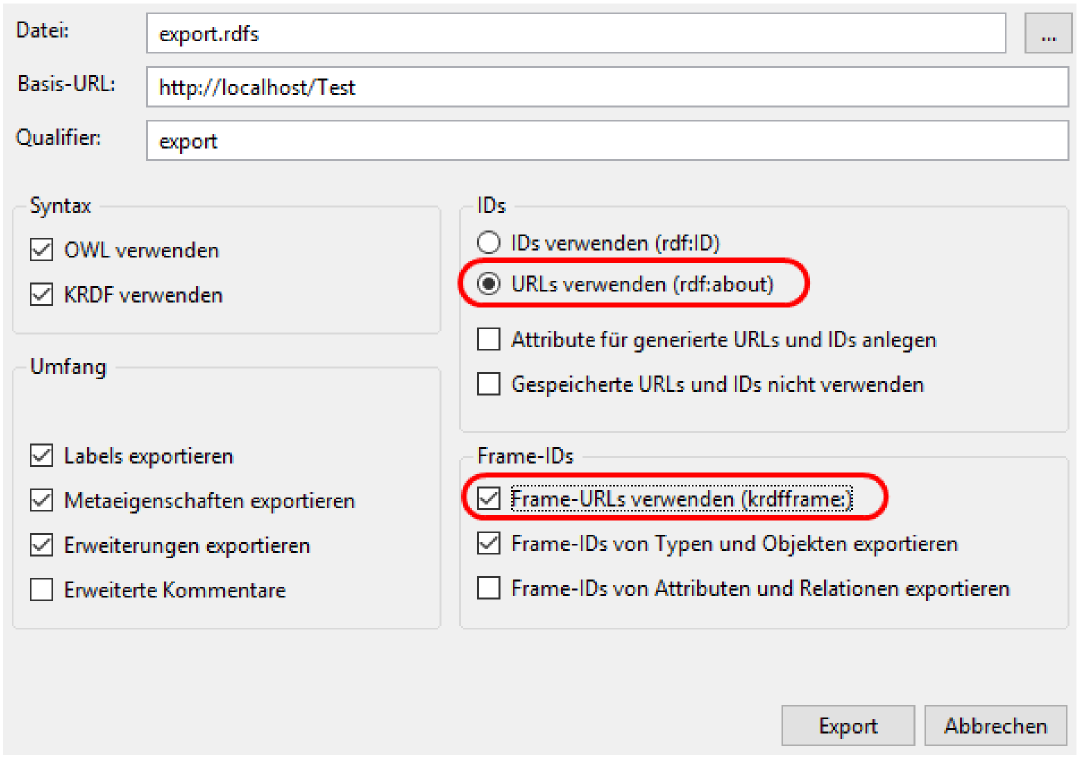
- **Export Frame-IDs of attributes and relations:** Exports frame-IDs of properties (attributes and relations) as well. As for exporting frame-IDs of objects and types, this option is useful for (partial) duplicating networks, but not for reuse into foreign networks

1.5.5 Restore deleted individuals from a back up

The RDF export and import is suitable for restoring deleted individuals from a backup network. Proceed as follows to do so:

1. Open the backup network in the Knowledge Builder

2. Create a new folder and save the individuals to be restored to it. To do so, right-click to open the context menu in the list view of the individuals to be copied, and select “Copy content to new folder” while selecting the new folder as the destination.
3. Open the RDF export on the newly created folder using the context menu
4. Specify a file name in the export dialog, select the options “Use URLS (rdf:about)” and “Use frame URLs (krdfframe:)” and execute the export:



Datei: export.rdfs
Basis-URL: http://localhost/Test
Qualifier: export

Syntax
- [✓] OWL verwenden
- [✓] KRDF verwenden

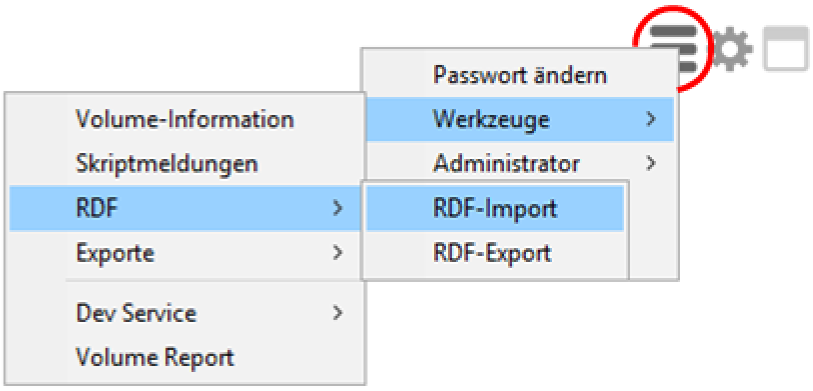
Umfang
- [✓] Labels exportieren
- [✓] Metaeigenschaften exportieren
- [✓] Erweiterungen exportieren
- [ ] Erweiterte Kommentare

IDs
- ( ) IDs verwenden (rdf:ID)
- (●) URLs verwenden (rdf:about)
- [ ] Attribute für generierte URLs und IDs anlegen
- [ ] Gespeicherte URLs und IDs nicht verwenden

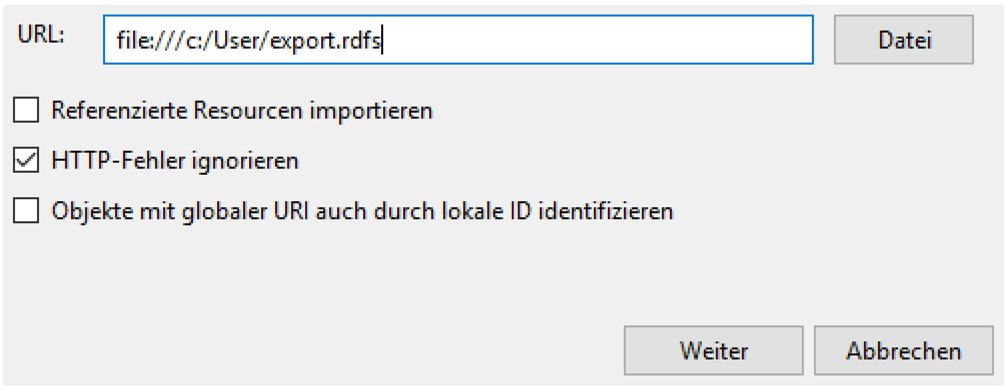
Frame-IDs
- [✓] Frame-URLs verwenden (krdfframe:)
- [✓] Frame-IDs von Typen und Objekten exportieren
- [ ] Frame-IDs von Attributen und Relationen exportieren

Note: the option “Use KRDF” results in i-views additionally copying specific content that cannot be mapped in full by means of RDF syntax.

5. Close the Knowledge Builder and open the target network in the Knowledge Builder
6. Open the RDF import dialog in the main menu under Tools > RDF > RDF import:



7. Select the file and press “Next”:

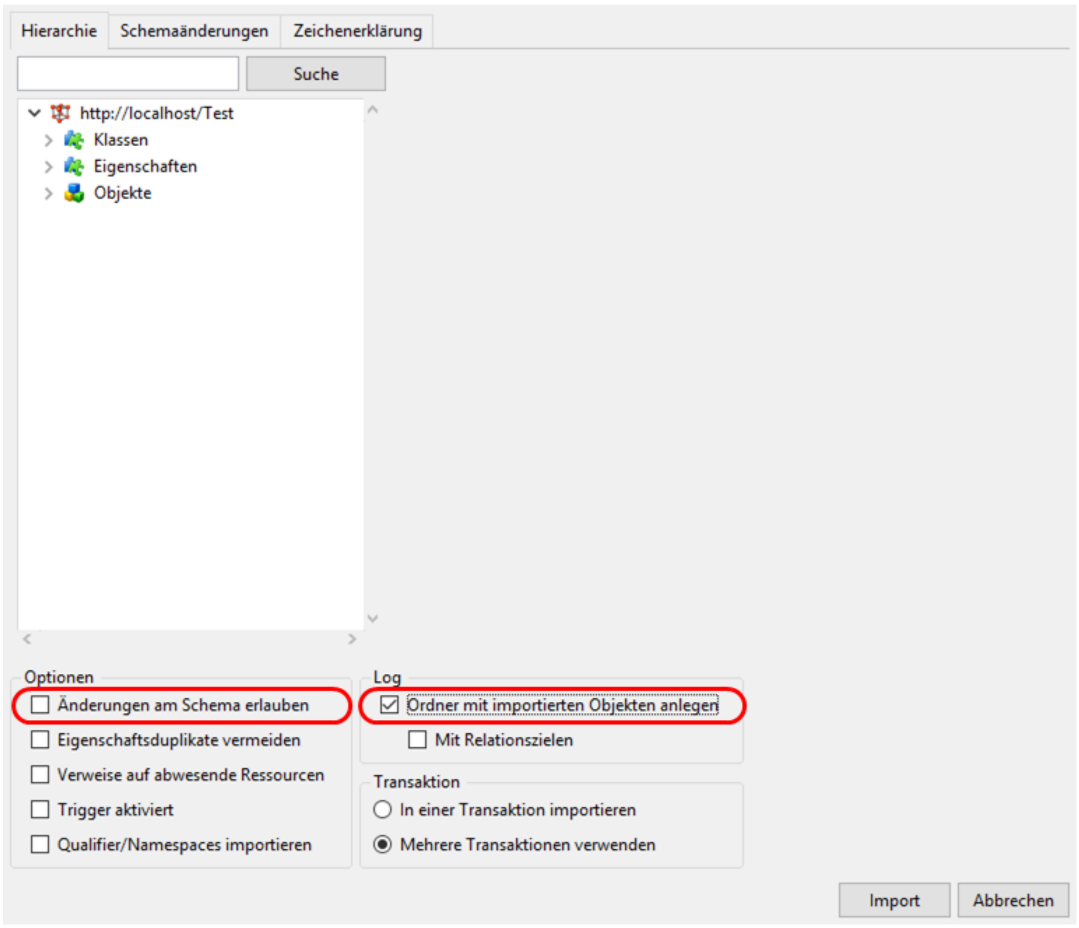

URL: file:///c:/User/export.rdfs

☐ Referenzierte Resourcen importieren

☑ HTTP-Fehler ignorieren

☐ Objekte mit globaler URI auch durch lokale ID identifizieren

8. Deactivate the option “Allow changes to the schema” in the selection dialog, and activate “Create folder with imported objects”:



Optionen
- ☐ Änderungen am Schema erlauben
- ☐ Eigenschaftsduplikate vermeiden
- ☐ Verweise auf abwesende Ressourcen
- ☐ Trigger aktiviert
- ☐ Qualifier/Namespaces importieren

Log
- ☑ Ordner mit importierten Objekten anlegen
- ☐ Mit Relationszielen

Transaktion
- ○ In einer Transaktion importieren
- ● Mehrere Transaktionen verwenden

9. Execute import
10. Check the restored individuals

1.5.6 Transport selected schema

The Admin tool can be used to transfer the entire schema of a semantic network from one semantic network to another via RDF export and import. However, if you only want to transfer selected types, you should consider using the “Copy schema to folder” function, which is available for all types via the context menu. This function creates a reference to the selected type together with all other (property) types that are required to create the selected type or objects of this type in the target network.

Once you have collected all required information in a folder, you can export this and import it into the target network in the same way as described in the previous chapter. However, the “Allow changes to schema” option should be deactivated in this case.

1.6 Access rights and triggers

This attribute handles the checking of access rights and triggers:

- **Access rights** regulate which operations on the semantic model 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 semantic graph database. 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.

1.6.1 Check of access right

We use rights to regulate user access to the data in the semantic network. 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 network-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 semantic network 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: "Paul [user] is not allowed to delete [operation] descriptions [element]”.

1.6.1.1 The activation of the rights system

In a newly created semantic network 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.)
4. Once you have connected the i-views knowledge portal, enter a user (object of the previously defined person object type) in the *Standard web user* field.

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.

Please 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 semantic model. 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.

Please 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.

1.6.1.2 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).

1.6.1.3 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**
	Grant access	Access is granted in the access situation to be checked.
	Deny access	Access is not granted in the access situation to be checked.

In general, there are two different deciders, a positive one called Grant access and a negative one called Deny access.

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.

1.6.1.4 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:

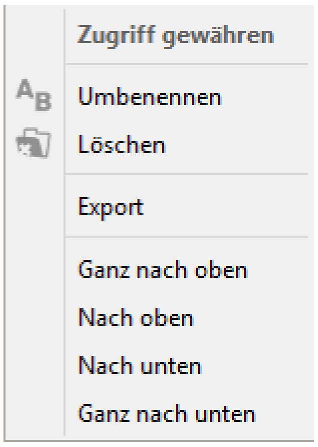
Symbol	Function	Description
	New operation filter	A new operation filter is generated.
	New query filter	A new query filter is generated.
	New property filter	A new property filter is generated.
	New organizing folder	A new organizing folder is generated.

	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

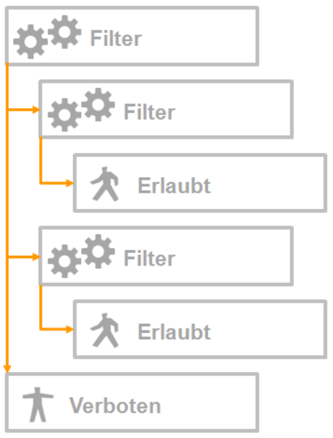
Assembling filters and deciders in the rights tree creates a large number of possible combinations for defining rights. By principle, there are 3 different procedures for defining rights:

- 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.

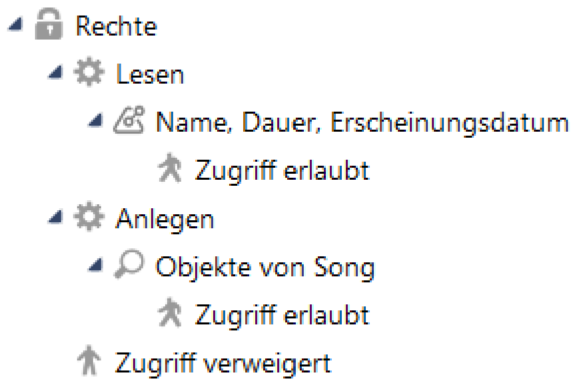
1.6.1.4.1 Positive configuration of rights

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:

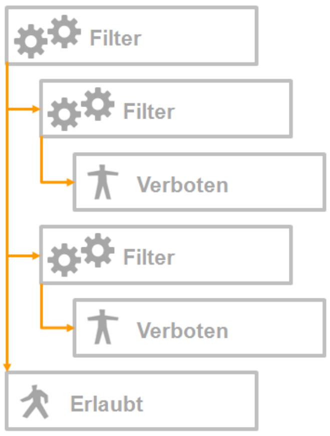


- Rechte
  - Lesen
    - Name, Dauer, Erscheinungsdatum
      - Zugriff erlaubt
  - Anlegen
    - Objekte von Song
      - Zugriff erlaubt
  - Zugriff verweigert

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.

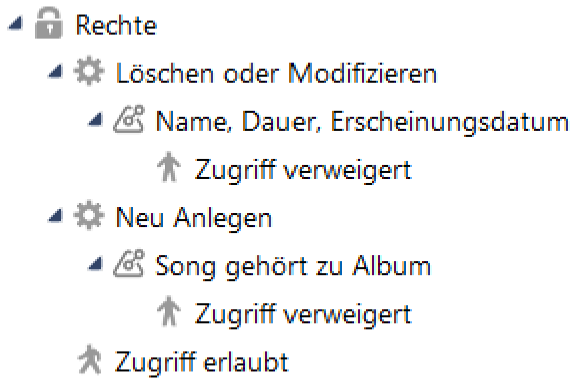
1.6.1.4.2 Negative configuration of rights

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:

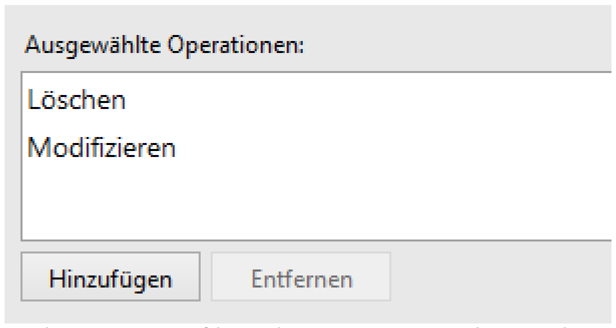

- Rechte
  - Löschen oder Modifizieren
    - Name, Dauer, Erscheinungsdatum
      - Zugriff verweigert
  - Neu Anlegen
    - Song gehört zu Album
      - Zugriff verweigert
  - Zugriff erlaubt

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.

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

Why 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.

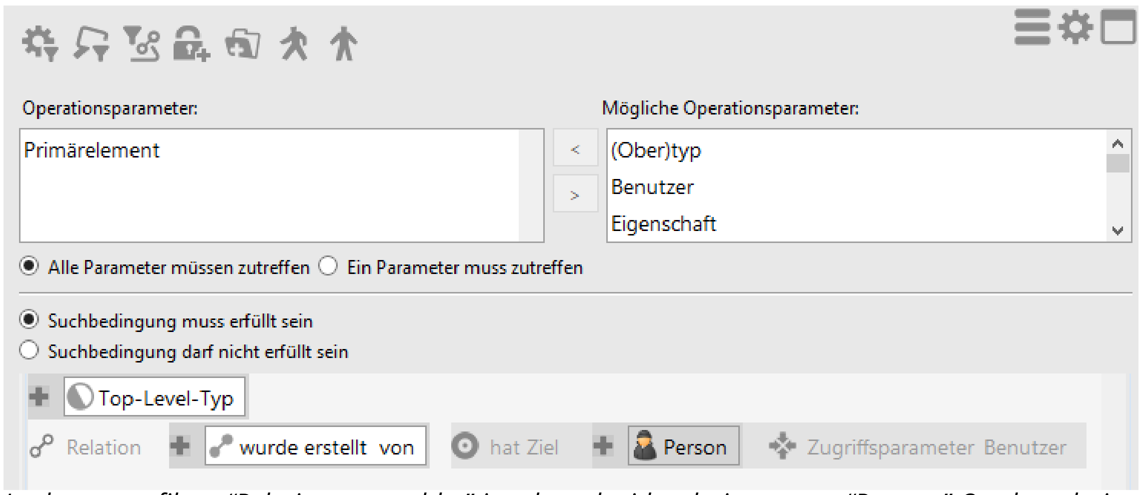
Operation filter


Ausgewählte Operationen:

Löschen
Modifizieren

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

Query filter


Operationsparameter: Primärelement

Mögliche Operationsparameter: (Ober)typ, Benutzer, Eigenschaft

[✓ Alle Parameter müssen zutreffen] Ein Parameter muss zutreffen

[✓ Suchbedingung muss erfüllt sein]
Suchbedingung darf nicht erfüllt sein

Top-Level-Typ
Relation – wurde erstellt von – hat Ziel – Person – Zugriffsparameter Benutzer

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 semantic element" was selected.

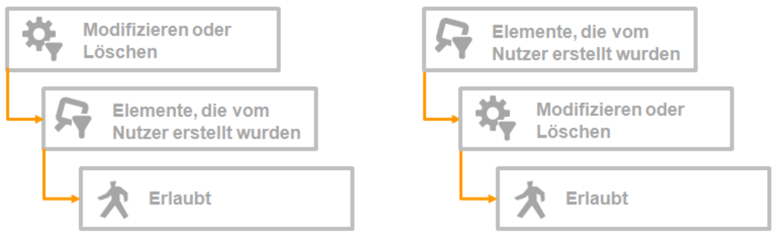
A question relating to the schema is: On which elements is the relation *was created by* defined? There are different options for implementing this relation in a semantic network:

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 semantic 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 semantic 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.

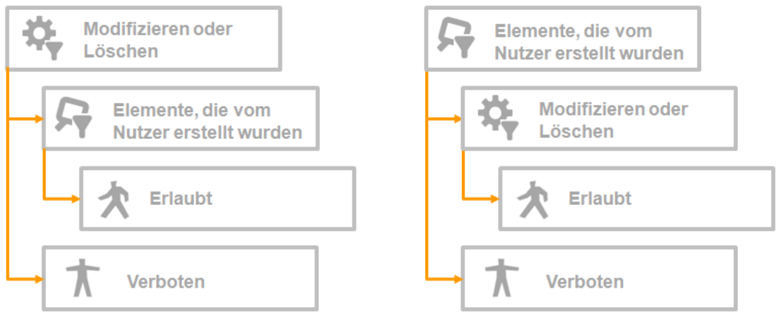

Modifizieren oder Löschen
Elemente, die vom Nutzer erstellt wurden
Erlaubt
Elemente, die vom Nutzer erstellt wurden
Modifizieren oder Löschen
Erlaubt

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.


Modifizieren oder Löschen
Elemente, die vom Nutzer erstellt wurden
Erlaubt
Verboten
Elemente, die vom Nutzer erstellt wurden
Modifizieren oder Löschen
Erlaubt
Verboten

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.

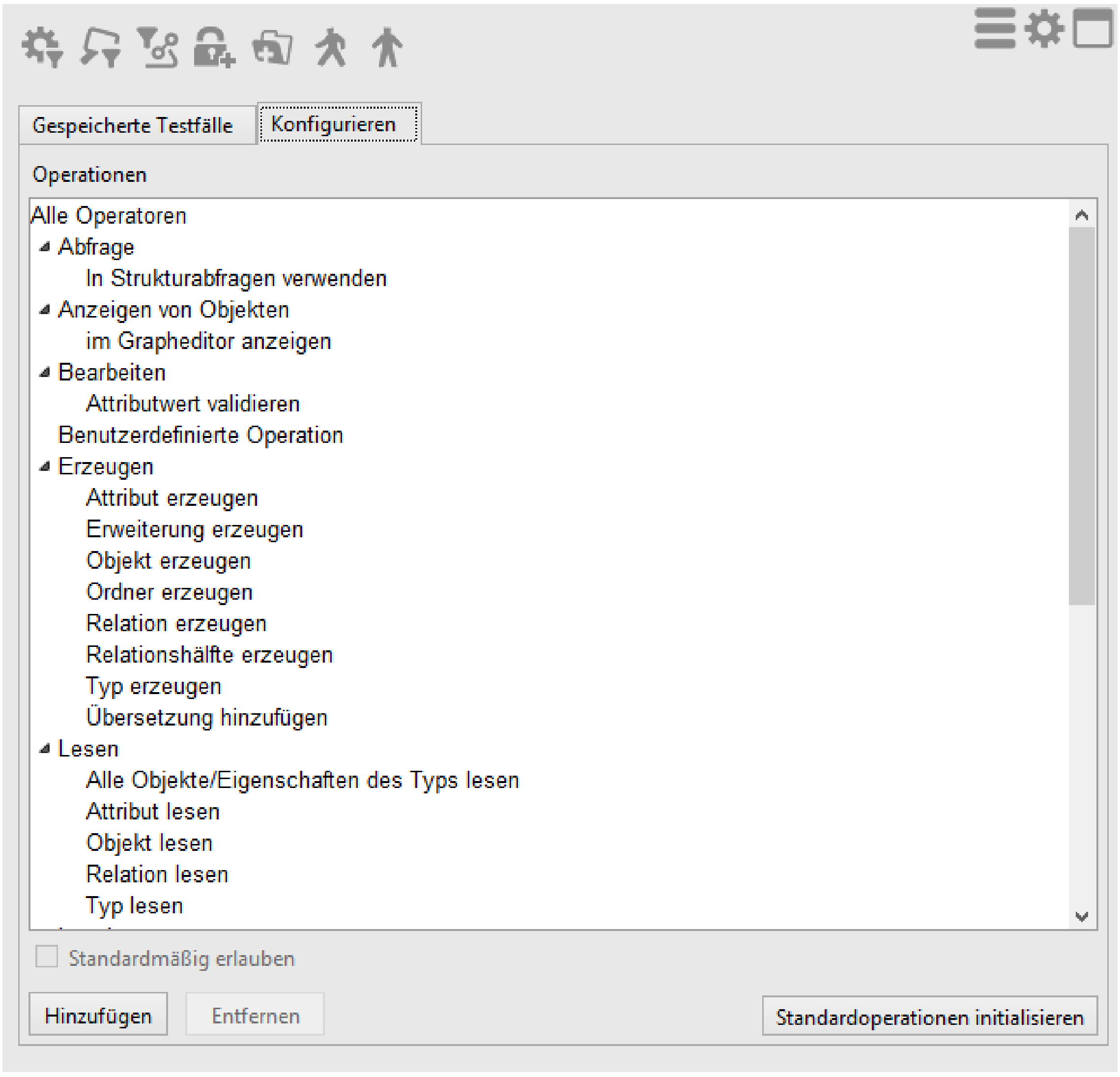
Version 2 (right)

- 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.

1.6.1.5 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.



Alle Operatoren
- Abfrage
  - In Strukturabfragen verwenden
- Anzeigen von Objekten
  - im Grapheditor anzeigen
- Bearbeiten
  - Attributwert validieren
- Benutzerdefinierte Operation
- Erzeugen
  - Attribut erzeugen
  - Erweiterung erzeugen
  - Objekt erzeugen
  - Ordner erzeugen
  - Relation erzeugen
  - Relationshälfte erzeugen
  - Typ erzeugen
  - Übersetzung hinzufügen
- Lesen
  - Alle Objekte/Eigenschaften des Typs lesen
  - Attribut lesen
  - Objekt lesen
  - Relation lesen
  - Typ lesen

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*.

1.6.2 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 semantic network. 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 semantic network. Most triggers are therefore usually project and network-specific; a consultation should be performed for each individual case.

1.6.2.1 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 semantic network 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.

1.6.2.2 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

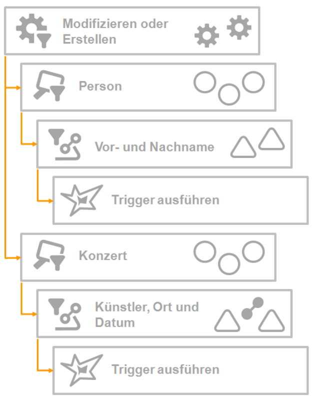
The order in which you define the triggers when designing the trigger tree usually has no effect on the performance of i-views. There are design recommendation for the rights tree, but these cannot be applied to the trigger tree, as the trigger tree is further traversed after a trigger action has been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

Sym-bol	Function	Description
📁	Organizing folder	Organizing folder for grouping subtrees

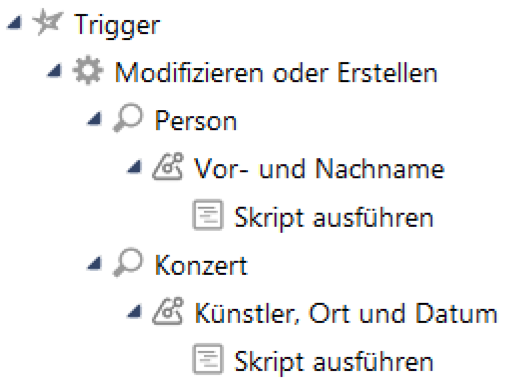
Example: trigger tree

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 concert type objects. The property filter only lets operations through if they are performed on the attributes or relations of the date, the event location or the artist. If these conditions apply, the corresponding script that compiles the name of the concert is executed.

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



Trigger
Modifizieren oder Erstellen
Person
Vor- und Nachname
Skript ausführen
Konzert
Künstler, Ort und Datum
Skript ausführen

1.6.2.3 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 semantic graph database 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.

1.6.2.4 Trigger actions

Trigger actions are used to perform intelligent operations in the semantic graph database, which, for example, automate or support work flows. However, they are only executed when the access situation and the links in the semantic network 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.

1.6.2.4.1 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.

Please 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 network.

Execution during data refactoring

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

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("geaendertAm", new Date());
    var userName = $k.user().name();
    if (userName)
        parameter.setAttributeValue("geaendertVon", userName);
    else
        parameter.attributes("geaendertVon").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)

1.6.2.4.2 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 semantic network, 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.

Log entry lines	State of the sem. network at the time
#pre	before triggering
#post	after triggering
#end	at the end of the transaction
#commit	when the transaction is successfully ended

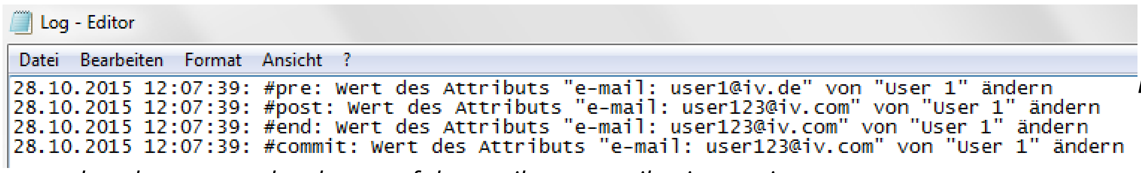
Log entries are used to retrace whether a trigger was executed in a specific access situation that actually occurred, and what it did. In contrast to this, a test can be performed in the test environment to determine whether a trigger would be triggered or not in a specific access situation, without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

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 *Enter log* in the trigger tree directly in front

of the script trigger.

Example:



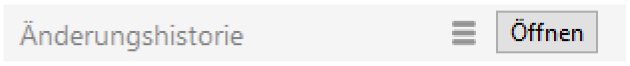
```
28.10.2015 12:07:39: #pre: Wert des Attributs "e-mail: user1@iv.de" von "User 1" ändern
28.10.2015 12:07:39: #post: Wert des Attributs "e-mail: user123@iv.com" von "User 1" ändern
28.10.2015 12:07:39: #end: Wert des Attributs "e-mail: user123@iv.com" von "User 1" ändern
28.10.2015 12:07:39: #commit: Wert des Attributs "e-mail: user123@iv.com" von "User 1" ändern
```

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

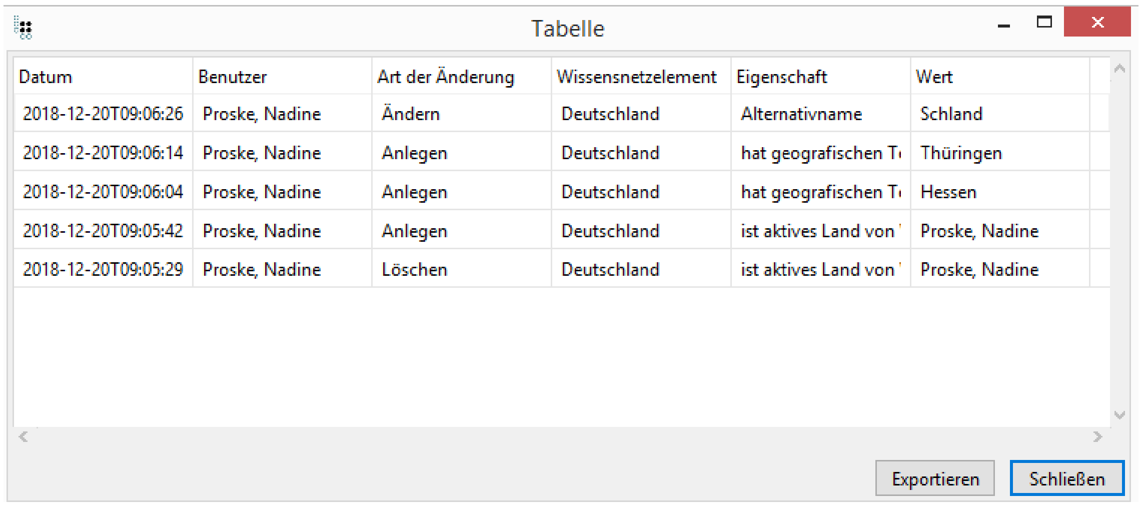
1.6.2.4.3 ChangeLog Trigger

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.



Datum	Benutzer	Art der Änderung	Wissensnetzelement	Eigenschaft	Wert
2018-12-20T09:06:26	Proske, Nadine	Ändern	Deutschland	Alternativname	Schland
2018-12-20T09:06:14	Proske, Nadine	Anlegen	Deutschland	hat geografischen T	Thüringen
2018-12-20T09:06:04	Proske, Nadine	Anlegen	Deutschland	hat geografischen T	Hessen
2018-12-20T09:05:42	Proske, Nadine	Anlegen	Deutschland	ist aktives Land von	Proske, Nadine
2018-12-20T09:05:29	Proske, Nadine	Löschen	Deutschland	ist aktives Land von	Proske, Nadine

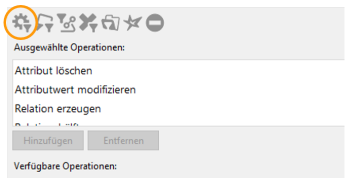
The trigger must contain the operation filters that will log the change history, and the elements where the attribute is to be visible.

The trigger script looks like this:

```
/**   * Perform the trigger   * @param parameter The chosen parameter, usually a semantic element   *
```

Example

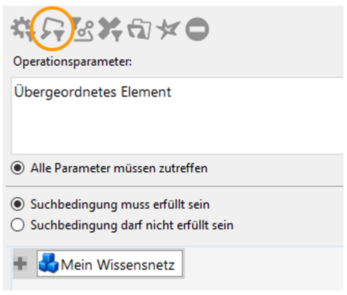
A change log is to be saved in all objects in a semantic network. The aim is to log the modification, creation and deletion of properties in the objects. First, an operation filter is created that reacts to the operations “Delete attribute”, “Modify attribute value”, “Create relation”, “Create relation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


Ausgewählte Operationen:

Attribut löschen
Attributwert modifizieren
Relation erzeugen

Verfügbare Operationen:

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


Operationsparameter:

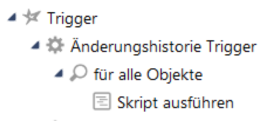
Übergeordnetes Element

(●) Alle Parameter müssen zutreffen

(●) Suchbedingung muss erfüllt sein
( ) Suchbedingung darf nicht erfüllt sein

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
  Änderungshistorie Trigger
    für alle Objekte
      Skript ausführen
```

1.6.3 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?

Sym-bol	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
✖	Delete filter	Filters the deletion of elements

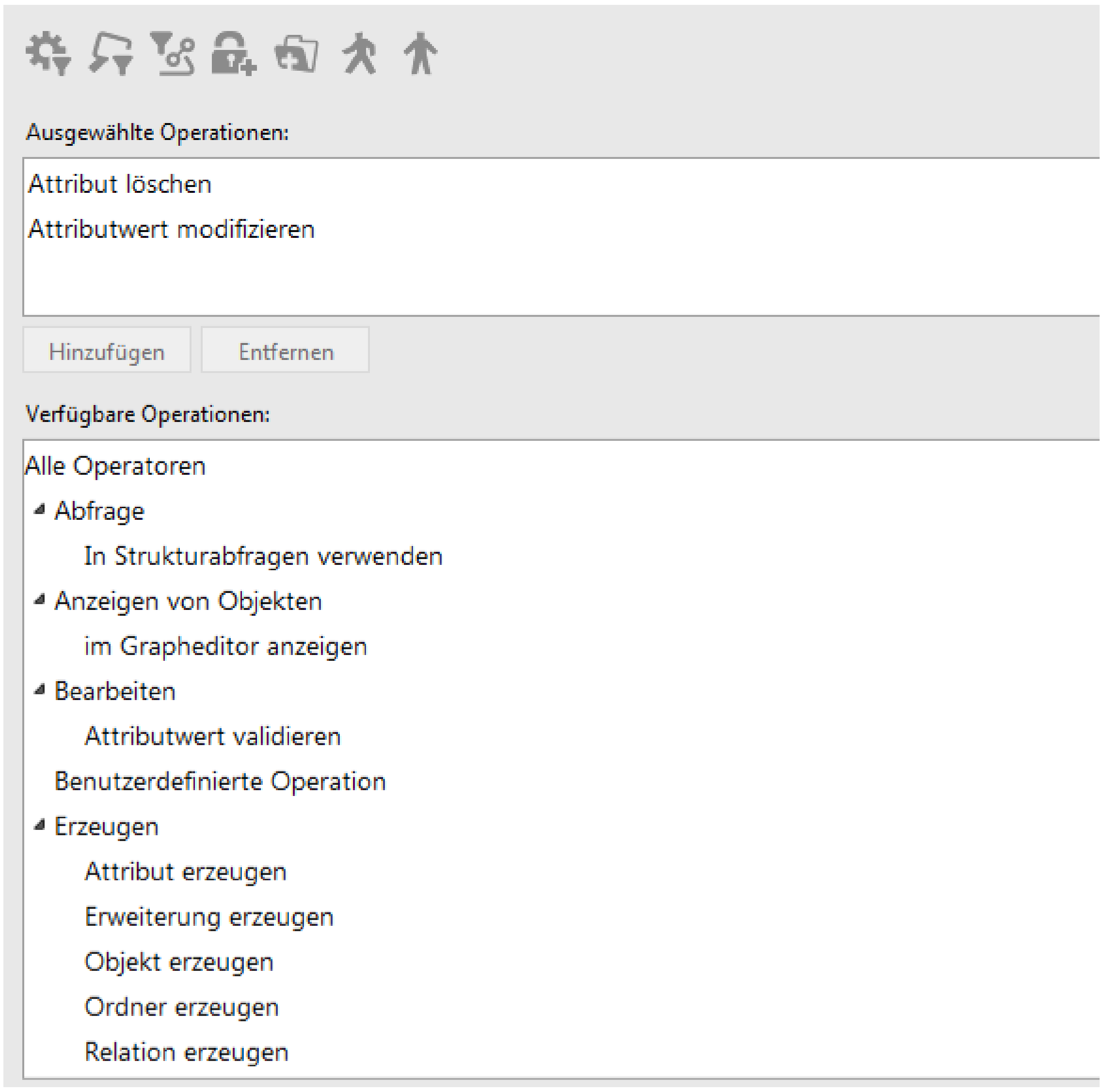
Operations can only be determined using an operation filter. Users can only be determined using a query filter. Properties can be determined using either query filters or property filters. The use of property filters makes sense when properties should be filtered regardless of other properties in the semantic model 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.

1.6.3.1 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.



Ausgewählte Operationen:

Attribut löschen
Attributwert modifizieren

Verfügbare Operationen:

Alle Operatoren
- Abfrage
  - In Strukturabfragen verwenden
- Anzeigen von Objekten
  - im Grapheditor anzeigen
- Bearbeiten
  - Attributwert validieren
- Benutzerdefinierte Operation
- Erzeugen
  - Attribut erzeugen
  - Erweiterung erzeugen
  - Objekt erzeugen
  - Ordner erzeugen
  - Relation erzeugen

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*.

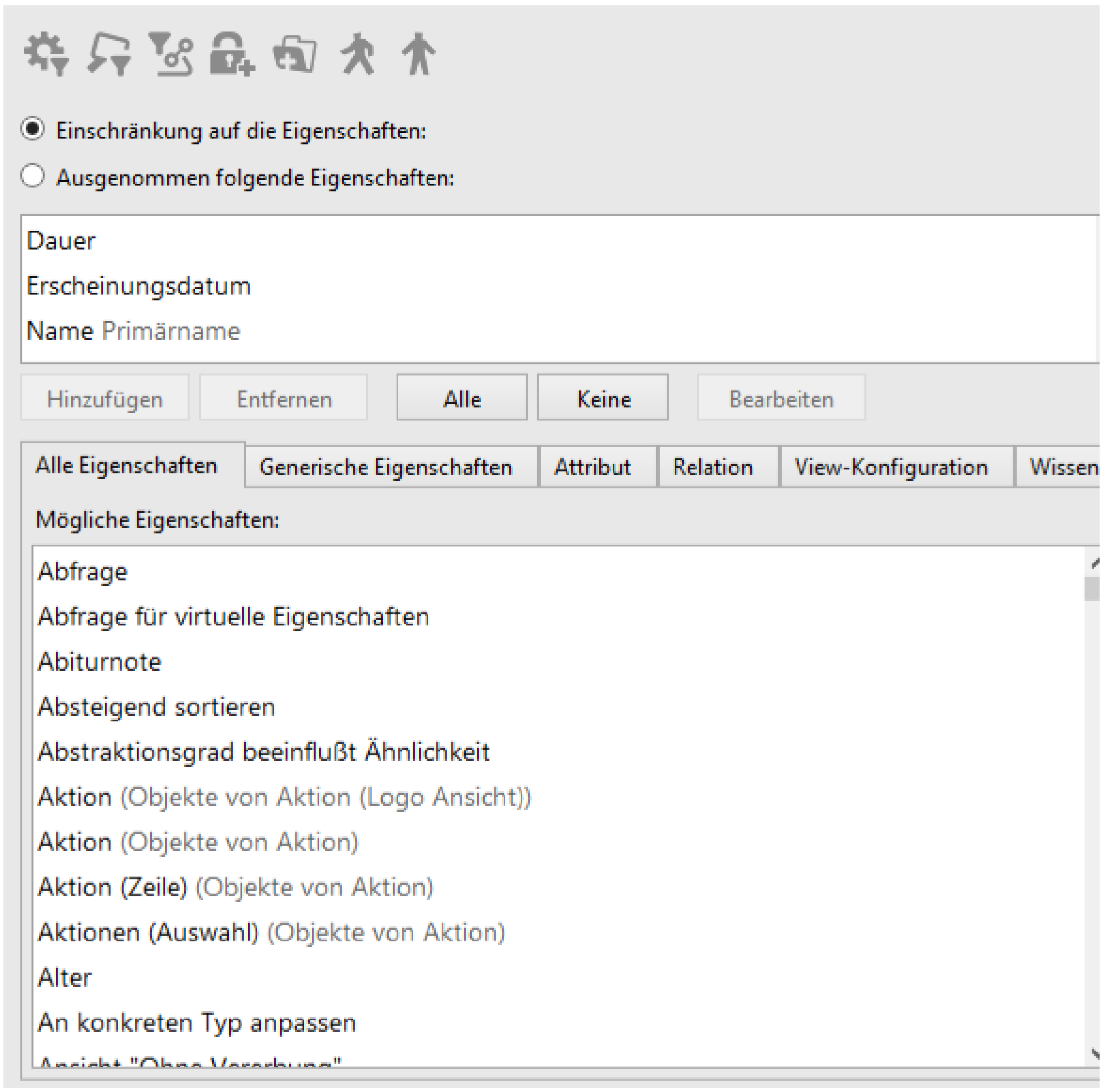
The Operations chapter lists all available operations and also specifies which operation parameters can be used in combination. The various operation parameters are explained accordingly in the Operation parameters chapter.

1.6.3.2 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.



[● Einschränkung auf die Eigenschaften:]
○ Ausgenommen folgende Eigenschaften:

Dauer
Erscheinungsdatum
Name Primärname

Mögliche Eigenschaften:

Abfrage
Abfrage für virtuelle Eigenschaften
Abiturnote
Absteigend sortieren
Abstraktionsgrad beeinflußt Ähnlichkeit
Aktion (Objekte von Aktion (Logo Ansicht))
Aktion (Objekte von Aktion)
Aktion (Zeile) (Objekte von Aktion)
Aktionen (Auswahl) (Objekte von Aktion)
Alter
An konkreten Typ anpassen

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 *Semantic network* are designed to help users find the filtering properties more quickly. The *Semantic network* tab shows all relations and attributes that the user has created.

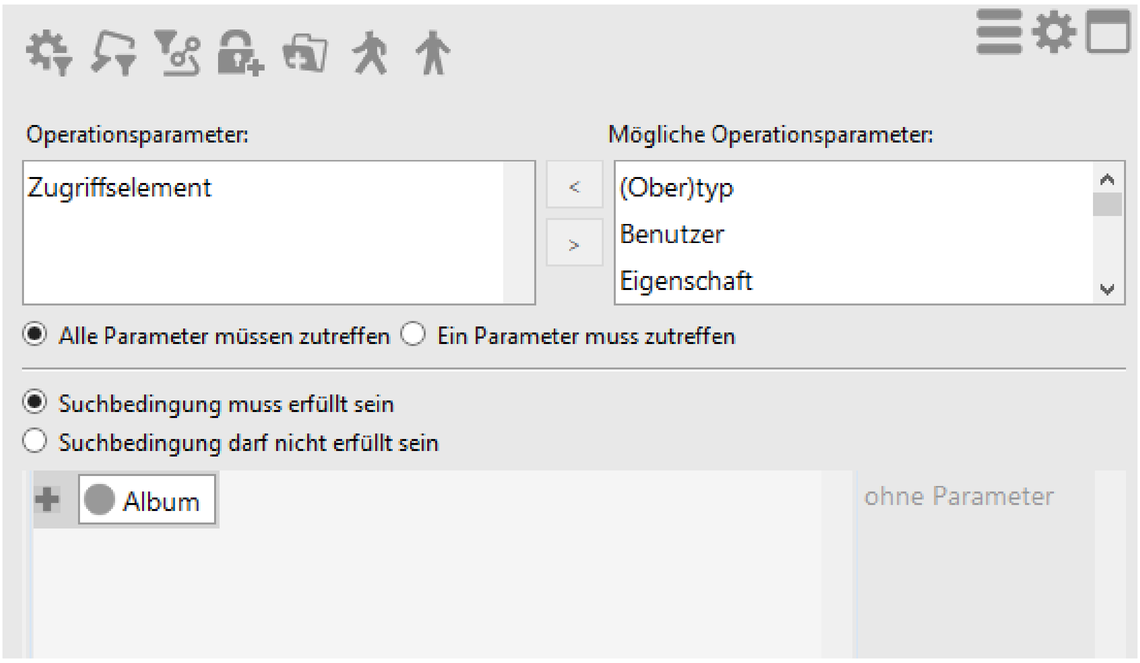
1.6.3.3 Query filter

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.



Operationsparameter: Zugriffselement

Mögliche Operationsparameter: (Ober)typ, Benutzer, Eigenschaft

(●) Alle Parameter müssen zutreffen ( ) Ein Parameter muss zutreffen

(●) Suchbedingung muss erfüllt sein
( ) Suchbedingung darf nicht erfüllt sein

Album

ohne Parameter

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.

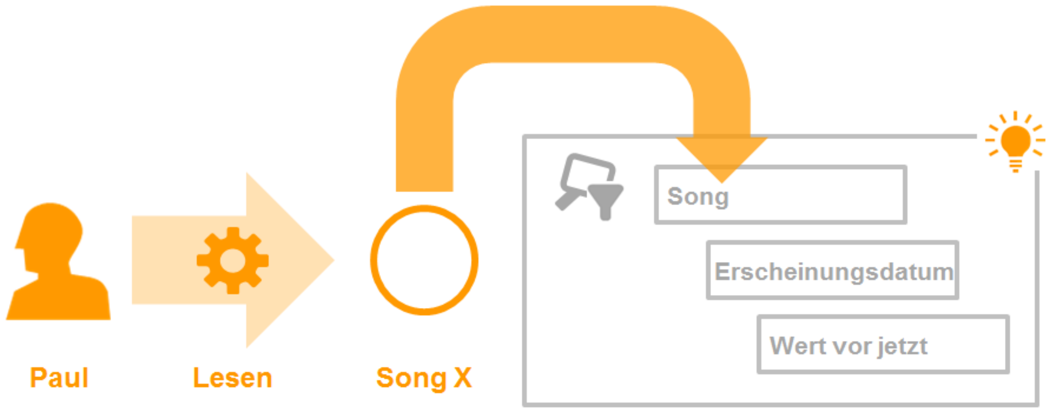
During selection of the operation parameter it is possible to configure whether

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

Please 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 published songs may be viewed by everyone; unpublished songs, in contrast, may not.

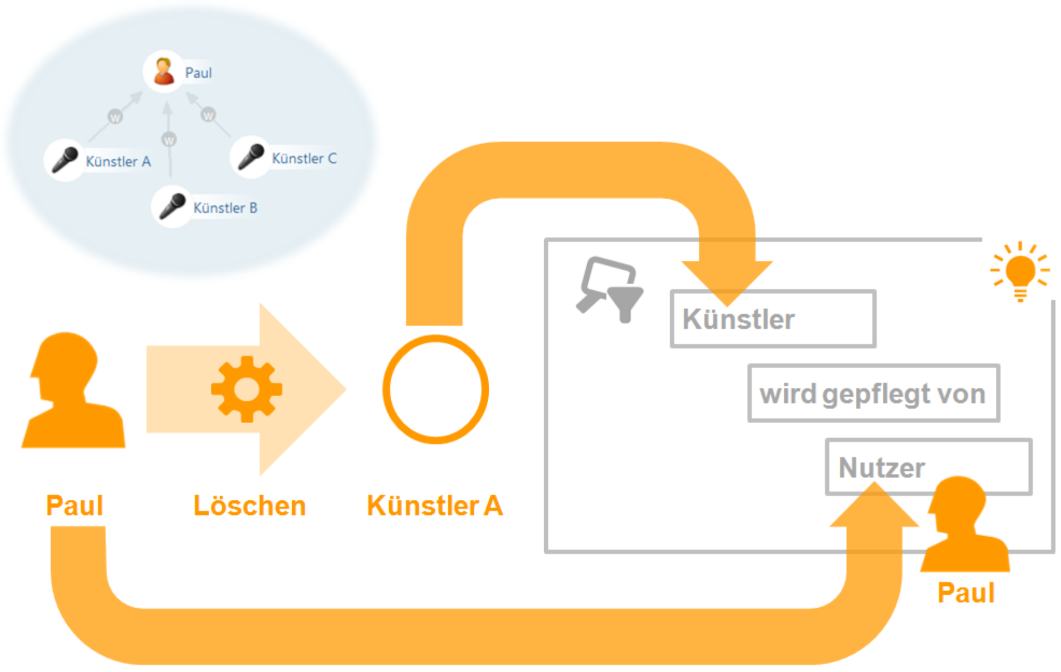


In this example, the user Paul would like to read song X. This operation is now checked by the rights system. A query filter has been defined in the rights system which checks whether the song has already been published. The structured query of the query filter searches of objects of the "Song" type, with the restriction that the attribute "Publication date" is in the past. The structured query delivers all songs that meet this condition. If song X 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.

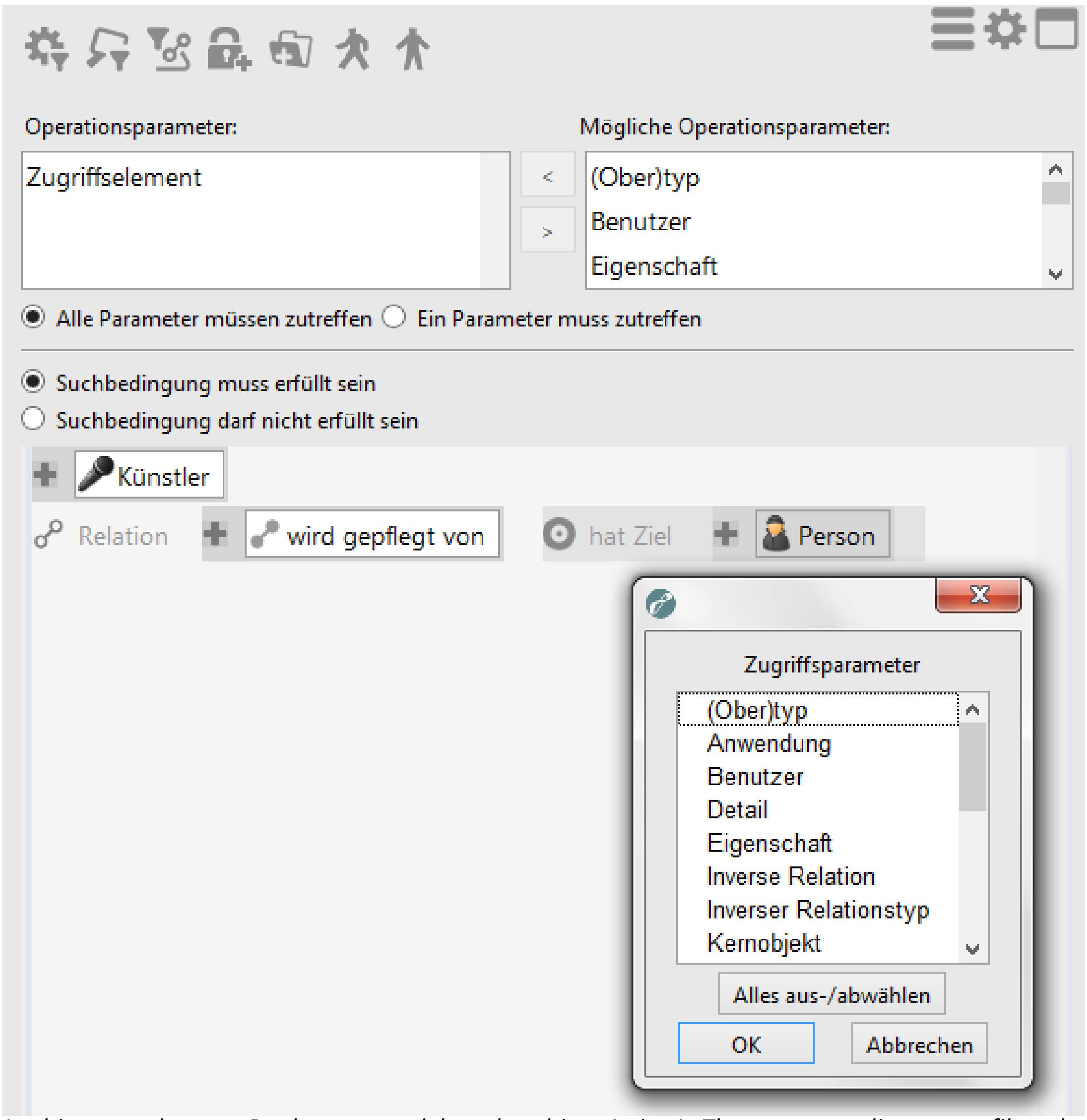
In the case of the query filter, the search condition settings must be met, and "All parameters must apply" must be selected.

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 artist may edit and delete this artist."



The left side shows a section of the semantic network: The object Paul is linked to the objects Artist A, Artist B and Artist C via the relation Maintains. The inverse relation of "maintains" is "maintained by," which exists between the objects Artist A, Artist B and Artist C and the object Paul, and is queried in the query filter. This relation in the semantic network represents that one person is responsible for data maintenance relating to an artist.


Operationsparameter: Zugriffselement

Mögliche Operationsparameter: (Ober)typ, Benutzer, Eigenschaft

[✓ Alle Parameter müssen zutreffen] Ein Parameter muss zutreffen

[✓ Suchbedingung muss erfüllt sein] Suchbedingung darf nicht erfüllt sein

Künstler — Relation: wird gepflegt von — hat Ziel: Person

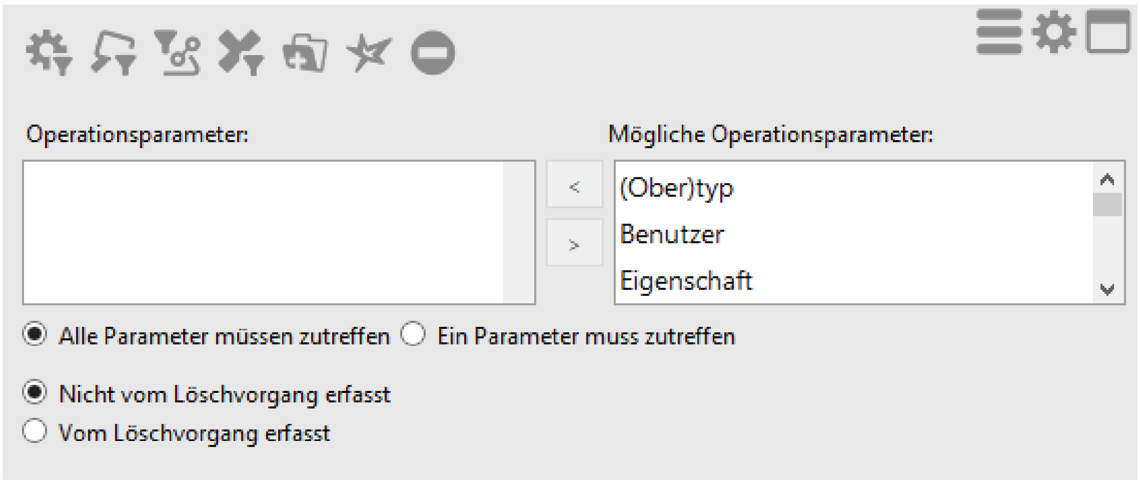
Zugriffsparameter: (Ober)typ, Anwendung, Benutzer, Detail, Eigenschaft, Inverse Relation, Inverser Relationstyp, Kernobjekt

In this example, user Paul wants to delete the object Artist A. The corresponding query filter delivers all artists 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 artists that were maintained by Paul. Since Artist 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 artist to be deleted and the user. The query filter can thus be defined in two different ways. The artist 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 company is used as the access parameter "Accessed element" in the structured query.

1.6.3.4 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.


Operationsparameter:

Mögliche Operationsparameter:

(Ober)typ
Benutzer
Eigenschaft

[✓ Alle Parameter müssen zutreffen] Ein Parameter muss zutreffen

[✓ Nicht vom Löschvorgang erfasst]
Vom Löschvorgang erfasst

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

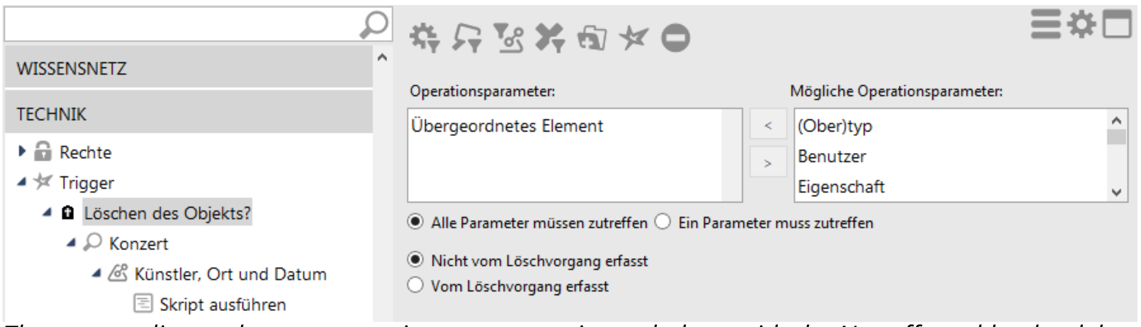
- *All parameters must apply*: All specified operation parameters must apply. For example, if two operation parameters are specified (accessed element and primary element), then it is checked whether the delete operation applies to both the accessed element and the primary element. This can only be the 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 artist, location or date of an event is modified or deleted, but not if the object containing the properties is deleted. The setting *Not affected by the delete operation* is used for this purpose. If the delete operation affects the superordinate accessed element, which in this case is the concert object itself, then the checking of the subtree is aborted because the filter has returned a negative result.


WISSENSNETZ
TECHNIK
Rechte
Trigger
Löschen des Objekts?
Konzert
Künstler, Ort und Datum
Skript ausführen

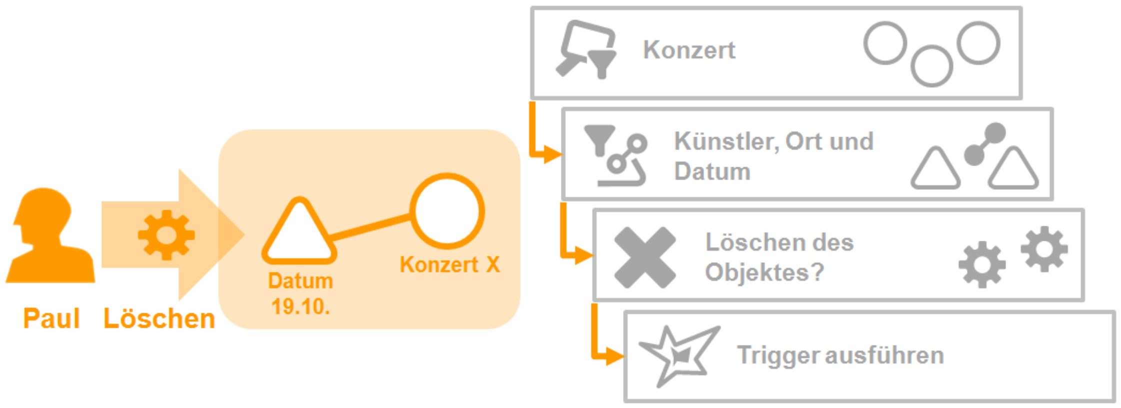
Operationsparameter: Übergeordnetes Element

Mögliche Operationsparameter: (Ober)typ, Benutzer, Eigenschaft

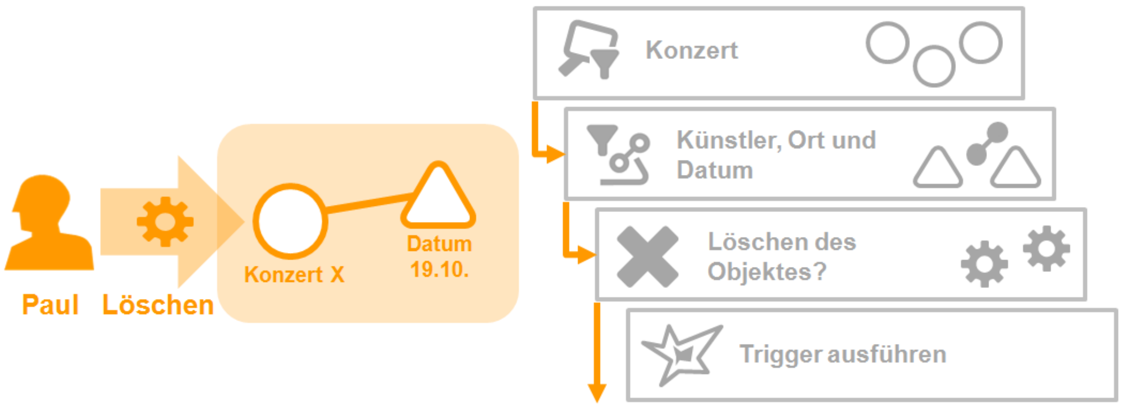
[✓ Alle Parameter müssen zutreffen] Ein Parameter muss zutreffen

[✓ Nicht vom Löschvorgang erfasst]
Vom Löschvorgang erfasst

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



In this example access situation, the Date attribute with the value "19.10." in the "Concert X" object is deleted. The object itself is not deleted. The "Concert" query filter, which is defined by the "Superordinate accessed element" operation parameter, and the "Artist, location and date" 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 affected by the delete operation" setting of the delete filter.



In this access situation the "Concert X" object is deleted by user Paul. 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 "Concert" query filter and the "Artist, location and date" 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 “Concert X” object containing the “Date 19.10.” 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).

1.6.4 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.
User	The *user* is the object of the users which executes the operation.
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.
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.

Core object	If the higher-level element is an extension, then the *core object* is the object on which the extension is stored. Otherwise, the *core object* is identical to the accessed element.
Folder	The *Folder* operation parameter is the folder affected by the operation.
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 semantic core object	If the primary semantic element is an extension, then the *primary semantic core object* is the core object of the extension. Otherwise, the *primary semantic core object* is identical to the core object.
Primary relation target	The *primary relation target* is the primary semantic element of the relation target.
Primary semantic element	If the superordinate accessed element is a property, the *primary semantic element* is the object, the type or the extension on which the property is stored (transitive). Otherwise, the *primary semantic element* is identical to superordinate element.
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.)
Superordinate element	The *semantic element* is the object, the type or the extension affected by the operation. In the case of properties, the *semantic element* is the object, the type or the extension on which the property is saved.
Accessed element	The *accessed element* is the element affected by the operation.

1.6.4.1 Operation parameter accessed element

The accessed element is the element of the semantic network 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.

1.6.4.2 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 semantic network 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

1.6.4.3 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

1.6.4.4 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

1.6.4.5 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

1.6.4.6 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

1.6.4.7 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
Object, type, extension or attribute	Blank

1.6.4.8 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

1.6.4.9 Operation parameter primary element

The primary semantic element always delivers an object, type or extension. If the primary 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	Primary 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)

1.6.4.10 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

1.6.4.11 Operation parameter core semantic element

The core object is used when work is done with extensions. Instead of the extension, the core object 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

1.6.4.12 Operation parameter primary semantic element

If you want the corresponding object or type to be processes for an element, you must use the primary semantic core object. In contrast to the primary semantic element, no extensions are permitted. In case of extensions, the core object is output.

Accessed element	Primary semantic core object
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)

1.6.4.13 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

1.6.4.14 Operation parameter folder

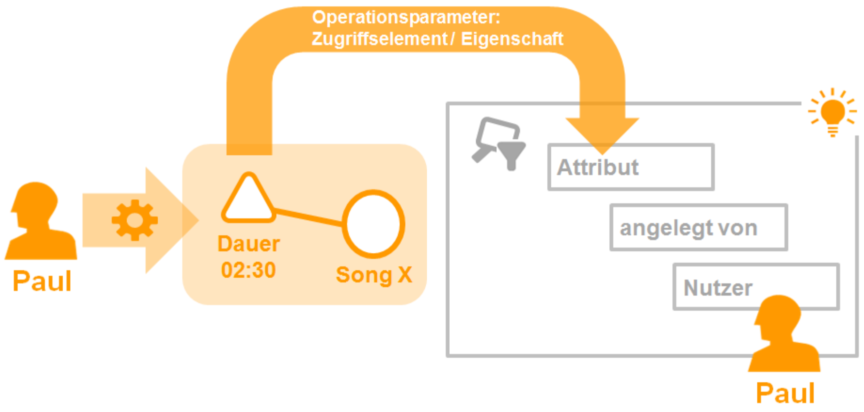
If a folder from the *Folder* area of the semantic network is to be transferred to the search as a parameter, the Folder operation parameter must be used.

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

1.6.4.15 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: User Paul wants to change the attribute Duration of song X.

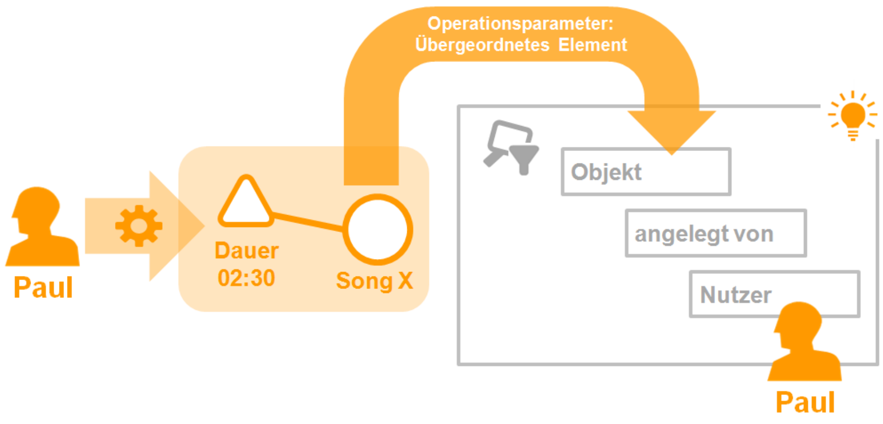
Query filter: All attributes created by a certain user are filtered. In the structured query, the access parameter “User” is used, which restricts the objects of user to the person who wants to execute the operation. This corresponds to all attributes that were created by Paul.

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

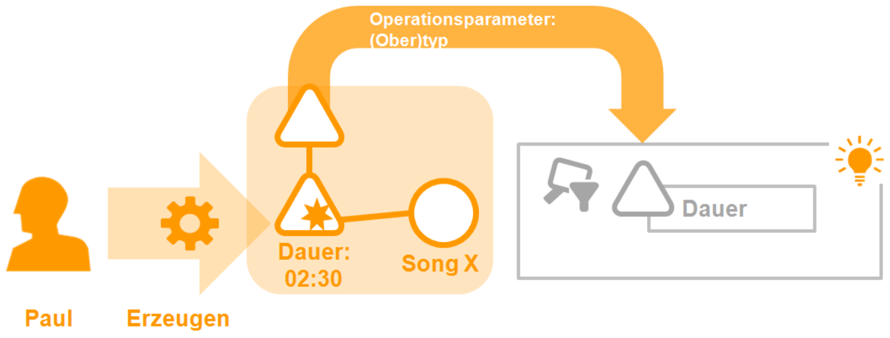
Query filter: The query filter is defined in such a way that it searches for all objects that were created by a specific user; the currently logged-in user is the accessed element. Accordingly, the query filter finds all the objects created by Paul.

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.

1.6.5 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 Query (operation: use in structured query), 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*.

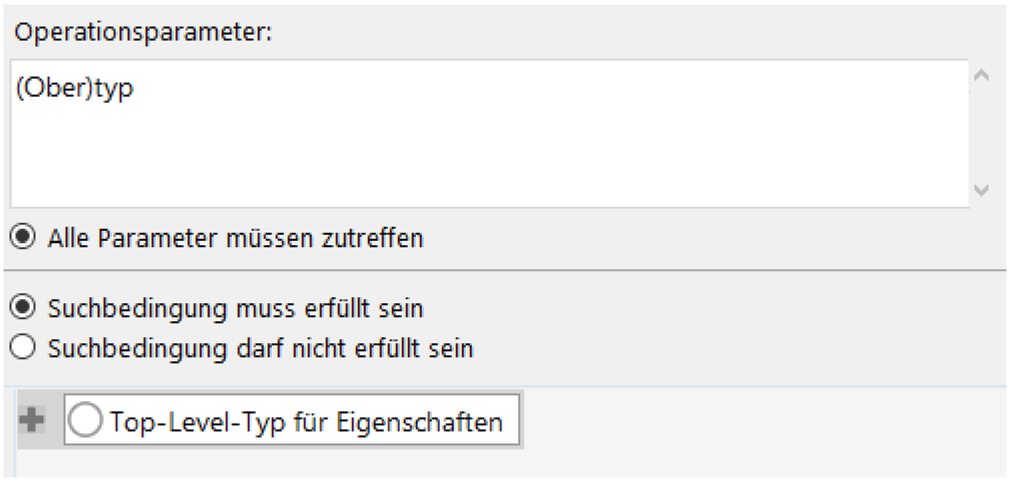
Operation group	Operation	Operation parameter
Query	Use in structured query	Accessed element
Display of objects	Display in graph editor	Accessed element
Edit	Validate attribute value	Accessed element, property, superordinate element, (parameter to be checked: attribute value)
User-defined operation		
Generate	Generate attribute	(Super) type, superordinate element
	Generate extension	(Super) type, superordinate element, core object
	Generate object	(Super) type
	Generate folder	Folder
	Generate relation	(Super) type, superordinate element, relation target, inverse relation type
	Generate relation half	(Super) type, superordinate element, relation target
	Generate 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
	Modify schema	Accessed element, superordinate element
	Change type	Accessed element, superordinate element
Use tools	Export	
	Import	
	Edit/execute script	

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 employ-

ees 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 semantic networks, permission can be issued for all property types ("top-level type for properties").


Operationsparameter:

(Ober)typ

[✓ Alle Parameter müssen zutreffen]

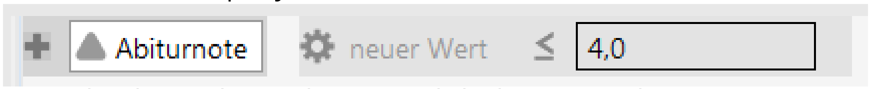
[✓ Suchbedingung muss erfüllt sein]
Suchbedingung darf nicht erfüllt sein

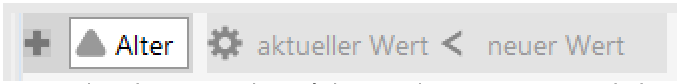
Top-Level-Typ für Eigenschaften

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".

Use in structured query (obsolete)
If a negative access right has been defined for an element that is filtered for the operation *Use in structured query*, then the element may not be used in a structured query. It will not be factored into structured queries even when the (abstract) super-type is specified.

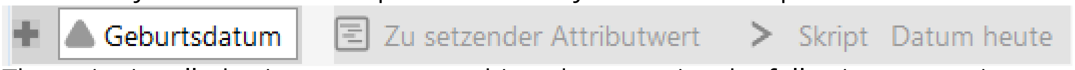
Validate attribute value
The operation *Validate attribute value* is used when the attribute value to be set must satisfy certain conditions. The definition of the condition for the attribute value is made in a structured query. Two possible definitions are available there for validation of the attribute 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.
  
  Abiturnote neuer Wert ≤ 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.
  
  Alter aktueller Wert < neuer Wert
  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.



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

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.

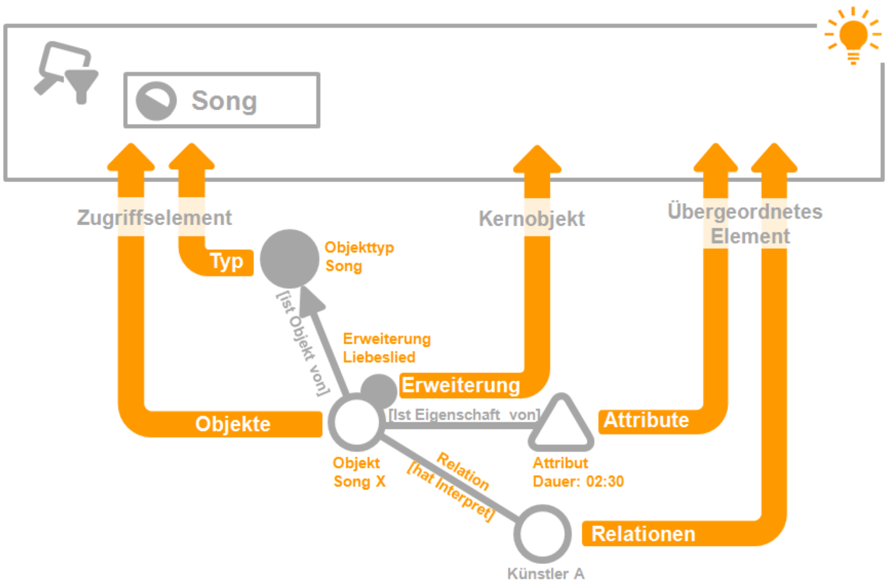
Modify schema
The modify schema operation concerns changes to the definition area of relations and changes to the type hierarchy (*is a subtype of* and *is a super-type of* relations).

1.6.5.1 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 Song type and its objects. This includes the following actions:

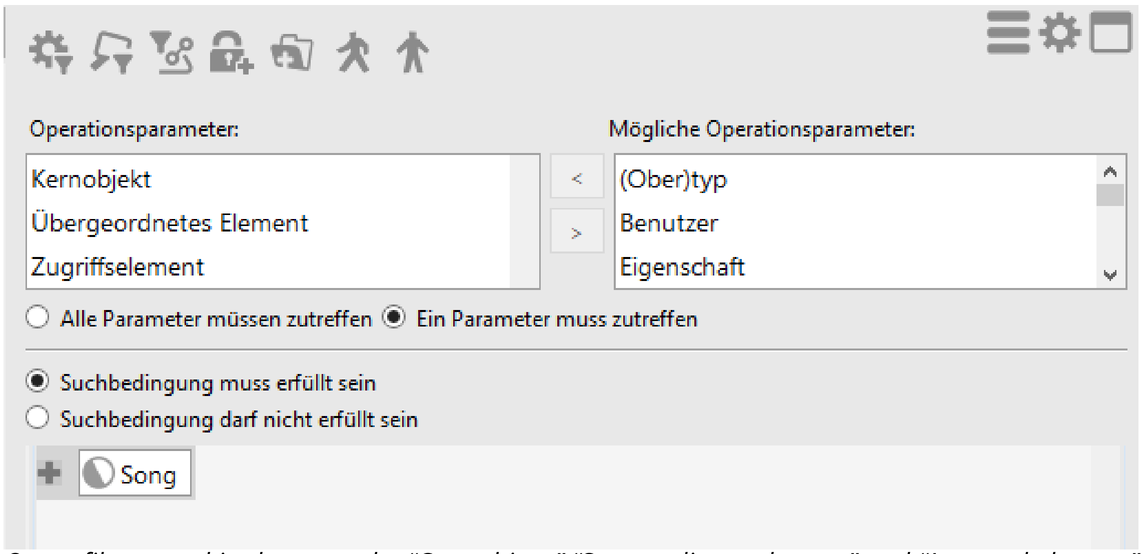
- Deletion of the object type Song
- Deletion of specific songs (objects of Songs)
- Deletion of attributes that occur on a Song
- Deletion of relations that occur on a Song (relation target and source)
- Deletion of extensions that extend objects of Song
- Deletion of attribute and relation types that have objects or subtypes of Song 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 Song, for which the setting Objects and Subtypes is selected. The operation parameter "Accessed element" covers the object type "Song" *and all objects that belong to this type. The parameter Core object covers the extension objects that belong to songs. Attributes and relations are covered by the operation parameter "Superordinate 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."



Operationsparameter:

Kernobjekt
Übergeordnetes Element
Zugriffselement

Mögliche Operationsparameter:

(Ober)typ
Benutzer
Eigenschaft

Alle Parameter müssen zutreffen [✓ Ein Parameter muss zutreffen]

[✓ Suchbedingung muss erfüllt sein]
Suchbedingung darf nicht erfüllt sein

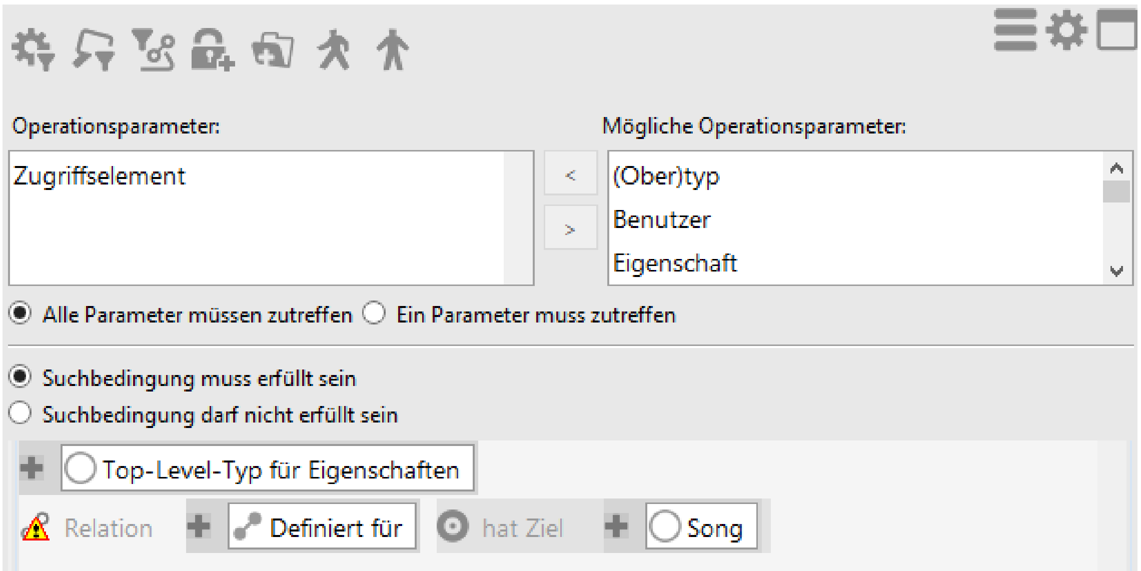
Song

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:



Operationsparameter:

Zugriffselement

Mögliche Operationsparameter:

(Ober)typ
Benutzer
Eigenschaft

[✓ Alle Parameter müssen zutreffen] Ein Parameter muss zutreffen

[✓ Suchbedingung muss erfüllt sein]
Suchbedingung darf nicht erfüllt sein

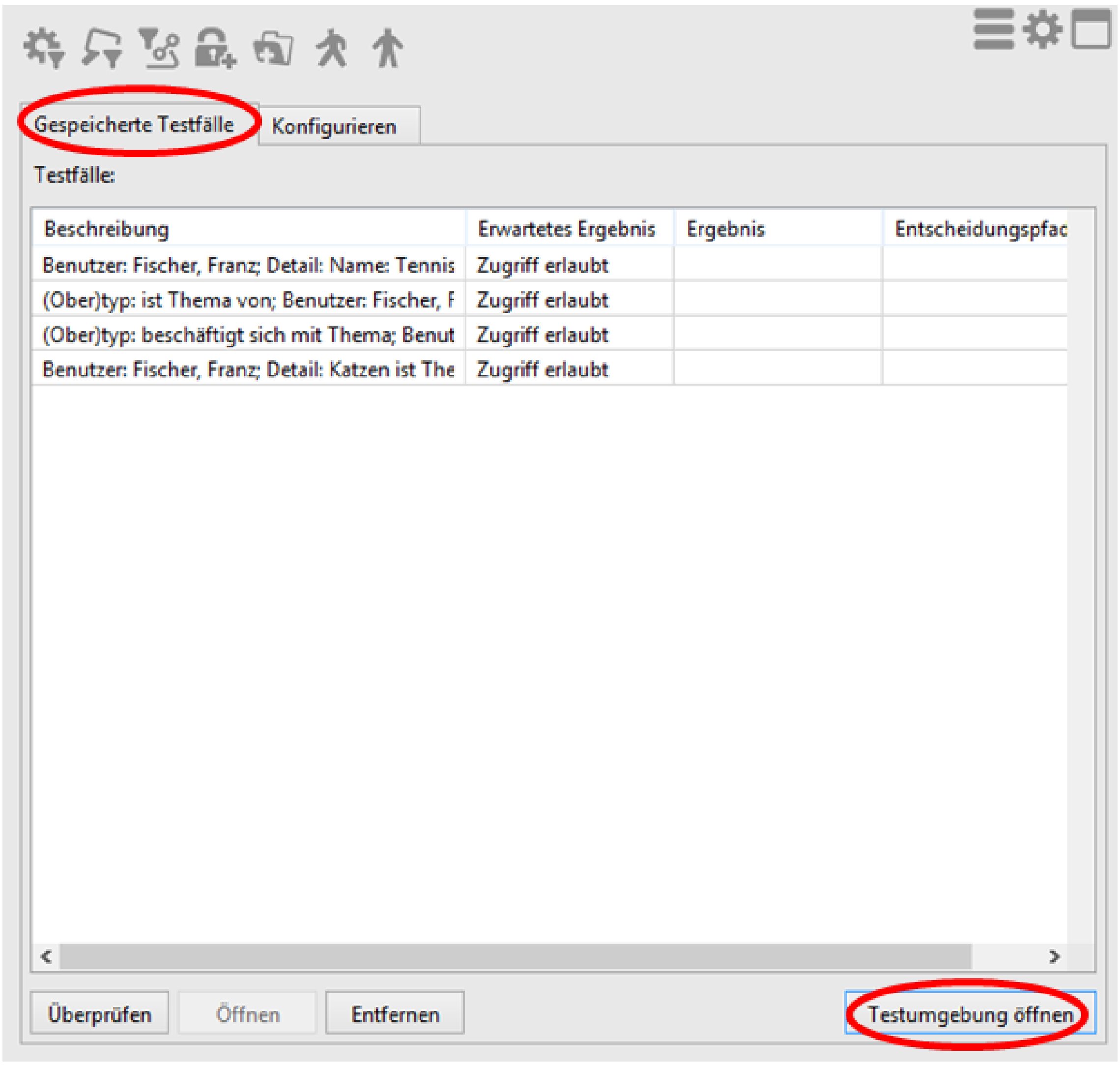
Top-Level-Typ für Eigenschaften
Relation — Definiert für — hat Ziel — Song

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

1.6.6 Test surrounding

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 test environment* button.



Beschreibung	Erwartetes Ergebnis	Ergebnis	Entscheidungspfac
Benutzer: Fischer, Franz; Detail: Name: Tennis	Zugriff erlaubt		
(Ober)typ: ist Thema von; Benutzer: Fischer, F	Zugriff erlaubt		
(Ober)typ: beschäftigt sich mit Thema; Benut	Zugriff erlaubt		
Benutzer: Fischer, Franz; Detail: Katzen ist The	Zugriff erlaubt		

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.

1.6.6.1 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

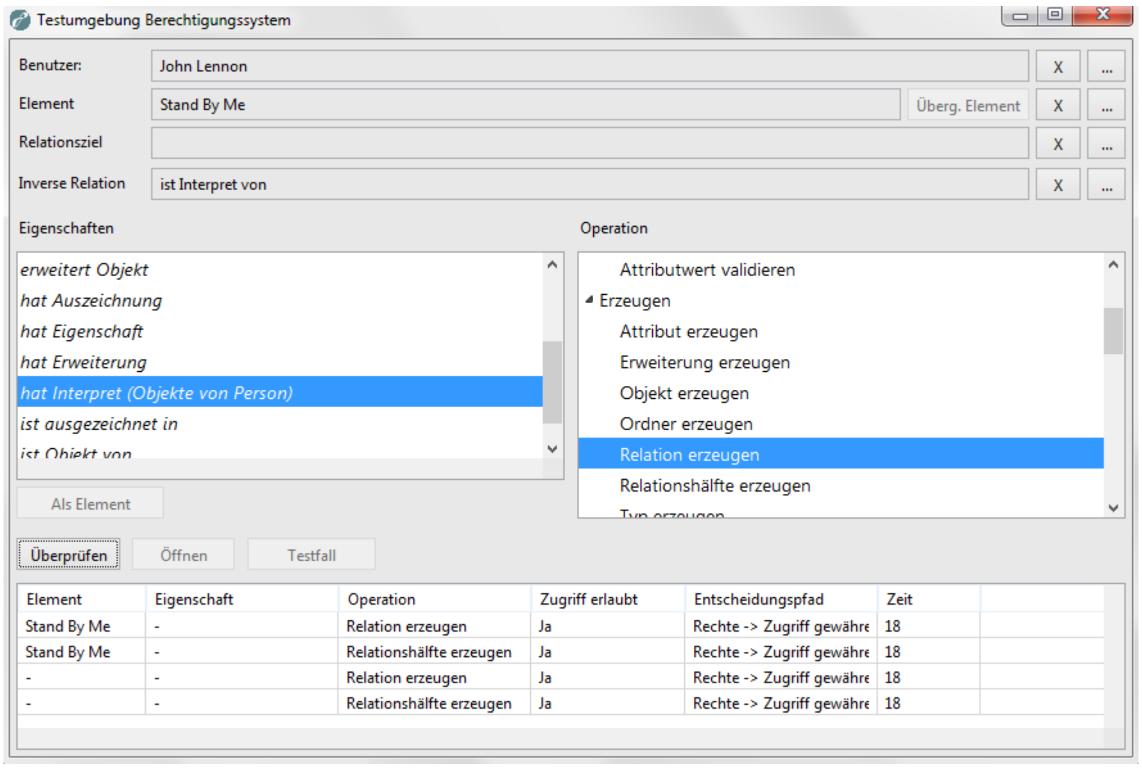
1. Select the folder *Rights* or *Triggers* in the *Technical* area in the Knowledge Builder.
2. If you are working in the rights system, select the *Saved test cases* tab in the main window.
3. Click *Open test environment* (bottom right) so that the test environment opens in a new window.

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.

Please 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.



Benutzer:	John Lennon
Element	Stand By Me
Relationsziel	
Inverse Relation	ist Interpret von

Eigenschaften

erweitert Objekt
hat Auszeichnung
hat Eigenschaft
hat Erweiterung
hat Interpret (Objekte von Person)
ist ausgezeichnet in
ist Objekt von

Operation

Attributwert validieren
Erzeugen
Attribut erzeugen
Erweiterung erzeugen
Objekt erzeugen
Ordner erzeugen
Relation erzeugen
Relationshälfte erzeugen
Typ erzeugen

Element	Eigenschaft	Operation	Zugriff erlaubt	Entscheidungspfad	Zeit
Stand By Me	-	Relation erzeugen	Ja	Rechte -> Zugriff gewähre	18
Stand By Me	-	Relationshälfte erzeugen	Ja	Rechte -> Zugriff gewähre	18
-	-	Relation erzeugen	Ja	Rechte -> Zugriff gewähre	18
-	-	Relationshälfte erzeugen	Ja	Rechte -> Zugriff gewähre	18

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

Please note: When testing relations, the relation, the inverse relation and the both relations halves are generally tested separately.

1.6.6.2 Define test cases

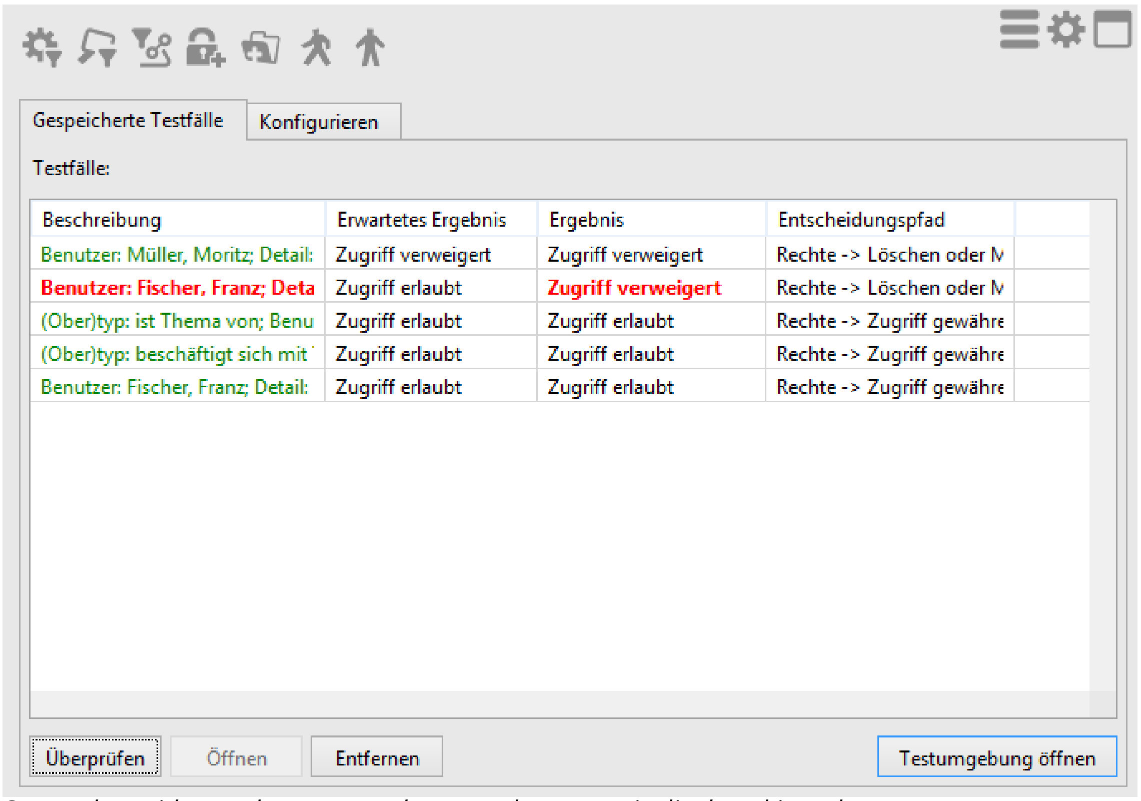
In order to monitor the functionality of the rights system, it is possible to save test cases. This is particularly important if changes are made to the rights system and you want to check afterwards whether the new result still matches the expected result. All saved test cases are displayed on the *Saved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



Beschreibung	Erwartetes Ergebnis	Ergebnis	Entscheidungspfad
Benutzer: Müller, Moritz; Detail:	Zugriff verweigert	Zugriff verweigert	Rechte -> Löschen oder M
Benutzer: Fischer, Franz; Deta	Zugriff erlaubt	Zugriff verweigert	Rechte -> Löschen oder M
(Ober)typ: ist Thema von; Benu	Zugriff erlaubt	Zugriff erlaubt	Rechte -> Zugriff gewähre
(Ober)typ: beschäftigt sich mit	Zugriff erlaubt	Zugriff erlaubt	Rechte -> Zugriff gewähre
Benutzer: Fischer, Franz; Detail:	Zugriff erlaubt	Zugriff erlaubt	Rechte -> Zugriff gewähre

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.

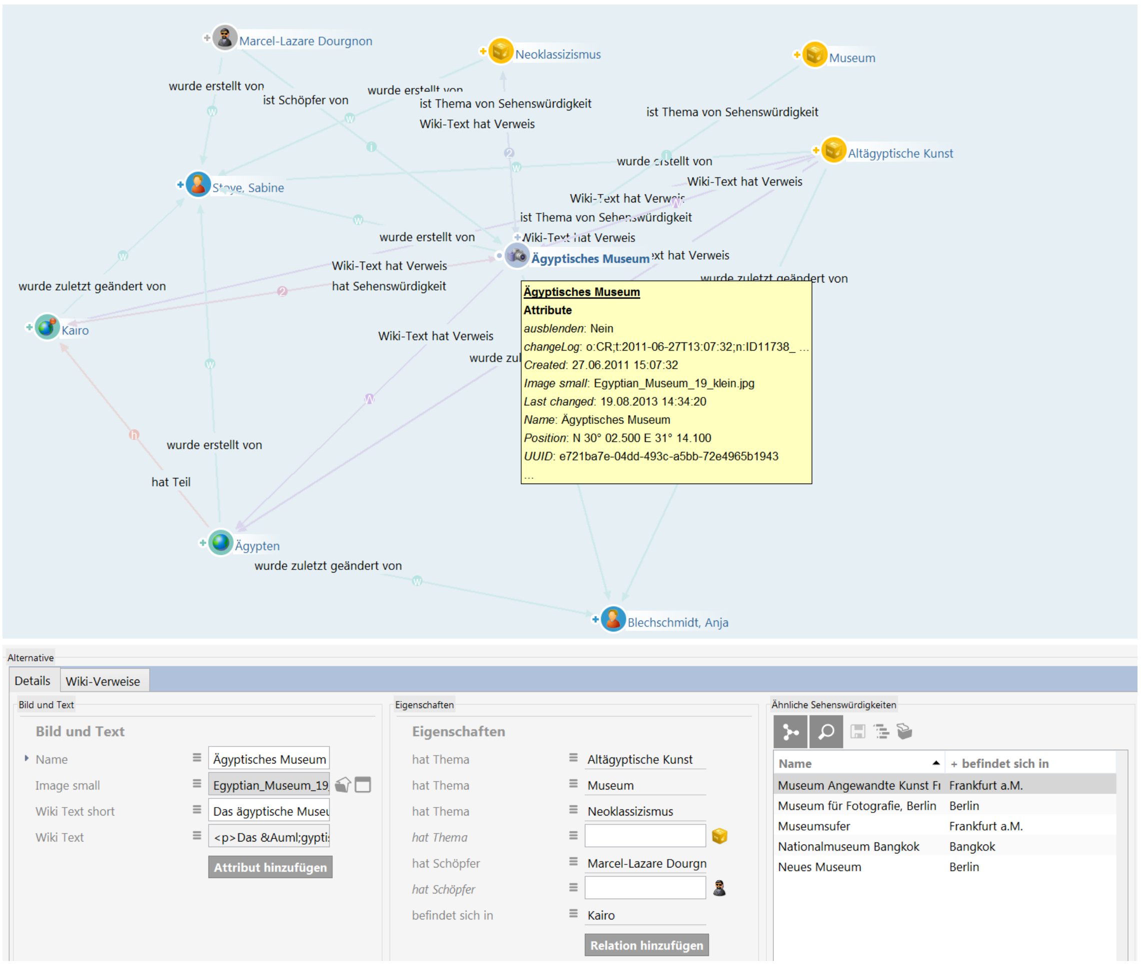
1.7 View Configuration

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 semantic model 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.



Ägyptisches Museum
Attribute
ausblenden: Nein
changeLog: o:CR;t:2011-06-27T13:07:32;n:ID11738_ ...
Created: 27.06.2011 15:07:32
Image small: Egyptian_Museum_19_klein.jpg
Last changed: 19.08.2013 14:34:20
Name: Ägyptisches Museum
Position: N 30° 02.500 E 31° 14.100
UUID: e721ba7e-04dd-493c-a5bb-72e4965b1943
...

Bild und Text

Bild und Text	
Name	Ägyptisches Museum
Image small	Egyptian_Museum_19
Wiki Text short	Das ägyptische Museu
Wiki Text	<p>Das &Auml;gypti:

Attribut hinzufügen

Eigenschaften

Eigenschaften	
hat Thema	Altägyptische Kunst
hat Thema	Museum
hat Thema	Neoklassizismus
hat Thema	
hat Schöpfer	Marcel-Lazare Dourgn
hat Schöpfer	
befindet sich in	Kairo

Relation hinzufügen

Name	+ befindet sich in
Museum Angewandte Kunst Fı	Frankfurt a.M.
Museum für Fotografie, Berlin	Berlin
Museumsufer	Frankfurt a.M.
Nationalmuseum Bangkok	Bangkok
Neues Museum	Berlin

Example of a view configuration. Upper screenshot: Unconfigured section of an object in the graph view with all its properties. Lower 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.

1.7.1 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.

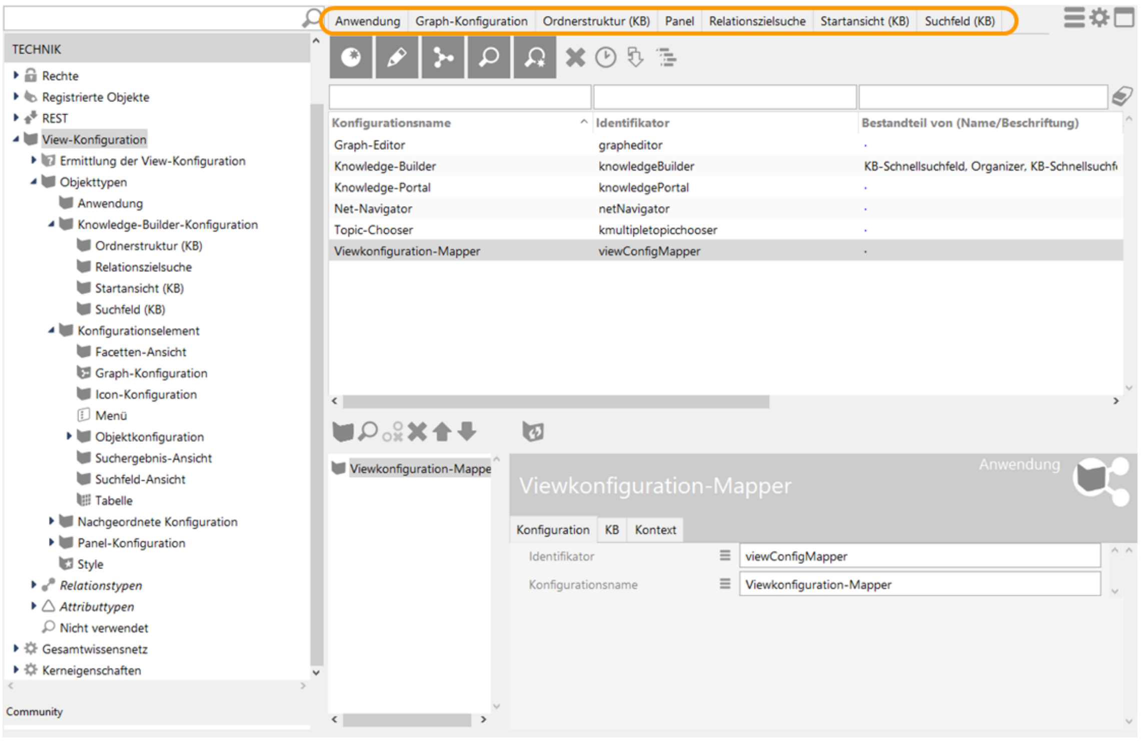
1.7.1.1 View Configuration

The purpose of the view configuration is to format the data of the semantic network 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 semantic network, 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)



Konfigurationsname	Identifikator	Bestandteil von (Name/Beschriftung)
Graph-Editor	grapheditor	-
Knowledge-Builder	knowledgeBuilder	KB-Schnellsuchfeld, Organizer, KB-Schnellsuchf
Knowledge-Portal	knowledgePortal	-
Net-Navigator	netNavigator	-
Topic-Chooser	kmultipletopicchooser	-
Viewkonfiguration-Mapper	viewConfigMapper	-

Identifikator	viewConfigMapper
Konfigurationsname	Viewkonfiguration-Mapper

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

1.7.1.2 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.

1.7.1.3 Create and update the view configuration

Create

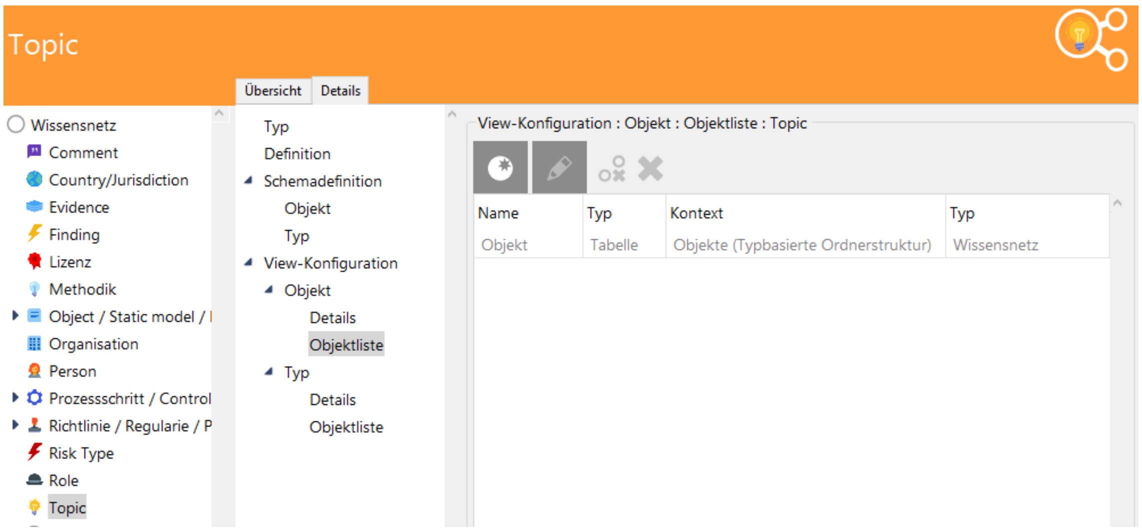
In Knowledge Builder, there are two places where you can create a new view configuration:

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	Typ	Kontext	Typ
Objekt	Tabelle	Objekte (Typbasierte Ordnerstruktur)	Wissensnetz

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

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 semantic network or for which a new view configuration can be created.

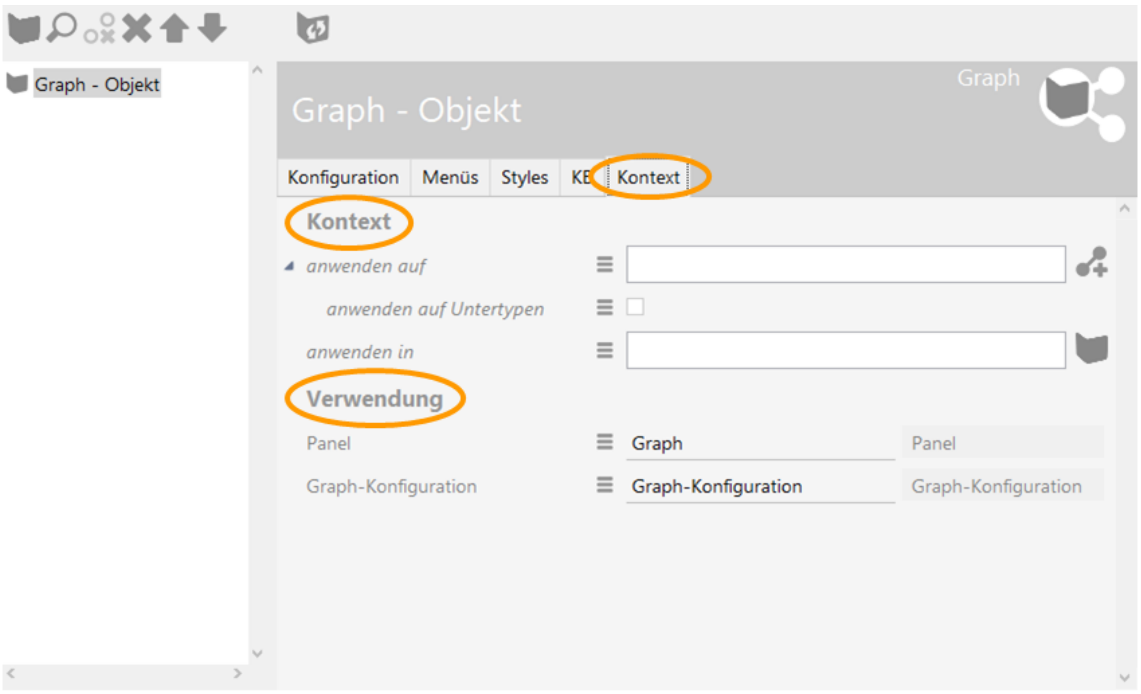
To configure a web front-end, use the panel configuration *Technical* > *View configuration* > *View configuration mapper*. For more information, see chapter 3 “View configuration mapper.”

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.

1.7.1.4 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.



Graph - Objekt

Kontext

anwenden auf		
anwenden auf Untertypen	☐	
anwenden in		

Verwendung

Panel	Graph	Panel
Graph-Konfiguration	Graph-Konfiguration	Graph-Konfiguration

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 group 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.

- **Knowledge portal:** The knowledge portal is a component of i-views which can be used as a front-end. It shows the objects of the semantic network on details pages and in context boxes on the basis of their semantic contexts.
- **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, in contrast to the knowledge portal, uses the 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.

1.7.1.5 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.

1.7.2 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 if standard actions can be added. This is currently possible for table and search configurations. This function offers the option to reactivate some or all default menu entries and to change the order of the individual actions if the parameter *Replaces standard menu* is set.

Notes

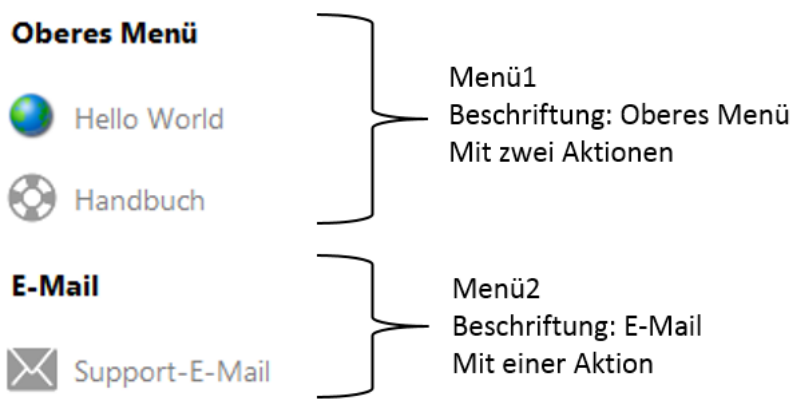
- If the parameter *Replaces standard menu* is not set, the actions that are not included in the menus are appended sequentially.
- If the order of the standard actions is supposed to be changed, the parameter *Replaces standard menu* must be set. Following that, standard actions can be added using the *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.  Oberes Menü Hello World Handbuch E-Mail Support-E-Mail Menü1 Beschriftung: Oberes Menü Mit zwei Aktionen Menü2 Beschriftung: E-Mail Mit einer Aktion
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"`

1.7.3 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)

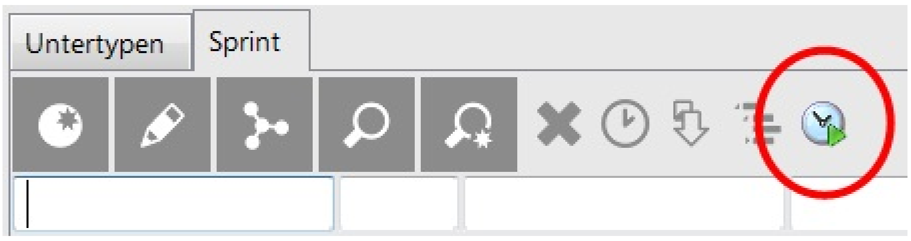
Depending on the action type and application, additional configurations are required, for example creating additional panels for displaying the results of an action.

1.7.3.1 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 is executed when selection occurs.

The actions configured are generally displayed as buttons in the web front-end (ViewConfiguration Mapper). Actions can be summarized in a menu, or be defined directly for a view configuration.



Action on an object 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.
Please note: If no symbol is specified, no button is displayed in the Knowledge Builder.

Buttons with a label and/or a symbol image are possible in another application. A tooltip can also be configured.

Important note: 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.

Setting options

Name	Value
Action type	The type of action. The different types are explained further down. A script overwrites the action defined by the action type.
execute in the view (VCM-specific)	View in which the action is to be executed.

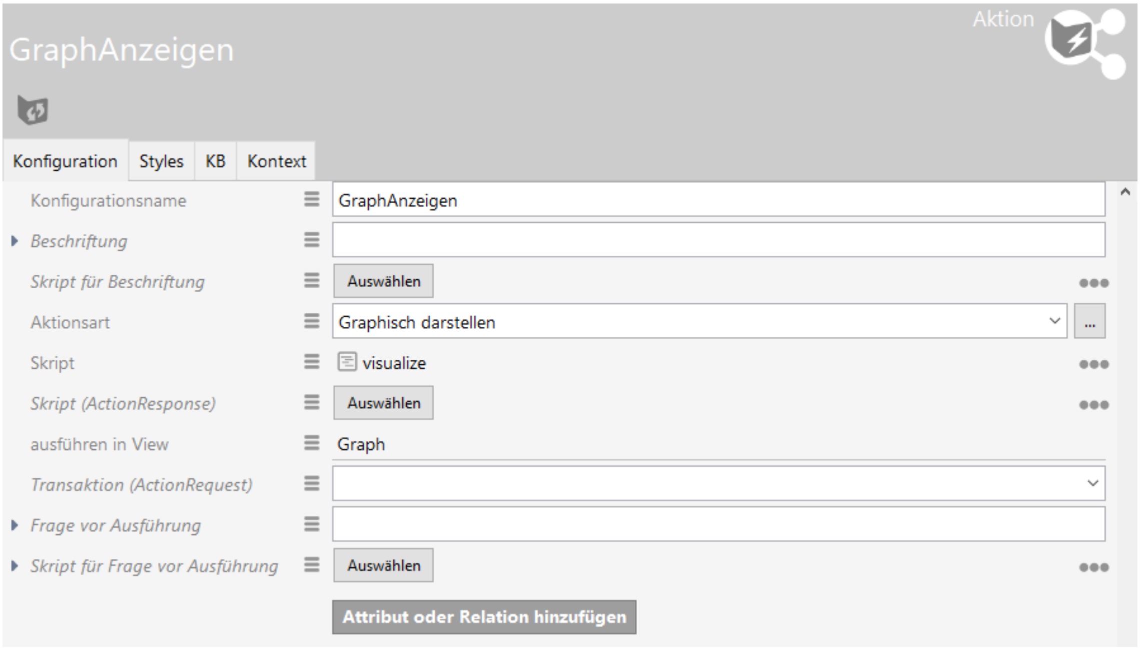
Notification	Text shown in a notification that appears after the action.
Label	A label can be defined for the button for the action here.
Show result in panel (VCM-specific)	A panel in which the result of the action is to be displayed.
Script for target object (VCM-specific)	.
Prompt before execution	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.
Context of	.
Close panel (VCM-specific)	Defines whether the panel is to be closed after the action.
Script	The script that is to be executed for this action. Not available for all action types.
Script (ActionResponse) (VCM-specific)	A script specified here executes a so-called *ActionResponse* after the action. Not available for all action types.
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.
Script for notification	A script can be used here to determine the content of the notification.
Script for label	A script can be used here to define the label.
Script for prompt before execution	A script can be used here to determine the text for the confirmation dialog for the action. If a blank string is returned, the dialog does not appear.
Script for tooltip	A script can be used here to determine the content of the tooltip for the action, instead of using the text of the label.
Start semantic element of	.
Symbol	A symbol can be selected here that is to be displayed on the button for the action.
Tooltip	The content of the tooltip for the action can be defined here, instead of using the text of the label.
Use original position	.

1.7.3.2 Universally applicable actions

Universally applicable actions can be used in both the Knowledge Builder and in the web front-end using the ViewConfiguration Mapper. This includes the action types "Display graphically", "Delete" and "Search".

1.7.3.2.1 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:

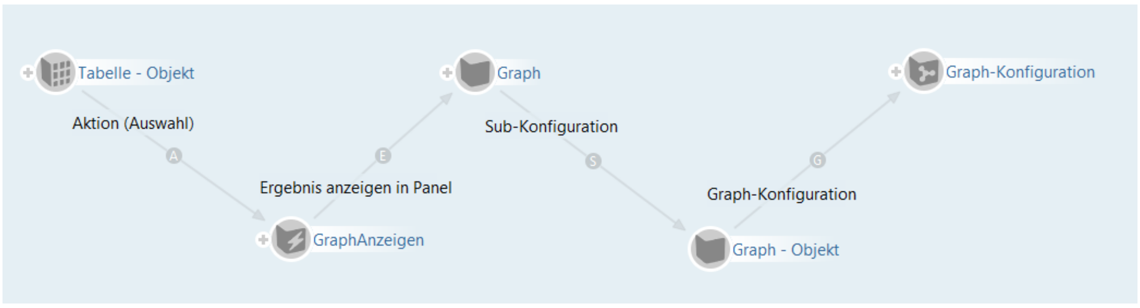


GraphAnzeigen

Aktion

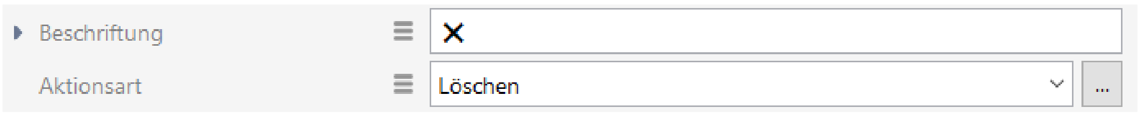
Konfigurationsname	GraphAnzeigen
Beschriftung	
Skript für Beschriftung	Auswählen
Aktionsart	Graphisch darstellen
Skript	visualize
Skript (ActionResponse)	Auswählen
ausführen in View	Graph
Transaktion (ActionRequest)	
Frage vor Ausführung	
Skript für Frage vor Ausführung	Auswählen

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:



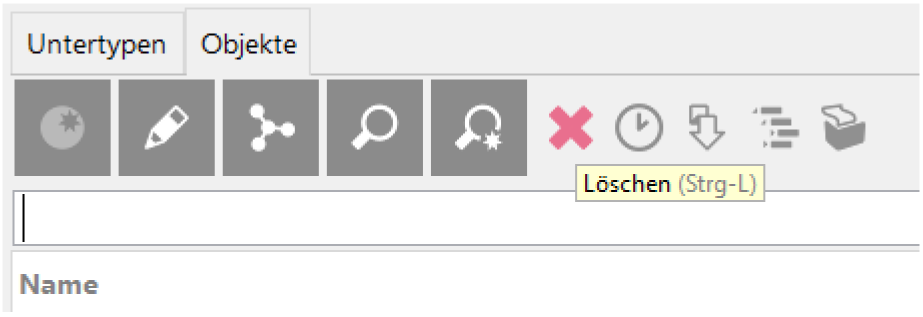
1.7.3.2.2 Action type "Delete"

This action type deletes the respective element.



Beschriftung	✕
Aktionsart	Löschen

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

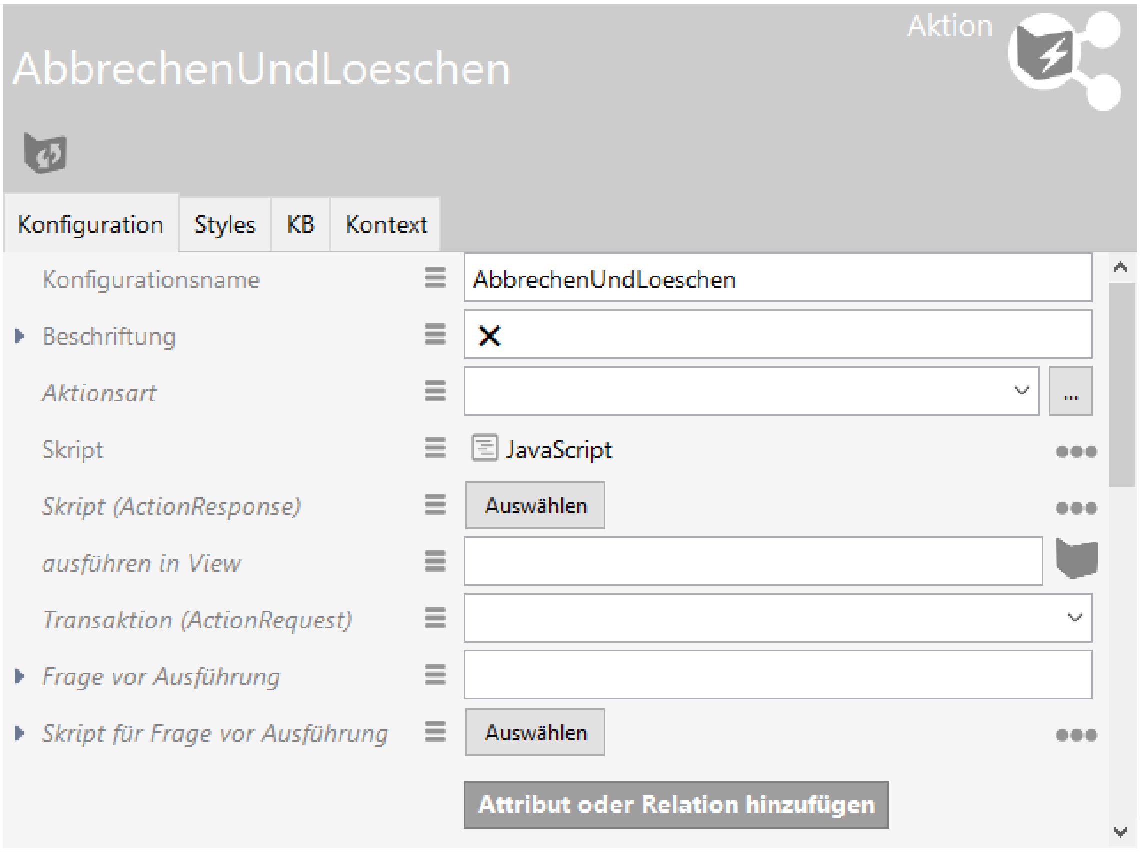


Like any other configuration in Knowledge Builder, the default configuration can be replaced with a customized configuration. In this case, use “Delete” as the action type.

Deleting by action with script

In the web front-end, action type “Delete” is not practical as the deleted element is no longer displayed afterwards, i.e. there is nothing left to see. Hence, we almost always implement an action with a script in the web front-end.

The script for deleting an element is created under the “Script” entry of the action:



Konfigurationsname	AbbrechenUndLoeschen
Beschriftung	✕
Aktionsart	
Skript	JavaScript
Skript (ActionResponse)	Auswählen
ausführen in View	
Transaktion (ActionRequest)	
Frage vor Ausführung	
Skript für Frage vor Ausführung	Auswählen

The syntax for this could look as follows:

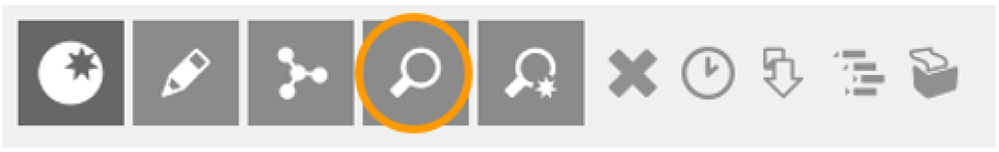
```
function onAction(element, context) {
    element.remove();
    return;
}
```

One possible use for this is the configuration of a dialog panel for creating new objects whose Cancel button then deletes the temporarily generated object.

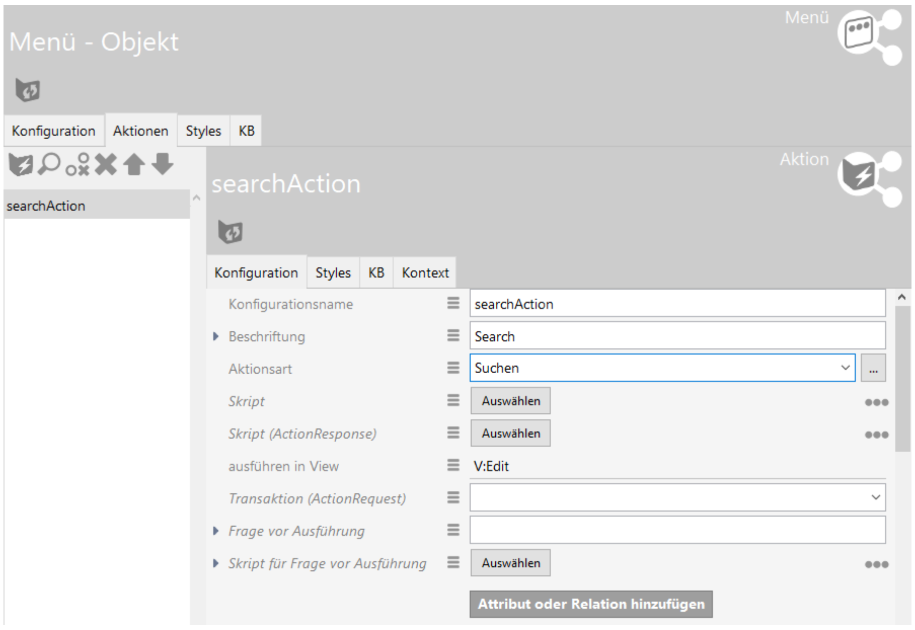
Keep in mind that no action type for the action has to be deleted with this script as the script overwrites the action type.

1.7.3.2.3 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:”



Konfigurationsname	searchAction
Beschriftung	Search
Aktionsart	Suchen
Skript	Auswählen
Skript (ActionResponse)	Auswählen
ausführen in View	V:Edit
Transaktion (ActionRequest)	
Frage vor Ausführung	
Skript für Frage vor Ausführung	Auswählen

Tip: If a search function with string input (keyword search) is required, then the search field element in the view configuration can be used as an alternative. An input line and search button are preconfigured here; the search result can be displayed in combination with the search result view.

1.7.3.3 Actions for the knowledge builder

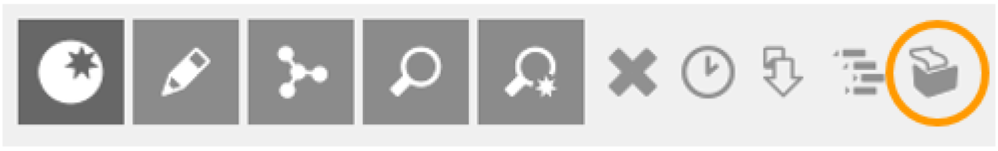
These action types can only be used for configurations in the Knowledge Builder.
Note: The KB-specific action types are only available in the “KB” tab of an action from KB version 5.2.2 or higher.

1.7.3.3.1 Action type "Update"

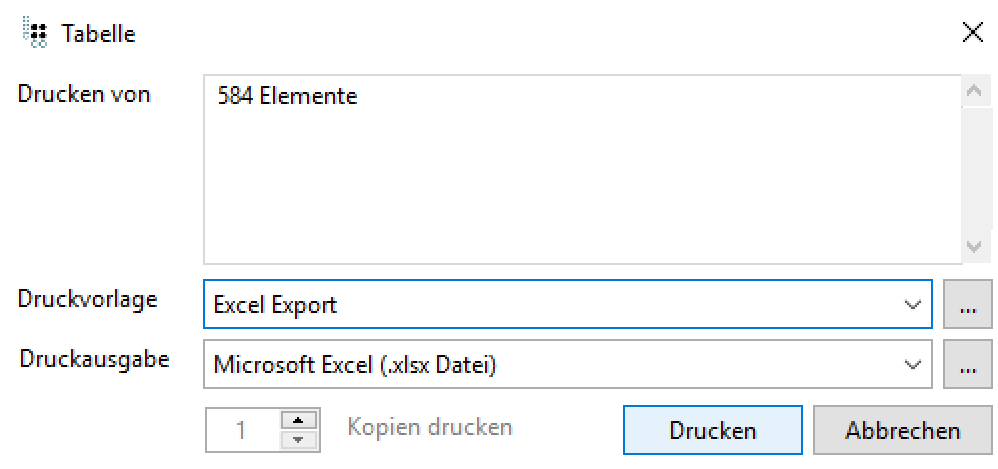
In KB, this action recalculates the visible content of table cells. This option is available via the “Update” button in the object list menu bar (shortcut: F5).

1.7.3.3.2 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.

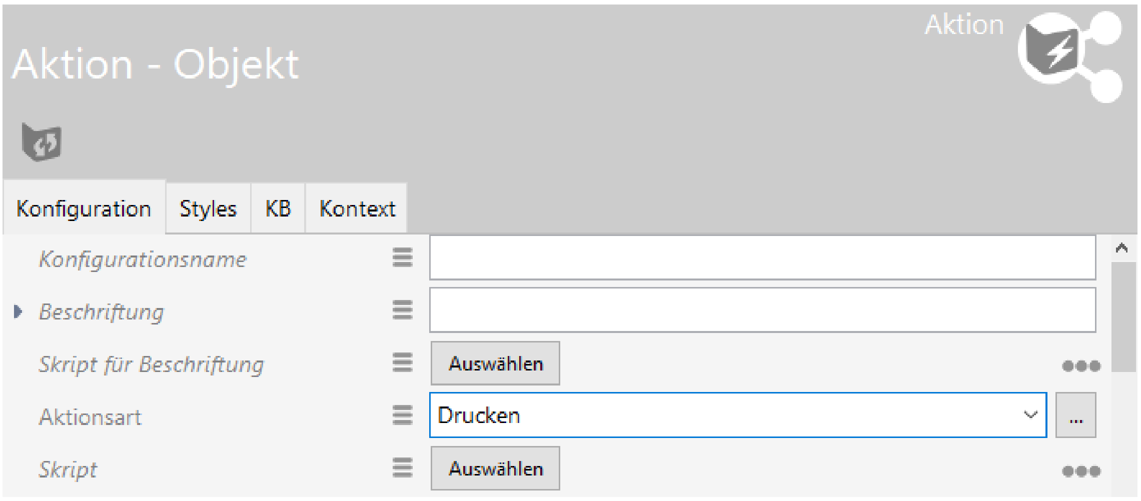


Tabelle

Drucken von	584 Elemente
Druckvorlage	Excel Export
Druckausgabe	Microsoft Excel (.xlsx Datei)

1 Kopien drucken

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



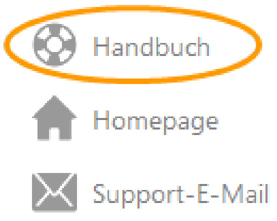
Aktion - Objekt

Konfigurationsname	
Beschriftung	
Skript für Beschriftung	Auswählen
Aktionsart	Drucken
Skript	Auswählen

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.

1.7.3.3.3 Action type "User guide"

This action opens the i-views web manual in the browser.



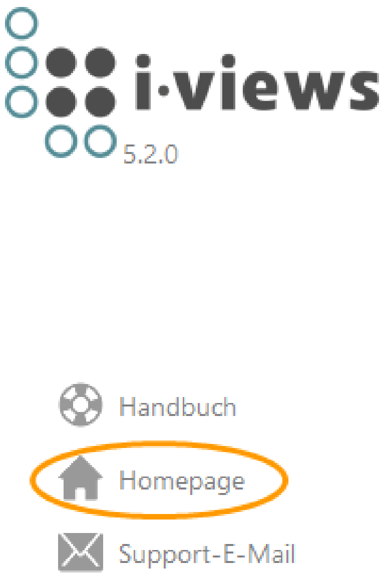
In contrast to the “Web-link” action type, this is a link to a preconfigured address, like the “Manual” action type.

Setting options

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

1.7.3.3.4 Action type "Homepage"

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

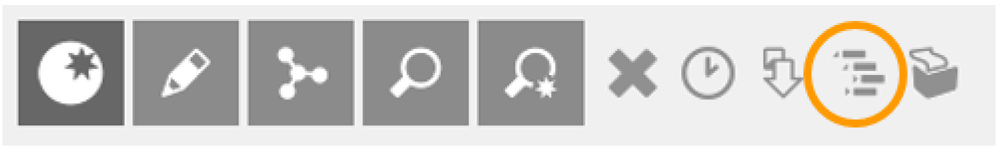


Setting options

Name	Value
URL	Link to a website

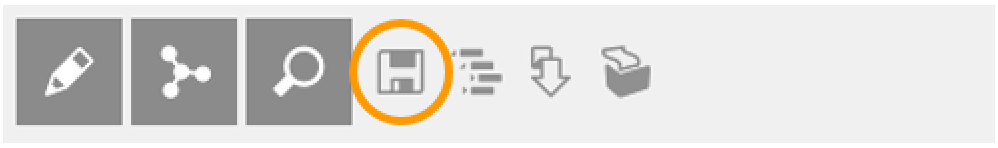
1.7.3.3.5 Action type "Show in tree"

The Show in tree action can be used to display the location of an element from the semantic network. 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.

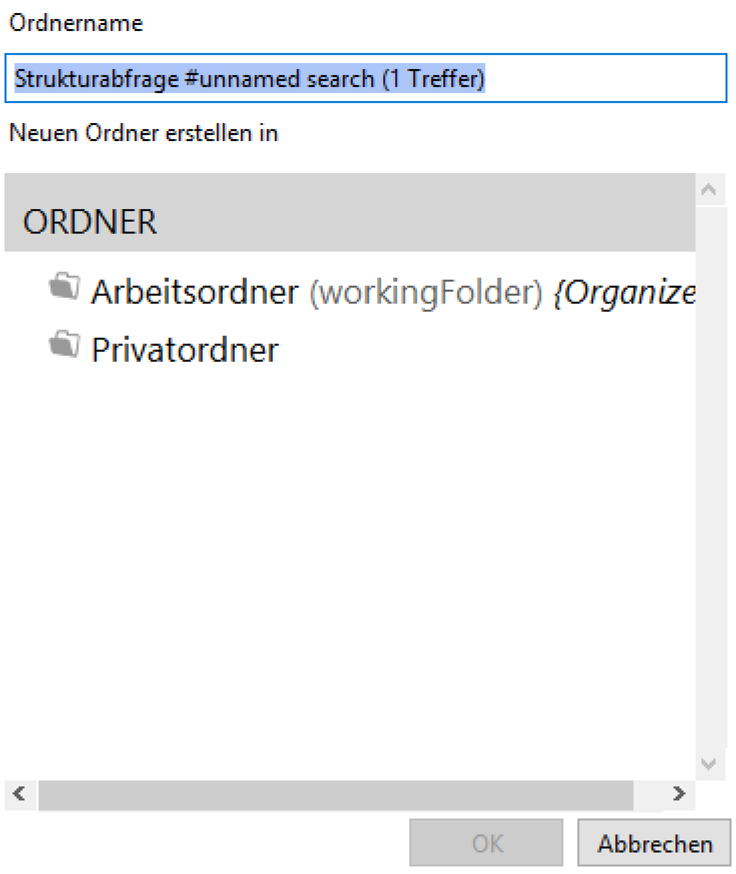


1.7.3.3.6 Action type "Save query result"

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:



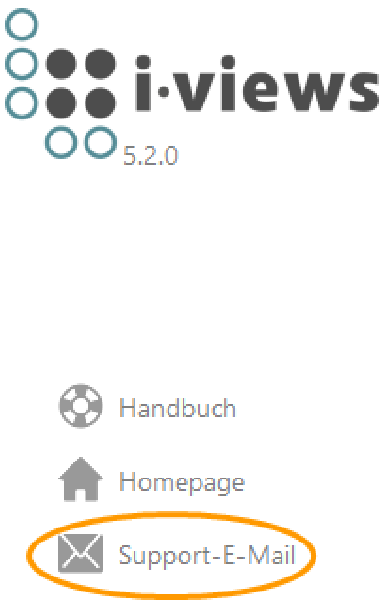
This action saves the query result in a folder you can choose:

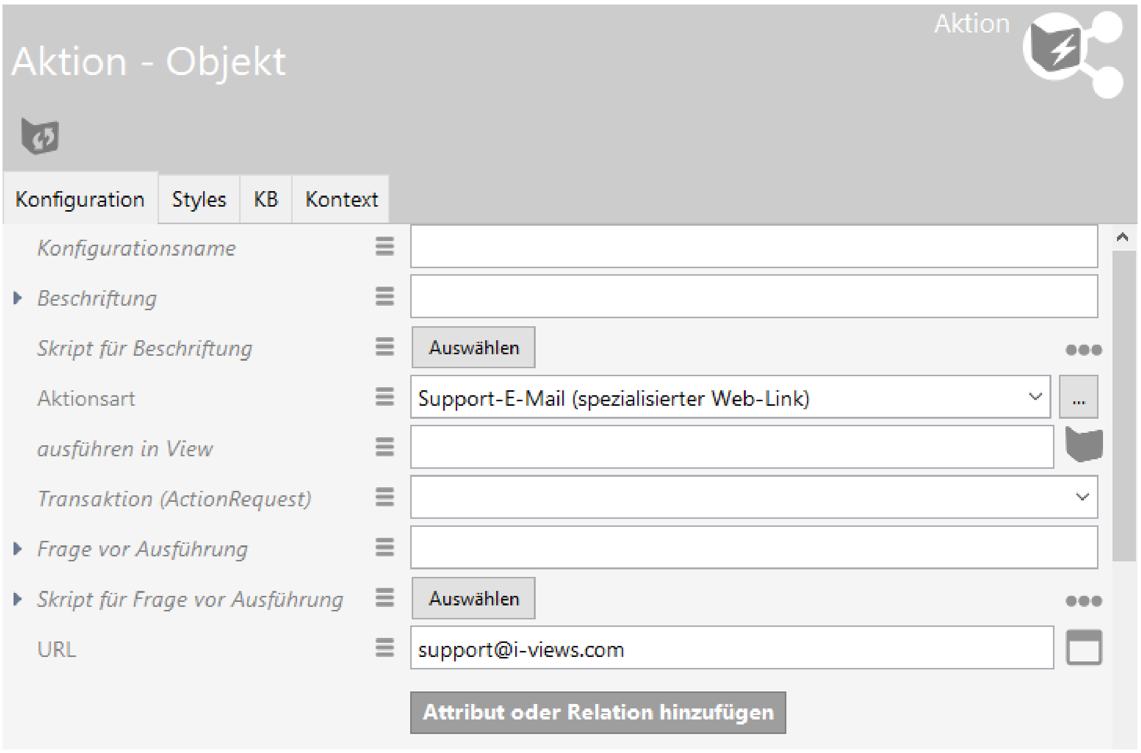

Ordnername
Strukturabfrage #unnamed search (1 Treffer)
Neuen Ordner erstellen in
ORDNER
Arbeitsordner (workingFolder) {Organize
Privatordner
OK
Abbrechen

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.

1.7.3.3.7 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.





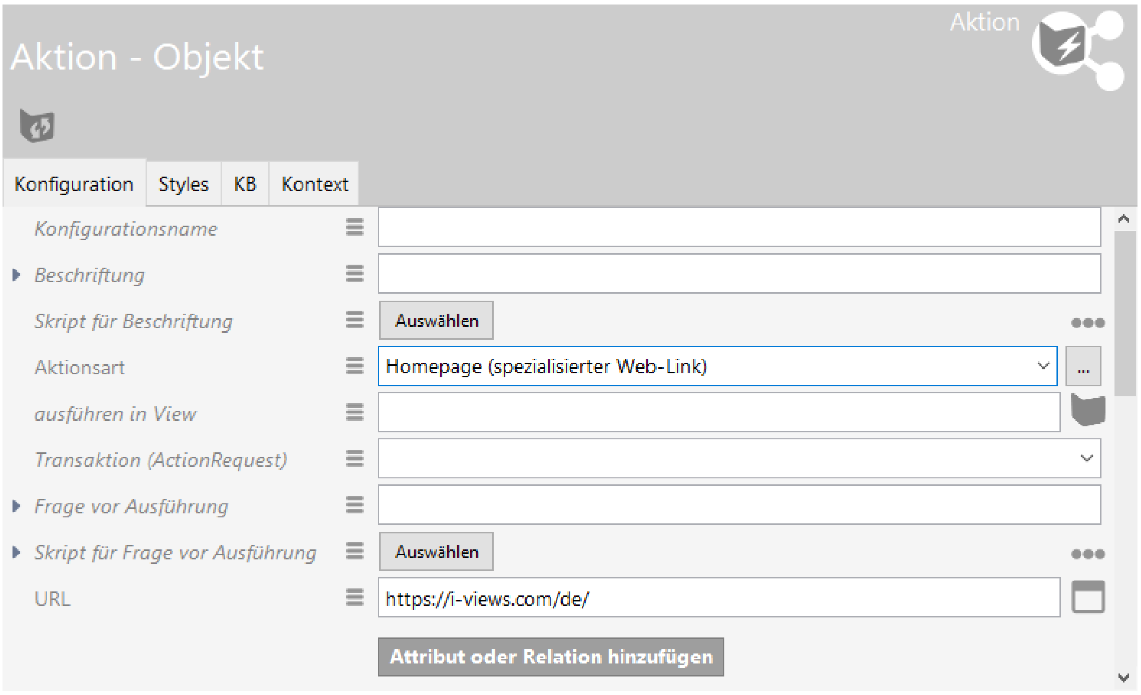
Konfigurationsname	
Beschriftung	
Skript für Beschriftung	Auswählen
Aktionsart	Support-E-Mail (spezialisierter Web-Link)
ausführen in View	
Transaktion (ActionRequest)	
Frage vor Ausführung	
Skript für Frage vor Ausführung	Auswählen
URL	support@i-views.com

Setting options

Name	Value
URL	Email link

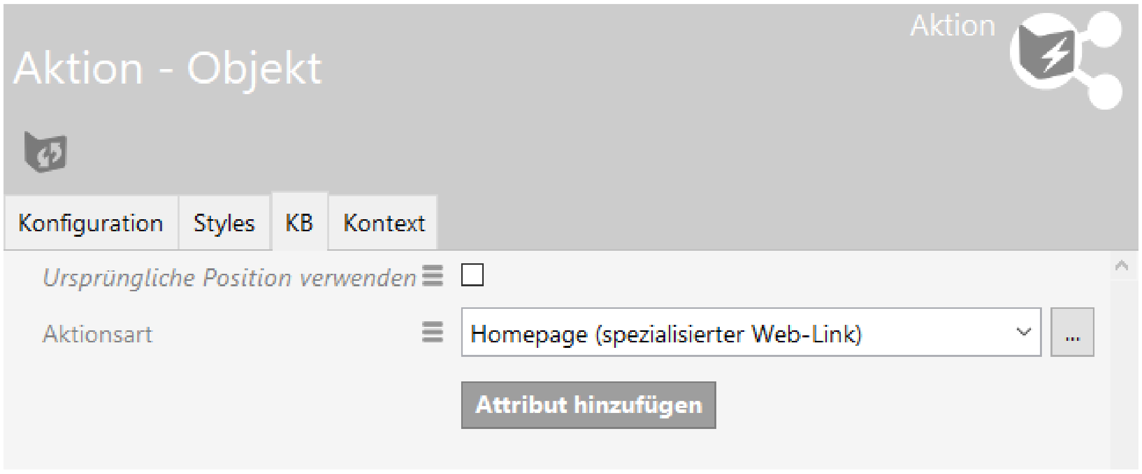
1.7.3.3.8 Action type "Web link"

The “web link” action type can be used for the start view of the KB. It differs from the “manual” action type in that you can assign any web address as the hyperlink.



Konfigurationsname:
Beschriftung:
Skript für Beschriftung:
Aktionsart: Homepage (spezialisierter Web-Link)
ausführen in View:
Transaktion (ActionRequest):
Frage vor Ausführung:
Skript für Frage vor Ausführung:
URL: https://i-views.com/de/

Note: In later KB versions (KB 5.2.2) the “home page” action type is only available on the “KB” tab.



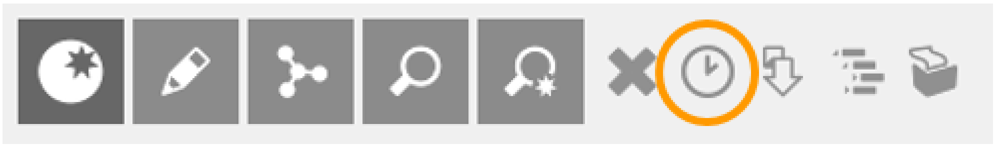
Ursprüngliche Position verwenden: ☐
Aktionsart: Homepage (spezialisierter Web-Link)

Setting options

Name	Value
URL	Address of the web link.

1.7.3.3.9 Action type "Recently accessed objects"

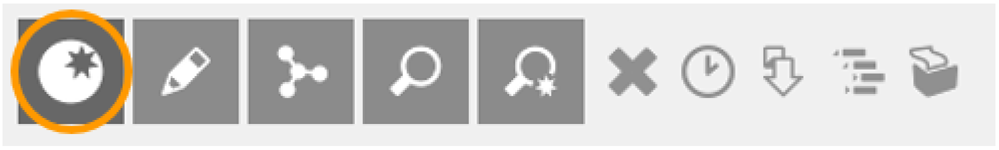
Shows the objects (semantic elements) that were last used in the respective table. Objects might be filtered depending on the definition of the table.



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

1.7.3.3.10 Action type "New"

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



Note: Instead of the new action, a script must be used in the web front-end. For more information, see the chapter "JavaScript-API".

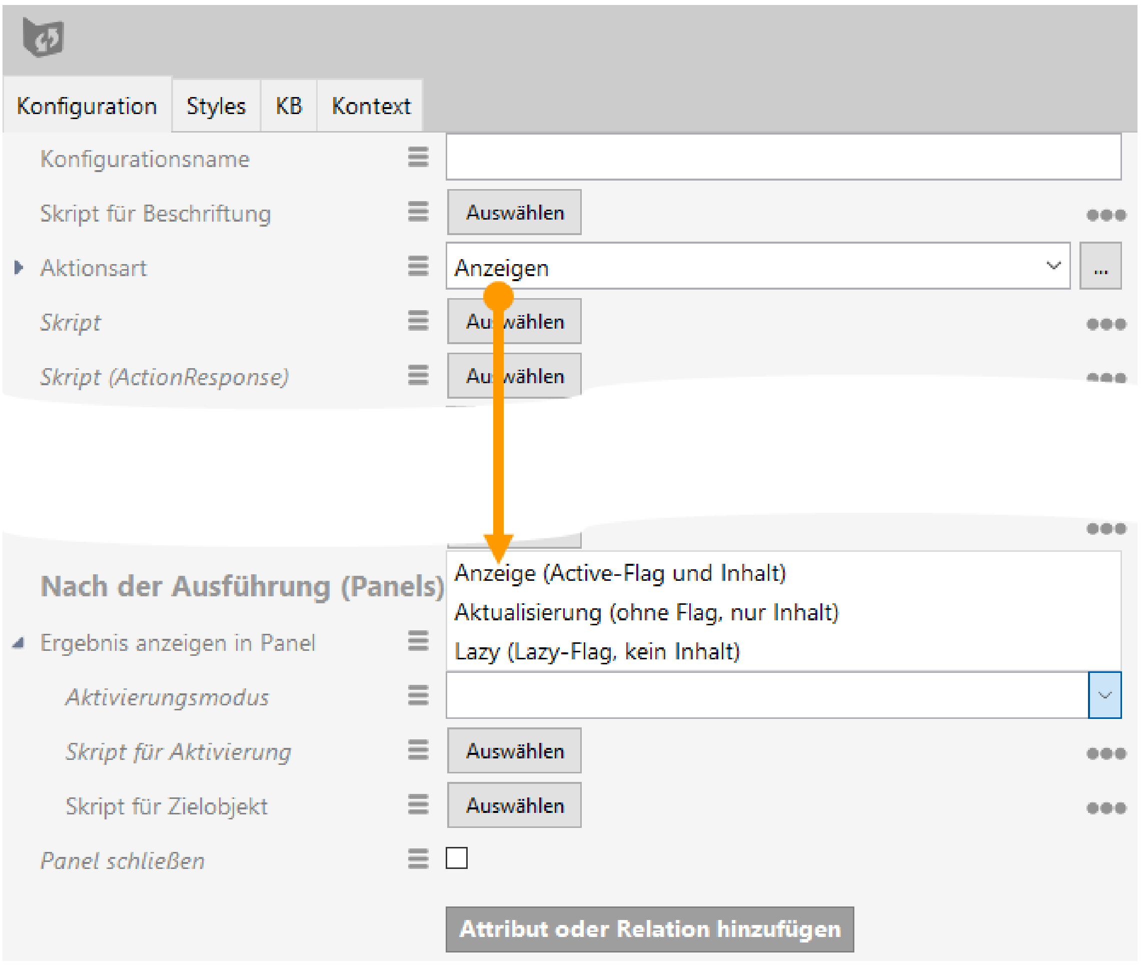
1.7.3.4 Actions for the viewconfiguration mapper

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

1.7.3.4.1 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.



Konfigurationsname
Skript für Beschriftung
Aktionsart: Anzeigen
Skript
Skript (ActionResponse)

Nach der Ausführung (Panels)
Ergebnis anzeigen in Panel
Aktivierungsmodus
- Anzeige (Active-Flag und Inhalt)
- Aktualisierung (ohne Flag, nur Inhalt)
- Lazy (Lazy-Flag, kein Inhalt)

Skript für Aktivierung
Skript für Zielobjekt
Panel schließen
Attribut oder Relation hinzufügen

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

Display (Active flag and content) = “Push response"	The view is recalculated if the display action is influenced or if any change is made to the content, irrespective of whether or not the panel is activated (= visible). This mode makes sense, for example, when calling up a dialog panel.

Update (without flag, content only) = "Delta load"	This view is only recalculated if the content changes: • When the panel is called up for the first time, the view is calculated. • If a view exists, it is kept even if another panel is called up in the meantime. The view is "stored" for the duration of the session until it has to be recalculated because the content has changed. An example of this is the display of a graph: As long as no changes are made to the graph, the same, unchanged graph panel is displayed every time the graph is called up.
Lazy (lazy flag, no content) = "Pull response"	If the content changes, the view is not calculated until the panel is called up. By setting the Lazy flag, a separate request only loads the new view if the panel is activated. An example of this is a shopping cart: The composition of a shopping cart can change constantly but the content of the shopping cart only needs to be displayed at certain times.

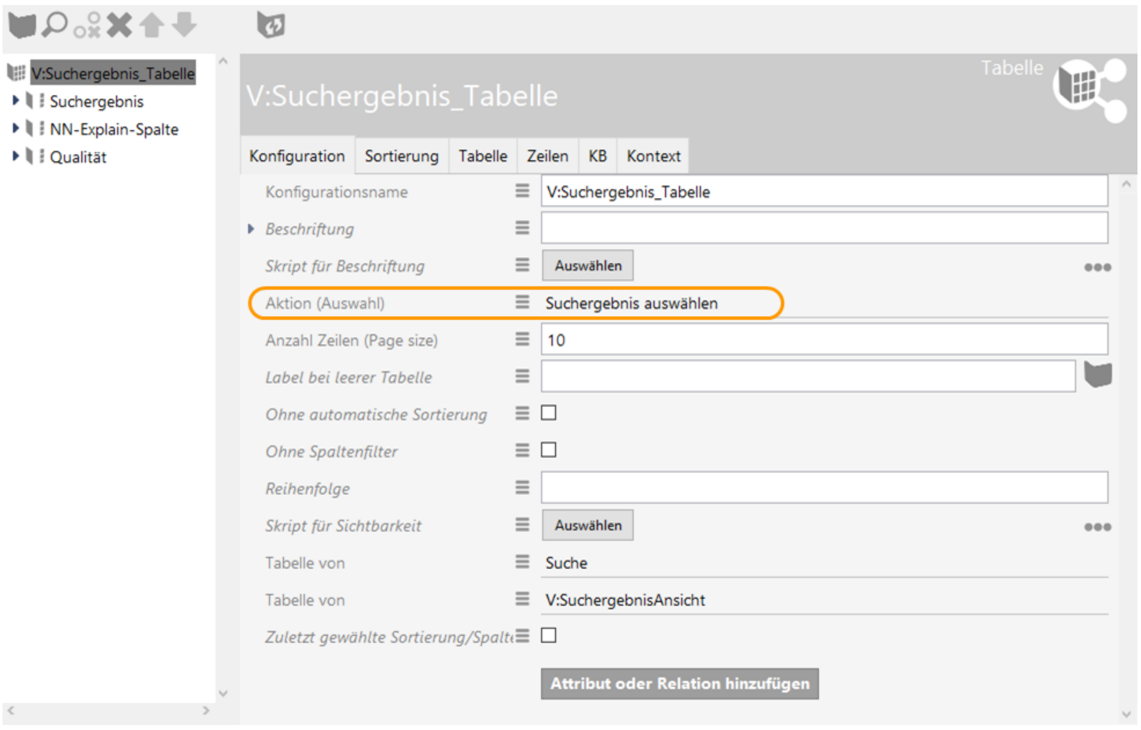
If no activation mode has been selected, the "Display (active flag and content)" mode applies by default.

1.7.3.4.2 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. **Please 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.

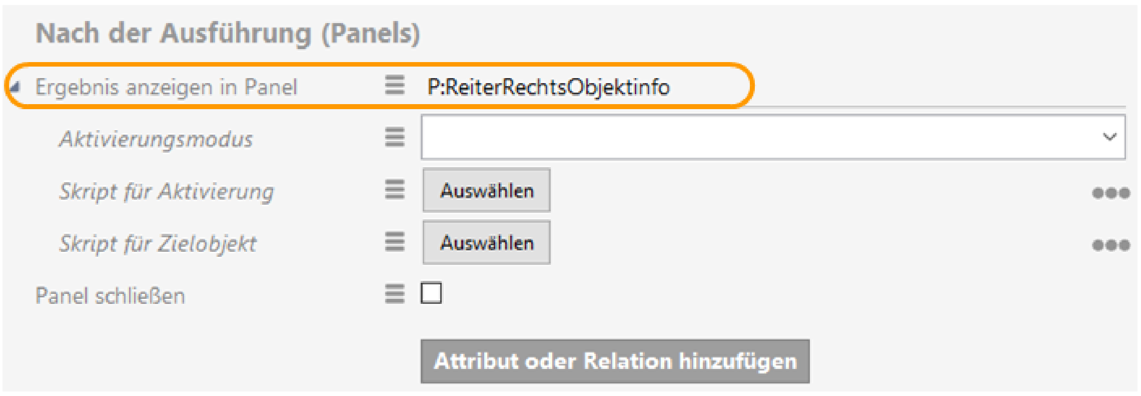
Example



V:Suchergebnis_Tabelle

Konfigurationsname	V:Suchergebnis_Tabelle
Beschriftung	
Skript für Beschriftung	Auswählen
Aktion (Auswahl)	Suchergebnis auswählen
Anzahl Zeilen (Page size)	10
Label bei leerer Tabelle	
Ohne automatische Sortierung	☐
Ohne Spaltenfilter	☐
Reihenfolge	
Skript für Sichtbarkeit	Auswählen
Tabelle von	Suche
Tabelle von	V:SuchergebnisAnsicht
Zuletzt gewählte Sortierung/Spalte	☐

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.”



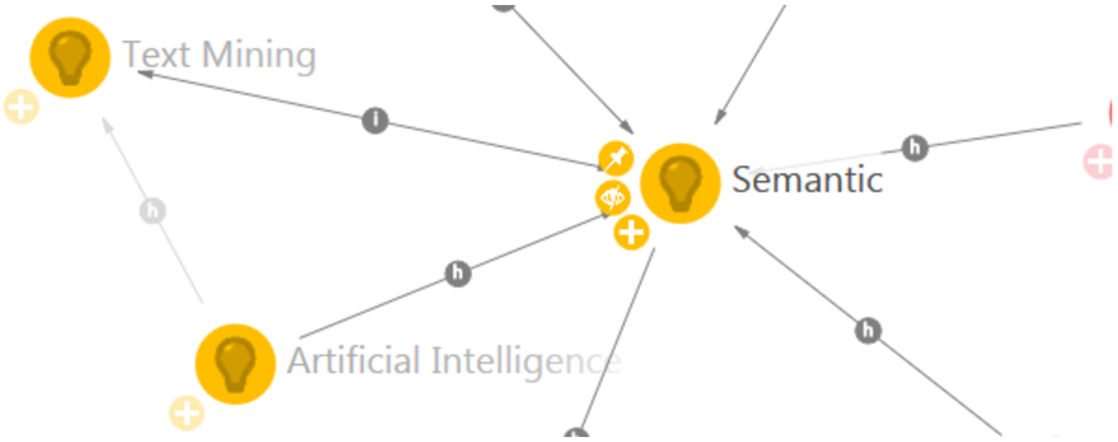
Nach der Ausführung (Panels)

Ergebnis anzeigen in Panel	P:ReiterRechtsObjektinfo
Aktivierungsmodus	
Skript für Aktivierung	Auswählen
Skript für Zielobjekt	Auswählen
Panel schließen	☐

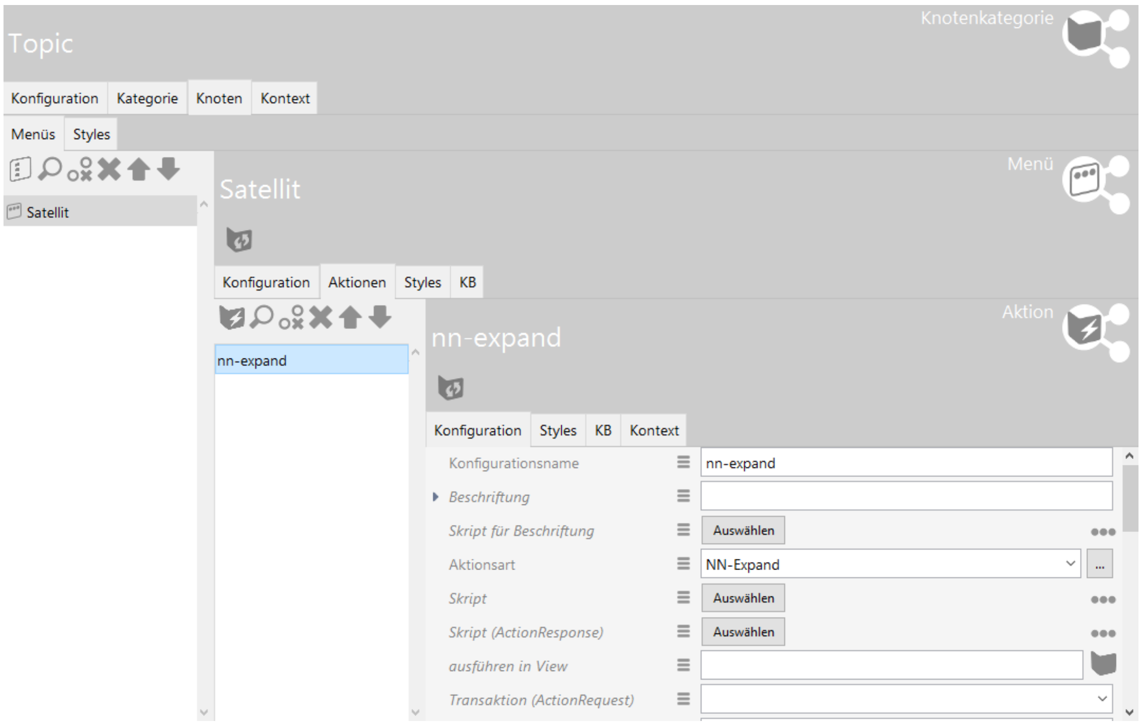
1.7.3.4.3 Action type "NN-Expand"

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.



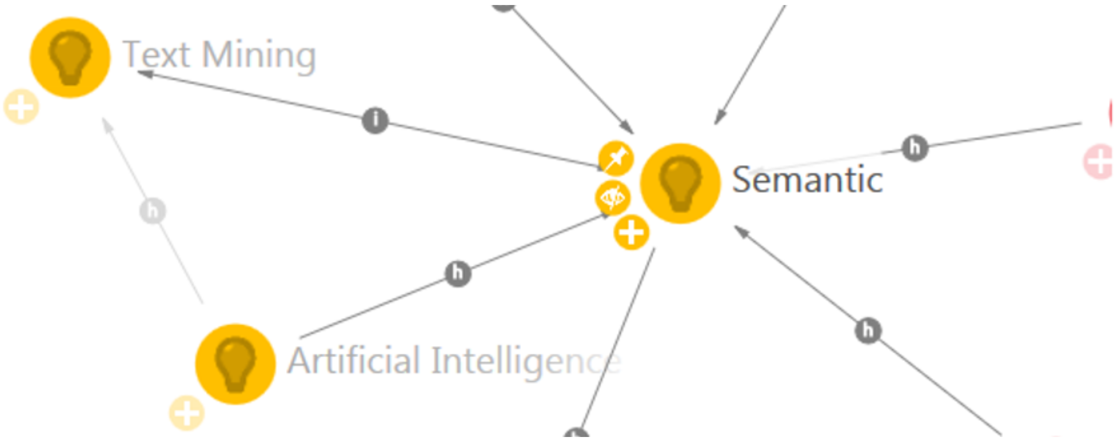
In the graph configuration this action is attached to all node categories that are supposed to be equipped with it. A menu that can contain all NN actions is created on the “Node” tab. In the action itself, it is only necessary to select the “NN-Expand” action type, all other specifications are optional. The action type may not be available in the drop-down menu; further action types are available from the neighboring “...” button.



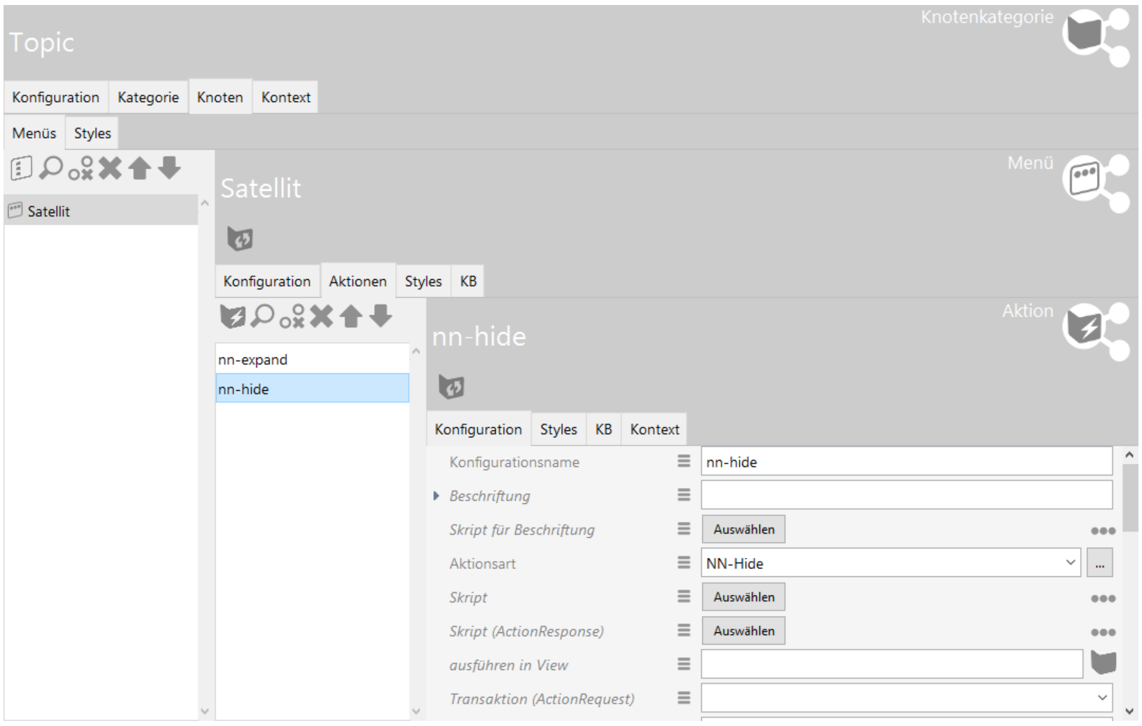
Konfigurationsname	nn-expand
Beschriftung	
Skript für Beschriftung	Auswählen
Aktionsart	NN-Expand
Skript	Auswählen
Skript (ActionResponse)	Auswählen
ausführen in View	
Transaktion (ActionRequest)	

1.7.3.4.4 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.

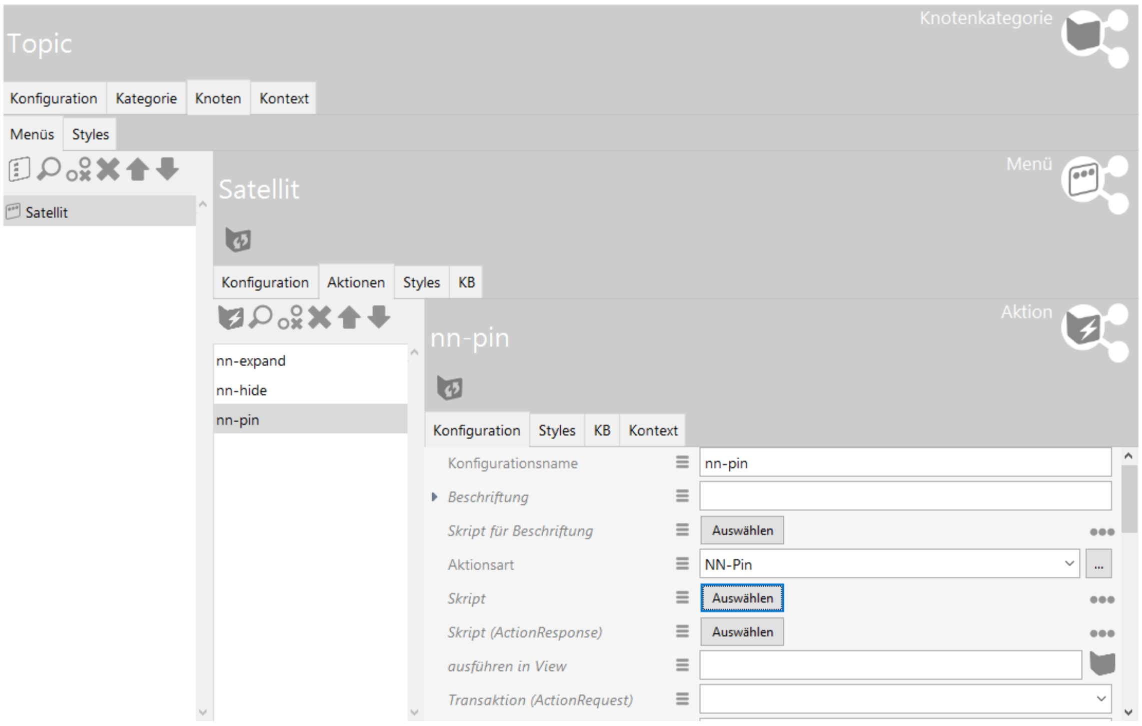


Konfigurationsname	nn-hide
Beschriftung	
Skript für Beschriftung	Auswählen
Aktionsart	NN-Hide
Skript	Auswählen
Skript (ActionResponse)	Auswählen
ausführen in View	
Transaktion (ActionRequest)	

1.7.3.4.5 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”.


Topic

Knotenkategorie

Konfiguration | Kategorie | Knoten | Kontext

Menüs | Styles

Satellit

Menü

Konfiguration | Aktionen | Styles | KB

nn-expand
nn-hide
nn-pin

Aktion

nn-pin

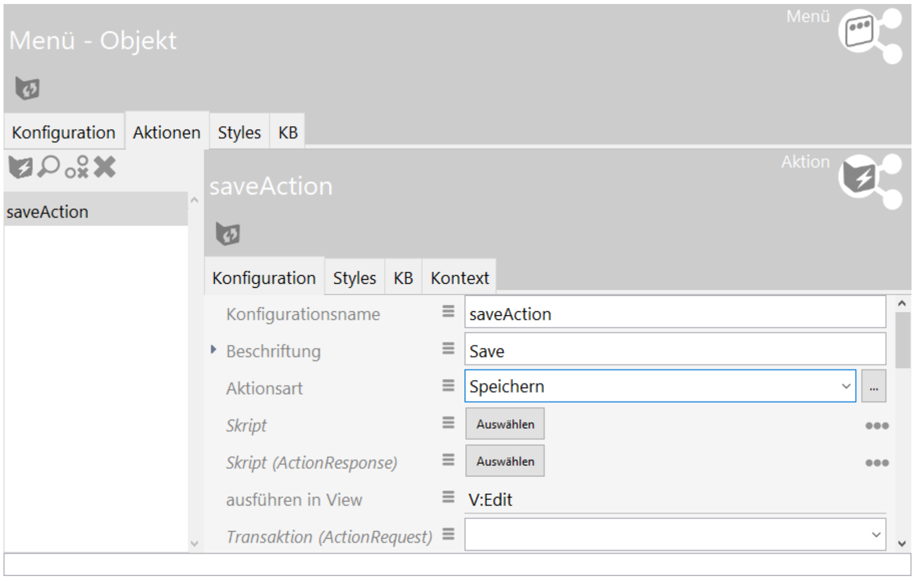
Konfiguration | Styles | KB | Kontext

Konfigurationsname	nn-pin
Beschriftung	
Skript für Beschriftung	Auswählen
Aktionsart	NN-Pin
Skript	Auswählen
Skript (ActionResponse)	Auswählen
ausführen in View	
Transaktion (ActionRequest)	

1.7.3.4.6 Action type "Save"

The Save action stores the form data from the web front-end in the semantic network. 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:


Menü - Objekt

Menü

Konfiguration | Aktionen | Styles | KB

saveAction

Aktion

saveAction

Konfiguration | Styles | KB | Kontext

Konfigurationsname	saveAction
Beschriftung	Save
Aktionsart	Speichern
Skript	Auswählen
Skript (ActionResponse)	Auswählen
ausführen in View	V:Edit
Transaktion (ActionRequest)	

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.

1.7.3.5 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:

- Show action
- Sorting 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.

1.7.3.6 Scripts of actions

1.7.3.6.1 Script (onAction)

This script is executed when the action is executed. The return value is passed on to the optional ActionResponse script.

```
function onAction(element, context) { return element;
}
```

Arguments

el-e-ment	The semantic element, in the context of which the action is executed. The “Selection” action forms the only exception - in this case, “Element” is equivalent to the element selected, and is therefore identical to “context.selectedElement.”
con-text	Other predefined variables that describe the context of the action in more detail.

The script of an action can access the following predefined variables contained in *context*:

Detail editor

Variable	Value
selectedElement	Object selected or type selected
type	Object type. If the element is a type, the actual type is used

Object list

Variable	Value
selectedElement	Object selected or type selected. Undefined if no elements, or several elements, were selected.
selectedElements	Elements selected
elements	All elements of the object list
type	Object list type

Transactions

A transaction is required for write changes. This is automatically the case when executed using the ViewConfigMapper.

By principle, no transaction is active in the Knowledge Builder. The script itself must control transactions.

Knowledge Builder

Another variable for interacting with the user is available in the Knowledge Builder:

Variable	Value
ui	Object $k.UIObject

For example, an alert can show:

```
ui.alert(‘‘Current Element: " + element.name());
```

1.7.3.6.2 Script (actionResponse)

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, actionResult) { var actionResponse = new $k.ActionRespon

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

    return actionResponse;
}
```

Arguments

element	The semantic element in the context of which the action is executed
context	More predefined variables that describe the context of the action in more detail (see previous section)
actionResult	The return value of the onAction script or, if not defined, the return value of the configured action type.

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
showelement	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.

1.7.3.6.3 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) { retu
    });
}
```

1.7.3.6.4 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) { retu
    });
}
```

1.7.3.6.5 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.

1.7.3.7 Action sequences

Often you will want to summarize the changes that the user makes to the semantic network 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 semantic network. 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 semantic network 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 - start” and the final action with “Transaction - end”.

Careful: The transaction is started only if the first action actually modifies the semantic network. 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 end 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 semantic network 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.

1.7.4 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.

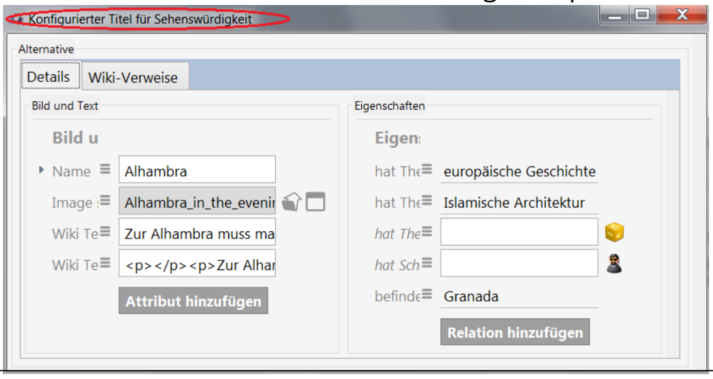
The individual view configuration elements can, in part, be plugged together in any way. The configurations can also be used multiple times as a sub-configuration.

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
Group	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.
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.  Konfigurierter Titel für Sehenswürdigkeit Alternative Details | Wiki-Verweise Bild und Text: Name: Alhambra; Image: Alhambra_in_the_evenir; Wiki Te: Zur Alhambra muss ma; Wiki Te: <p></p><p>Zur Alhar Eigenschaften: hat The: europäische Geschichte; hat The: Islamische Architektur; hat The: ; hat Sch: ; befinde: Granada Bild u Attribut hinzufügen Eigen: Relation hinzufügen

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

1.7.4.1 Alternative

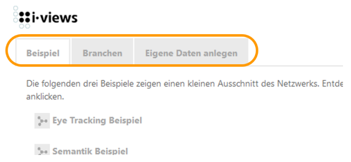
An alternative is used to configure many different alternative views on an object. You can use tabs to switch between the views in the application.

Setting options

Name	Value
Label	A label is only used if this configuration is embedded in another configuration, e.g. an additional *alternative*.
has default alternative	The sub-view that is supposed to be selected initially can be specified here.

Display in an application

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



Die folgenden drei Beispiele zeigen einen kleinen Ausschnitt des Netzwerks. Entde
anklicken.

Example of an alternative in an application: You can use the tabs to switch between the views "Example," "Branches" and "Create own data."

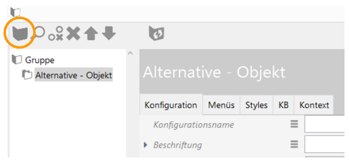
Display in Knowledge Builder

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 "Tab 1" and the view "Tab 2."

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 "Group" 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.

1.7.4.2 Groups

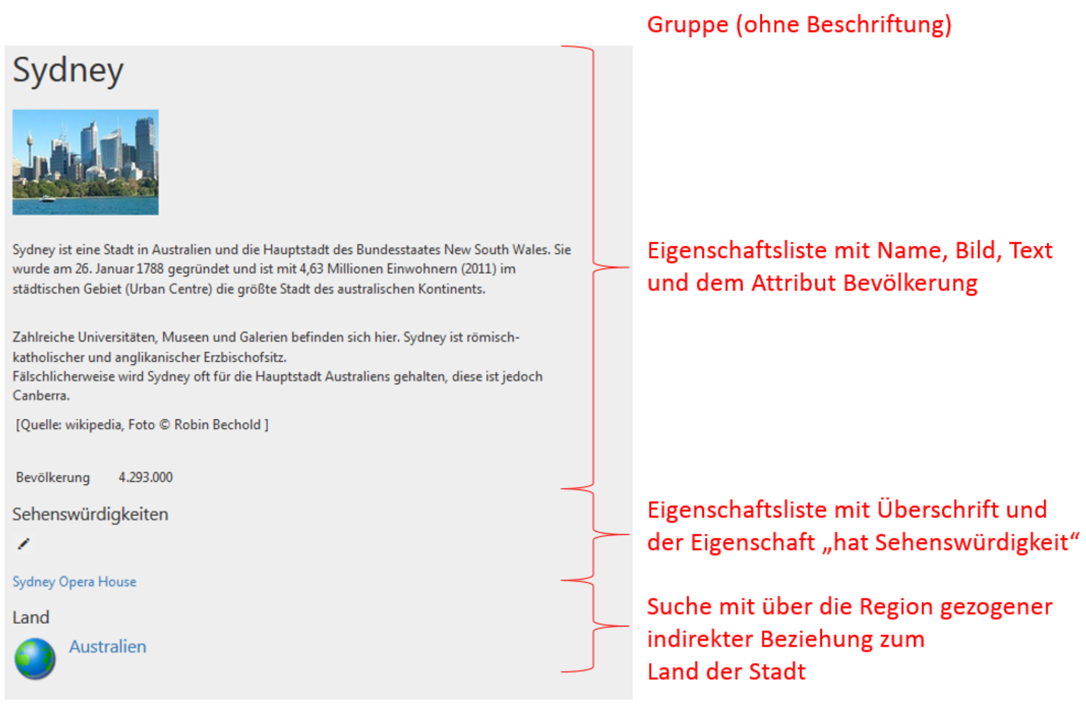
You can use a group to summarize different sub-configurations in one view. The subelements are then shown in order. However, there are exceptions that only apply for the front-end: The configuration element *Property* cannot be a direct sub-configuration of a group. This initially requires the *properties* configuration.

Setting options

Name	Value
Label	A label is only used if this configuration is embedded in another configuration, e.g. *Alternative*.

Display in an application

If the view is exported into JSON, the individual subviews are attached to the KEY *group* in an ARRAY.


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

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

Display in Knowledge Builder

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

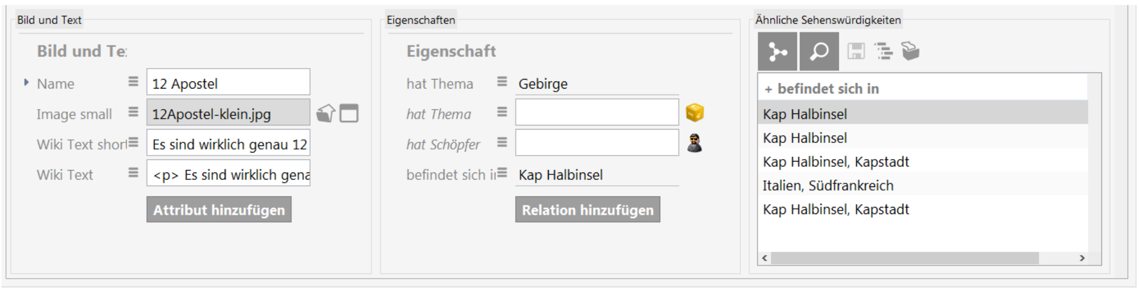


Bild und Te

Name	12 Apostel
Image small	12Apostel-klein.jpg
Wiki Text short	Es sind wirklich genau 12
Wiki Text	<p> Es sind wirklich gena

Eigenschaft

hat Thema	Gebirge
hat Thema	
hat Schöpfer	
befindet sich i	Kap Halbinsel

+ befindet sich in
Kap Halbinsel
Kap Halbinsel
Kap Halbinsel, Kapstadt
Italien, Südfrankreich
Kap Halbinsel, Kapstadt

A group with the following sub-configurations: the “Image and Text” property list, the “Properties” property list and the “Similar points of interest” query

1.7.4.3 Hierarchy

The configuration type “Hierarchy” displays elements of a semantic model 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

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.
Display the hierarchy root banner	Only relevant for the Knowledge Builder: Banner is displayed.
Action (selection)	Reference to an action that is called when a hierarchy element is clicked.
Hide details view	The details view of a selected object is displayed (Knowledge Builder) or output (json, as a *subview*) by default. By activating this option, no details view is displayed or output.
Script for visibility	
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.
Deny 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.

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). • *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.

Options for determining the elements that form the hierarchy

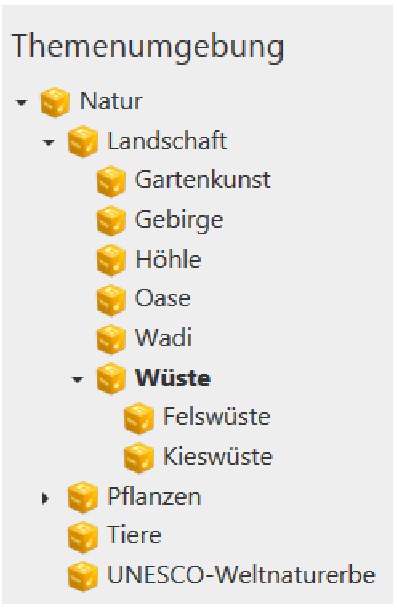
Name	Determining of...
Relation (descending)	Subelements
Relation (ascending)	Top-level elements
Structured query (descending)	Subelements
Structured query (ascending)	Top-level elements
Script (descending)	Subelements
Script (ascending)	Top-level elements

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.

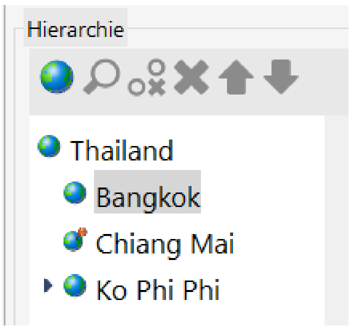
Display in an application

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*.



Notes

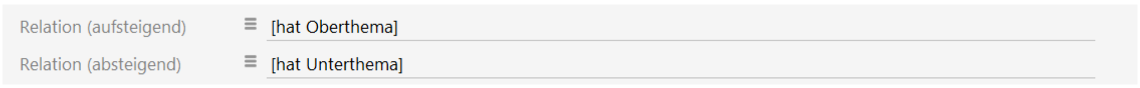
- Elements are not always represented by their name in hierarchies. It is not possible to display anything other than the name, or information supplementing the name, directly in the hierarchy.
- The values of all properties that can be filled out for forming the hierarchy are relations.
- The individual 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

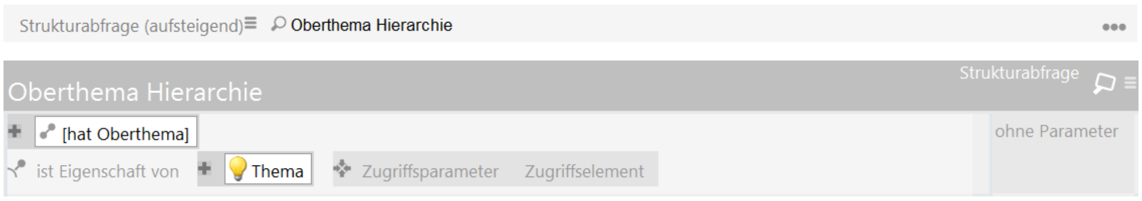
The most direct variant. The relations that form the hierarchy are entered.



Relation (aufsteigend)	[hat Oberthema]
Relation (absteigend)	[hat Unterthema]

2. Structured query that forms the hierarchy

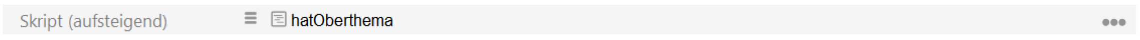
The relations can also be determined by means of a structured query.



Strukturabfrage (aufsteigend) Oberthema Hierarchie

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.



Skript (aufsteigend) hatOberthema

Script *has supertopic*

```
function relationsOf(element)
{
    return element.relations( hasSupertopic');
}
function targetsOf (element)
{
    return element.relationTargets( hasSupertopic');
}
```

1.7.4.4 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 semantic network 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.

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. Please 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.

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.

1.7.4.5 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 semantic network object or type.

Setting options

Name	Value
Label	Display name of the collection of properties. If no label is specified, the string Properties is used in Knowledge Builder.

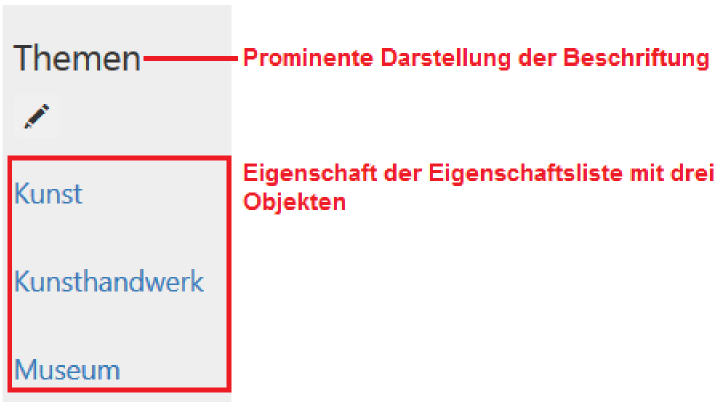
Script for label	Alternatively, the display name can also be determined via a script.
Script for visibility	Control of the visibility of the properties by a script.
Initially expanded	If this configuration in included e.g. as a meta-configuration, this parameter can be used if this is supposed to be expanded already when opening Knowledge Builder. **Note:** The web front-end does not display the affected meta-property if the checkmark is not set here.

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). • *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.

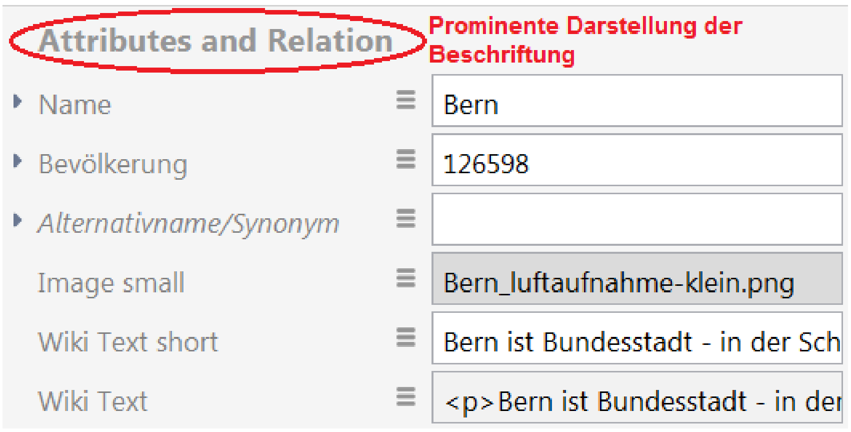
Display in applications

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*.


Themen
Prominente Darstellung der Beschriftung
Eigenschaft der Eigenschaftsliste mit drei Objekten
Kunst
Kunsthandwerk
Museum

Display in Knowledge Builder

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



Name	Bern
Bevölkerung	126598
Alternativname/Synonym	
Image small	Bern_luftaufnahme-klein.png
Wiki Text short	Bern ist Bundesstadt - in der Sch
Wiki Text	<p>Bern ist Bundesstadt - in der

Note

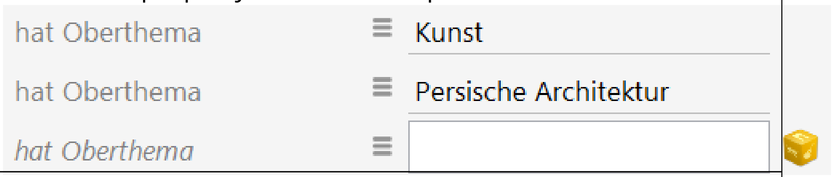
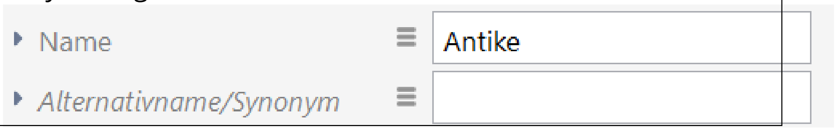
Meta properties are appended using the same process.

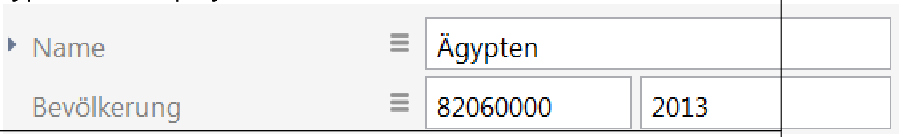
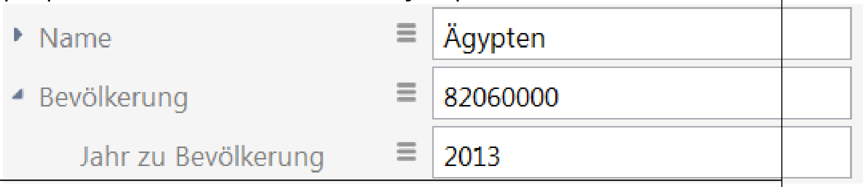
1.7.4.6 Property

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
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.

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 *Selection* or *Relational structure*. **2.** The property is a file attribute: Selection option for the display of the value in a file attribute. Selection options: • *Icon (topic*Icon): Icon of the relation target / file as an icon • *Icon and string* • *String (name attribute*: Name of the relation target / name of the file
Label	Display name of the property. If no label is specified, the name of the property type is output.
Property	Link to the property type that is to be displayed.
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. Careful: if you set this flag, this can have a significant effect on performance, depending on the script.
Display 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 *Display relation target* option, the relation targets are displayed directly, which means not just their names, but also all their properties.
Show additional properties	Only relevant in the view for editing objects: If this option is set, another property is shown in addition to the property, so this can be filled quickly and conveniently. It must be permitted for the property to occur multiple times.  hat Oberthema ≡ Kunst hat Oberthema ≡ Persische Architektur *hat Oberthema* ≡
Show new properties	Only relevant in the view for editing objects: 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 ≡ Antike *Alternativname/Synonym* ≡

Display 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.
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: Ägypten; Bevölkerung: 82060000, 2013
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: Ägypten; Bevölkerung: 82060000; Jahr zu Bevölkerung: 2013
Relation target view	If a relation is chosen as the property, this parameter can be used to define the view of the relation targets: • *Selection*: 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. • *Relational 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.
Script for label	The label can be determined by means of a script specified here.

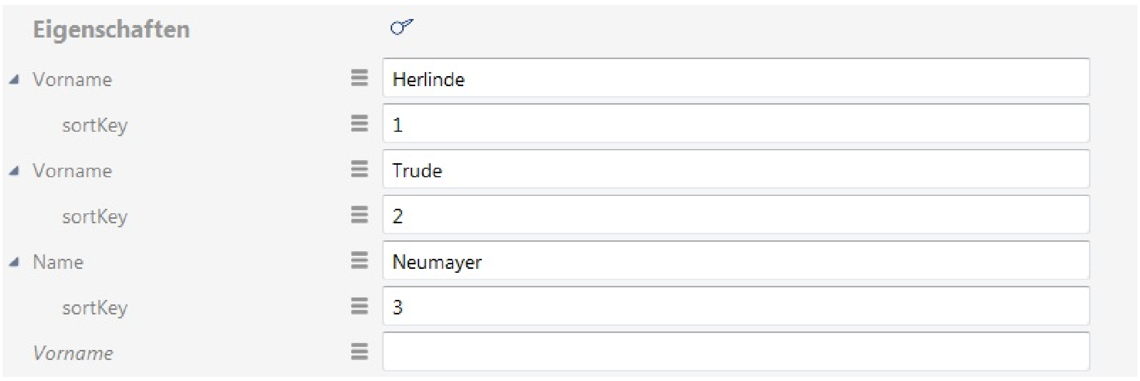
Script for relation target identifier	This script can be used to control how the relation target is displayed for relations. If no script is entered, the object name of the relation target is used for the display. Application example: A person belongs to a department with the name Dpt. IV . Using a suitable script, it is possible to change the output for the person from Dpt. IV to Darmstadt city administration, Dpt. IV .
Script for sorting	The script is used to determine a value for sorting. See the example below.
Search for target selection	The search determines which semantic elements are offered as possible relation targets. For configuration of the search, see also the Search / queries chapter.
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.

Example of a *script for sorting*

Precondition: The *sortKey* attribute can be attached to all properties.

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

Property display for a person

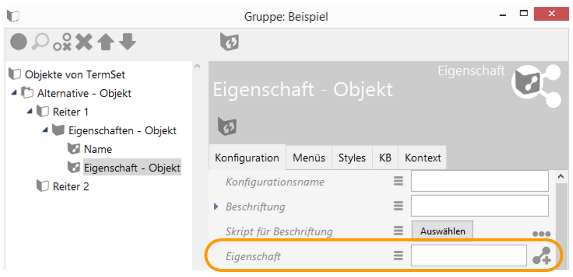


Eigenschaften

Vorname	Herlinde
sortKey	1
Vorname	Trude
sortKey	2
Name	Neumayer
sortKey	3
Vorname	

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.



Gruppe: Beispiel

- Objekte von TermSet
  - Alternative - Objekt
    - Reiter 1
      - Eigenschaften - Objekt
        - Name
        - Eigenschaft - Objekt
    - Reiter 2

Eigenschaft - Objekt

Konfiguration | Menüs | Styles | KB | Kontext

Konfigurationsname
Beschriftung
Skript für Beschriftung Auswählen
Eigenschaft

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

Note

In case of the *Name* attribute type, the value 3 is entered for *sortKey* , so this temporary value is at the end of the list.

1.7.4.7 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
Label	A label is only used if this configuration is embedded in another configuration, e.g. *Alternative*.
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.

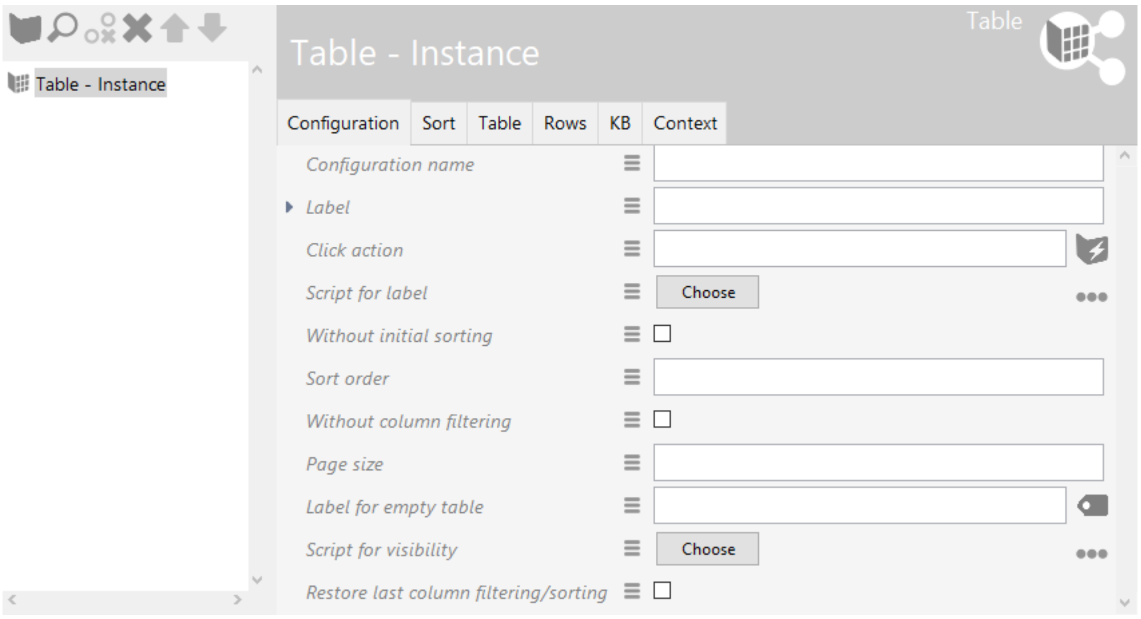
1.7.4.8 Table

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

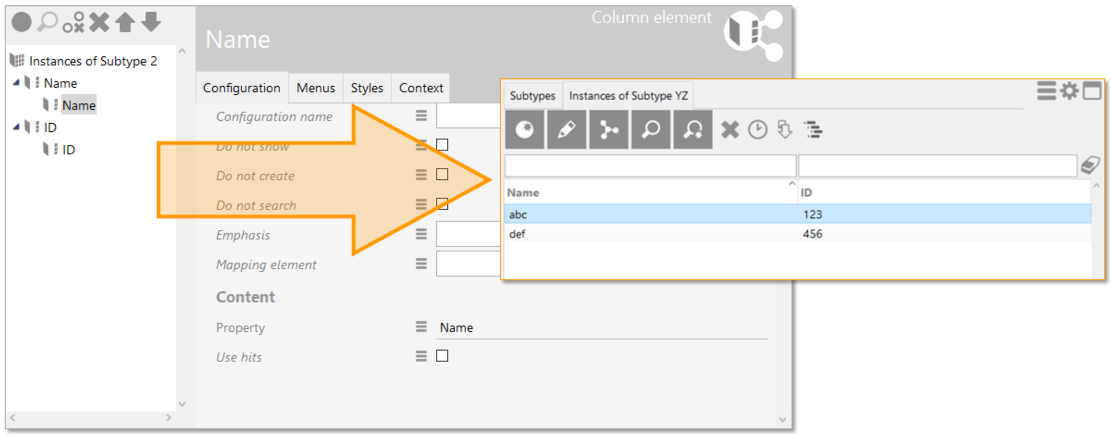
- Create and link a new subelement.
- Search through all potential subelements that already exist and link (= add) the slected subelement.
- Delete link again. When this occurs, the subelement is retained as an object and can be used again in other configurations.
- Delete complete subelement selected. 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
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.
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.
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*.

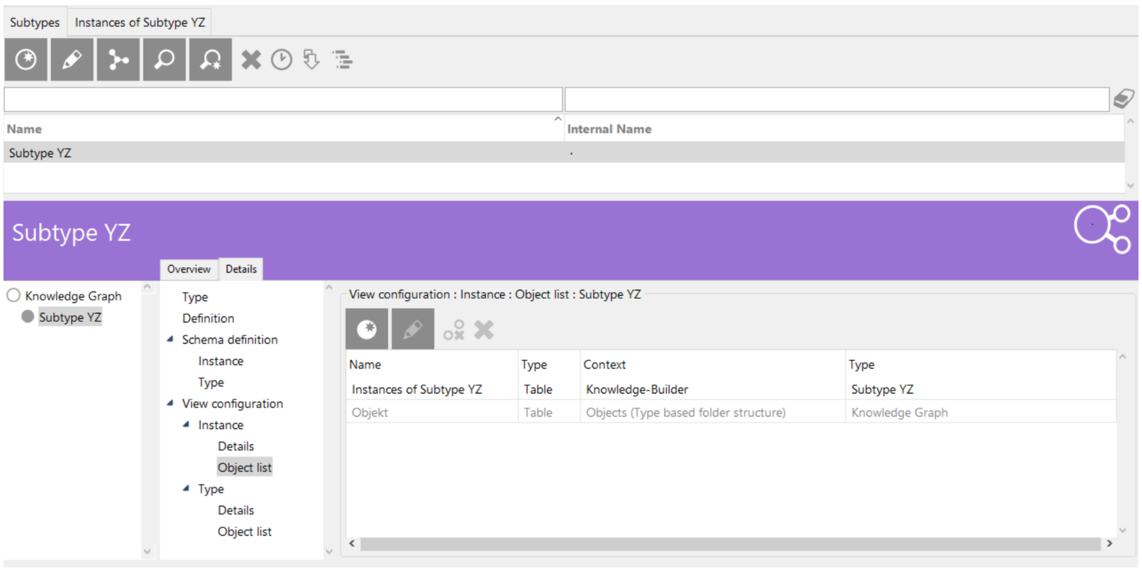
The property *Apply in* refers to an application. Several links are possible.

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.


Subtypes | Instances of Subtype YZ

Name	Internal Name
Subtype YZ	·

Subtype YZ

Overview | Details

- Knowledge Graph
  - Subtype YZ

- Type
- Definition
- Schema definition
  - Instance
  - Type
- View configuration
  - Instance
    - Details
    - Object list
  - Type
    - Details
    - Object list

View configuration : Instance : Object list : Subtype YZ

Name	Type	Context	Type
Instances of Subtype YZ	Table	Knowledge-Builder	Subtype YZ
Objekt	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.

1.7.4.8.1 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.

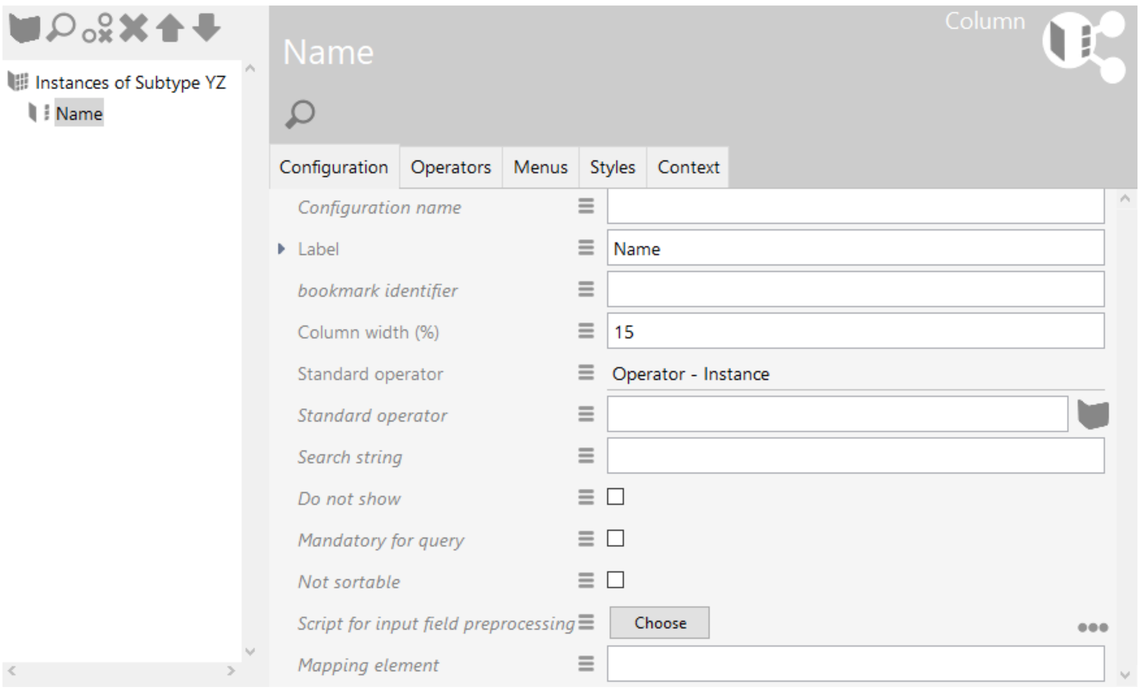
Setting options

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.
Operators	
Configuration name	The configuration name is used for identification and reuse of the configuration.
Icon	Symvol 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 genereated 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	

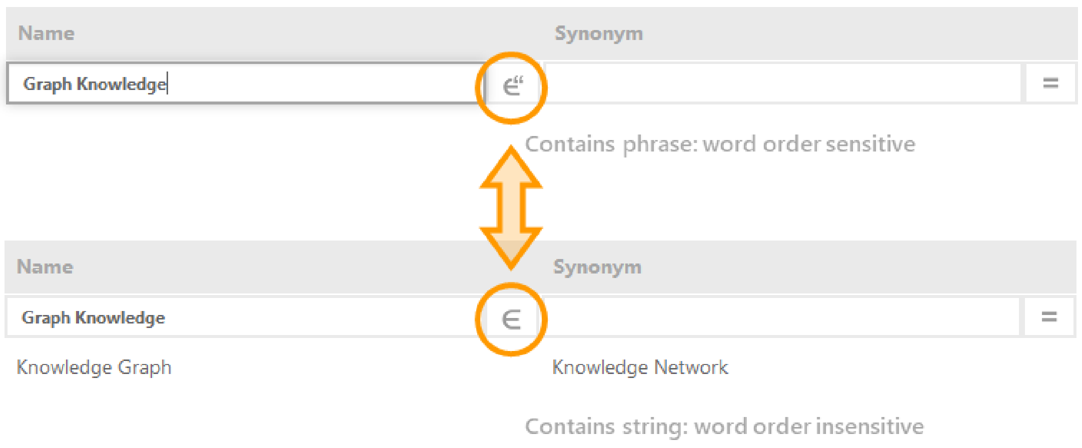
Column configuration for the Name column

1.7.4.8.2 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.

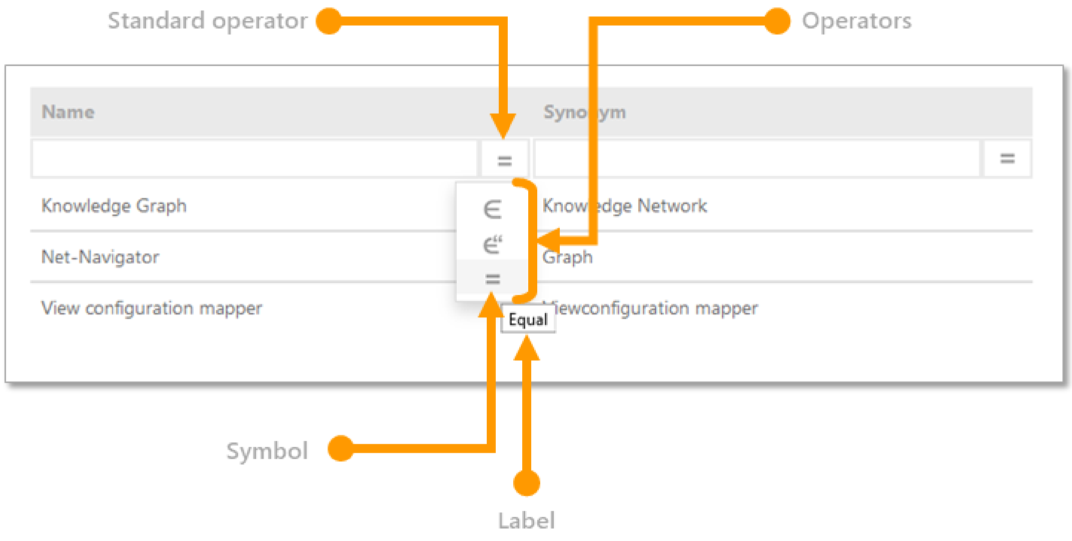
For example, the difference between "contains phrase" and "contains string" is as follows:

- "**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

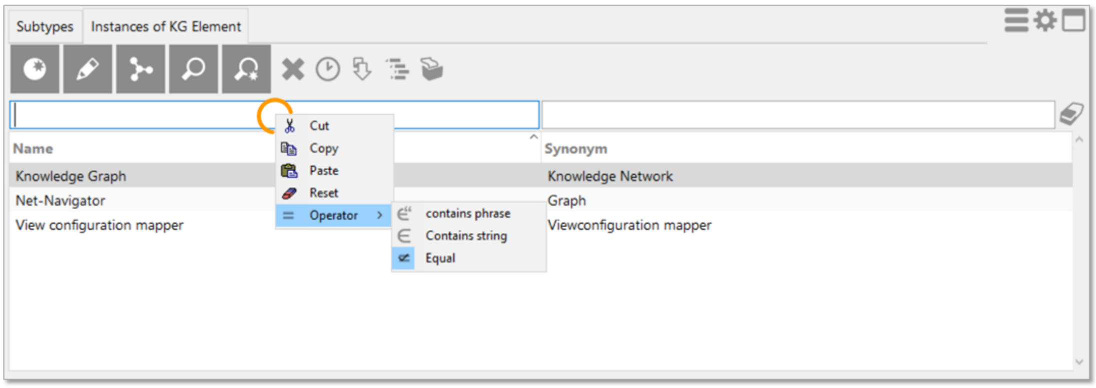


This allows to use different filtering behaviors when filtering large tables to narrow down the search results to specific content.

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	Viewconfiguration 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.

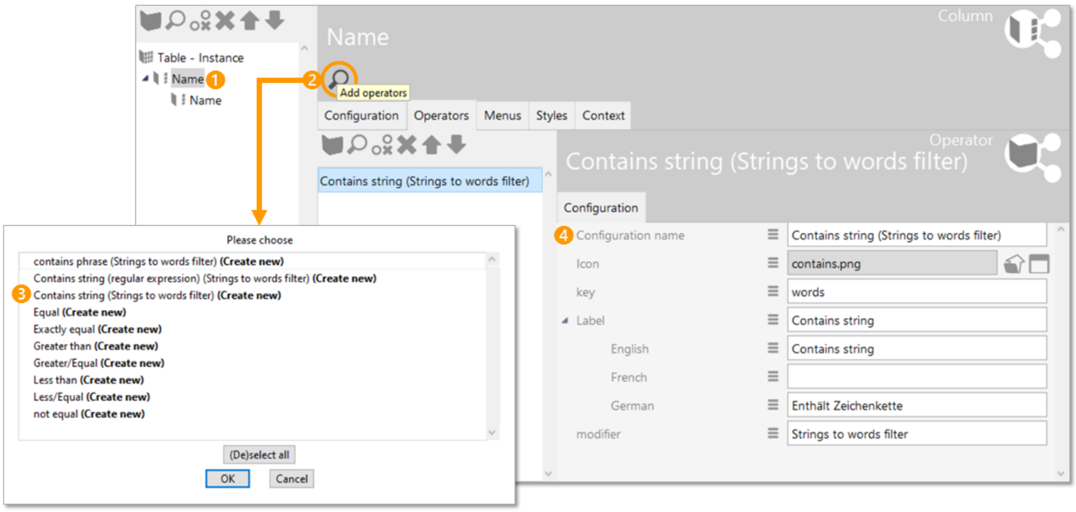
Note: Since the application of operators depends on the value type of the property to be filtered for and on the indices, the preset operators are only available if the property of the column element has been defined. If string operators are needed, a correctly configured index including index filter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

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

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).

Select the needed kind of operator.

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.

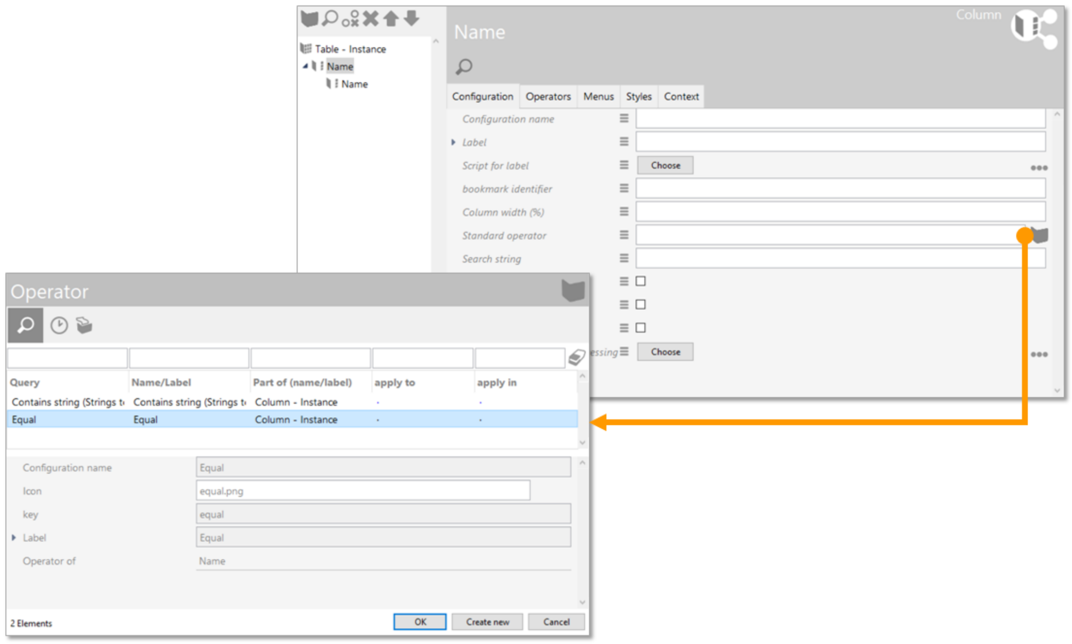


Contains string (Strings to words filter)

contains phrase (Strings to words filter) **(Create new)**
Contains string (regular expression) (Strings to words filter) **(Create new)**
3 Contains string (Strings to words filter) **(Create new)**
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)**

4 Configuration name	Contains string (Strings to words filter)
Icon	contains.png
key	words
Label	Contains string
English	Contains string
French	
German	Enthält Zeichenkette
modifier	Strings to words filter

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



Name

Configuration name	
Label	
Script for label	Choose
bookmark identifier	
Column width (%)	
Standard operator	
Search string	

Operator

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

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

2 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	Short term	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)
equalsTopicOneWay		filter with
fulltext		Contains string
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)

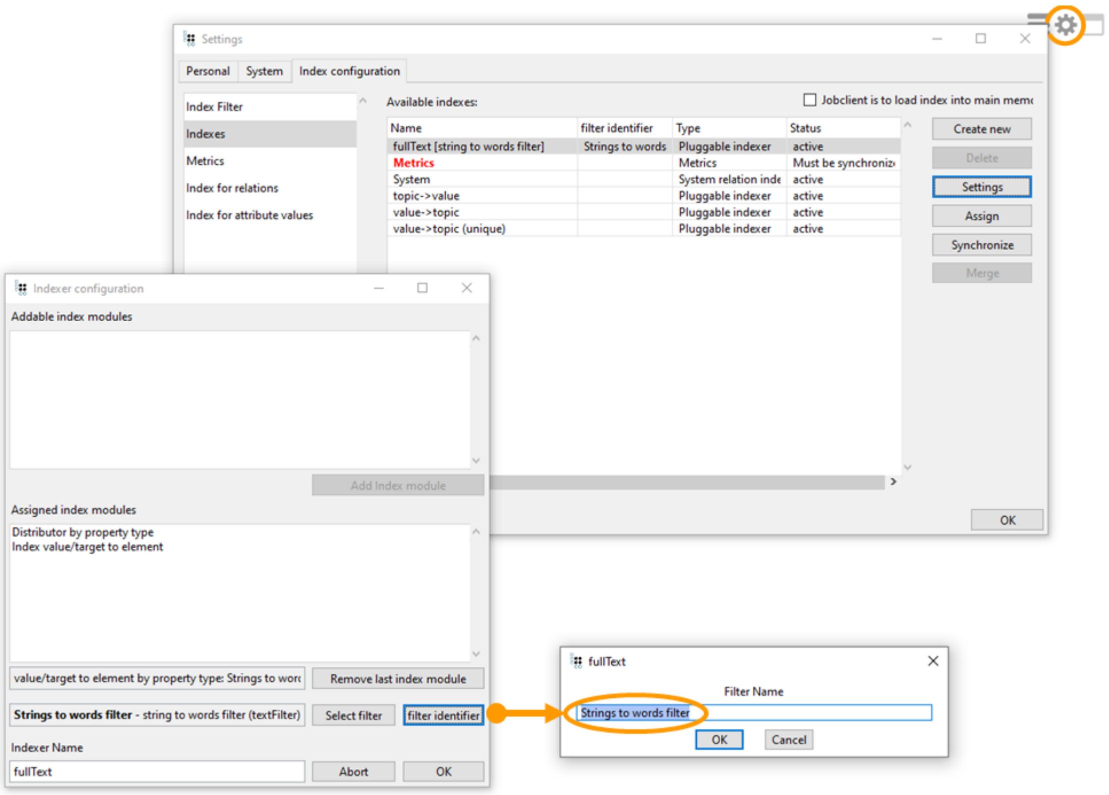
unmodifiedEqual		Exactly identical
words		Contains string

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:



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

Assigned index modules:
Distributor by property type
Index value/target to element

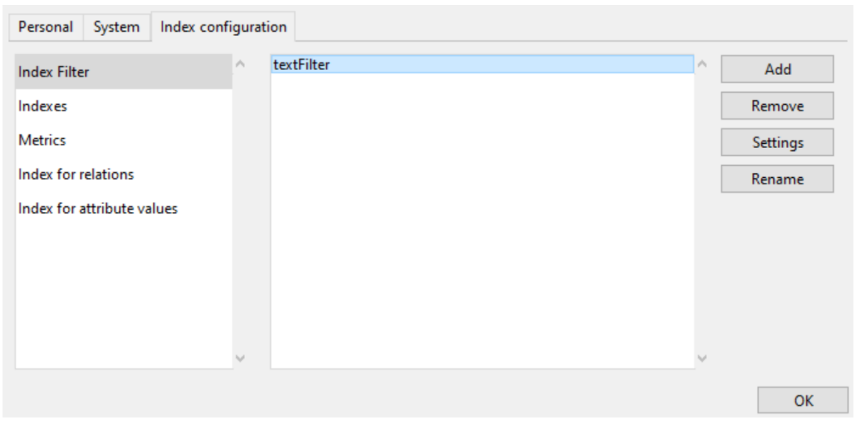
value/target to element by property type: Strings to wor

Strings to words filter - string to words filter (textFilter)

Indexer Name: fullText

Filter Name: Strings to words filter

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



1.7.4.8.3 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

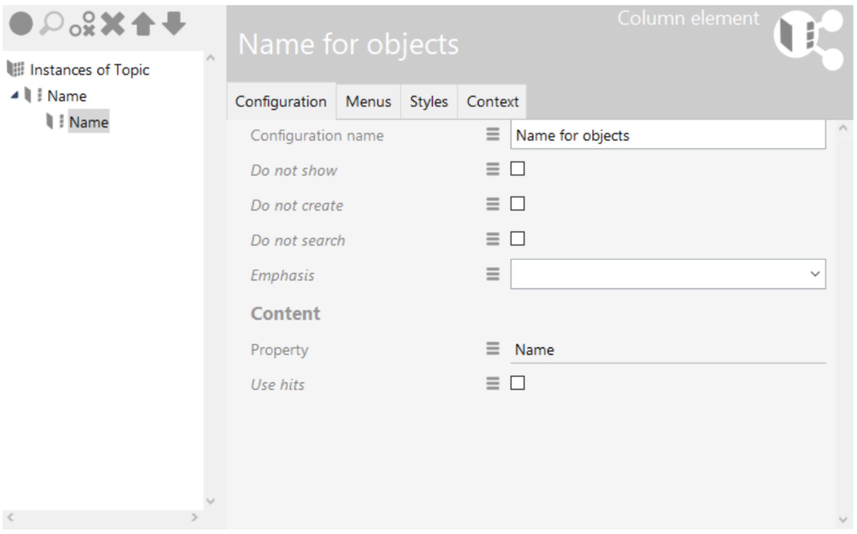
Name	**Value**
Configuration	
Configuration name	The configuration name is used for identification and reuse of the configuration.
Do not show	Use this Boolean attribute to control whether the values of the selected property should be displayed. By default all properties are displayed.
Do not create	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 find”, this has the same effect as “Hide”!
Emphasis	Here you can provide formatting specifications for the display of values; currently, the only available option is *underline*.
Property (obligatory or script)	Link to the property type that is to be displayed.
Script (obligatory or property)	Executes the *cellValues* script to determine the values to be displayed.

Quality	For the web frontend, this option displays a bar showing the hit quality incl. percentage value. Note: This option is used instead specifying the property and can only indicates a value when the option "Use hits" is activated because only hits carry a quality value.
Structured query element	A structured query can be used to determine the property.
Script	A script can be used which returns a property of an element or a hit instead of specifying the property.
Use hits	Objects are created by default. However, if you want to further process the hits in the *cellValues* script, you must switch on *Use hits*.
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.

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.

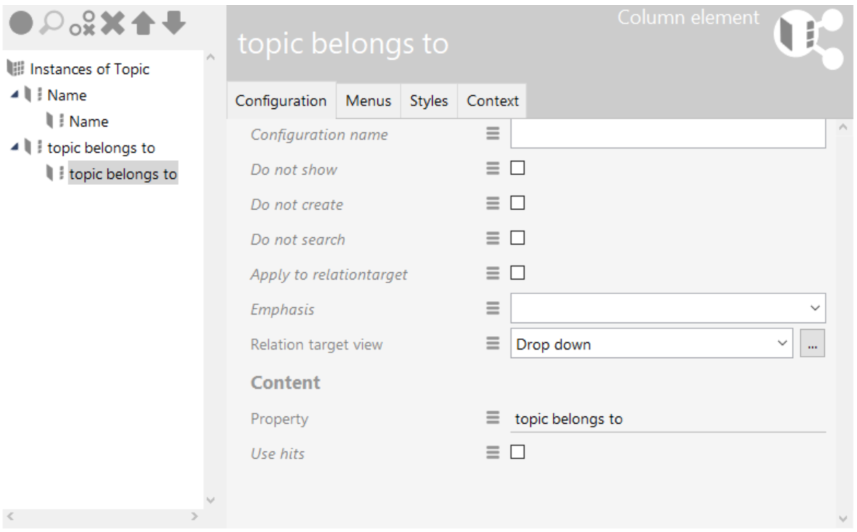
Example

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



Configuration name	Name for objects
Do not show	☐
Do not create	☐
Do not search	☐
Emphasis	
Property	Name
Use hits	☐

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


Column element

topic belongs to

Configuration | Menus | Styles | Context

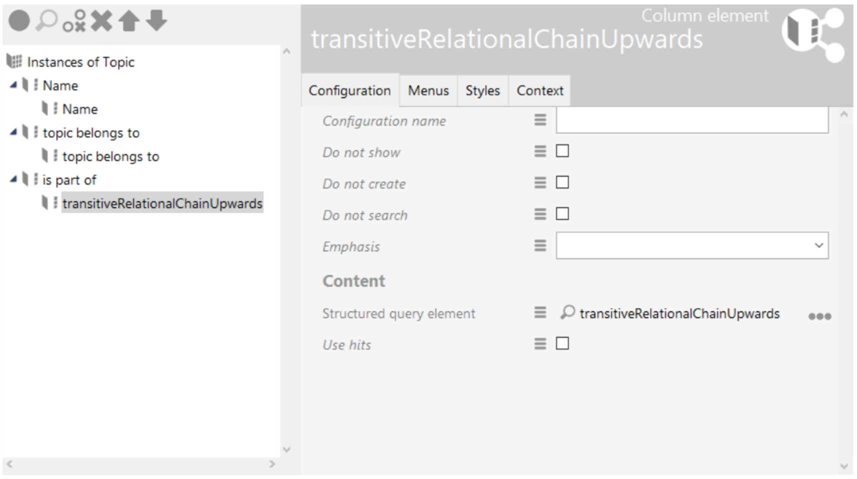
Instances of Topic
- Name
  - Name
- topic belongs to
  - topic belongs to

Configuration name:
Do not show: ☐
Do not create: ☐
Do not search: ☐
Apply to relationtarget: ☐
Emphasis:
Relation target view: Drop down

Content

Property: topic belongs to
Use hits: ☐

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


Column element

transitiveRelationalChainUpwards

Configuration | Menus | Styles | Context

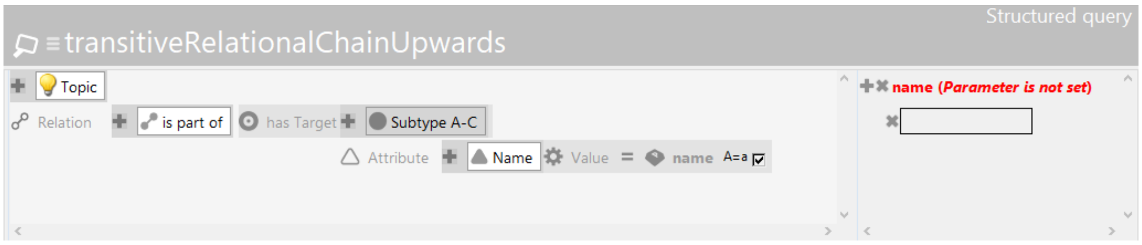
Instances of Topic
- Name
  - Name
- topic belongs to
  - topic belongs to
- is part of
  - transitiveRelationalChainUpwards

Configuration name:
Do not show: ☐
Do not create: ☐
Do not search: ☐
Emphasis:

Content

Structured query element: transitiveRelationalChainUpwards
Use hits: ☐

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).

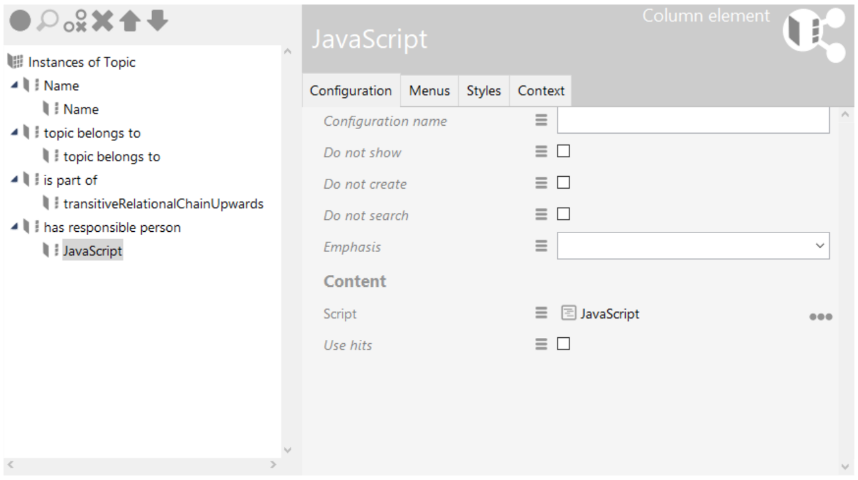
Unless further changes are made, the value displayed in the column is the value of the attribute used for filtering. If the displayed value does not result from the attribute used for filtering, there are two options:

- 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


Instances of Topic
Name
Name
topic belongs to
topic belongs to
is part of
transitiveRelationalChainUpwards
has responsible person
JavaScript

JavaScript
Column element
Configuration | Menus | Styles | Context

Configuration name:
Do not show: ☐
Do not create: ☐
Do not search: ☐
Emphasis:
Content
Script: JavaScript
Use hits: ☐

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" m
    return elements;
}

// Returns cell values for the given element function cellValues(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.

1.7.4.9 Search

The user can use the view configuration element “Search” to configure search options for the semantic network. The search can either be a predefined search with parameters, or be a search field input screen for the user.

The “search” can be selected as a sub-configuration of an *alternative* or a *group*. 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 “Search” (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 with facets, then the functional chain should be observed in the case of panel influencing.
Search field element –> Facet –> Search result.

Setting options

Name	Value
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.
Label	A label is only used if this configuration is embedded in another configuration, e.g. *Alternative*.

Table	A table configuration is specified here which is used to display the search results.
Script for label	The label can, alternatively, be determined using a script.
Script for visibility	A script can be used to control whether the configuration element should be displayed.
Script for table configuration	The table can also be determined using a script.

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.

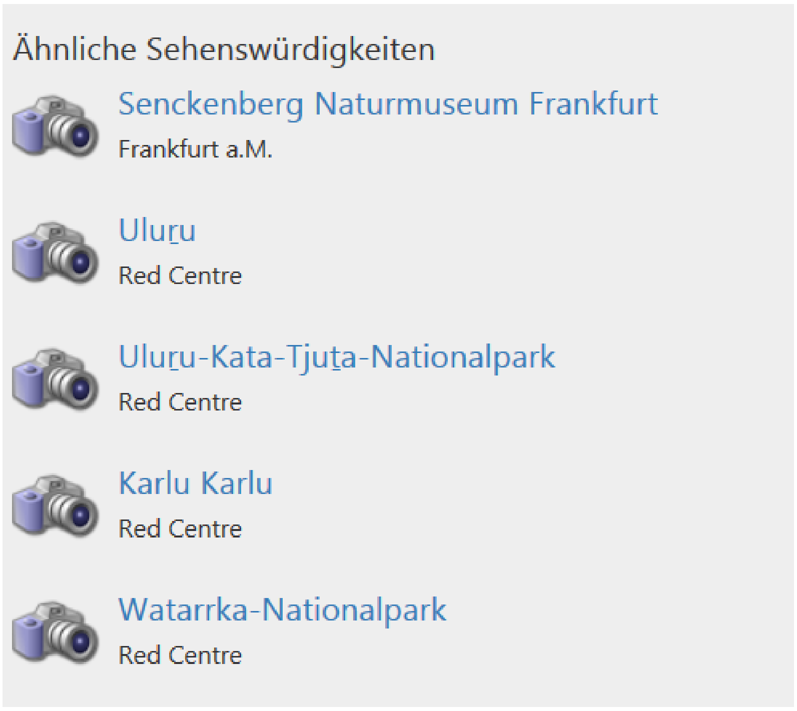
Setting options for a parameter name

The following parameters are maintained as meta properties for a *parameter name*:

Name	Value
Script	The script *parameterValue* is used for determining the search value for the specified parameter name.
Value determination	Specifies the value determination path *Script*: The value is determined from the script and must not be overwritten by the user. *Script, can be overwritten*: The script determines the value. The user may overwrite it. *User input*: No script evaluation. User input only.
Type	xsd-type
Label	During output to JSON this value ends up in *label.*

Display in an application

Search results are output in a table by default.



Ähnliche Sehenswürdigkeiten

Senckenberg Naturmuseum Frankfurt
Frankfurt a.M.

Uluṟu
Red Centre

Uluṟu-Kata-Tjuṯa-Nationalpark
Red Centre

Karlu Karlu
Red Centre

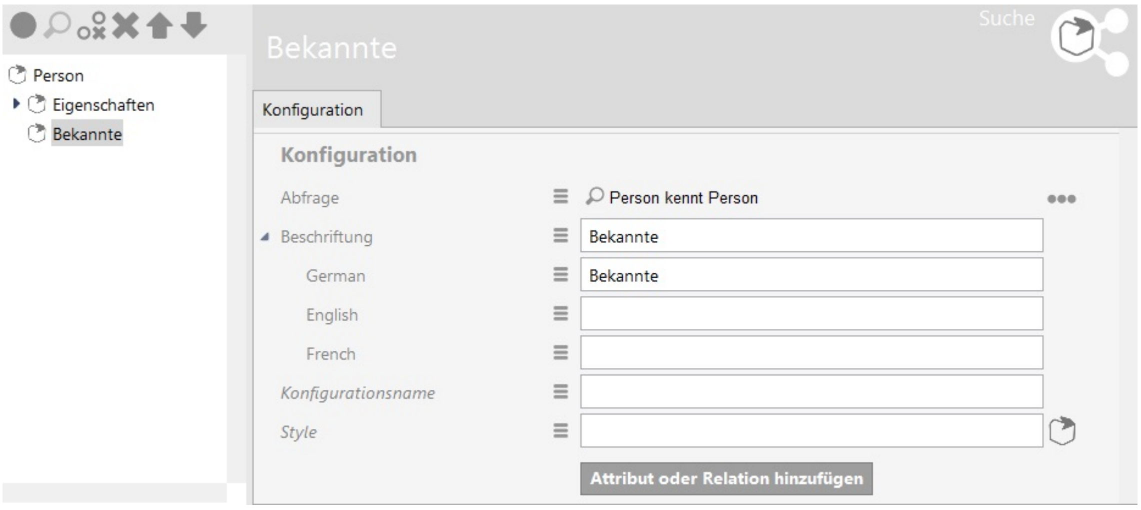
Watarrka-Nationalpark
Red Centre

In this example, search results are output 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 (in this case, the location of the point of interest).

Display in Knowledge Builder

The results of any query are always shown in an object list in the Knowledge Builder.

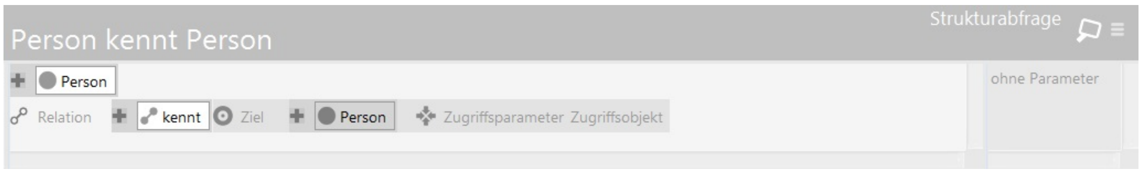
Example:



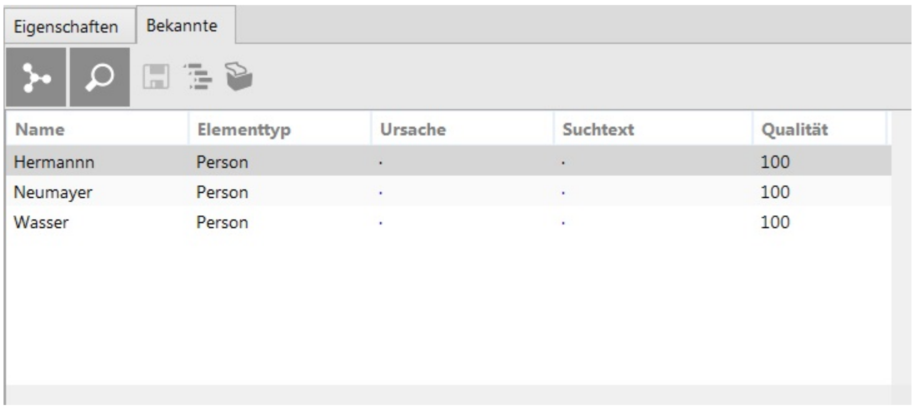
Konfiguration

Abfrage	Person kennt Person
Beschriftung	Bekannte
German	Bekannte
English	
French	
Konfigurationsname	
Style	

The "Properties" and "Friends" tabs are defined in the view configuration. "Friend" is a configuration element of the type "Search". A search can be selected, or a new one be created directly, under "Query".



Definition of the search



Name	Elementtyp	Ursache	Suchtext	Qualität
Hermannn	Person	·	·	100
Neumayer	Person	·	·	100
Wasser	Person	·	·	100

The result of the query is displayed in the "Friends" tab in the Knowledge Builder for objects of the type "Person".

Instead of the individual configuration element, searches can be split into several separate configurations.

1.7.4.10 Graph

The contents of the semantic database are plotted in a graph with their objects and connections (see *Knowledge Builder* > *Basics* > *Graph editor*).

Setting options

Name	Value
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.
Hide key	This defines whether the key for the node types is to be displayed.
Width/height	This defines the width and height of the configuration element, either as a percentage or exact to the pixel.
Script for visibility	The visibility of the configuration element can be defined in a script referenced here.
Structured query for start elements	A query that determines the objects to be displayed.

1.7.4.10.1 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
Label	A label is only used if this configuration is embedded in another configuration, e.g. an additional *alternative*.
Maximum path length	.
Steps until node is hidden	.

1.7.4.10.2 Node categroy

Node categories are subordinated to graph configurations.

They are assigned subordinate link elements.

Setting options

Name	Value
Label	A label is only used if this configuration is embedded in another configuration, e.g. an additional *alternative*.
Display extensions initially	.
Color	Color assigned to the nodes of this category
Show in key	*Always*, *If necessary* or *Never*
Node size	.
Draw icon only	.

1.7.4.10.3 Links

What exactly are links?

They are subordinate to a node category.

Setting options

Name	Value

Query for link	.
Label	A label is only used if this configuration is embedded in another configuration, e.g. an additional *alternative*.
Expand as preference	.
Color	.
Initially expanded	.
Relation	.
Script for label	Here it is possible to reference a script that determines the label.
Script for link	A script referenced here can be used to define the link.

1.7.4.11 Text

This configuration element outputs a simple text. This is either configured fixed or determined via a script.

Setting options

Name	Value
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.
Text	Text that is to be output.
Script for text	A script that returns the text to be displayed.

1.7.4.12 Image

Static graphics can be integrated with the aid of this configuration element.

Name	Value
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.
Width/height	Scales the image file to the dimensions specified.
Script for visibility	A script is used to determine whether the graphics are to be displayed.

1.7.4.13 Script generated view/HTML

Script-generated view

A view created using a script saved in the semantic network. 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
Label	A label is only used if this configuration is embedded in another configuration, e.g. *Alternative*.
Script	Script for generation of the view.
Script for visibility	Script for determining the visibility.
Script for label	Script for determining the label.
viewType	Name of the partial.

Script-generated HTML

This view configuration shows an HTML fragment that is generated using a script stored in the semantic network. 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
Label	A label is only used if this configuration is embedded in another configuration, e.g. *Alternative*.
Script	Script for generating HTML output.
Script for visibility	Script for determining the visibility.
Script for label	Script for determining the label.

Example of a script that generates simple HTML output:

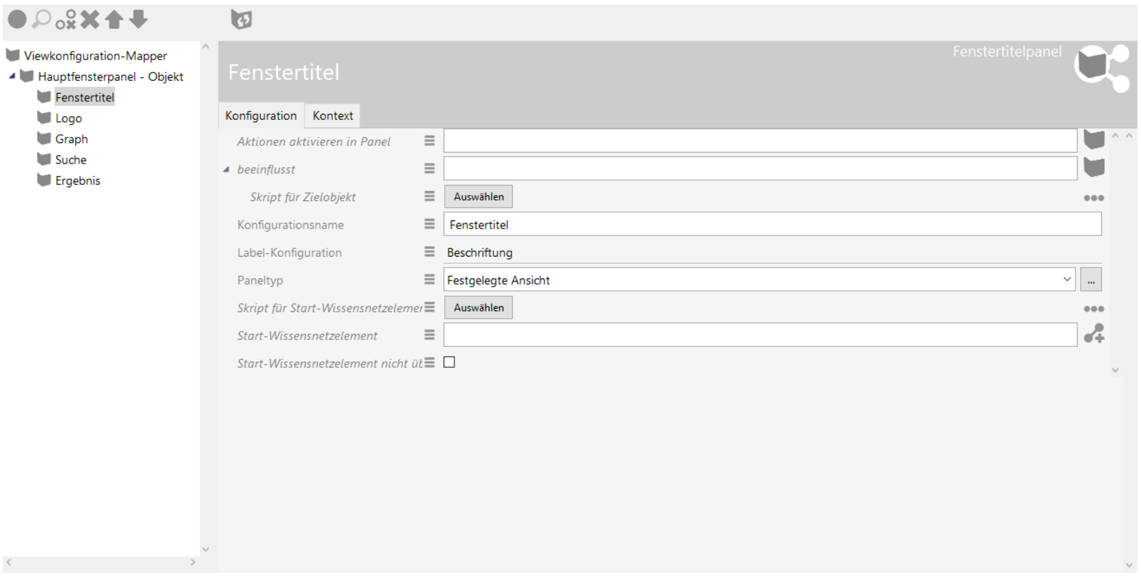
```
function render(element, document) {
  var writer = document.xmlWriter();
  writer.startElement("div");
    writer.startElement("h2");
      writer.cdata(element.name());
    writer.endElement();
  writer.endElement();
}
```

Output:

```
<div>
  <h2>Hermann</h2>
</div>
```

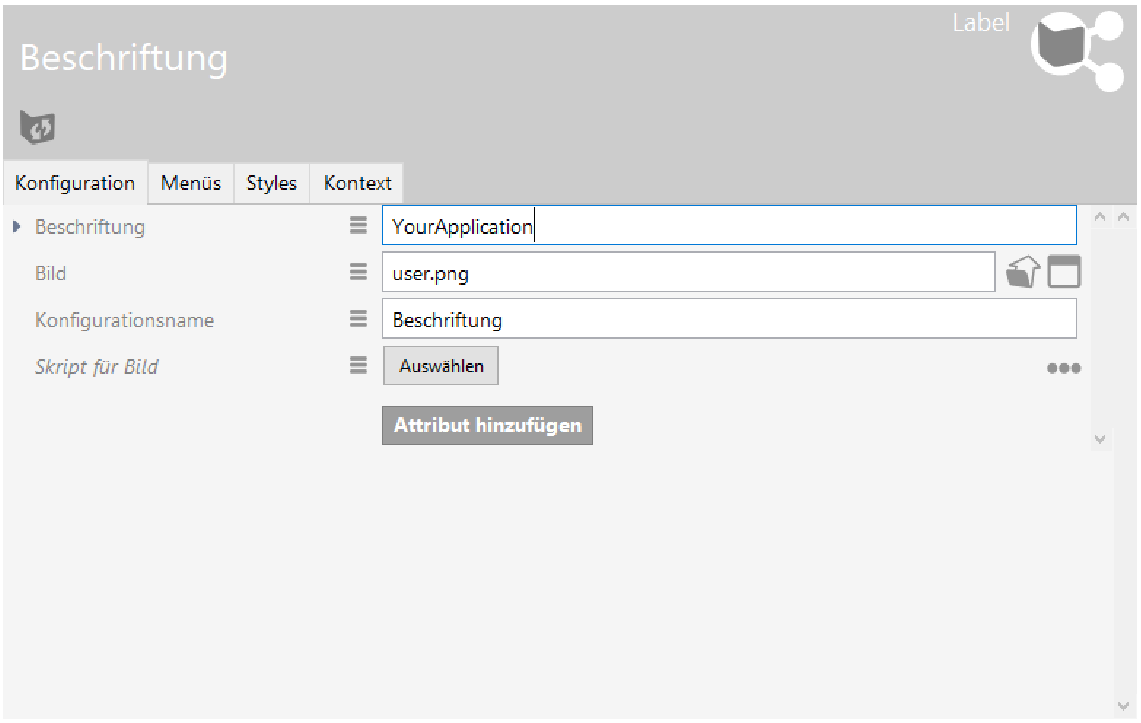
1.7.4.14 Label

The label configuration allows, for example, the labeling of a website or the labeling of a table to be configured. The label configurations are managed in the category “Subordinate configuration” in the Knowledge Builder.
Labels are used in the window title panel; this requires creation of a new object under “Label configuration:”



Aktionen aktivieren in Panel	
beeinflusst	
Skript für Zielobjekt	Auswählen
Konfigurationsname	Fenstertitel
Label-Konfiguration	Beschriftung
Paneltyp	Festgelegte Ansicht
Skript für Start-Wissensnetzelemer	Auswählen
Start-Wissensnetzelement	
Start-Wissensnetzelement nicht üt	☐

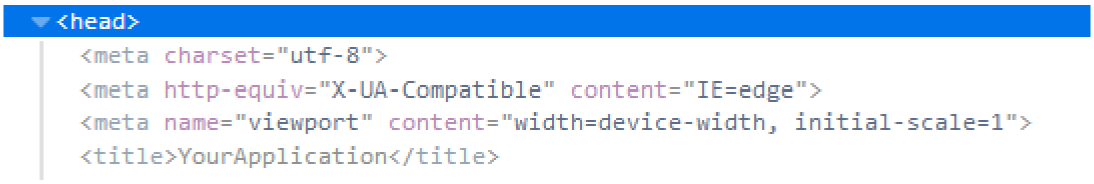
The entries can then be made under “Label” and “Image”, which later appear on the browser tab of the website:



Beschriftung	YourApplication
Bild	user.png
Konfigurationsname	Beschriftung
Skript für Bild	

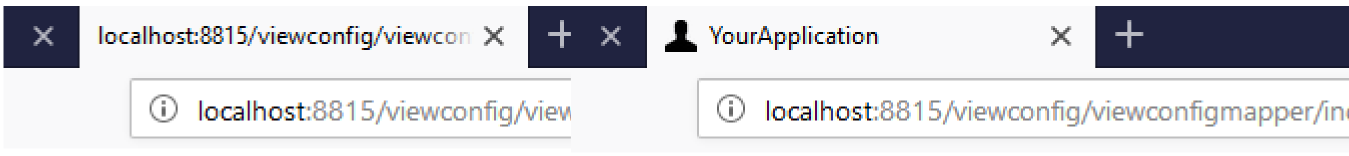
Note: The view configuration element ("Label") is titled "Label - Object" by default in the Knowledge Builder. If a string is entered under "Label", then this appears as the element name of the view configuration element. When a configuration name ("Label") is assigned, this appears as the element name. As a rule, configuration names should be assigned so that configuration elements can be found again and are easier to distinguish from each other.

In regard to the web front-end that is generated by the ViewConfiguration Mapper, this is equivalent to the <title> element in the <head> section:



```
<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/viewcon
localhost:8815/viewconfig/view
YourApplication
localhost:8815/viewconfig/viewconfigmapper/in

1.7.5 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, 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)
  - Changing layout (only one of the subordinate panels is displayed at the same time)
- 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).
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)

1.7.5.1 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:

A. The main window panel of the application is always active when an application starts

B. 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 option “Lazy” is provided for this case.

In the same way, the option “Update” can be used to avoid triggering the activation chain. In this case, only the content of the panel is calculated again.

The option “Display” is the default setting when neither of the two options described above were selected.

1.7.5.2 Layout panels

The application is divided into different areas using layout panels. Linear layouts arrange subordinate panels either next to each other or one above the other. Changing 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)

1.7.5.3 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 network.
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.

1.7.5.4 Dialog panels

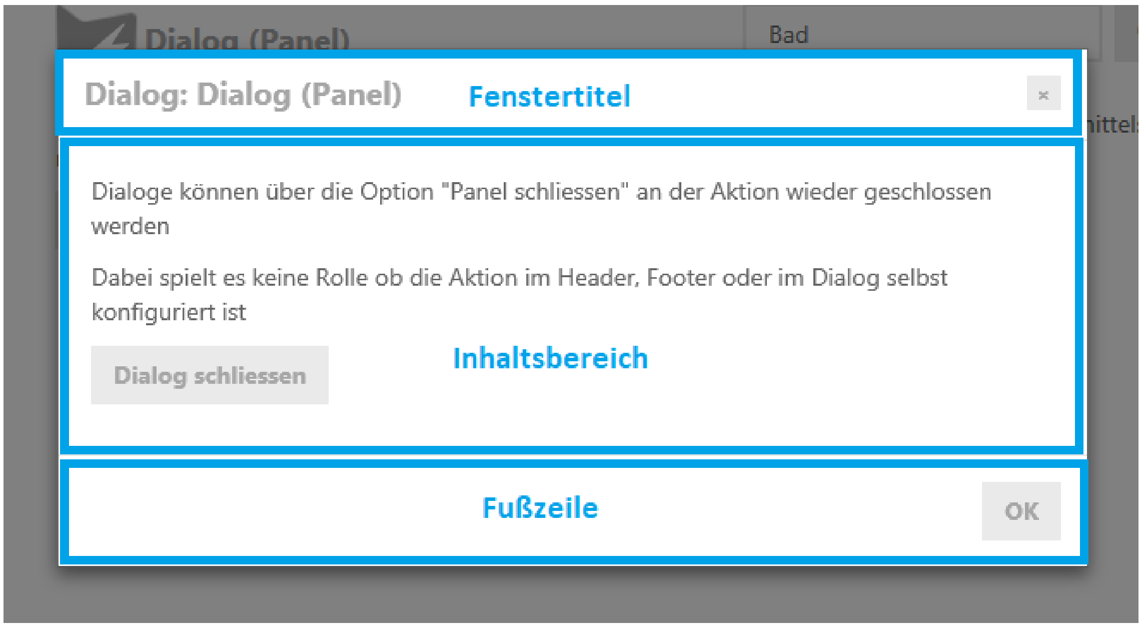
Dialog panels are special display areas whose contents are displayed in a dialog box. Dialog boxes appear automatically when the corresponding dialog panel is activated. Just like with other panels, activation is also possible via certain actions (see relation “Show result in panel” in Action configurations) 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).



Dialoge können über die Option "Panel schliessen" an der Aktion wieder geschlossen werden

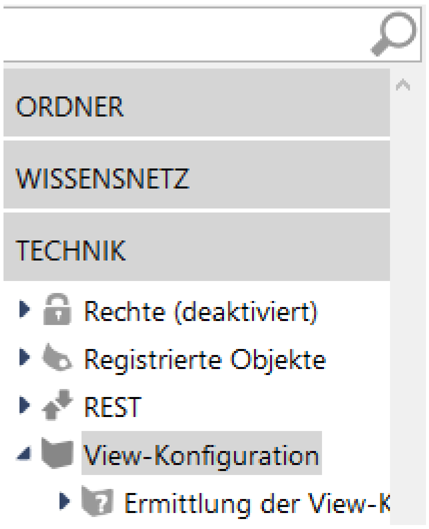
Dabei spielt es keine Rolle ob die Aktion im Header, Footer oder im Dialog selbst konfiguriert ist

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.

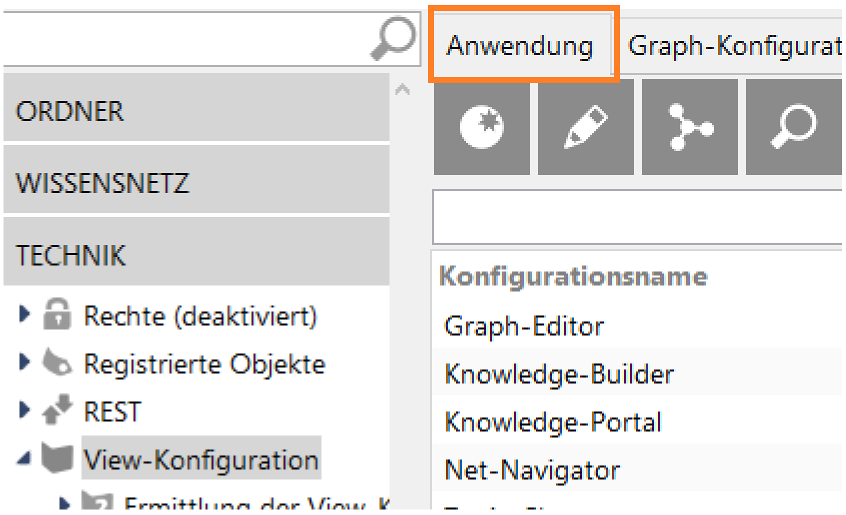
Dialog panels can be created as follows in Knowledge Builder:

1. Use a user account that has administrator rights to log on to Knowledge Builder

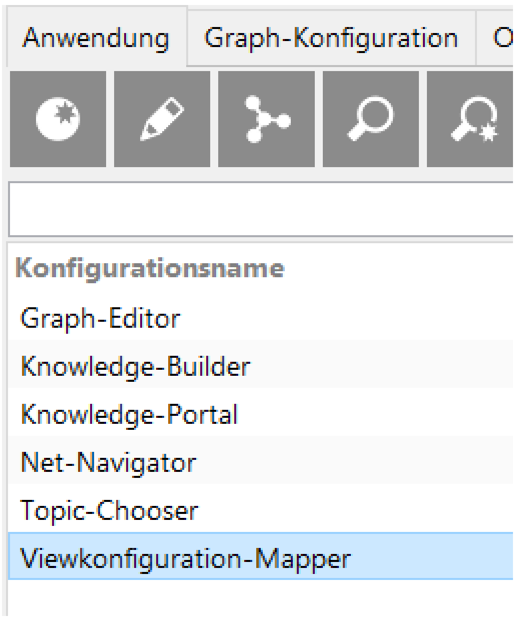
2. In the navigation area, on the left, open the “Technical” category and select the sub-item “View configuration.”



3. Select the “Application” tab on the right window.

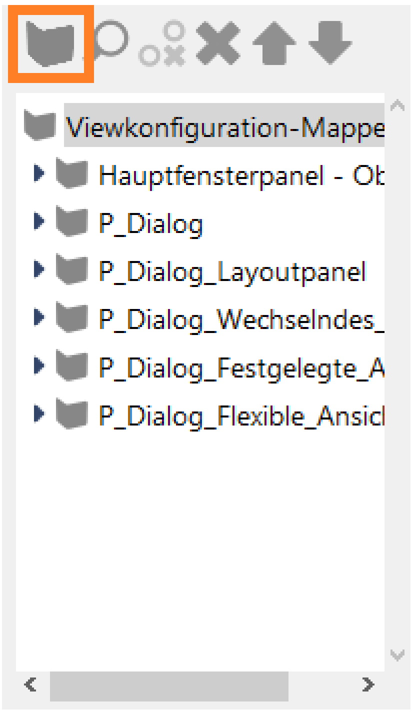


4. In the list underneath, select the application to which you would like to add the dialog panel (usually “View configuration mapper”).



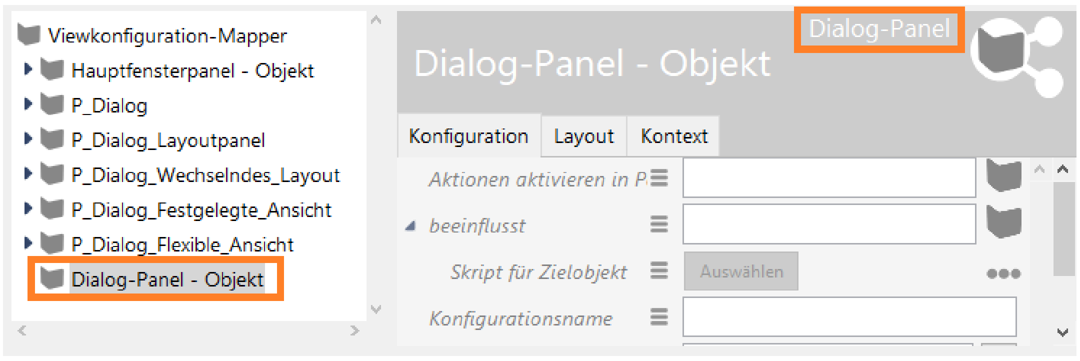
Konfigurationsname
Graph-Editor
Knowledge-Builder
Knowledge-Portal
Net-Navigator
Topic-Chooser
Viewkonfiguration-Mapper

5. Select the top element in the panel tree below and click on the Create icon



Viewkonfiguration-Mappe
Hauptfensterpanel - Ob
P_Dialog
P_Dialog_Layoutpanel
P_Dialog_Wechselndes_
P_Dialog_Festgelegte_A
P_Dialog_Flexible_Ansicl

6. 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


Viewkonfiguration-Mapper
- Hauptfensterpanel - Objekt
- P_Dialog
- P_Dialog_Layoutpanel
- P_Dialog_Wechselndes_Layout
- P_Dialog_Festgelegte_Ansicht
- P_Dialog_Flexible_Ansicht
- Dialog-Panel - Objekt

Dialog-Panel - Objekt
Dialog-Panel

Konfiguration | Layout | Kontext

Aktionen aktivieren in P
beeinflusst
Skript für Zielobjekt: Auswählen
Konfigurationsname

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.

1.7.6 Sessions

A session in the sense of the view configuration is used as temporary storage for variable values, which can be read and written to within scripts of the view configuration. This makes it possible to map the current status of an application.

Sessions form a stack. The first session also 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.

The activation of panels marked “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 using the "element()" function on this session.

Use function "$k.Session.current()” to reach the top session of the stack.

1.7.7 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.

1.7.7.1 Folder structure

The left part of the main window in Knowledge Builder is used for navigating through the semantic model. 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.

1.7.7.1.1 Default folder structure

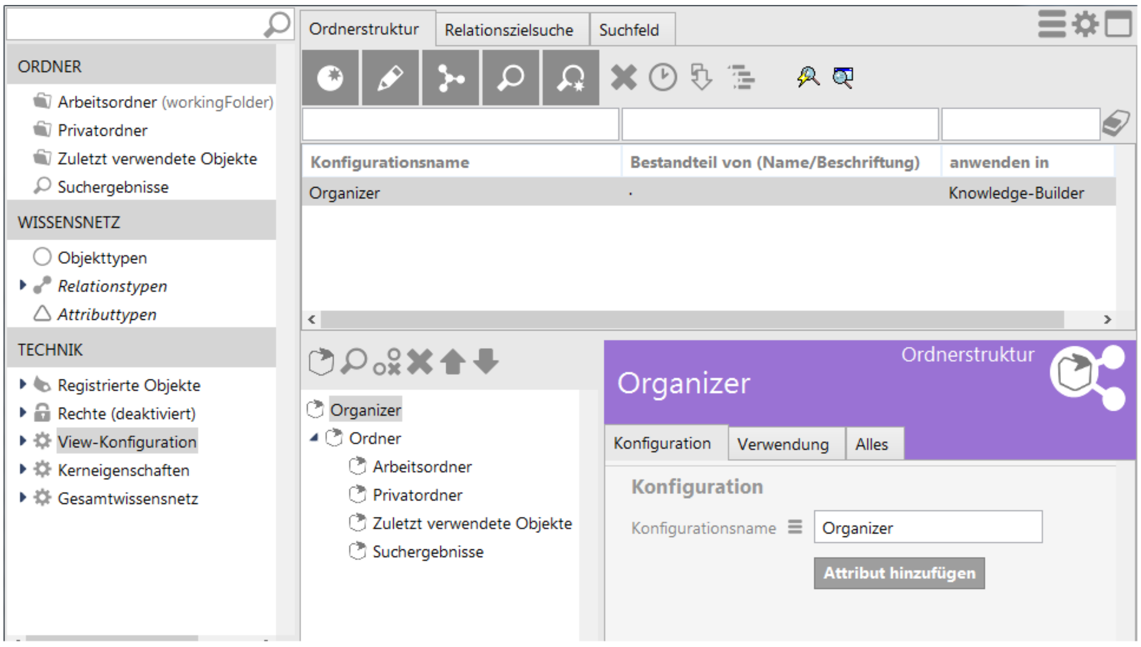
The configuration of the standard folder structure provides folders, making it possible to navigate the semantic model and store contents there. Three main areas are available to administrators.

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 “**Semantic network”** 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 semantic network. These include, among others, registered objects, the rights system and triggers.



Konfigurationsname	Bestandteil von (Name/Beschriftung)	anwenden in
Organizer	-	Knowledge-Builder

Konfiguration

Konfigurationsname: 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.

1.7.7.1.2 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 elements* 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.

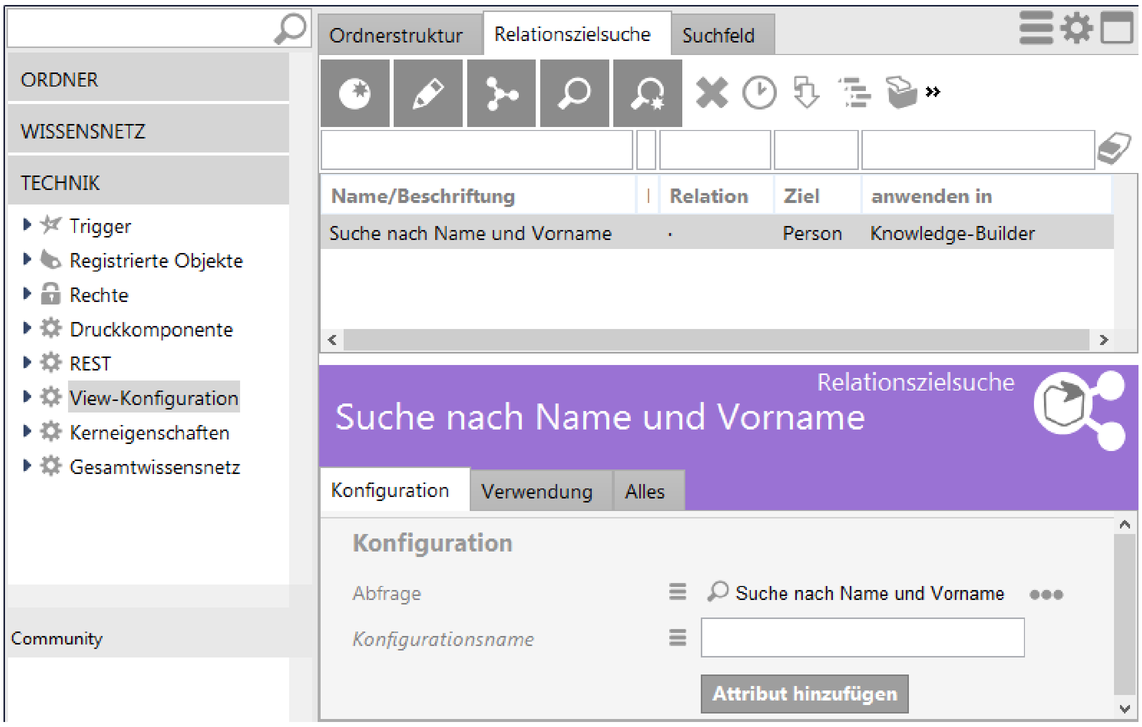
Only the configuration type *Virtual folder* can contain additional sub-configurations, and it is the only one for which sub-configurations make sense.

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.

1.7.7.2 Relations destination search

The configuration of relation targets makes it possible to influence the strategy used to search for possible relation targets. If a semantic model 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/Beschriftung	Relation	Ziel	anwenden in
Suche nach Name und Vorname	·	Person	Knowledge-Builder

Suche nach Name und Vorname

Konfiguration

Abfrage: Suche nach Name und Vorname

Konfigurationsname

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.” The properties “apply to target” and “apply in” are no longer taken into account as of version 5.2.

1.7.7.3 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

1.7.7.4 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.

1.7.7.4.1 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.

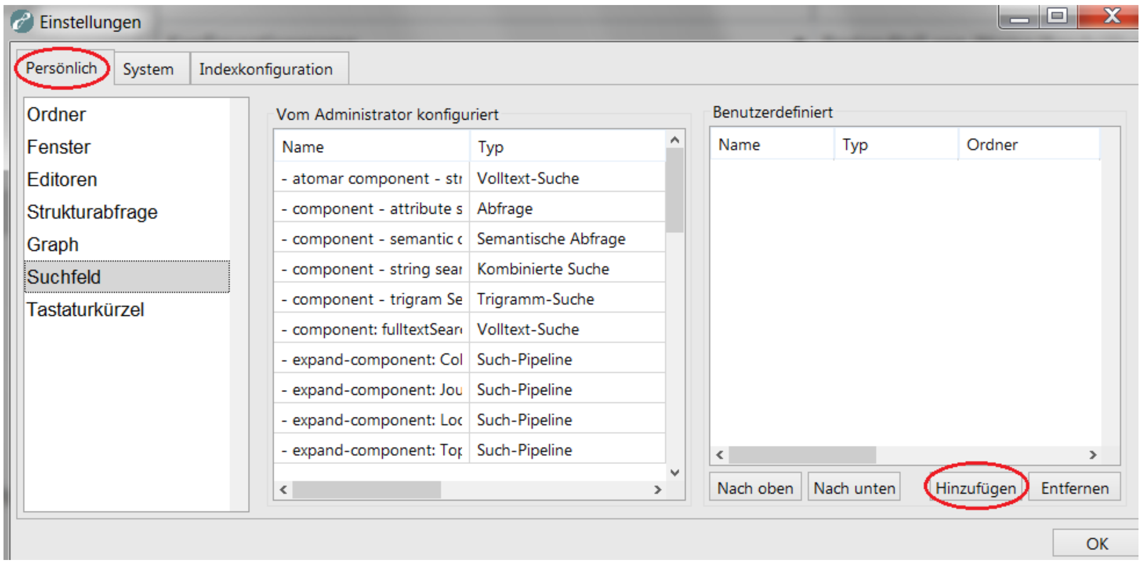
Newly created networks feature a search field configuration that is the same for all users. The administrator can expand this search field configuration to make other queries accessible to all users. Moreover, each user can add further queries to their quick search field, which are then only visible 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.

1.7.7.4.2 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	Typ
- atomar component - str	Volltext-Suche
- component - attribute s	Abfrage
- component - semantic c	Semantische Abfrage
- component - string sear	Kombinierte Suche
- component - trigram Se	Trigramm-Suche
- component: fulltextSear	Volltext-Suche
- expand-component: Col	Such-Pipeline
- expand-component: Jou	Such-Pipeline
- expand-component: Loc	Such-Pipeline
- expand-component: Top	Such-Pipeline

Name	Typ	Ordner

1.7.8 Style

The view configuration is responsible for the structural formatting of elements of the semantic model 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.

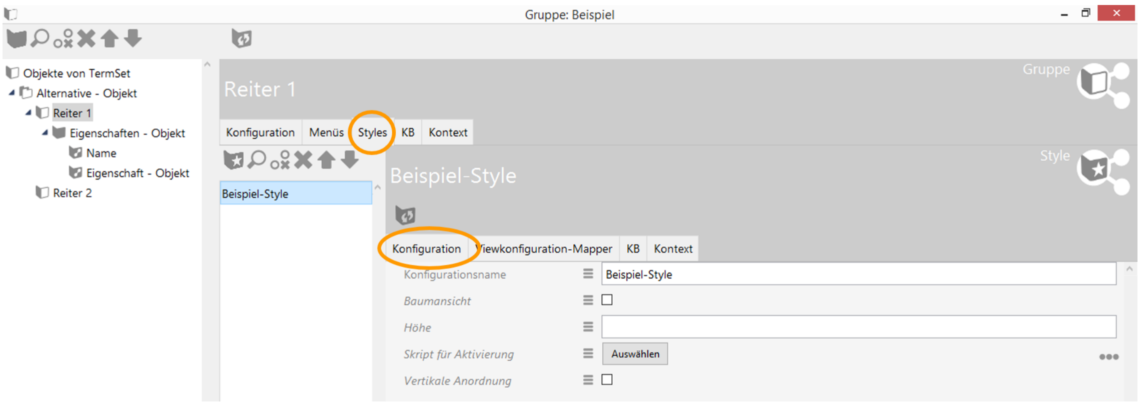
A style element can be filled with any number of style properties. The style properties are always distributed across several tabs, which are described in the sections below.

Note

Not all properties of a style make sense for all configurations. The tables of the following sections then also have 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.

1.7.8.1 Style properties in applications and in the knowledge kuilder

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.



Konfigurationsname	Beispiel-Style
Baumansicht	☐
Höhe	
Skript für Aktivierung	Auswählen
Vertikale Anordnung	☐

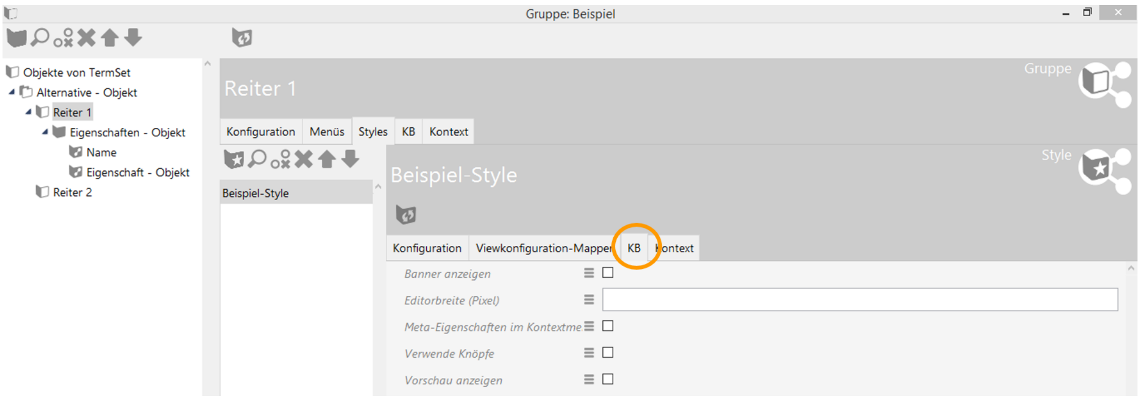
Style property	**Configuration type**	**Effect**
Tree view	Group	Show elements of the group as a tree. The default is *no*, which means the group elements are displayed next to each other or underneath each other.
Height	Property	Height in lines for string attributes
Height	Group	Height in pixels of a group element if group elements are shown next to each other.
Script for activation	All	The style can be activated in dependence on the active element by means of a script.
Vertical layout	Group	Show elements of the group next to each other. By default they are shown *underneath each other*.

1.7.8.2 Style properties in applications

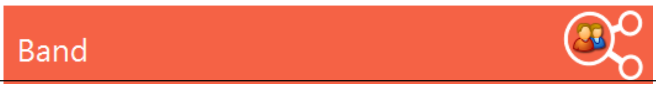
The “ViewConfiguration Mapper” tab is only displayed when the component “ViewConfiguration Mapper” has been installed. The style properties available for this component are included in the chapter Style of the ViewConfiguration Mapper (chapter 3).

1.7.8.3 Style properties in the knowledge kuilder

This chapter describes the “KB” tab of a style element, which contains the style properties used only in the Knowledge Builder.



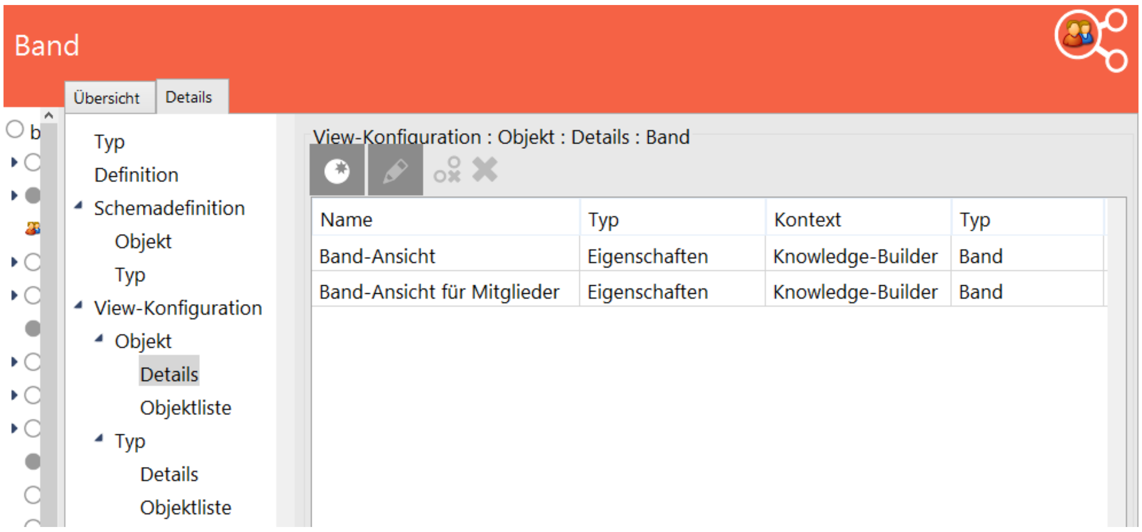
Banner anzeigen ☐
Editorbreite (Pixel)
Meta-Eigenschaften im Kontextme ☐
Verwende Knöpfe ☐
Vorschau anzeigen ☐

Style property	**Configuration type**	**Effect**
Display banner	Object configuration	Display banner with name of objects and buttons for editing. The default is *no*. 
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).
Buttons used	Property (relations only!)	
Show preview	Table	Controls whether an editor is shown underneath the table.

1.7.9 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.

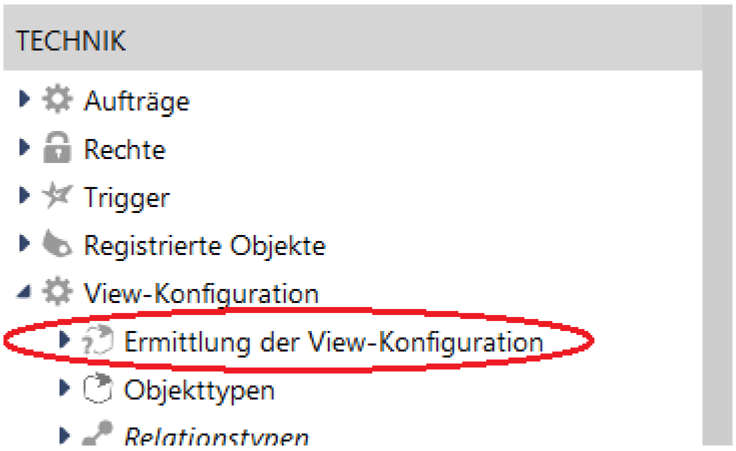


Name	Typ	Kontext	Typ
Band-Ansicht	Eigenschaften	Knowledge-Builder	Band
Band-Ansicht für Mitglieder	Eigenschaften	Knowledge-Builder	Band

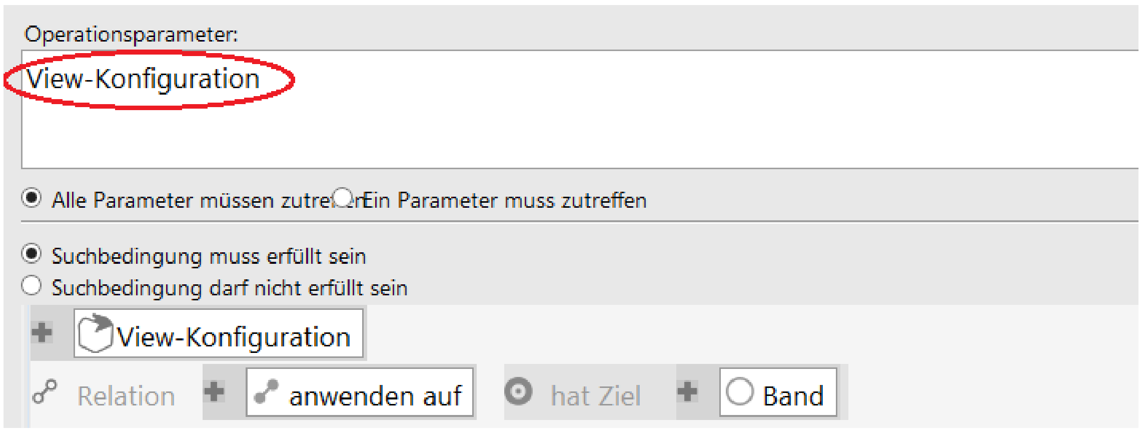
Users who are members of the band which they wish to access should see the “Band view for members”. All users who are not members of the band which they wish to access should see the “Band view”. The conditions that determine show the views are used are defined in the detector system.

Creation of a view configuration determination

The detector system is located in the folder hierarchy on the left in the “Technology” section, and has been designated as “View configuration determination” under “View configuration”.



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: “Band view for members”), 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:



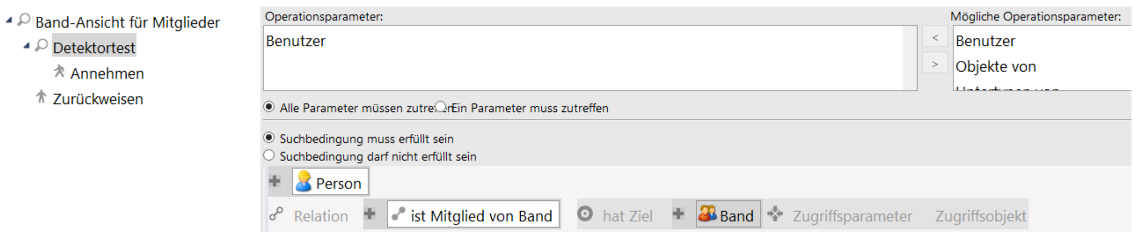
Operationsparameter:

View-Konfiguration

[✓ Alle Parameter müssen zutre] Ein Parameter muss zutreffen

[✓ Suchbedingung muss erfüllt sein]
Suchbedingung darf nicht erfüllt sein

A new query filter must now be created under the query filter that is searching for the view configuration “Band view for members” and which describes the condition for this view configuration: the view configuration “Band view for members” should only be visible to users who are members of the band that they are currently viewing. The second query filter therefore checks whether the active user is a member of the band. By clicking on , the set of search results is then permitted to view the configuration “Band view for members”. The following diagram shows the query filter for users who are members of the band that they are currently viewing and the folder hierarchy that was created so far on the left-hand side.



Operationsparameter: Benutzer

Mögliche Operationsparameter: Benutzer, Objekte von

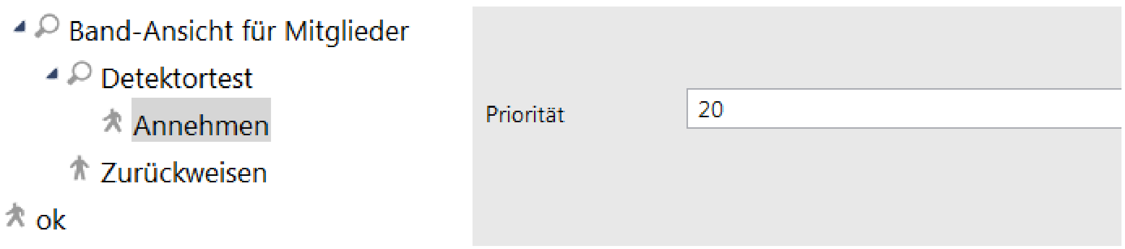
[✓ Alle Parameter müssen zutre] Ein Parameter muss zutreffen

[✓ Suchbedingung muss erfüllt sein]
Suchbedingung darf nicht erfüllt sein

The view configuration “Band view” is automatically used for those users who are not members of the band that they are currently viewing.

Weighting of the configurations in the detector system

The configurations in the detector system “View configuration determination” 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.



Priorität: 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.

1.8 JavaScript API

1.8.1 Introduction

The JavaScript-API is a server-side API for accessing a semantic net. The API is used in triggers, REST services, reports etc.

1.8.1.1 API Reference

The API reference is available here:

http://documentation.i-views.com/

1.8.1.2 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 semantic network, or

```
$k.user())
```

which returns the current user.

1.8.1.3 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.

1.8.1.4 Working with semantic elements

Semantic elements are usually retrieved from the registry or by a query.

```
// Get the person type by its internal name
var personType = $k.Registry.type("Person");


// Perform the query named "articles",
// with the query parameter "tag" set to "Sailing"
```

```
var 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"
var familyName = person.attributeValue("familyName");
// Get the target of the relation "bornIn"
var birthplace = person.relationTarget("bornIn");
```

A shortcut to access the value of the name attribute is the function name()

```
var 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. The current language of the environment is used if no language is specified.

```
var englishTitle = book.attributeValue("title", "en");
var swedishTitle = book.attributeValue("title", "swe");
var currentTitle = book.attributeValue("title");
```

1.8.1.5 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(function() {
    return $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(
    function() { return $k.Registry.type("Article").createInstance() },
    function() { 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(function() {
    return $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.

Please 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.

1.8.1.6 Modify elements

1.8.1.6.1 Create elements

```
// Create a new instance
var person = $k.Registry.type("Person").createInstance();
```

```
// Create a new type
var blogType = $k.Registry.type("CommunicationChannel").createSubtype();
blogType.setName("Blog");
```

1.8.1.6.2 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");
```

1.8.1.6.3 Add relations

A relation between two elements can be created with createRelation():

```
var places = $k.Registry.query("places").findElements({name: "Hoboken"});
if (places.length == 1)
    person.createRelation("bornIn", places[0]);
```

1.8.1.6.4 Delete elements

Any element can be deleted with the remove() function:

```
person.remove();
```

This also deletes all properties of the element.

1.8.2 Examples

1.8.2.1 Queries

Search for elements:

```
// Perform the query "articles" with parameter tag = "Soccer"
var topics = $k.Registry.query("articles").findElements({tag: "Soccer"});
for (var t in topics)
    $k.out.print(topics[t].name() + "\n");
```

Return hits. A hit wraps an element and adds a quality value (between 0 and 1) and additional metadata.

```
// Perform the query "mainSearch" with the search string "Baseball"
var hits = $k.Registry.query("mainSearch").findHits("Baseball");
hits.forEach(function(hit) {
    $k.out.print(hit.element().name() + " (" + (Math.round(hit.quality() * 100))+ "%)\n");
});
```

Convert query results to JSON:

```
var topics = $k.Registry.query("articles").findElements({tag: "Snooker"});
var jsonTopics = topics.map(function(topic) {
    return {
        name: topic.name(),
        id: topic.idNumber(),
        type: topic.type().name()
    }
});
$k.out.print(JSON.stringify(jsonTopics, undefined, "\t"));
```

1.8.2.2 Runtime generated queries

The JavaScript API also makes it possible to generate queries dynamically. Here are several examples from a film network:

Search for films by year + name

```
var 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

Year + number of directors >= 3

```
var query = new $k.StructuredQuery("imdb_film");
query.addAttributeValue("imdb_film_year", "year");
query.addCardinality(‘‘imdb_film_director", 3, ">=");
query.findElements({year: "1958"});
```

Year + name of director

```
var query = new $k.StructuredQuery("imdb_film");
query.addAttributeValue("imdb_film_year", "year", ">=");
var directorQuery = query.addRelationTarget(‘‘imdb_film_director").targetQuery();
directorQuery.addAttributeValue("name", "director");
query.findElements({year: "1950", director: "Hitchcock, Alfred"});
```

Alternatives (Or conditions)

```
var query = new $k.StructuredQuery("imdb_film");
query.addAttributeValue("imdb_film_year", "year");
var alternatives = query.addAlternativeGroup();
alternatives.addAlternative().addAttributeValue("name", "name");
alternatives.addAlternative().addAttributeValue("imdb_film_alternativeTitel", "name");
query.findElements({year: "1958", name: "Vert*"});
```

Possible operators:

Operator name	**Short term**	**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)
equalsTopicOneWay		filter with
fulltext		Contains string
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

1.8.2.3 Create and change elements

Create a person

```
// Get the person type by its internal name
var personType = $k.Registry.type("Person");
// Create a new instance
var person = personType.createInstance();
```

```
// Set attribute values
person.setAttributeValue("familyName", "Norris");
person.setAttributeValue("firstName", "Chuck");
```

Set the full name of a person

```
var familyName = person.attributeValue("familyName");
var firstName = person.attributeValue("firstName");
if (familyName && firstName)
{
    var fullName = familyName + ", " + firstName;
    person.setAttributeValue("fullName", fullName);
}
```

Set the value of an attribute

```
// Boolean attribute
topic.setAttributeValue("hasKeycard", true);

// Choice attribute
// - internal name
topic.setAttributeValue("status", "confirmed");
// - choice object
var choiceRange = $k.Registry.attributeType("status").valueRange();
var choice = choiceRange.choiceInternalNamed("confirmed");
topic.setAttributeValue("status", choice);

// Color attribute
topic.setAttributeValue("hairColor", "723F10");

// Date / Time / DateAndTime attribute
topic.setAttributeValue("dateOfBirth", new Date(1984, 5, 4));
topic.setAttributeValue("lastModification", new Date());
topic.setAttributeValue("teatime", new Date(0, 0, 0, 15, 30, 0));

// FlexTime attribute
// - $k.FlexTime (allows imprecise values)
topic.setAttributeValue("start", new $k.FlexTime(1984, 6));
// - Date (missing values are set to default values)
topic.setAttributeValue("start", new Date(1984, 5, 3));

// Number (integer / float) attribute
topic.setAttributeValue("weight", 73);

// Interval
topic.setAttributeValue("interval", new $k.Interval(2, 4));

// String attribute
// - untranslated
topic.setAttributeValue("familyName", "Norris");
// - translated (language is an ISO 639-1 or 639-2b code)
topic.setAttributeValue("welcomeMessage", "Welcome", "en");
topic.setAttributeValue("welcomeMessage", "Bienvenue", "fre");
```

Create a new attribute

```
person.createAttribute("nickName", "Ground Chuck");
```

Create a new relation

```
var places = $k.Registry.query("places").findElements({name: "Oklahoma"});
if (places.length == 1)
    person.createRelation("bornIn", places[0]);
```

Delete an element, including its properties

```
person.remove()
```

Convert a string to an attribute valu. 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.

```
var statusRange = $k.Registry.type("status").valueRange();
var statusConfirmed = statusRange.parse("Confirmed", "eng");
```

Set change metadata

```
topic.setAttributeValue("lastChangeDate", new Date());
var userInstance = $k.user().instance();
// Ensure that a single relation to the user instance exists
if (topic.relationTarget("lastChangedBy") !== userInstance)
{
    var relations = topic.relations("lastChangedBy");
    for (var r in relations)
        relations[r].remove();
    topic.createRelation("lastChangedBy", userInstance);
}
```

1.8.2.4 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");
}
```

Restlet that returns a blob

```
function respond(request, parameters, response)
```

```
{
    var name = parameters["name"];
    if (name)
    {
        var 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");
    }
}
```

Restlet that creates an instance with an uploaded blob

```
function respond(request, parameters, response)
{
    var formData = request.formData();
    var name = formData.name;
    var picture = formData.picture;
    if (name && picture)
    {
        var 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");
    }
}
```

1.8.2.5 XML

Transforms query results into XML elements

```
function respond(request, parameters, response)
{
    var name = parameters["name"];
    if (name)
    {
        // Find points of interest
        var topics = $k.Registry.query("rest.poi").findElements({name: name});
        // Write XML
        var document = new $k.TextDocument();
        var writer = document.xmlWriter();
        writer.startElement("result");
        for (var t in topics)
        {
            writer.startElement("poi");
            writer.attribute("name", topics[t].name());
            writer.endElement();
        }
        writer.endElement();
        response.setContents(document);
        response.setContentType("application/xml");
    }
    else
    {
        response.setCode($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>
```

Use qualified names

```
var document = new $k.TextDocument();
var 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>
```

Define a default namespace

```
var document = new $k.TextDocument();
var writer = $k.out.xmlWriter();
writer.startElement("root");
writer.defaultNamespace("http://www.i-views.de/kinfinity");
writer.startElement("child").endElement();
writer.endElement();
```

XML

```
<root xmlns="http://www.i-views.de/kinfinity">
    <child/>
</root>
```

1.8.2.6 HTTP client

Load a picture via HTTP and store it as a blob

```
var http = new $k.HttpConnection();
var imageUrl = "http://upload.wikimedia.org/wikipedia/commons/e/e7/2007-07-06_GreatBriain_Portree.
var imageResponse = http.getResponse(new $k.HttpRequest(imageUrl));
if (imageResponse && imageResponse.code() == $K.HttpResponse.OK )
{
    var portree = $k.Registry.type("City").createInstance();
    portree.setAttributeValue("image", imageResponse);
    portree.setName("Portree");
}
```

Update the weather report of all cities

```
var instances = $k.Registry.type("City").instances();
var http = new $k.HttpConnection();
for (var i in instances)
{
    var city = instances[i];
    var weatherUrl = "http://api.openweathermap.org/data/2.5/weather";
    var weatherRequest = new $k.HttpRequest(weatherUrl);
    weatherRequest.setQueryData({q: city.name()});
    try {
        var weatherResponse = http.getResponse(weatherRequest);
        if (weatherResponse.code() == $k.HttpResponse.OK)
        {
            var json = JSON.parse(weatherResponse.text());
            var weather = json.weather[0].description;
            city.setAttributeValue("weather", weather);
        }
    } catch (e) {
    }
}
```

1.8.2.7 Send e-mails

E-mails can be sent with the MailMessage object. To do so, an SMTP server must be configured in the network (Settings -> System -> SMTP).

```
var 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.

1.8.2.8 Views

JSON structures can also be generated using the view configuration; this is possible for individual objects as well as for lists of objects.

In the most straightforward case, an object is converted to JSON using the standard configuration without additional context:

```
var data = element.renderJSON();
```

All structures defined by the configuration are then converted to JSON:

```
{
    "viewType" : "fieldSet",
    "label" : ‘‘Bern",
    "elementType" : "instance",
    "modNum" : 26,
    "elementId" : "ID17361_141538476",
    "type" : {
        "elementType" : "instance",
        "typeId" : "ID10336_319205877",
        "internalName" : "City",
        "typeName" : ‘‘City"
    },
    "properties" : [{
            "values" : [{
                    "value" : "Bern",
                    "propertyId" : "ID17361_137824032"
                }
            ],
            "schema" : {
                "label" : "Name",
                "elementType" : "attribute",
                "internalName" : "name",
                "maxOccurrences" : 1,
                "attributeType" : "string",
                "viewId" : "ID20838_426818557",
                "typeId" : "ID4900_317193164",
                "minOccurrences" : 0
            }
        }, {
            "values" : [{
                    "typeId" : "ID4900_79689320"
                }
            ],
            "schema" : {
                "label" : ‘‘AlternativeName/Synonym",
                "elementType" : "attribute",
                "internalName" : "alternativeName",
                "attributeType" : "string",
                "rdf-id" : "alternativeName",
                "viewId" : "ID20839_64952366",
                "typeId" : "ID4900_79689320",
                "minOccurrences" : 0
            }
        }, {
            "values" : [{
                    "target" : {
                        "typeId" : "ID10336_493550611",
                        "label" : ‘‘Museum of Fine Arts Bern",
                        "elementId" : "ID17362_205182965"
                    },
                    "propertyId" : "ID17361_395925739"
                }, {
                    "target" : {
                        "typeId" : "ID10336_493550611",
                        "label" : ‘‘Swiss National Library",
                        "elementId" : "ID20401_126870015"
                    },
                    "propertyId" : "ID17361_9264966"
                }
            ],
            "schema" : {
                "targetDomains" : [{
                        "elementType" : "instance",
                        "typeId" : "ID10336_493550611",
                        "internalName" : "point_of_interest",
                        "typeName" : ‘‘Point of interest"
                    }
                ],
                "label" : ‘‘contains point of interest",
                "elementType" : "relation",
                "internalName" : "contains_poi",
                "viewId" : "ID20840_182208894",
                "typeId" : "ID2052_332207092",
                "minOccurrences" : 0
            }
        }
    ]
}
```

You can also define a context in the form of an application or configuration object. A suitable configuration for this context is then selected. The application “Android” is specified in the following example:

```
var application =  $k.Registry.elementAtValue("viewconfig.configurationName", "Android");
var data = element.renderJSON(application);
```

However, it is also possible to specify a configuration and let this configuration convert the element. To do so, you generate a $k.ViewConfiguration from the configuration object.

```
var configurationElement = $k.Registry.elementAtValue("viewconfig.configurationName", ‘‘Android Art
var data = $k.ViewConfiguration.from(configurationElement).renderJSON(element);
```

Since the JSON structure is rather extensive, you can also leave out certain properties in the conversion by specifying the keys as additional parameters:

```
var application =  $k.Registry.elementAtValue("viewconfig.configurationName", "Android");
var data = element.renderJSON(application, ["rdf-id", "viewId", "typeId", "propertyId", "modNum",
```

```
{
    "viewType": "fieldSet",
    "label": "Bern",
    "elementType": "instance",
    "elementId": "ID17361_141538476",
    "type": {
        "elementType": "instance",
        "internalName": "City",
        "typeName" : ‘‘City"
    },
    "properties": [
        {
            "values": [
                {
                    "value": "Bern"
                }
            ],
            "schema": {
                "elementType": "attribute",
                "label": "Name",
                "internalName": "name",
                "attributeType": "string",
                "maxOccurrences": 1
            }
        },
        {
            "schema": {
                "elementType": "attribute",
                "label": ‘‘AlternativeName/Synonym",
                "internalName": "alternativeName",
                "attributeType": "string"
            }
        },
        {
            "values": [
                {
                    "target": {
                        "label": ‘‘Museum of Fine Arts Bern",
                        "elementId": "ID17362_205182965"
                    }
                },
                {
                    "target": {
                        "label": ‘‘Swiss national library",
                        "elementId": "ID20401_126870015"
                    }
                }
            ],
            "schema": {
                "elementType": "relation",
                "targetDomains": [
                    {
                        "elementType": "instance",
                        "internalName": "point_of_interest",
                        "typeName" : ‘‘Point of interest"
                    }
                ],
                "label": ‘‘contains point of interest",
                "internalName": "contains_poi"
            }
        }
    ]
}
```

1.8.2.9 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
    var templateId = parameters["templateId"];
    var templateTopic = $k.Registry.elementAtValue("template.id", templateId);
    var templateText = templateTopic.attributeValue("template.file").text("utf-8");

    // Find elements
    var topics = $k.Registry.query("rest.articles").findElements(parameters);

    // Prepare template parameters
    var topicsData = topics.map(function(topic) {
                   return {
                      name: topic.name(),
                      id: topic.idNumber(),
                      type: topic.type().name()
                   }
                })
    var templateParameters = {
        topics: topicsData
    };

    // Render with Mustache
    var output = Mustache.render(templateText, templateParameters);

    // Return the rendered document
    response.setText(output);
    response.setContentType(templateTopic.relationTarget("template.contentType").name());
}
```

1.8.2.10 Java native interface

Java can be accessed via JNI (Java Native Interface).

Caution: 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.

```
[JNI]
classPath=tika\tika-app-1.5.jar
libraryPath=C:\Program Files\Java\jre7\bin\server\jvm.dll
```

Basic example

```
// Import the StringBuilder class, without namespace
$jni.use(["java.lang.StringBuilder"], false);
// Create a new instance
var builder = new StringBuilder();
// Javascript primitives and Strings are automatically converted
builder.append("Welcome to ");
builder.append($k.volume());
// toJS() converts Java objects to Javascript objects
$k.out.print(builder.toString().toJS());
```

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
var blob = $k.Registry.elementAtValue("uuid", "f36db9ef-35b1-48c1-9f23-1e10288fddf6").attributeVal
// Blobs have to be explicitely converted to Java byte arrays
var bufferedInputStream = new BufferedInputStream(new ByteArrayInputStream($jni.toJava(blob)));
// Parse the blob
try {
    var parser = new AutoDetectParser();
    var writer = new StringWriter();
    var metaData = new Metadata();
    parser.parse(bufferedInputStream, new BodyContentHandler(writer), metaData, new ParseContext()
    var string = writer.toString().toJS();
    // Print extracted metadata
    var metaNames = metaData.names().toJS().sort(
        function(a,b) { return a.localeCompare(b) });
    for (n in metaNames)
        $k.out.print(metaNames[n] + " = " + metaData.get(metaNames[n])).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();
}
```

1.8.3 Modules

1.8.3.1 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 topics
     */
    jsonify: function(topics) {
        return topics.map(function(topic) {
            return {
                name: topic.name(),
                id: topic.idString(),
                type: topic.type().name()
            };
        });
    }
});
```

define() allows to specify dependencies from other modules. The following script defines a module that uses another module.

```
$k.define(["rest.common"], function(common) {
    return {
        stringify: function(topics) {
            return JSON.stringify(common.jsonify(topics), undefined, "\t")
        }
    }});
```

1.8.3.2 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 ojects. require() returns the return value of the callback function

```
var topics = $k.Registry.query("rest.poi").findElements({name: "Madrid"});
var json = $k.require(["rest.common"], function(common) {
     return common.jsonify(topics);
});
$k.out.print(JSON.stringify(json, undefined, "\t"));
```

module() expects the name of a module and returns the module object.

```
var json = $k.module("rest.common").renderTopics(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.

1.8.3.3 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.

1.8.4 Debugger

1.8.4.1 Debugging in the knowledge kuilder

The following wrapper script is an example for testing a restlet in the Knowledge Builder. It can be defined in the script editor on the "Execute Script" tab as "Additional test script".

```
// Prepare request and response
var testRequest = new $k.HttpRequest("http://localhost");
var testResponse = new $k.HttpResponse();
// Call the restlet function respond() with the testbench parameters,
// which are available as a property named $k.testbenchParameters
respond(testRequest, $k.testbenchParameters, testResponse);
// Print the response header and contents
$k.out.print(testResponse.debugString());
```

Breakpoints can be set on the tab "Debug".

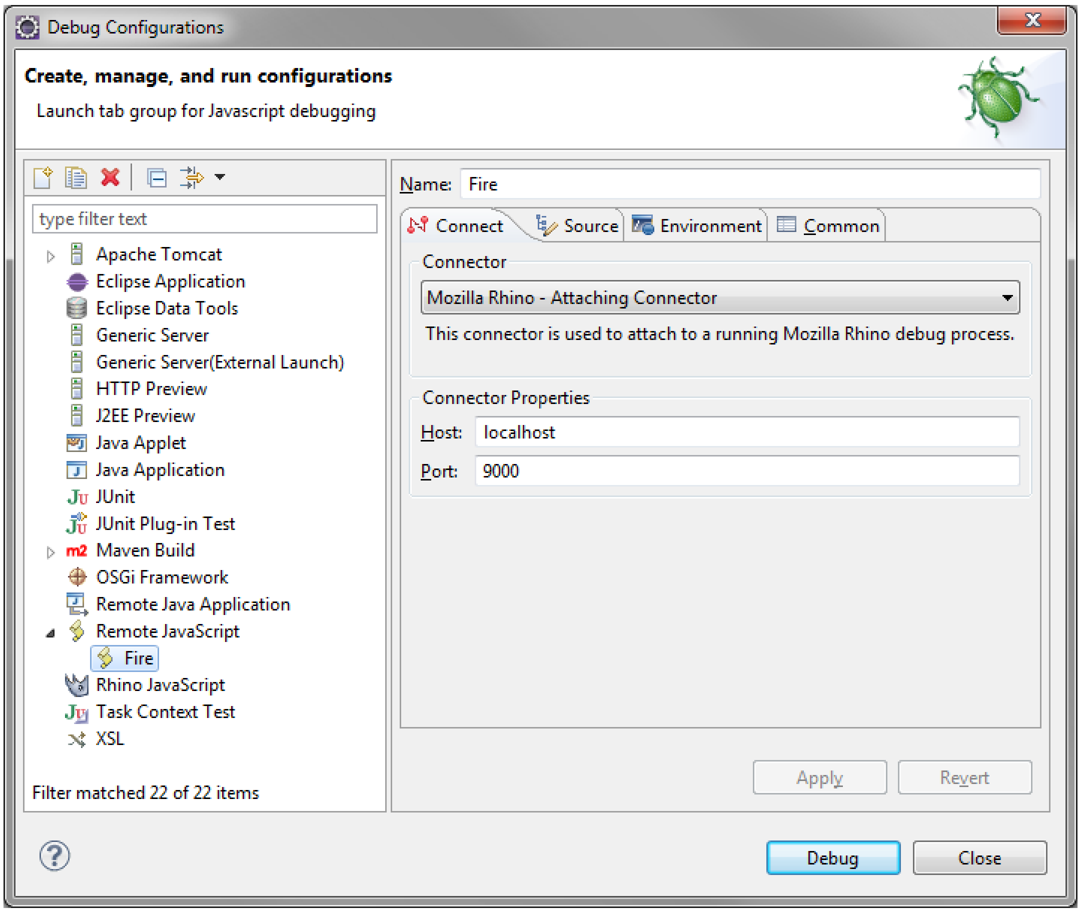
1.8.4.2 Remote Debugger

Restlet scripts can be also be debugged with a remote debugger.

1. Enable a debug port in the REST bridge .ini file

   ```
   [KRemoteDebuggerBridge]
   port=9000
   ```

2. Install Eclipse and JavaScript Development Tools
3. Create a JavaScript debug configuration in Eclipse with a Mozilla Rhino connector



Debug Configurations

Create, manage, and run configurations

Launch tab group for Javascript debugging

Name: Fire

Connect | Source | Environment | Common

Connector

Mozilla Rhino - Attaching Connector

This connector is used to attach to a running Mozilla Rhino debug process.

Connector Properties

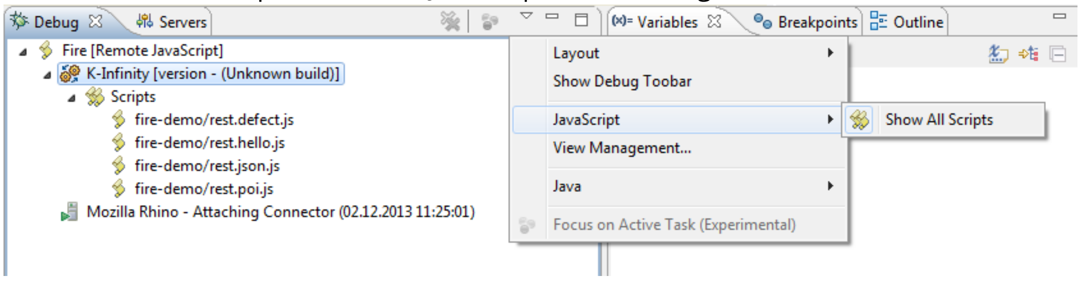
Host: localhost

Port: 9000

Apache Tomcat
Eclipse Application
Eclipse Data Tools
Generic Server
Generic Server(External Launch)
HTTP Preview
J2EE Preview
Java Applet
Java Application
JUnit
JUnit Plug-in Test
Maven Build
OSGi Framework
Remote Java Application
Remote JavaScript
Fire
Rhino JavaScript
Task Context Test
XSL

Filter matched 22 of 22 items

4. Start debugging
5. Enable "Show all scripts" to see all Javascripts of the bridge



Fire [Remote JavaScript]
K-Infinity [version - (Unknown build)]
Scripts
fire-demo/rest.defect.js
fire-demo/rest.hello.js
fire-demo/rest.json.js
fire-demo/rest.poi.js
Mozilla Rhino - Attaching Connector (02.12.2013 11:25:01)

6. Select "Open source" to view the source of a script (double click does not seem to work)
7. Set breakpoints in the source code

1.8.5 API extensions

1.8.5.1 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(function(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());
    var inverse = this.inverseRelationType();
    if (inverse)
    {    var inverseDomains = inverse.domains();
        if (inverseDomains.length > 0 )
        {
            $k.out.print(" (Relation to ");
            var 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) {
    var definedProperties = this.definedProperties();
    if (definedProperties.length > 0)
    {
        $k.out.print(label + "\n");
        definedProperties.sort(function(p1, p2) { return p1.name().localeCompare(p2.name()) });
        definedProperties.forEach(function(propertyType) {
            propertyType.logPropertySchema();
        });
    }
}
```

```
// Print the entire schema
$k.rootType().printSchema();
```

1.8.5.2 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, protoype).

Example: A basket prototype

```
// Define a Basket prototype with a function totalPrice()
function Basket() { }

Basket.prototype.totalPrice = function() {
    return this.relationTargets("contains").reduce(
        function(sum, item) {
            return 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
var baskets =  $k.Registry.type("Basket").instances();
for (var b in baskets)
    $k.out.print(baskets[b].totalPrice() + "\n");
```

For using within other scripts, the module needs to be loaded first:

```
$k.module('myBasketSkript');
var basket = $k.Registry().elementWithID('ID_123');
$k.out.print(basket.totalPrice() + "\n");
```

1.9 REST services

The REST interface can be used for read and write access to the semantic network. To do so, you define **resources** (which describe the interface behavior when accessing a resource) in the semantic network and **services** (which summarize several resources).

The behavior of a resource is controlled using scripts. In addition, predefined resources may also be used.

Access takes place via HTTP requests that are structured according to the pattern

```
https://<hostname>:<port>/<service-id>/<resource-path-and-parameter>
```

1.9.1 Configuration

The REST components must be added in the semantic network. These define 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 network. 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 semantic network do not automatically affect interfaces that are already running. This occurs 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 semantic network (volume) and the REST service ID is entered in this. The line with “services” can be omitted entirely, and the resources of all existing service objects are then automatically activated.

```
[Default]
host=localhost
loglevel=10

[KHTTPRestBridge]
volume=demo
port=8086
services=core,extra
```

1.9.2 Services

Services combine several resources. Resources may be featured by several services.

The service editor in the Knowledge Builder shows the resources in its structure view. A new resource is created using “Link new” and is added to the service. A resource that has already been defined is added to the service using “Link existing”.

1.9.3 Resources

Resources describe the response in the event of an HTTP prompt at 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	Delivers 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

1.9.3.1 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.

Method	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 enquiry.
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 semantic network 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.

1.9.3.2 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)
{
    var albums = $k.Registry.query("albums").findElements(parameters);
    var albumData = albums.map(function(album) {
        return {
            name: album.name(),
            id: album.idString(),
        }});
    response.setText(JSON.stringify(albumData, undefined, "\t"));
    response.setContentType("application/json");
}
```

You could use this script, for example, in the resource

```
albums/{genre}/{year}
```

and use the search parameters “genre” and “year” as the search conditions in the “albums” query.

1.9.3.3 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	**Methoc**	**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).
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 Re-sourceWithFilename	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 supplement 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.

BlobResource

This integrated 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.

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 semantic network.

http://{server:port}/baseService/topicIcon/{topicID}?size=10

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.

Setting the starting set of the list via KPath

The optional query parameters "**elementsPath**" and "**startTopic**" can be used to calculate the initial elements of the list. If these parameters are not set, the initial set consists of all individuals or all subtypes (in case of a type object list) of the type specified via "conceptLocator."

Here "elementsPath" is a KPath expression and "startTopic" is a reference to the topic with which the evaluation of the KPath is to be started. The form of the "startTopic" parameters corresponds to that of the "conceptLocator."

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"
 }]
}
```

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 semantic network.

“**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.

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.

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"
```

1.9.3.4 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:8815/test/icons/bullet.png

1.9.3.5 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:

Parameter name	Name of the parameter

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 specified
Fixed value	Default value when no parameter was specified.

1.9.4 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
```

1.9.5 OpenAPI documentation

i-views offers the possibility to generate OpenAPI 2.0 documentation for configured services. For this purpose, service configurations and resource configurations can be enriched with documentation data.

1.9.5.1 Configuration

Service

Property	**Description**	**Mapping to OpenAPI 2.0**

Service Description	Free text description of the service; supports GitHub Flavored Markdown	`info.description`
Service Version	Version specification in forms of a free text. It is recommended to follow the Semantic Versioning.	`info.version`
Service ID		`info.title`
Swagger Definitions	Script which generates reusable OpenAPI 2.0 definitions in forms of a JSON object.	`definitions`

Resource

The *Resource Description* is **not** used for OpenAPI documentation, since descriptions in OpenAPI are intended to be used on method level only.

Method

The mappings to OpenAPI elements are specified relatively to paths.{path}.{method}

Property	Description	Mapping to OpenAPI 2.0
Method description	Free text description of the resource; supports GitHub Flavored Markdown	`.description`
Input media type	The name of the media type configured here (e.g. application/json) describes the format which is required as input for the request.	`.consumes.0`
Output media type	The name of the media type configured here (e.g. application/json) describes the format in which the request will respond.	`.produces.0`
Request Body	See section *Request Body*	special parameter with property in: "body"

Response	See section *Response*	`.responses.{code}`

Parameter

The mappings to OpenAPI elements are specified relatively to paths.{path}.{method}.parameters.{index}

Property	Description	Mapping to OpenAPI 2.0
Parameter Description	Free text description of the parameter; supports GitHub Flavored Marlkdown	`.description`
Parameter name		`.name`
Repeating	In case of path parameters, this option MUST NOT be enabled.	`.collectionFormat: "multi"`
Required	In case of path parameters, this option MUST be enabled.	`.required`
Style		`.in`
Type		`.type`

Request Body

In OpenAPI 2.0, the request body is represented in forms of a special parameter. The mappings relate to this parameter object:

Property	Description	Mapping to OpenAPI 2.0
Name	Denomination of the special request body parameter for OpenAPI 2.0	`.name`
Request Body Description	Free text description of the body; supports GitHub Flavored Markdown	`.description`
Request Model	Script which generates a JSON schema object, which in turn describes the format of the body.	`.schema`

Response

For a valid OpenAPI documentation, at least one possible response needs to be documented for each request. The specified mappings relate to the response object.

Property	Description	Mapping to OpenAPI 2.0
Response Code	HTTP status code of the response	Key
Response Description	Free text description of the response; supports GitHub Flavored Markdown	`.description`
Response Model	Script which generates a JSON object, which in turn describes the format of the response.	`.schema`

JSON schema definitions

For generating JSON schema for a further description of input and output, scripts can be used at different spots. The scripts allow usage of a subset of the JSON schema standard, which can be looked up at the OpenAPI specification website.

Example script *Swagger Definitions*:

```
function swaggerDefinitions() {
    return {
        "Example": {
            "properties": {
                "id": { "type": "integer" },
                "name": { "type": "string" }
            }
        }
    }
}
```

Example script *Request Model* with reference to the definition above:

```
function swaggerJSONSchema() {
    return {
        "$ref": "#/definitions/Example"
    }
}
```

1.9.5.2 Generating the API documentation

Manual generation within the KB

For generating a .json file manually using the OpenAPI documentation within the Knowledge Builder, the button *Export as OpenAPI 2.0* is available above the list of the services.

CLI

The same export is provided by means of the command line interface:

```
bridge-64.exe -exportBuiltInRequestAPI {filename} {serviceID}
```

As REST API endpoint

By means of the script resource, a REST endpoint can be created which returns the API documentation. The script resource requires a GET method with *Output media type*

```
application/json; charset=utf-8
```

and a *script* with the following content:

```
function respond(request, parameters, response) {
    // Here, the internal ID of the service configuration object needs to be inserted, NOT the val
    var service = $k.Registry.elementWithID(’<ID der Service-Konfiguration>’)
    var openAPI = service.callBehaviour(’asSwaggerObject’)
    delete openAPI.definitions.Context
    delete openAPI.definitions.RenderContext
    // Extract the service base path from the request
    openAPI.basePath = /(http[s]?:\/\/)?([^\/\s]+)(\/.*)\/.+/gm.exec(request.url()).pop()
    response.setContents(JSON.stringify(openAPI, null, ’\t’))
    response.setCodeOk()
}
```

1.10 Reports and printing

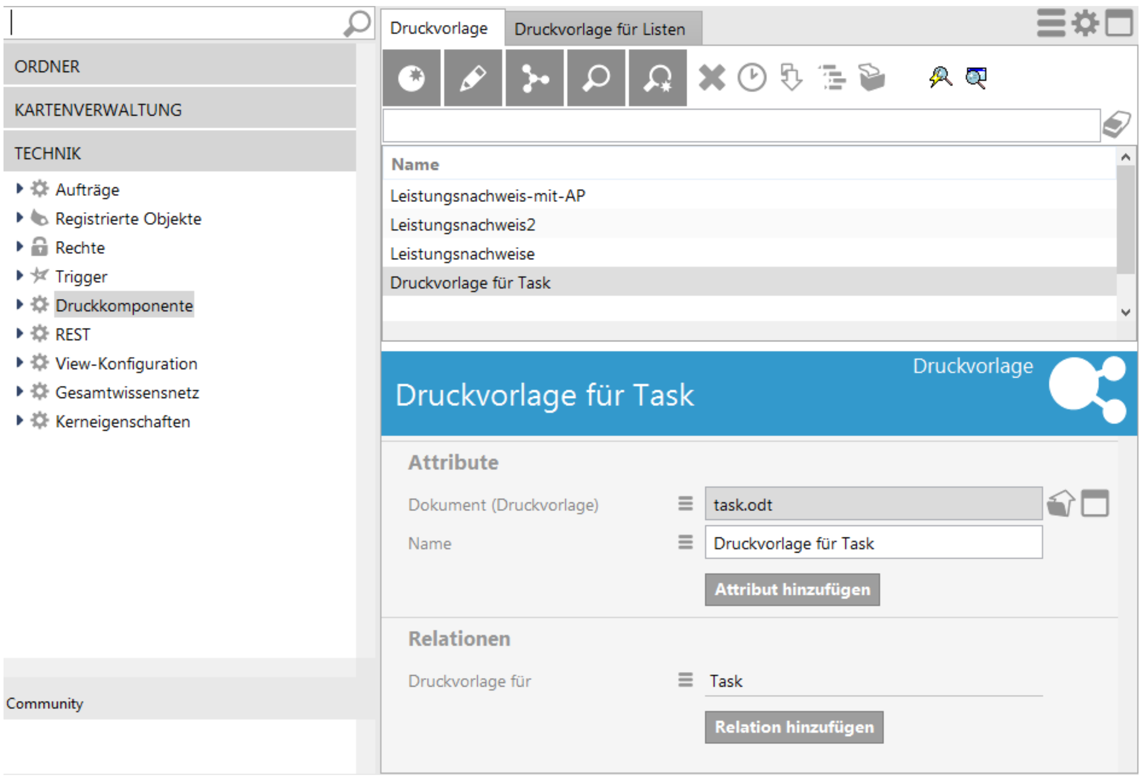
You can use the printing component to use document templates (ODT/DOCX/XLSX/RTF files) with KPath expressions on objects or object lists and then use them to generate an adapted output file, which can be either printed or stored.

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 semantic network. The existence of this component is prerequisite for the print function being available in Knowledge Builder or via the REST interface.

1.10.1 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
Leistungsnachweis-mit-AP
Leistungsnachweis2
Leistungsnachweise
Druckvorlage für Task

Druckvorlage

Druckvorlage für Task

Attribute

Dokument (Druckvorlage): task.odt

Name: Druckvorlage für Task

Relationen

Druckvorlage für: Task

The following chapters explain how print template documents are created.

1.10.1.1 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 gmbh
Julius-Reiber-Str. 17
64293 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.

Example:
The template is:

Parts (<KPATH_EXPAND topic()/~$hatParts$/size()> pieces)	**Note**

<KPATH_EXPAND topic()><KPATH_ROWS topic()/~$hatPart$/target()/sort(@$name$, true)>	<KPATH_EXPAND topic()/@$note$>

After the evaluation the output file says:

Parts (3 pieces)	**Note**
RTF print	
ODT print	Replaces RTF print
Conversion service	Optional service

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

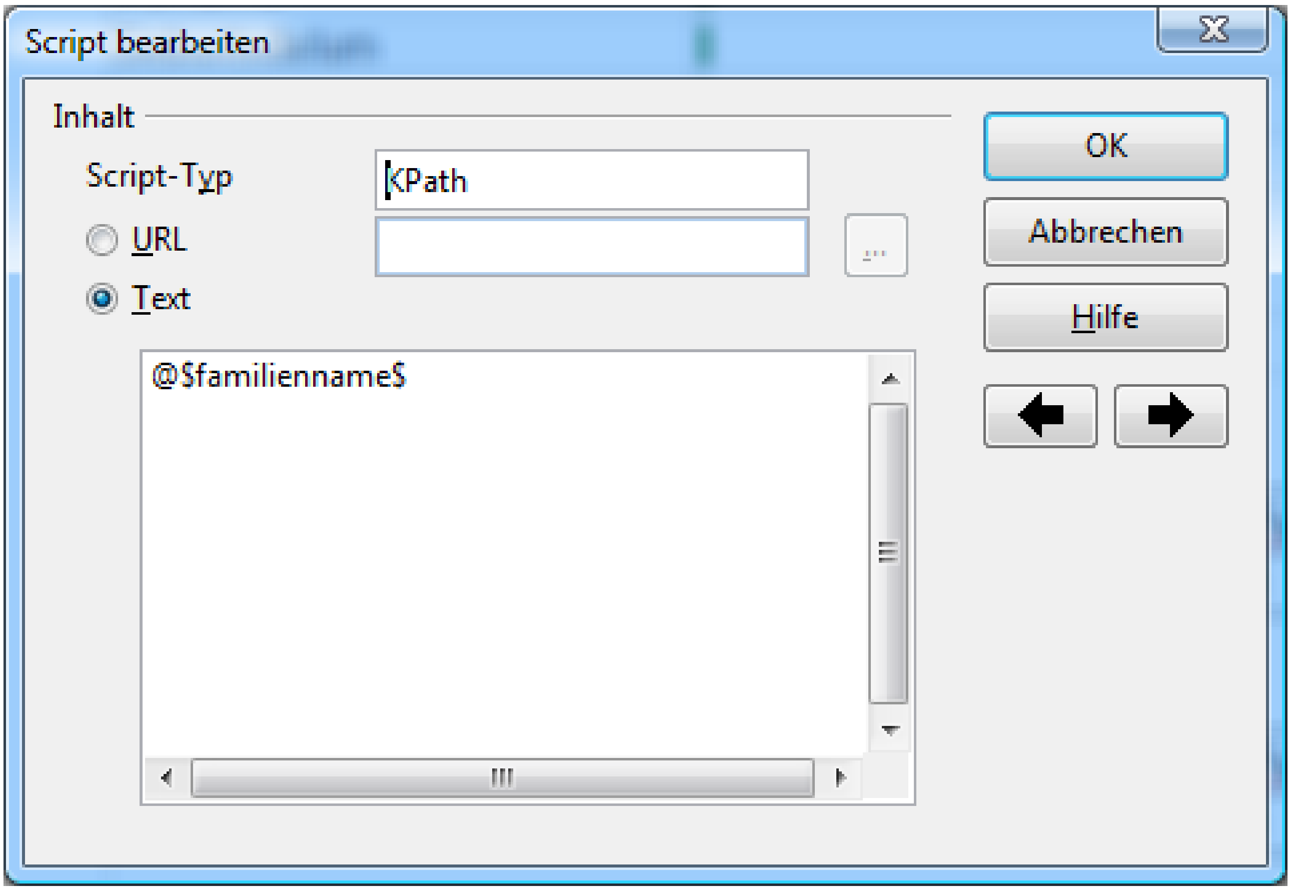
1.10.1.2 Create ODT documents (OpenOffice)

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.



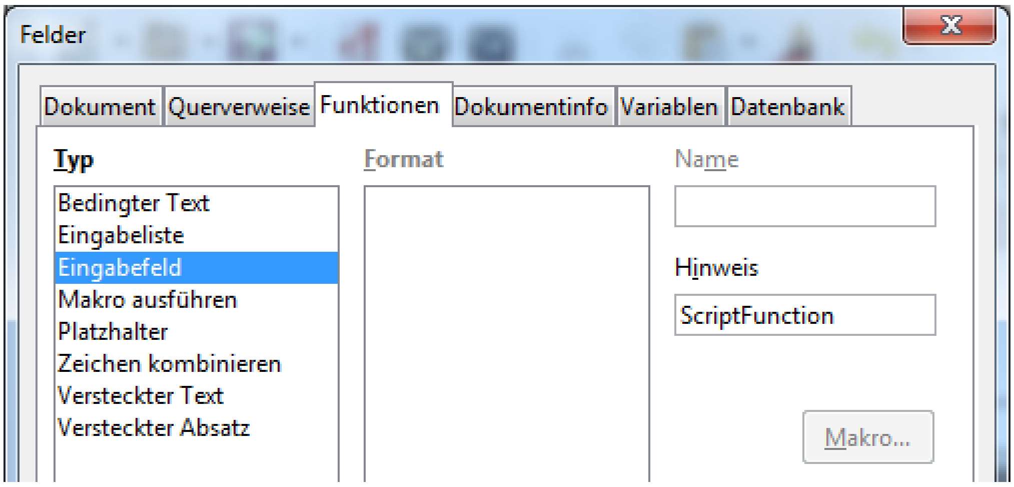
Script-Typ: KPath
URL
[✓ Text]

```
@$familienname$
```

The script field can no longer be integrated in LibreOffice 5. As an alternative to this, the “Input field” can be used:

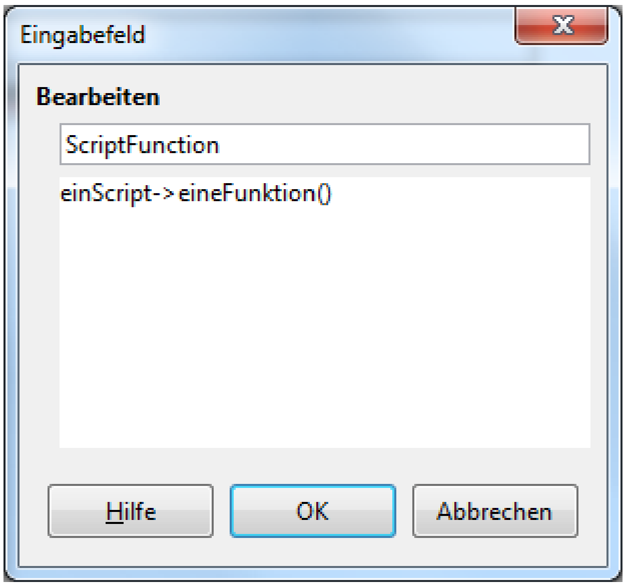
Insert > Field command > Other field commands (alternative keyboard shortcut Ctrl+F2)

The input field is found there on the “Functions” tab.



Hinweis: ScriptFunction

“Note” is equivalent to the previous “Script type”; after clicking on insert, another window opens in which the script can be entered.



ScriptFunction

einScript->eineFunktion()

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.

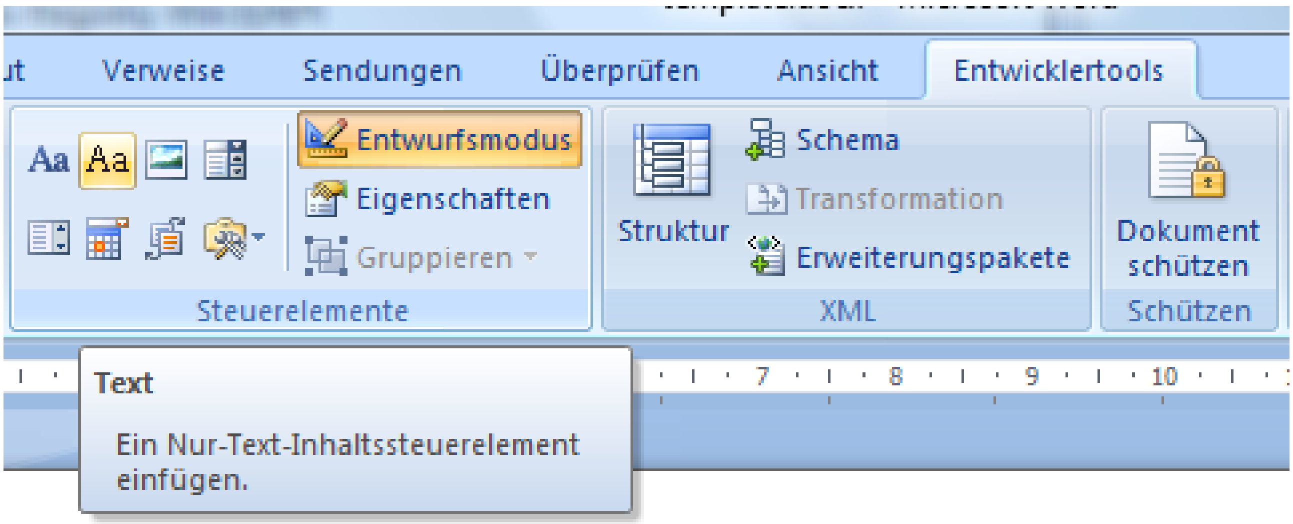
Embedded images are always sourced in their original size (at 96d dpi). If another size should appear in the printout, a frame with the required dimensions (absolute dimensions in cm must be used!) must be built around the script element. The resulting embedded image is then fit into the frame so that the frame dimension is not exceeded while retaining the image aspect ratios.

1.10.1.3 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.

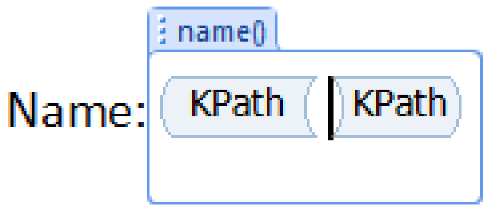
To insert the control elements, it is first necessary to activate the developer tools in Word. To do so, go to the Office menu, open the **Word options**, go to the **Popular commands** category and activate the option **Show Developer tab in the ribbon**. Now go to the **Developer tools** tab and activate **Design mode**.

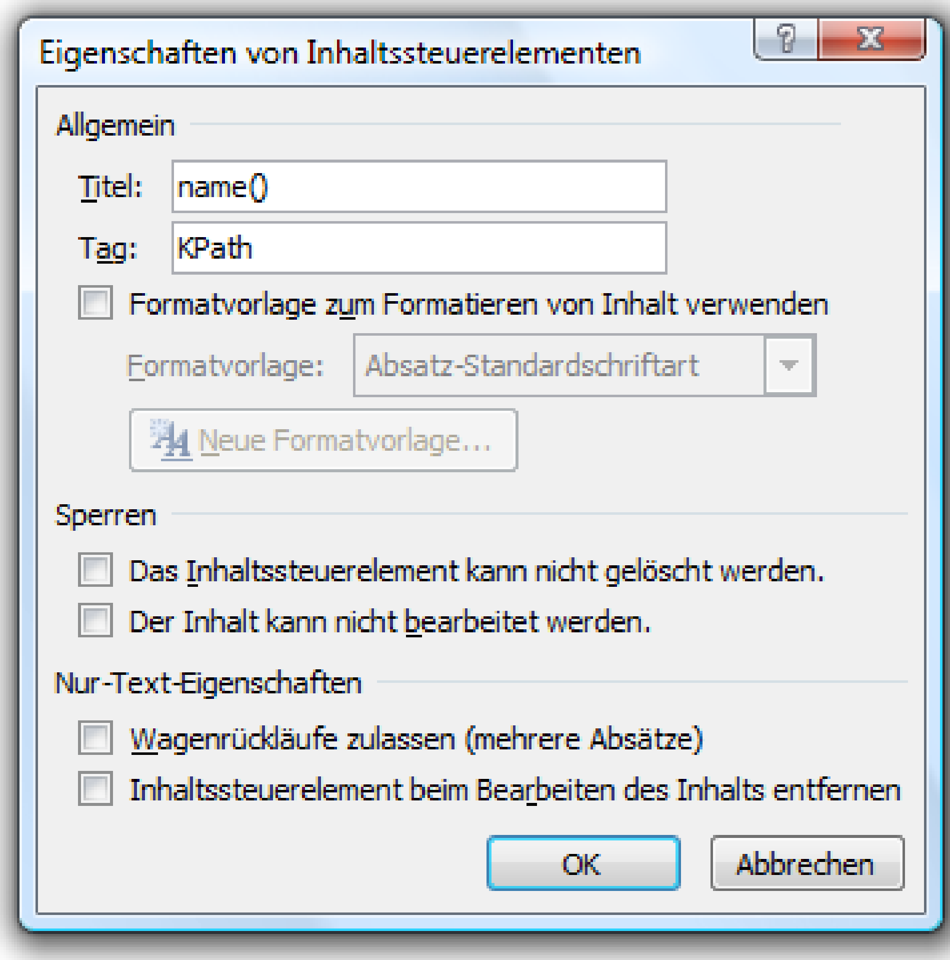


Text

Ein Nur-Text-Inhaltssteuerelement einfügen.

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**.**




Eigenschaften von Inhaltssteuerelementen

Allgemein

Titel:	name()
Tag:	KPath

☐ Formatvorlage zum Formatieren von Inhalt verwenden

Formatvorlage: Absatz-Standardschriftart

Neue Formatvorlage…

Sperren

☐ Das Inhaltssteuerelement kann nicht gelöscht werden.

☐ Der Inhalt kann nicht bearbeitet werden.

Nur-Text-Eigenschaften

☐ Wagenrückläufe zulassen (mehrere Absätze)

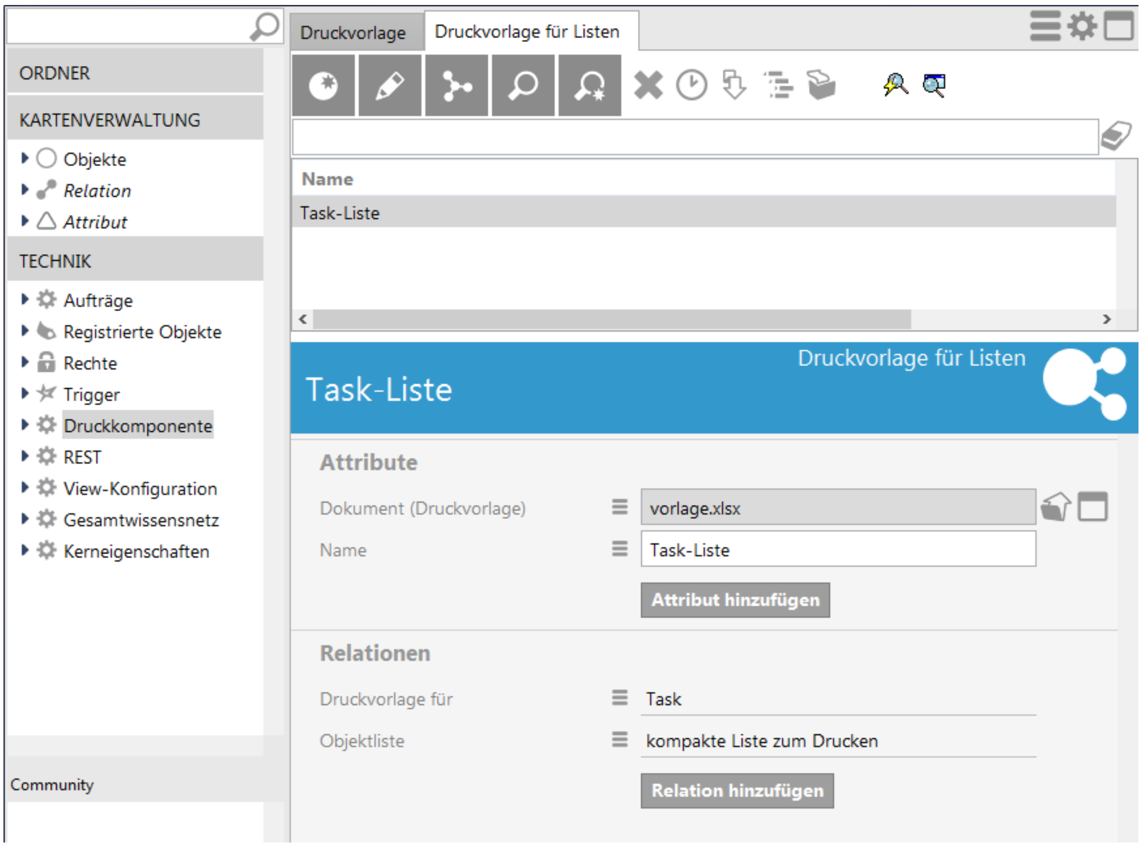
☐ Inhaltssteuerelement beim Bearbeiten des Inhalts entfernen

1.10.2 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.



Task-Liste — Druckvorlage für Listen

Attribute

Dokument (Druckvorlage)	vorlage.xlsx
Name	Task-Liste

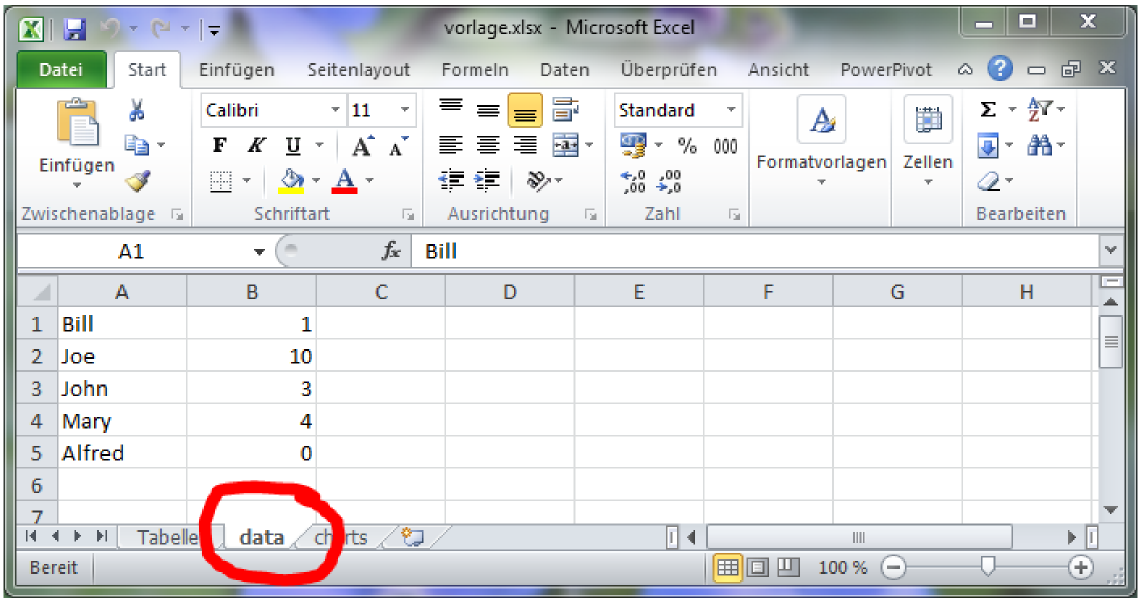
Relationen

Druckvorlage für	Task
Objektliste	kompakte Liste zum Drucken

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.



A1 f_x Bill

	A	B	C	D	E	F	G	H
1	Bill	1						
2	Joe	10						
3	John	3						
4	Mary	4						
5	Alfred	0						
6								
7								

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.

1.10.3 Document format conversion with Open / 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 network.

The important thing is that not all formats can be converted into all formats. The most important ones are:

Name	Exten-sion	Mime type
Portable Document Format	pdf	application/pdf
OpenDocument Text	odt	application/vnd.oasis.opendocument.text
Microsoft Word	doc	application/msword

1.11 Tagging

The tagging component allows objects from the semantic network (persons, topics, etc.) to be found or be created in documents.

Tagging requires:

- A configured tagging component in the semantic network
- A tagging software (Intrafind, OpenNLP) that finds potential objects in a text

Tagging is performed in three steps

1. The document text for tagging is defined (e.g. the value of a text attribute)
2. The text is passed on to the tagging software, which analyzes the text and delivers a series of tags
3. The configuration is used to search for existing objects in the semantic network for each tag, and to create any potentially new objects. The objects are linked with the document by means of a relation.

1.11.1 Configuration

To use tagging, you need to use the Tagging component which can be added in the Admin tool. This component sets up the required schema.

Following that, you can configure it in Knowledge Builder under “Technical” > “Tagging.”

Every tagging configuration consists of:

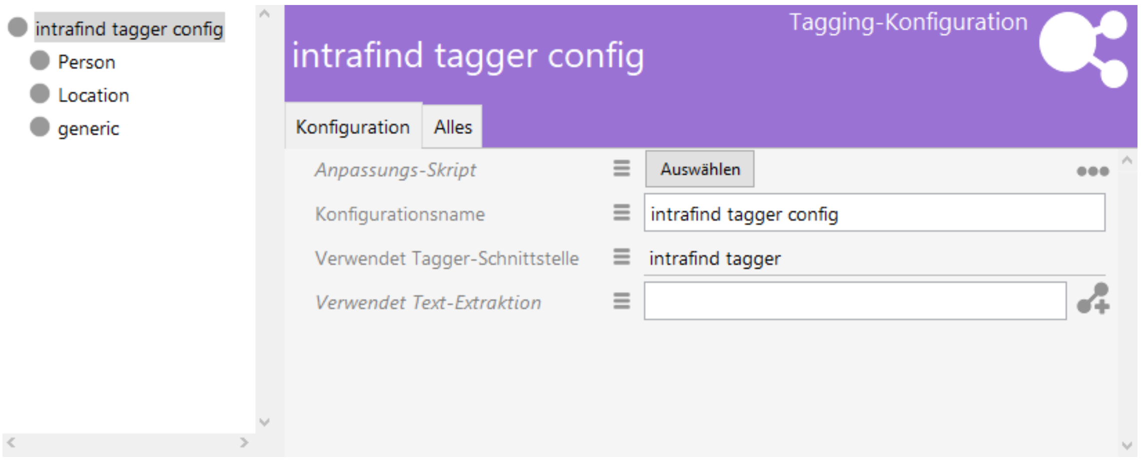
- An interface configuration of the tagging software to be used (intrafind, OpenNLP)
- Configuration of the text extraction that determines the text to be tagged in a document
- Tag configurations that determine how objects are found, created and linked in the semantic network

1.11.1.1 Tagging configuration

The tagging configuration bundles all the information required for tagging.

It is however mandatory to specify the tagger interface to be used.

Specification of the text extraction to be used is optional. Alternatively this can also be determined dynamically (see the corresponding sub-chapter).



Anpassungs-Skript	Auswählen
Konfigurationsname	intrafind tagger config
Verwendet Tagger-Schnittstelle	intrafind tagger
Verwendet Text-Extraktion	

Furthermore, it is possible to specify an adjustment script that can be used to influence tagging. Additional adjustments can also be made in the configurations for tags and for text extraction.

Newly created adjustment scripts contain commented-out function bodies. In order to activate them you only need to remove the comment signs.

1.11.1.2 Interface configuration

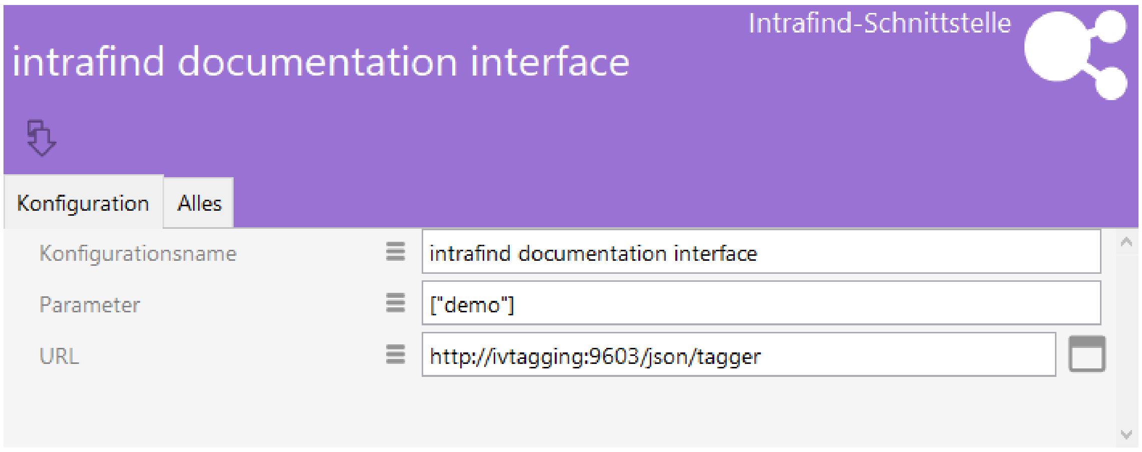
The Intrafind interface has the following setting:

Configuration name	Freely selectable name

Parameter	(optional) This is transferred to Intrafind using the interface, and controls tagging
URL	URL of the Intrafind tagger
Update-URL	(optional) URL of the Intrafind List Service, used for export of known tags, see also 1.11.1.5.

In the case of OpenNLP, only the URL of the REST service is required along with the optional name.

The interface "Internal tagger" is only intended for test purposes / internal demos for which connecting an external system is unwanted. This tagger makes no claim to returning results that make sense.



Konfigurationsname	intrafind documentation interface
Parameter	["demo"]
URL	http://ivtagging:9603/json/tagger

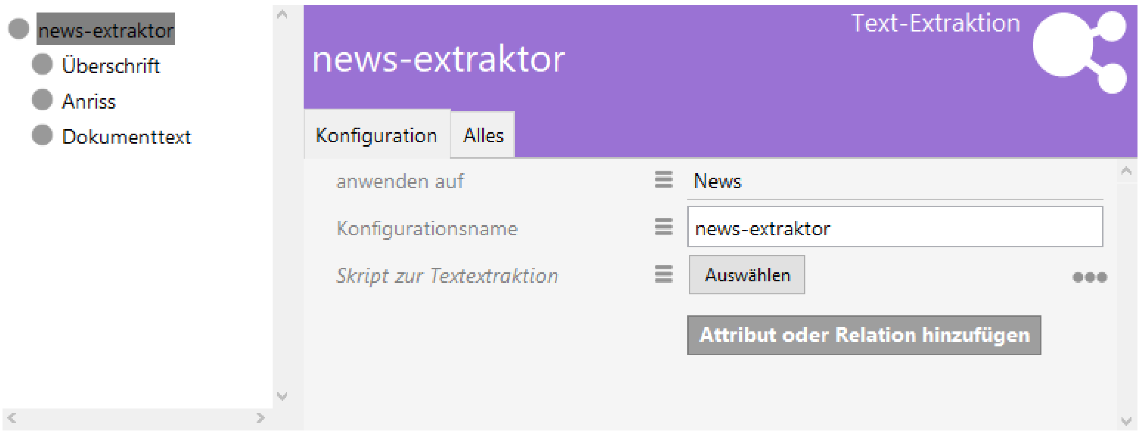
1.11.1.3 Text extraction

If the text to be tagged is not determined dynamically, e.g. because only the text of a very specific attribute type or the text of a document is to be extracted, text extraction must be configured.

This configuration can be added on the "Text extraction" tab.

Configuration name	Freely selectable name
apply to	Object type to which this configuration applies. Is used if no explicit text extraction is specified during the tagging configuration.
Script for text extraction	Optional script for determining text

To specify the attribute types to be tagged, one or more text part extractions (hierarchically sorted on the left side) are added to the text extraction. In each text part extraction, the attribute type to be tagged is stored under "extracts text from."



news-extraktor
- Überschrift
- Anriss
- Dokumenttext

Text-Extraktion

news-extraktor

Konfiguration | Alles

anwenden auf: News

Konfigurationsname: news-extraktor

Skript zur Textextraktion: Auswählen

Attribut oder Relation hinzufügen

In addition to strings, blobs can also be used as text part extractions. Text is extracted from these and forwarded to the tagging interface. To do this, text extraction must be configured in the client (bridge or KB) (see chapter i-views services > Text extraction).

The optional script has three arguments

textDocument	$k.TextDocument	Outputs the text to be tagged
element	$k.SemanticElement	The element whose text is to be extracted
attributes	$k.Attribute []	Array of attributes of the element. The attributes are collected according to the configuration.

The following example writes the values of the attributes in sequence:

```
function extractText(textDocument, element, attributes)
{
    attributes.forEach(function(attribute) {
        textDocument.println(attribute.valueString());
    });
}
```

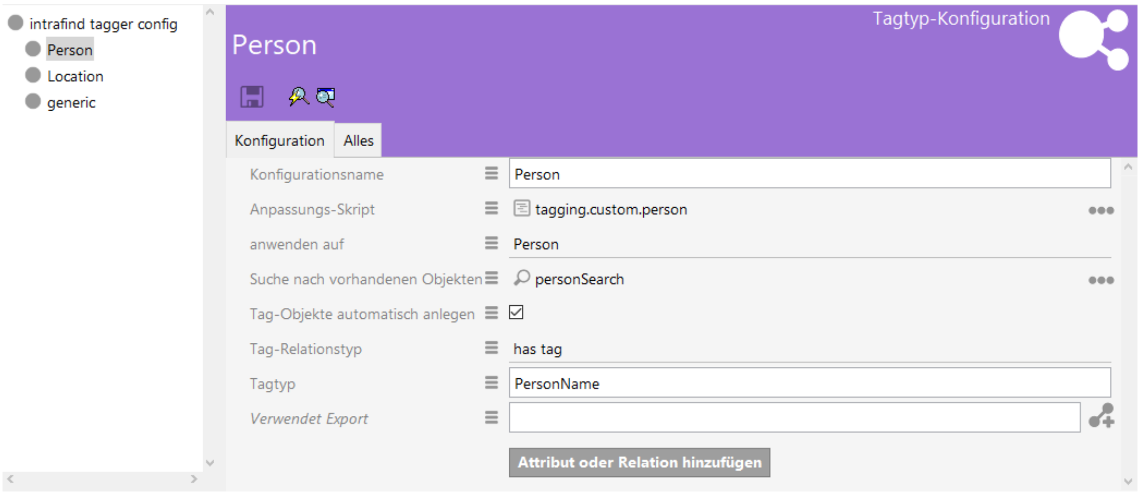
1.11.1.4 Tag types

The tag type configuration determines how objects are found, created and linked in the semantic network. To do this, you can specify a separate configuration for each tag type provided by the tagging interface. You can create a new configuration for the tagging configuration in the hierarchy view on the left side.

By default, the interfaces provide the following tag types:

Intrafind	PersonName, Location, TFIDF
OpenNLP	NP

A tag configuration can apply to one or more tag types.



Person

Konfigurationsname	Person
Anpassungs-Skript	tagging.custom.person
anwenden auf	Person
Suche nach vorhandenen Objekten	personSearch
Tag-Objekte automatisch anlegen	☑
Tag-Relationstyp	has tag
Tagtyp	PersonName
Verwendet Export	

The configuration offers the following settings:

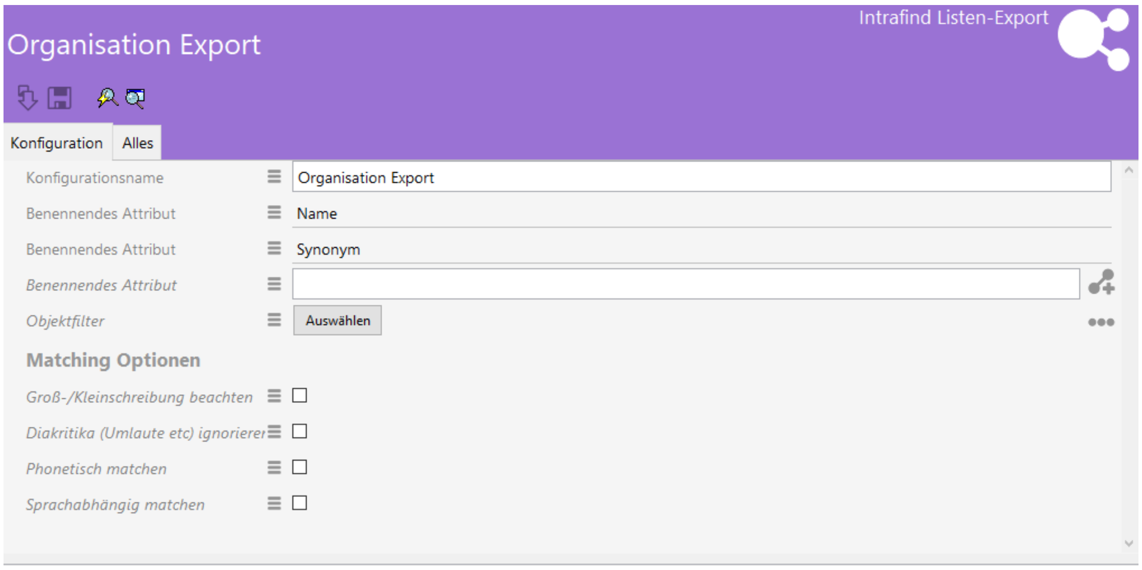
Adaptation script	Script to affect tagging. The template contains a row of functions that are commented out and can be activated.
Apply to	Type in the semantic network that corresponds to the tag type. If objects are to be searched/created and no additional configuration information is specified, this type is used.
Configuration name	Freely selectable name
Search for existing objects	Search that contains the text of the tag as the *searchString* parameter and searches for *one* suitable object in the semantic network. Several searches can be specified, e.g. to keep the individual searches more compact. If there are several hits, query search must return the suitable hit. If several hits of different quality are found, the highest quality hit is used. If no best hit can be determined, no object is assigned.

Create tag objects automatically	If no object was found and this option was activated, new objects are created. You have to ensure that the search for existing objects find these as new objects are created every time tagging takes place. If no adaptation script applies here, an object of the specified type is created for “apply to” and the text of the tag is set as its name.
Tag relation type	This relation type is used to link documents to the objects found by the tagger.
Tag type	The tag types specified above. If no tag type is defined, the configuration applies to all types of tags.
Uses export	Here, an export configuration can be specified which can be used to export all tags of the type or a subset thereof. Refer to the next section for details.

1.11.1.5 Export of known tags

There is an export function used to save information from the network in a tagging service, e.g. Intrafind. This is currently only supported for Intrafind, where it performs the following:

One, or several, lists can be generated that are then saved to the tagging interface. Each list export assigns naming attributes (e.g. name, synonym) to the semantic elements for export. The tagger then searches for these names in texts, and can deliver the suitable semantic element as well. For example, the list of known organizations can be exported this way, and the tagger can identify them reliably.



Organisation Export

Konfigurationsname	Organisation Export
Benennendes Attribut	Name
Benennendes Attribut	Synonym
Benennendes Attribut	
Objektfilter	Auswählen

Matching Optionen

Groß-/Kleinschreibung beachten	☐
Diakritika (Umlaute etc) ignorieren	☐
Phonetisch matchen	☐
Sprachabhängig matchen	☐

The *Intrafind list export* is configured for every tag type and is also influenced by the *tag type configuration*. Generation configuration options:

Configuration name	Freely selectable name
Naming attribute	(Optional) attribute that identifies the object. Multiple specifications possible. If no attribute has been specified, the name attribute is exported by default.
Object filter	(Optional) A search can be specified here that specifies the set of objects. If no search has been specified, all types that are assigned in the tag type configuration by means of *Apply to* are exported.

Intrafind-specific *matching options*. These have a direct influence on the performance of the tagging service:

Observe upper/lower case	Case-insensitive matching is activated by default. Case-sensitive matching can be activated here.

Ignore diacritics (umlauts, etc.)	[Presumably] This option is used to ignore characters with accents or umlauts, e.g. Geräte will match with Gerate.
Phonetic matching	[Presumably] For example, match “photography” with “fotografie.”
Language-dependent matching	This option activates the linguistic processing of the names transferred. In doing so, it is important that the data is maintained correctly according to language in the semantic network, as every language must be processed using its own linguistics.

Performing the export

There are three relevant buttons to performing the export:

- the zigzag arrow (found at the export config or the "top" tagging configuration) "refreshes" the configuration cache, such that the newly changed configuration will have an effect
- the floppy disk symbol found at the export config opens a dialog to save the exported list to a directory. The same symbol found at the top tagging configuration will export all lists at once. (hint: you have to select an *existing directory*, and the files will be written into it)
- the up-pointing arrow (found at the top tagging configuration if configured) is used to upload all lists via the intrafind list service. This option is only possible, if the list service was installed for the given environment, i.e. *if* the list service is configured. See also "Interface configuration" -> "Update-URL" above on how to configure that. After entering the correct credentials, the upload will take place (this may take a while with spinning cursor as feedback). On success, the response will indicate whether the service was restarted and how many files were uploaded.

1.11.1.6 Overlapping filter group

The tagger may deliver several tags for one text passage. In some cases, the user explicitly allows this overlap and have several tags displayed.

The overlap filter group does the following:

- All tags types that are summarized into a group like this must be free of overlaps.
- Within a group, a script can be used to specify a prioritization to influence the decision about which tag is displayed in the end
- In order to allow overlaps, at least two groups like this must exist
- All tag types without a group are summarized in the “Default” overlap filter group.

Prioritization with script

```
/**
 * When there are conflicting tags (e.g. overlapping), this function can influence the conflict re
 * The sortOrder compares the array from left to right, lower numbers are sorted before higher one
 * e.g.: [-1, 3] < [0, 0] < [1, -3] < [1, -2]
 *
 * @param {$k.Tag[]} tags
 * @param {$k.TaggingContext} taggingContext
 * @returns {integer[]} an array of numbers that is used to sort the conflicting tags.
 **/
function tagSortOrder(tag, taggingContext)
{
    var smallestSpanReducer = function(minPos, span){return Math.min(minPos, span.start)};
    var positionMinimum = tag.spans().reduce(smallestSpanReducer, Number.MAX_VALUE);
    return [-tag.tagTypePriority(), -tag.canonicalText().length,  positionMinimum ];
}
```

A script must return a list of integers, whereby the first element in this list has the most influence. In principle, it functions the same ways as sorting by several columns, meaning that the second element is only used when the same value occurs in the first element.

Default prioritization

If no script has been specified, or the tag type is grouped in the implicit “Default” group, then the following prioritization is used:

- Order of the tag types - higher priority first
- Longer tags given preference
- Position within the overlap (meaning in the case of “a red wall”, “a red” is given preference over “red wall”, because it is closer to the front)

Also compare the script template.

1.11.2 View configuration

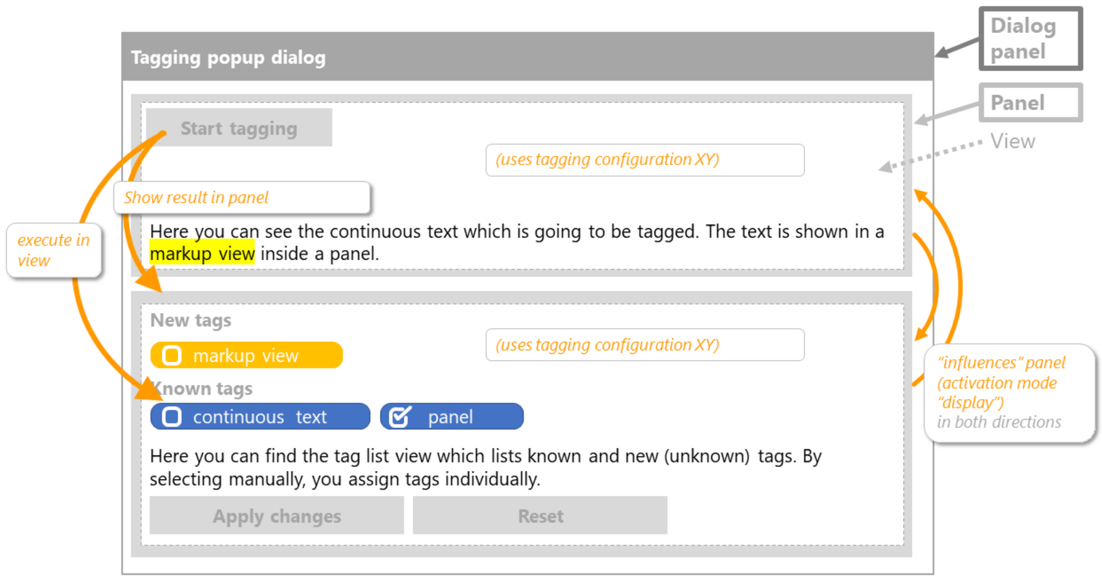
Two views are available for the display:

- Markup view
- Tag list

The markup view can be used in both the Knowledge Builder and in the ViewConfiguration Mapper. The view can be used everywhere that other views such as properties or hierarchies can be used.

The view has a permanently integrated tag button in the Knowledge Builder. There is an integrated action type “Tag” in the ViewConfiguration Mapper, which can also be used in a custom button.

The tag list is only available in the View-Configuration Mapper and is the content of a panel (e.g. as a sub-configuration of a panel with a fixed view) there. If a markup view with tag buttons was configured in another panel, its panel should be linked to the tag list panel using the relation “Influenced” so that the tag list is updated after tagging.



Here you can see the continuous text which is going to be tagged. The text is shown in a markup view inside a panel.

Here you can find the tag list view which lists known and new (unknown) tags. By selecting manually, you assign tags individually.

Both views have the obligatory configuration setting "Tagging configuration used", which connects the view to the tagging configuration.

1.11.2.1 Debug Log

The KB can output debug information during the tagging process. The information is written to the #tagging channel (see manual for documentation regarding channels) and can be output to a file, for example.

To do so, create a .txt file in the directory of the KB and rename it "kb.ini". Then add the following content:

```
[Default] logTargets=tagging [tagging] type=file format=plain channels=tagging loglevel=DEBUG file
```

This creates a "tagging.log" file where you can view the tags found by Intrafind for the tag types. This makes it possible to identify which strings are suggested by Intrafind as tags, and also which tag types (e.g. signifterm/tfidf or organization) are used to find them.

1.11.3 Tagging by Script

Tagging can also be performed by script. To do so, create an object of the type $k.TaggingConfiguration.

The tag(context) function is used to perform the tagging. Tagging is controlled by an object of the type $k.TaggingContext. Because it is stateful, a new one must be created every time tag() is called. The TaggingConfiguration object is stateless and can be reused.

```
var document = $k.Registry.elementAtValue("RDF-ID", "opennlp-testdocument");
var configElement = $k.Registry.elementAtValue("tagging.name", "opennlp tagger config");
var tagger = $k.TaggingConfiguration.from(configElement);
var context = new $k.TaggingContext();
context.setSource(document);
```

```
tagger.tag(context);
$k.out.print("Found " + context.tags().length + " tags");
```

1.11.4 Required software

The Intrafind tagger must be purchased and installed separately. The corresponding Intrafind List Service can be provided by i-views.

The OpenNLP connection is made using a REST interface to OpenNLP provided by i-views.

1.12 Development support

1.12.1 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.

1.12.2 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
```

1.13 Rule engine

1.13.1 What are rules?

Allgemein ausgedrückt können mit Regeln ein Element des Wissensnetzes (=Referenzobjekt) mit einer zweiten Menge von Elementen des Wissensnetzes (=Vergleichsmenge) über das Ablaufen von Relationen, den Vergleich von Werten und das Erfüllen von weiteren Bedingungen miteinander verglichen werden, um schließlich eine Ergebnismenge zum Referenzobjekt zu erhalten. Die Ergebnismenge ist dabei eine Teilmenge der Vergleichsmenge. Die Elemente der Ergebnismenge können zusätzlich gewichtet werden.

Auf Produktwelten bezogen, sind verschiedene Use Cases vorstellbar. Mit Regeln können Fachexperten zum Beispiel definieren, welche Produkte aufgrund von Eigenschaften zueinander passen (z.B. Stecker zu Steckverbinder, Zubehör zu Produkten, etc.), aber auch welche Teile von Produkten (Merkmale, Komponenten, Varianten, etc.) zueinander passen, um ein vollständiges valides Produkt zu ergeben (Konfigurator). Weitere Use Cases für Regeln sind, dass Produkte aufgrund von Eingaben eines Kunden empfohlen werden sollen, dass ausgedrückt werden kann welche Produkte zu einer Serie gehören oder auch aufgrund welcher Eigenschaften und Bedingungen zueinander ähnliche Produkte gefunden werden sollen.

In dieser Dokumentation werden die Regeln anhand des Beispiels potenzielle Freunde zu einer Person zu finden erklärt.

1.13.2 Where can rules be configured?

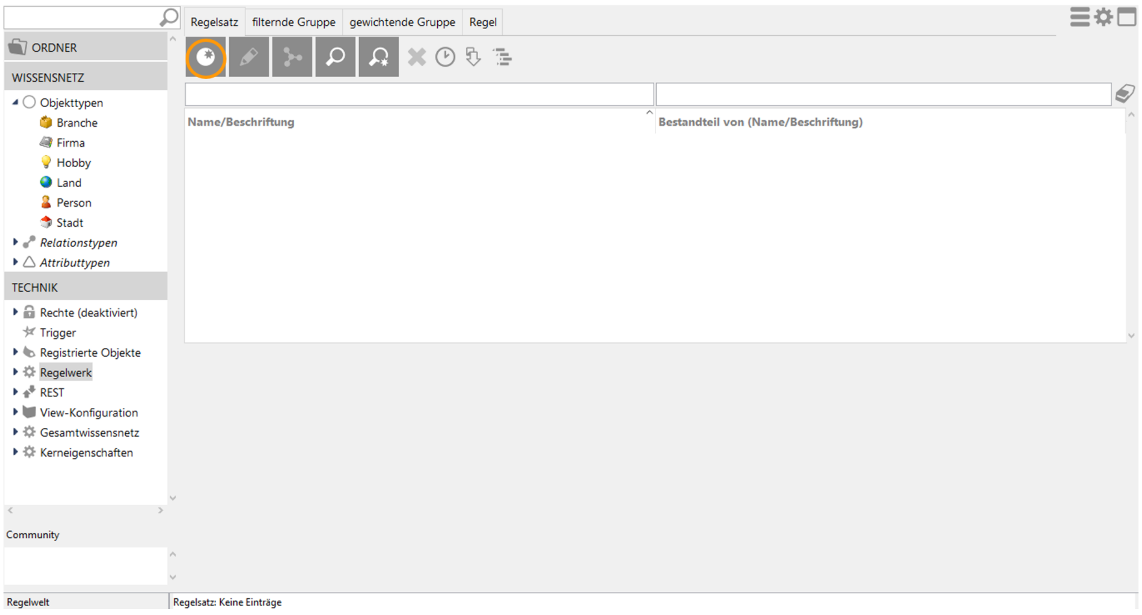
Regeln können in i-views über die Regelkomponente konfiguriert werden. Nach Installation im Admin-Tool befindet sich die Regelkomponente unter dem Namen „Regelwerk“ im Technik-Bereich. Ein Klick auf „Regelwerk“ öffnet im rechten Bereich des KB auf dem ersten Tab die Regelsätze. Um Regeln zu erstellen benötigt man zunächst einen Regelsatz, der die darin enthaltenen Bedingungen zusammenfasst.

1.13.3 How can rules be configured?

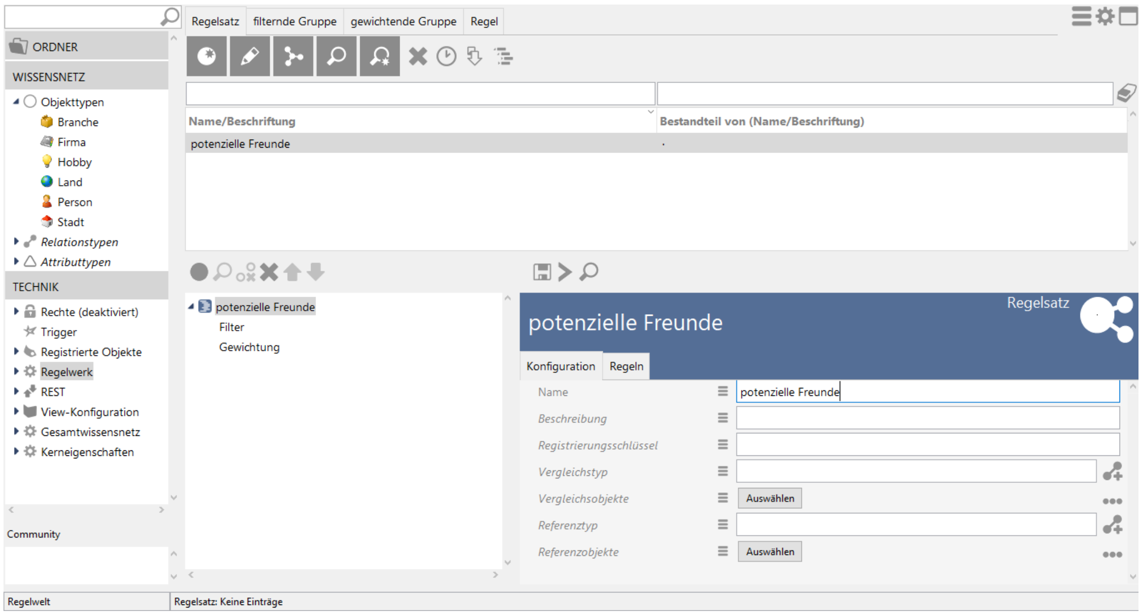
1.13.3.1 Rule set

Bevor die einzelnen Regeln definiert werden, muss ein Regelsatz angelegt werden, der die Regeln enthält und die Verwendung der Regeln definiert. Wann ein Regelsatz angewendet wird, wird über die Vergleichs- und Referenzmenge definiert (siehe nächstes Kapitel).

Ein Regelsatz kann aus filternden und/oder gewichteten Regeln, Ableitungen, Gruppen und Abzügen bestehen. Um einen neuen Regelsatz zu erstellen, klickt man im Reiter „Regelsatz“ auf „Neu erstellen“ und muss hier dem Regelsatz einen Namen geben.



In unserem Beispiel geben wir dem Regelsatz den Namen „potenzielle Freunde“, weil wir zu einer Person unter verschiedenen in diesem Regelsatz zusammengefassten Regeln potenzielle Freunde finden wollen.



potenzielle Freunde

Name	potenzielle Freunde
Beschreibung	
Registrierungsschlüssel	
Vergleichstyp	
Vergleichsobjekte	Auswählen
Referenztyp	
Referenzobjekte	Auswählen

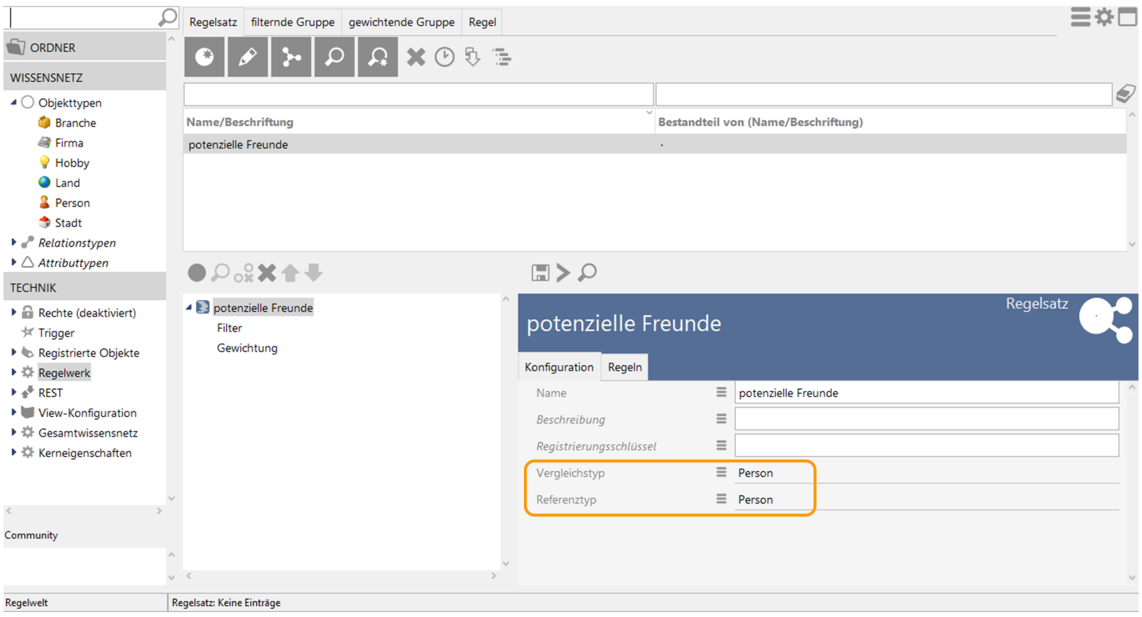
Der neue Regelsatz ist damit erstellt. Es können nun noch optional eine detaillierte Beschreibung und ein Registrierungsschlüssel (zum Beispiel zur Wiederverwendung in einer View-Konfiguration) gesetzt werden.

1.13.3.2 Comparative amount / Reference amount

Jeder Regelsatz braucht zwingend eine Vergleichs- und eine Referenzmenge. Eine Vergleichsmenge besteht immer aus Objekten. Diese Objekte können entweder über eine Strukturabfrage definiert werden (Vergleichsobjekte, Referenzobjekte) oder es kann ganz einfach der Typ der Objekte eingegeben werden (Vergleichstyp, Referenztyp). Ist keine Referenzmenge gesetzt, ist es dieselbe wie die Vergleichsmenge.

Die Referenzmenge ist die Menge an Objekten, aus der ein Objekt gewählt werden kann, für das der Regelsatz verwendet wird (ausgehende Menge). Die Vergleichsmenge definiert, woher die Ergebnisse entstammen dürfen.

In unserem Beispiel wählen wir hier sowohl für die Vergleichs- als auch für die Referenzmenge alle Objekte des Typs Person aus.



Name/Beschriftung	Bestandteil von (Name/Beschriftung)
potenzielle Freunde	-

potenzielle Freunde — Regelsatz

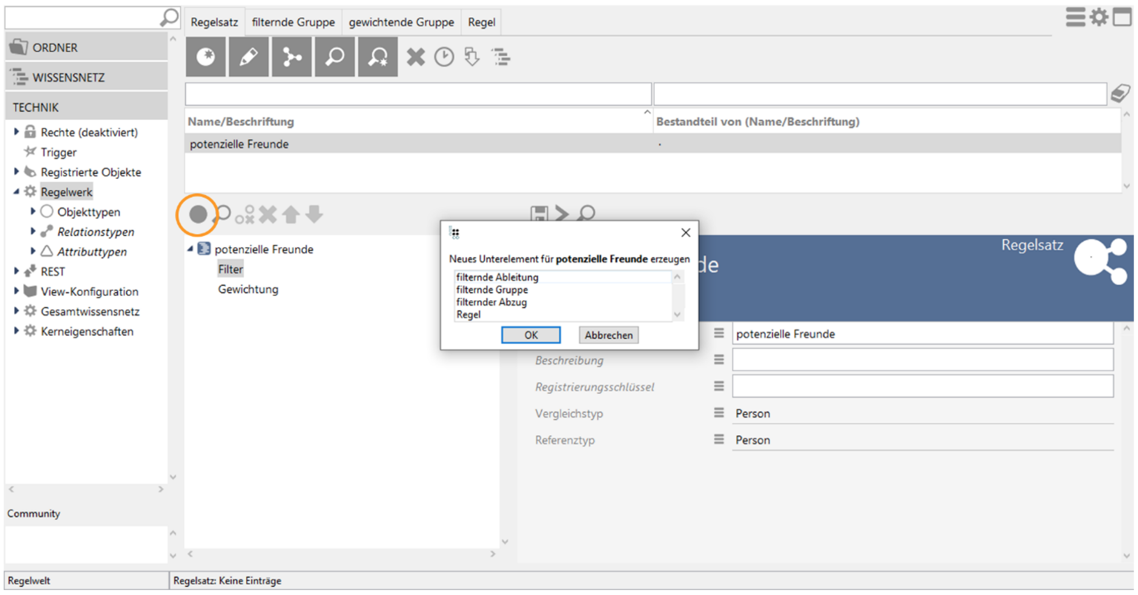
Name	potenzielle Freunde
Beschreibung	
Registrierungsschlüssel	
Vergleichstyp	Person
Referenztyp	Person

Regelwelt | Regelsatz: Keine Einträge

1.13.3.3 Filter/Weighting

Im nächsten Schritt müssen wir uns entscheiden, welche Regeln wir unter „Filter“ setzen und welche Regeln wir unter „Gewichtung“ setzen. Filternde Regeln filtern Objekte aus der Vergleichsmenge aus, sodass sie nicht in der Ergebnismenge auftauchen. Gewichtende Regeln verleihen den Objekten der Ergebnismenge, je nachdem wie viele Regeln zutreffen oder wie stark gewichtet sind, eine Gewichtung untereinander, sodass die Objekte in der Ergebnismenge nach diesem Gewicht sortiert werden können (Qualität).

Wenn wir nun filternde Regeln erstellen wollen, klicken wir auf „Filter“ und dann auf „neues Unterelement erzeugen“. Hier muss zunächst ausgewählt werden, welches Element des Bausatzes zum Regelwerk verwendet werden soll. Es wird zwischen filternde Ableitung, filternde Gruppe, filternder Abzug und Regel unterschieden.



Neues Unterelement für **potenzielle Freunde** erzeugen
- filternde Ableitung
- filternde Gruppe
- filternder Abzug
- Regel

OK | Abbrechen

Name	potenzielle Freunde
Beschreibung	
Registrierungsschlüssel	
Vergleichstyp	Person
Referenztyp	Person

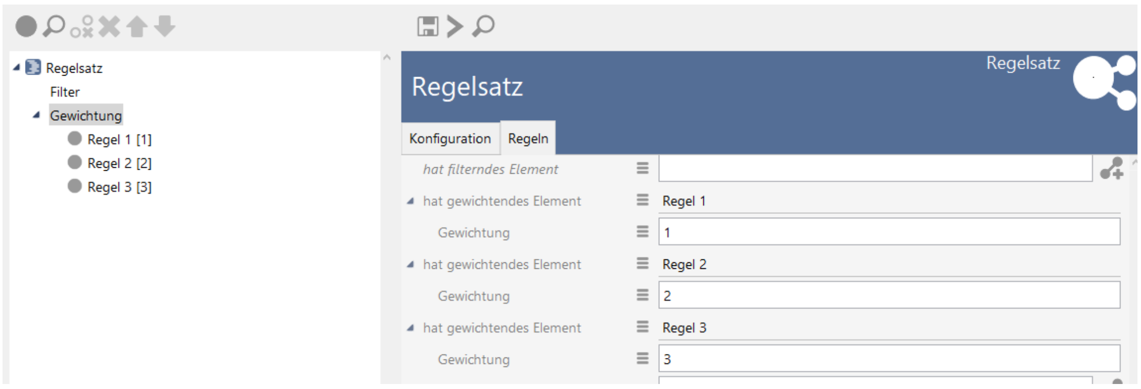
Regelwelt | Regelsatz: Keine Einträge

Bei gewichtenden Regeln wird zwischen gewichtender Ableitung, gewichtender Gruppe, gewichtendem Abzug und Regel unterschieden. Gewichtet wird dann, wenn Treffer mehr oder weniger gut sein können.

Angenommen wir haben 3 Regeln, wobei Regel 1 die Gewichtung 1, Regel 2 die Gewichtung 2 und Regel 3 die Gewichtung 3 hat, dann gibt es insgesamt 6 Anteile (1+2+3) = 100%. Das heißt für die einzelnen Regeln erfolgt eine Gewichtung wie folgt:

Regel 1 = 1/6 = 17%
Regel 2 = 2/6 = 33%
Regel 3 = 3/6 = 50%

Die Gewichtungsangabe wird im Reiter „Regeln“ bei „Gewichtung“ als Anteil an der Gesamtqualität angegeben.



hat filterndes Element
hat gewichtendes Element: Regel 1
Gewichtung: 1
hat gewichtendes Element: Regel 2
Gewichtung: 2
hat gewichtendes Element: Regel 3
Gewichtung: 3

Zu beachten ist hierbei auch der Unterschied zwischen Und-Gruppen und Oder-Gruppen: Bei Und-Gruppen müssen alle enthaltenen Regeln zutreffen. Trifft eine Regel nicht zu, fällt die Qualität unter 100% und erhält damit am Ende als Konsequenz eine Qualität von 0%. Im Grunde spielen bei Und-Gruppen Gewichte also keine Rolle. In Oder-Gruppen fallen die Qualitätswerte wie oben beschrieben ins Gewicht. Die anteiligen Gewichte der Kindelemente werden aufaddiert.

1.13.3.4 Rule types

Jeder Regelsatz besteht aus mindestens einer Regel. Jede Regel hat einen Namen und kann eine Beschreibung erhalten. Zu Testzwecken kann sie auch auf „deaktiviert“ gesetzt werden.

Es gibt 5 verschiedene Regeltypen:

- Attributvergleich
- Kardinalität
- Relationsziel
- Relationszielvergleich
- Vergleich durch Suche

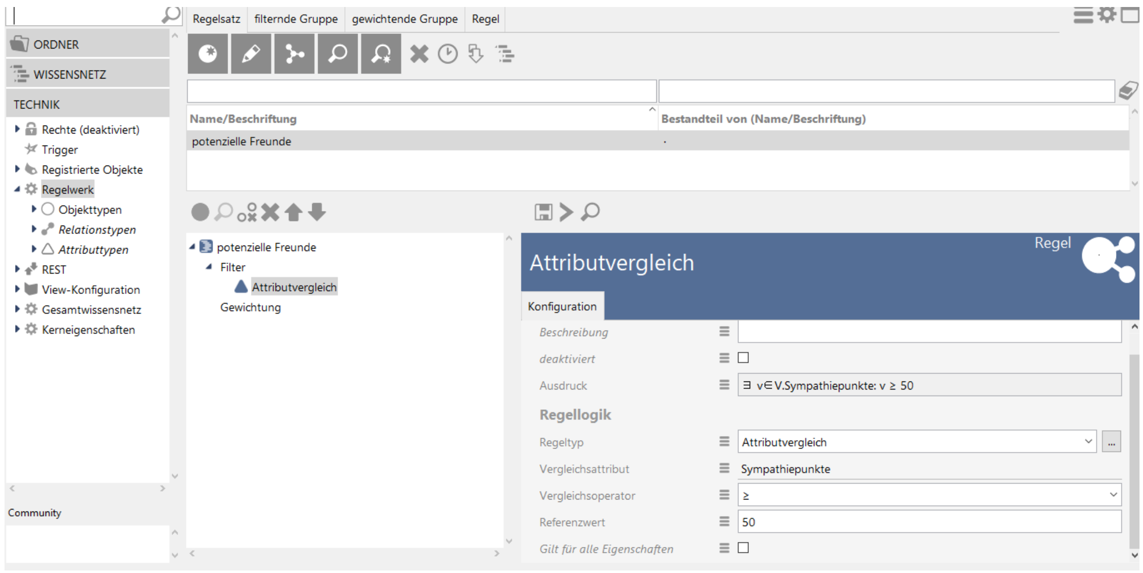
Sobald ein Regeltyp ausgewählt ist, wird die entsprechende Konfiguration zu diesem Regeltyp sichtbar (Regellogik).

Bei „Ausdruck“ wird die Regel als Prädikatenlogik abgebildet. Sie dient nur der Information und kann nicht bearbeitet werden.

1.13.3.4.1 Attribute comparison

Ein Attributvergleich vergleicht entweder ein Attribut der Vergleichsmenge mit einem Attribut der Referenzmenge oder das Attribut der Vergleichsmenge muss als Bedingung einen festen Wert erfüllen. Für den Vergleich der Attribute, bzw. den Vergleich zu einem festen Wert muss ein Vergleichsoperator gewählt werden (< , =, >, != , <= , >=).

Will man zum Beispiel ausdrücken, dass nur die Menschen als potenzielle Freunde vorgeschlagen werden, die mindestens 50 Sympathiepunkte haben, so muss das Vergleichsattribut „Sympathiepunkte“ sein und der Referenzwert „50“.



Name/Beschriftung	Bestandteil von (Name/Beschriftung)
potenzielle Freunde	-

Attributvergleich

Beschreibung	
deaktiviert	☐
Ausdruck	∃ v∈V.Sympathiepunkte: v ≥ 50
Regellogik	
Regeltyp	Attributvergleich
Vergleichsattribut	Sympathiepunkte
Vergleichsoperator	≥
Referenzwert	50
Gilt für alle Eigenschaften	☐

Hinweis: Nicht alle Operatoren machen für alle Attributtypen Sinn (z.B. können Intervall-Attribute auch überschneidende Operatoren haben). Hier ist der Regeleditor in der aktuellen Version noch nicht fertig.

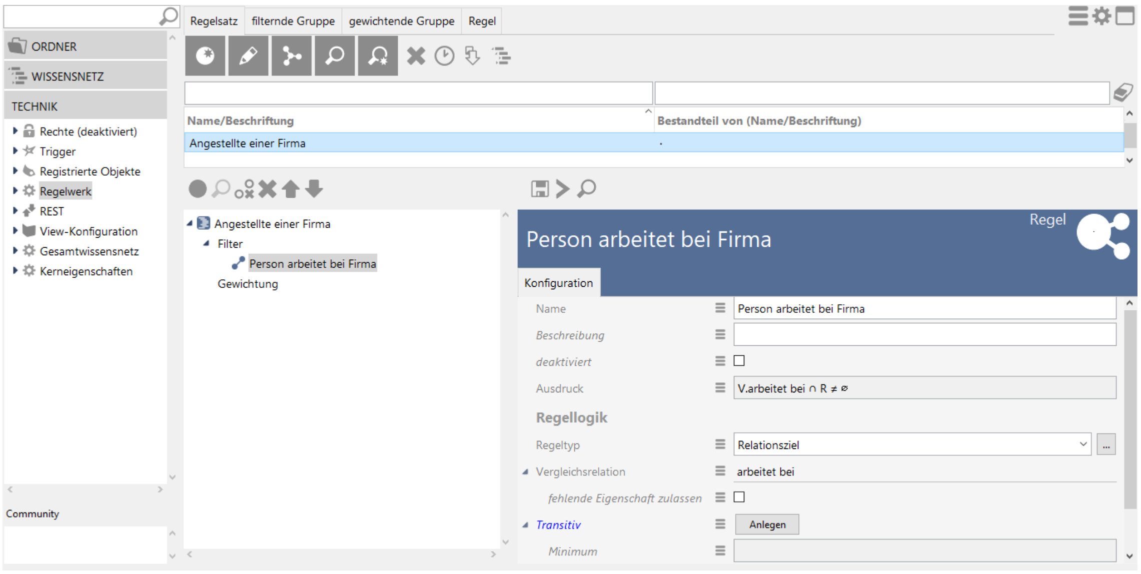
„Gilt für alle Eigenschaften" muss angehakt werden, wenn das zu vergleichende Attribut mehrfach vorkommt und alle Werte gleich sein sollen. Ist es nicht angehakt, reicht es der Regel, wenn einer der Werte gleich ist. Die Bedingung gilt nur in einer Richtung. So müssen alle Attributwerte des Vergleichsobjektes denen des Referenzobjektes entsprechen, aber nicht alle Attributwerte des Referenzobjektes auch denen des Vergleichsobjektes.

1.13.3.4.2 Cardinality

Mit dem Regeltyp „Kardinalität“ kann das Vorkommen einer bestimmten Eigenschaft am Vergleichsobjekt gezählt werden. Die Eigenschaft muss über „Eigenschaftstyp“ angegeben werden. Mit dem Kardinalitätsoperator gibt man an, ob die Eigenschaft kleiner gleich, exakt gleich oder größer gleich dem bei „Kardinalität“ eingegebenen Wert vorkommen soll.

1.13.3.4.3 Relation target

Beim Regeltyp „Relationsziel” wird über eine Vergleichsrelation geprüft, ob das Relationsziel der Vergleichsrelation in der Referenzmenge enthalten ist. Die Vergleichsrelation kann dabei transitiv unter Angabe eines Minimums und Maximums abgelaufen werden.



Name/Beschriftung	Bestandteil von (Name/Beschriftung)
Angestellte einer Firma	·

Angestellte einer Firma
Filter
Person arbeitet bei Firma
Gewichtung

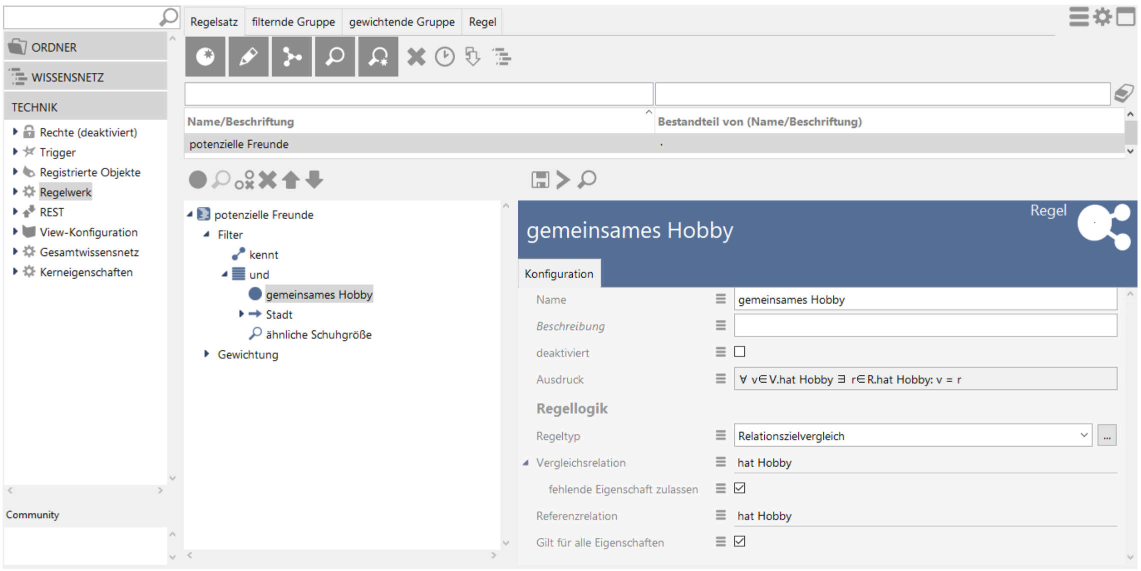
Person arbeitet bei Firma — Regel
Konfiguration

Name	Person arbeitet bei Firma
Beschreibung	
deaktiviert	☐
Ausdruck	V.arbeitet bei ∩ R ≠ ∅
Regellogik	
Regeltyp	Relationsziel
Vergleichsrelation	arbeitet bei
fehlende Eigenschaft zulassen	☐
Transitiv	Anlegen
Minimum	

„Fehlende Eigenschaft zulassen" ist ein Metaattribut an der Vergleichsrelation. Es muss angehakt werden, wenn man möchte, dass die Regel auch erfüllt ist, wenn das Vergleichsobjekt die Vergleichsrelation gar nicht besitzt.

1.13.3.4.4 Relation target comparison

Der Relationszielvergleich ist der am häufigsten verwendete Regeltyp. Hiermit kann geprüft werden, ob das Relationsziel sowohl das Ziel der Vergleichsrelation, als auch das Ziel der Referenzrelation ist. In unserem Beispiel können wir darüber abgleichen, ob Start- und Zielperson das gleiche Hobby haben.



Name/Beschriftung	Bestandteil von (Name/Beschriftung)
potenzielle Freunde	·

potenzielle Freunde
Filter
kennt
und
gemeinsames Hobby
Stadt
ähnliche Schuhgröße
Gewichtung

gemeinsames Hobby — Regel
Konfiguration

Name	gemeinsames Hobby
Beschreibung	
deaktiviert	☐
Ausdruck	∀ v∈V.hat Hobby ∃ r∈R.hat Hobby: v = r
Regellogik	
Regeltyp	Relationszielvergleich
Vergleichsrelation	hat Hobby
fehlende Eigenschaft zulassen	☑
Referenzrelation	hat Hobby
Gilt für alle Eigenschaften	☑

Alternativ zu einer Referenzrelation kann auch ein festes Referenzobjekt angegeben werden. Möchte man in unserem Beispiel ausdrücken, dass die Wohnorte der Personen in Deutschland liegen sollen, so muss man als Vergleichsrelation „liegt in" und als Referenzobjekt „Deutschland" angeben.

„Fehlende Eigenschaft zulassen" ist ein Metaattribut an der Vergleichsrelation. Es muss angehakt werden, wenn man möchte, dass die Regel auch erfüllt ist, wenn das Vergleichsobjekt die Vergleichsrelation gar nicht besitzt.

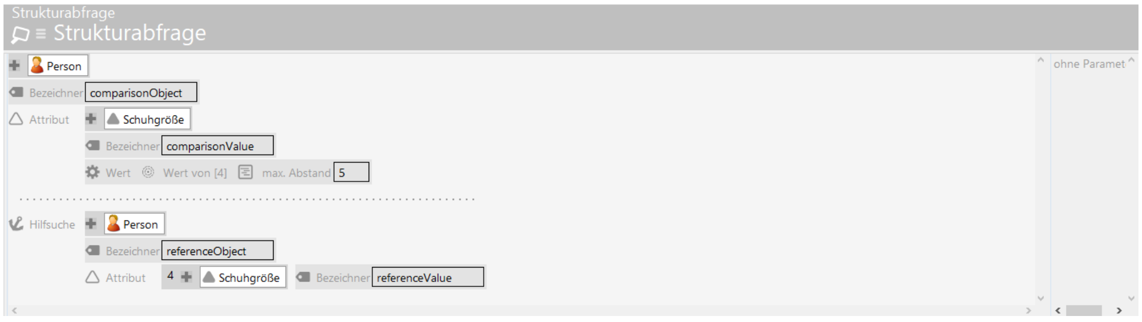
„Gilt für alle Eigenschaften” muss angehakt werden, wenn die zu vergleichende Relation mehrfach vorkommt und alle Relationsziele gleich sein sollen. Ist es nicht angehakt, reicht es der Regel, wenn eines der Relationsziele gleich ist. Die Bedingung gilt nur in einer Richtung. So müssen alle Relationsziele des Vergleichsobjektes auch ein Relationsziel des Referenzobjektes sein, aber nicht alle Relationsziele des Referenzobjektes auch ein Relationsziel des Vergleichsobjektes.

1.13.3.4.5 Comparison by structured queries

Um die ganze Mächtigkeit von i-views auch in den Regeln verwenden zu können, hat man schließlich auch noch die Möglichkeit eine Strukturabfrage für den Vergleich zweier Objekte und/oder zweier Werte zu nutzen. Dazu wählt man den Regeltyp „Vergleich durch Suche“. Wichtig hierbei ist, dass das Vergleichsobjekt durch den vordefinierten Bezeichner „comparisonObject“ und das Referenzobjekt durch den vordefinierten Bezeichner „referenceObject“ kenntlich gemacht werden. Der Vergleichswert muss durch den vordefinierten Bezeichner „comparisonValue“ und der Referenzwert durch den vordefinierten Bezeichner „referenceValue“ kenntlich gemacht werden. Möchte man, dass beim Testen des Regelsatzes die Tabelle zur Erklärung der Ursachen entsprechend befüllt ist, müssen die Bezeichner gesetzt sein.

Wenn die Regel nicht vergleicht, sondern generelle Aussagen trifft, dann können die Parameter weggelassen werden.

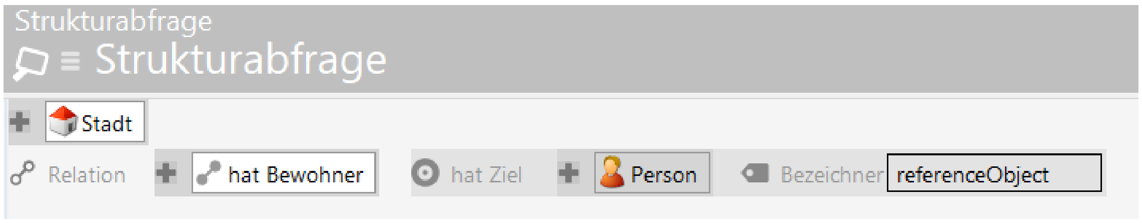
Eine Vergleichssuche, die die Schuhgrößen beider Personen vergleicht und dabei ausdrückt, dass sich diese um maximal 5 unterscheiden darf, würde wie folgt aussehen:


Strukturabfrage
Strukturabfrage
Person
Bezeichner comparisonObject
Attribut Schuhgröße
Bezeichner comparisonValue
Wert Wert von [4] max. Abstand 5
Hilfsuche Person
Bezeichner referenceObject
Attribut 4 Schuhgröße Bezeichner referenceValue
ohne Paramet

1.13.3.5 Derivation

Mit einer filternden oder gewichtenden Ableitung können - ausgehend von Vergleichs- und Referenzobjekten - weitere Objekte abgeleitet werden, auf die dann wiederum Regeln angewendet werden können. In der Produktwelt ist ein häufiger Anwendungsfall, dass Merkmale von Komponenten im Vergleich berücksichtigt werden sollen.

Um die abgeleiteten Objekte zu ermitteln, kann entweder jeweils eine Relation ausgehend vom Vergleichs- und/oder Referenzobjekt angegeben werden oder die abgeleiteten Objekte können über eine Strukturabfrage ermittelt werden. Wie auch bei dem „Vergleich durch Suche“ müssen dabei die vordefinierten Bezeichner verwendet werden. Beispiel:



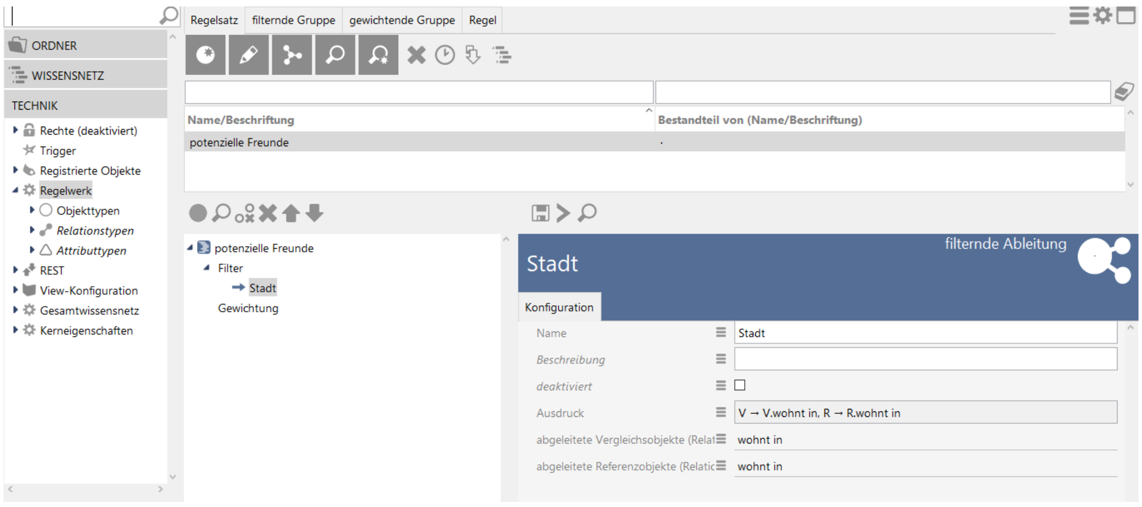
Stadt

Relation: hat Bewohner

hat Ziel: Person

Bezeichner: referenceObject

Angenommen, wir wollen als erste Bedingung für unsere Suche nach potenziellen Freunden ausdrücken, dass beide Personen aus einem Ort kommen sollen, der in Deutschland liegt, dann müssen wir hierzu eine filternde Ableitung erstellen, die - von einer Person ausgehend - die Stadt ermittelt. Zu der Stadt kann man dann in einer darunterliegenden Regel eine Bedingung ausdrücken.



Name/Beschriftung	Bestandteil von (Name/Beschriftung)
potenzielle Freunde	-

potenzielle Freunde
- Filter
  - Stadt
- Gewichtung

Stadt

Name	Stadt
Beschreibung	
deaktiviert	☐
Ausdruck	V → V.wohnt in, R → R.wohnt in
abgeleitete Vergleichsobjekte (Relat	wohnt in
abgeleitete Referenzobjekte (Relatic	wohnt in

1.13.3.6 Group

Eine filternde oder gewichtende Gruppe hat entweder den Gruppentyp „und“ oder „oder“. Eine „und“-Gruppe besagt, dass alle Regeln in dieser Gruppe zutreffen müssen. Eine „oder“-Gruppe besagt, dass nur eine der Regeln in der Gruppe zutreffen müssen, damit das Vergleichsobjekt als Ergebnis angezeigt wird. Ist keine Gruppe gesetzt, gilt per Default, dass alle Regeln zutreffen müssen („und“-Gruppe). Zu beachten ist das unterschiedliche Verhalten bei angegebenen Gewichtungen der enthaltenen Regeln (Siehe Kapitel Gewichtungen).

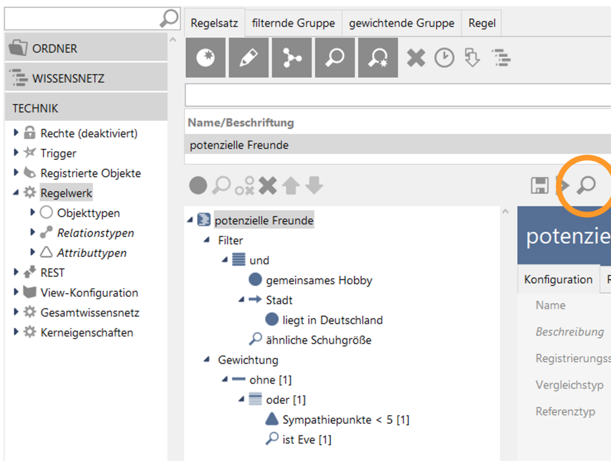
1.13.3.7 Subtraction

Ein filternder Abzug zieht alle Objekte von der Ergebnismenge ab, die als Ergebnis aus den darunter befindlichen Regeln hervorgehen.

Ein gewichtender Abzug zieht den Qualitätswert einer enthaltenen Regel vom Gesamtergebnis ab, wenn die Regel zutrifft. Wird zum Beispiel die Regel Sympathiepunkte $<$ 5 erfüllt und die Gewichtung der Regel beläuft sich auf 33% (Anteil von 1/3), so wird das Ergebnis nur noch 67% betragen.

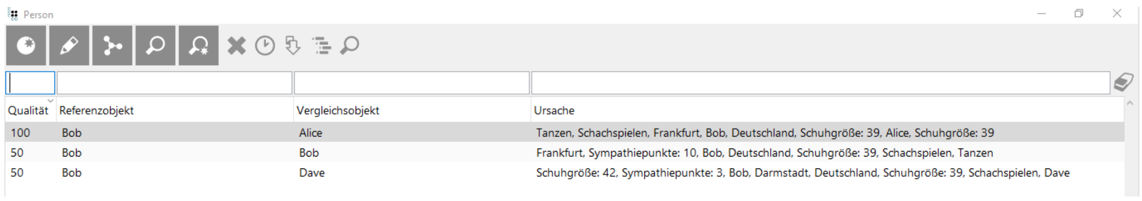
1.13.4 Testing rules

Zur Kontrolle der Ergebnisse kann die Suche geöffnet werden:



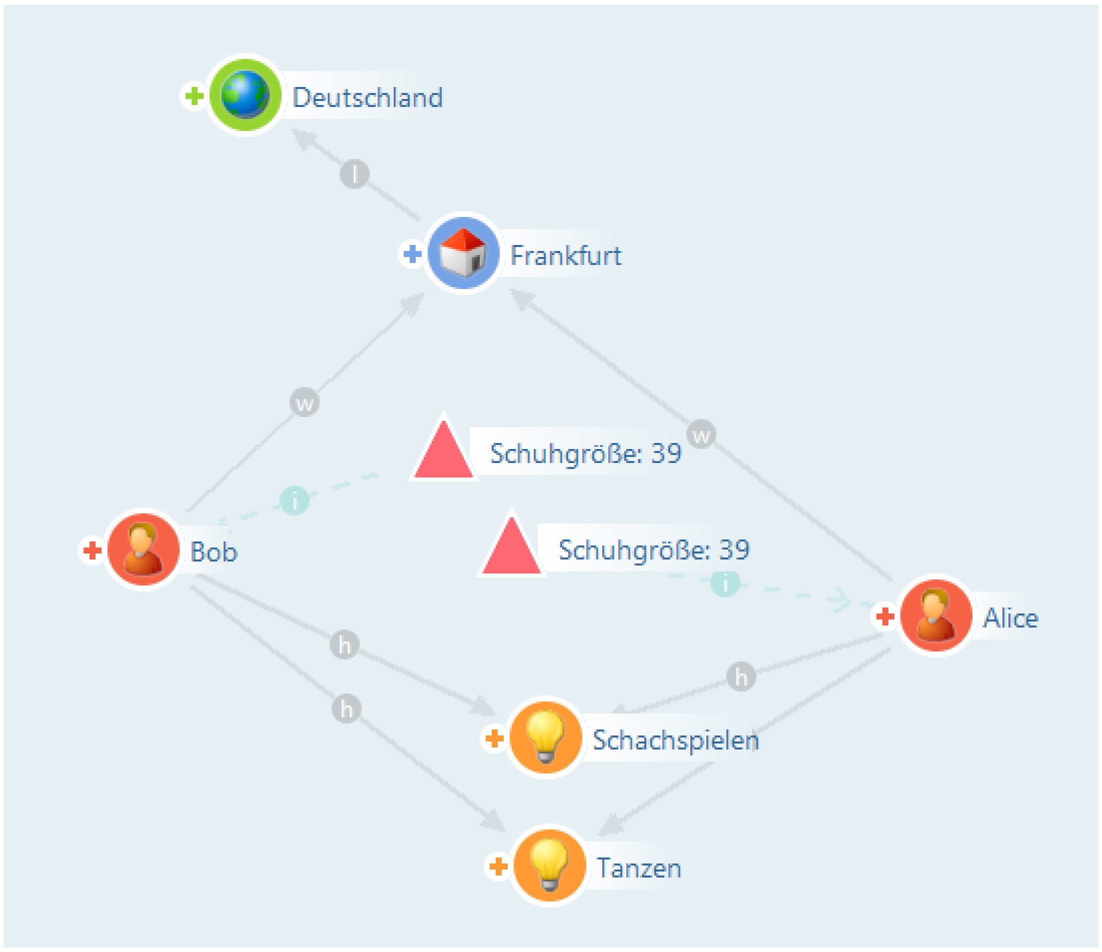
Hier muss ein Objekt ausgewählt werden, für welches der Regelsatz angewendet werden soll. Die Ergebnistabelle zeigt alle Ergebnisse unter Angabe von Qualität, dem gerade ausgewählten Referenzobjekt, dem gefundenen Vergleichsobjekt und den Ursachen.

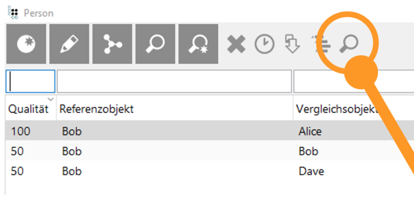
Hinweis: Solange keine gewichtete Regel enthalten ist, erhalten alle Treffer 100%.



Qualität	Referenzobjekt	Vergleichsobjekt	Ursache
100	Bob	Alice	Tanzen, Schachspielen, Frankfurt, Bob, Deutschland, Schuhgröße: 39, Alice, Schuhgröße: 39
50	Bob	Bob	Frankfurt, Sympathiepunkte: 10, Bob, Deutschland, Schuhgröße: 39, Schachspielen, Tanzen
50	Bob	Dave	Schuhgröße: 42, Sympathiepunkte: 3, Bob, Darmstadt, Deutschland, Schuhgröße: 39, Schachspielen, Dave

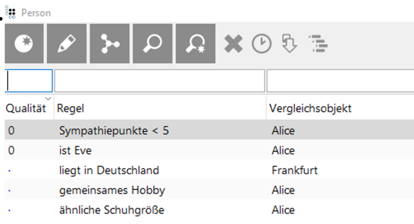
Die Treffererklärung lässt sich per Klick auf Graph auch grafisch visualisieren:





Qualität	Referenzobjekt	Vergleichsobjekt
100	Bob	Alice
50	Bob	Bob
50	Bob	Dave

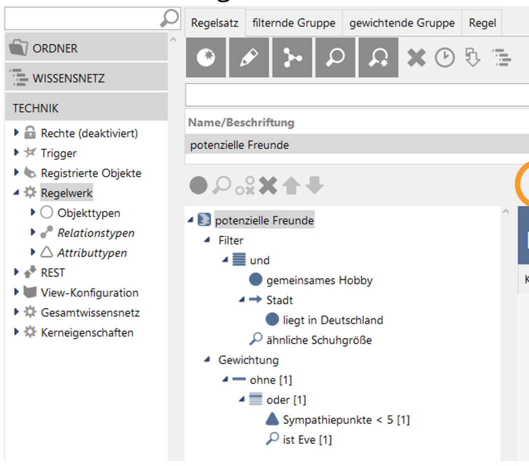
Ein Klick auf „Ursache“ öffnet die einzelnen Ursachen im Detail in einer Tabelle.



Qualität	Regel	Vergleichsobjekt
0	Sympathiepunkte < 5	Alice
0	ist Eve	Alice
·	liegt in Deutschland	Frankfurt
·	gemeinsames Hobby	Alice
·	ähnliche Schuhgröße	Alice

1.13.5 Executing rules

Regelsätze können an verschiedenen Stellen verwendet werden. Doch damit ein Regelsatz verwendet werden kann, muss dieser zunächst gespeichert werden.



Sobald man auf den Speichern-Button klickt, wird der Regelsatz in eine Such-Pipeline umgewandelt und als solche gespeichert. Zur Identifikation der Such-Pipeline wird der Registrierungsschlüssel verwendet. Es kann nur gespeichert werden, wenn ein Registrierungsschlüssel angegeben ist. Bei jeder Änderung am Regelsatz muss erneut gespeichert werden, damit sich die Such-Pipeline aktualisiert.

Im Viewconfigmapper kann die Such-Pipeline dann zum Beispiel als Suche in einer Zielauswahl verwendet werden, um die Ergebnismenge einzuschränken. Die Ergebnistabelle und die Treffererklärung kann ebenfalls verwendet werden, um sie beispielsweise in einer Webapp darzustellen.

2 Admin Tool

You can use the Admin tool to create new semantic networks, manage all semantic networks of a mediator and configure individual semantic networks.

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 an ini file.

The content of the ini file "admin.ini" for starting the admin tool with English UI is as follows:

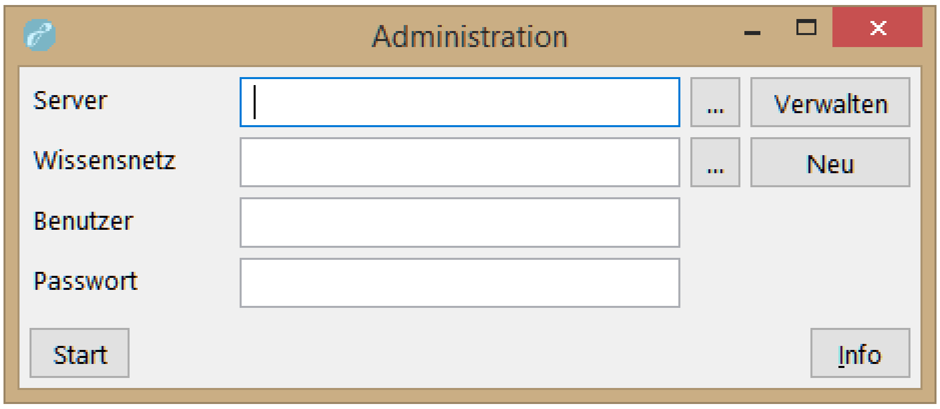
```
[Default]
language=eng
```

Please obey that without further configuration, the ini file needs to be located in the same directory as the admin tool itself to take effect.

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.

2.2 Launch window

After the Admin tool (Windows: *admin.exe*, Mac OS: *admin*, Linux: *admin-64.im*) has started, the **Start window** appears.



2.2.1 Server

The URL of the server is entered in the free text field **Server**. (If no protocol is specified, the protocol cnp:// is used). Valid URLs use one of the protocols *[cnp://,cnps://,http:// or https://] followed by [computer name or IP address]:[Port number]*. This format corresponds to the interface setting on the mediator.

If the mediator that is used to administrate the semantic networks 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 semantic networks 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 which authentication using the server password is required.

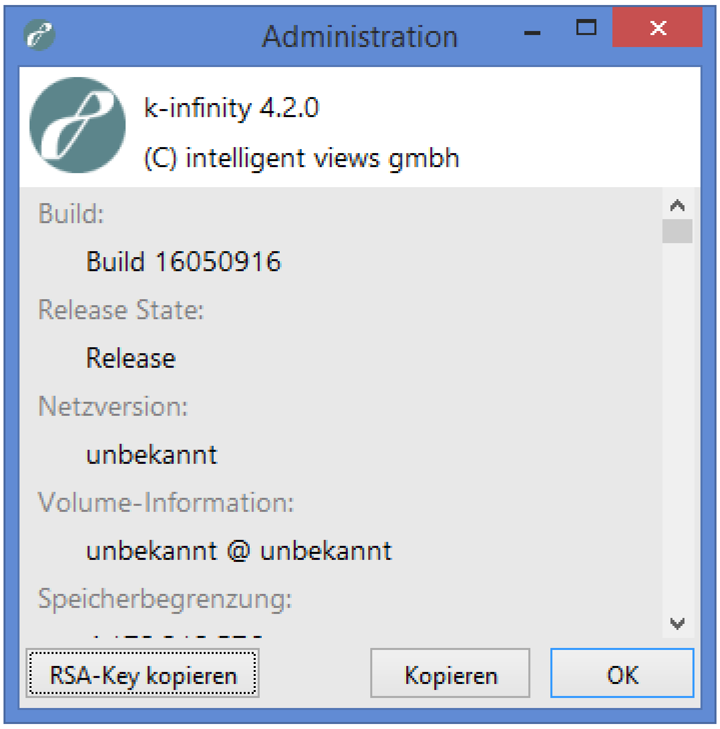
2.2.2 Knowledge network

The semantic network that is to be administrated is specified in the free text field **Semantic network**.

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 semantic networks, the user may be prompted to enter the server password.

2.2.3 Information

You can use the **Info** button to retrieve version-specific information in a separate window via the Admin tool.


Administration

k-infinity 4.2.0

(C) intelligent views gmbh

Build:

Build 16050916

Release State:

Release

Netzversion:

unbekannt

Volume-Information:

unbekannt @ unbekannt

Speicherbegrenzung:

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 limit*),
- The version number and the digital finger print of the execution environment used by the Admin tool (*VM version*),
- The language setting active in the operating system (*Locale*),
- The fonts provided in the Admin tool (*Fonts*),
- The semantic network components including version numbers supplied with the Admin tool (*software components*) and
- The small talk packages including version number used in the Admin tool (*Packages*).

Information on the *network version* and *volume information* is not decisive here.

The information is output in an invisible text field, which 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.

The **Copy** button copies all information to the clipboard of the operating system.

The **Copy RSA key** button copies the unique key for each compiled Admin tool to the clipboard of the operating system. This key can be entered into the initialization file of a mediator (default file name *mediator.ini*) and thus restricts this mediator s access via an Admin tool to Admin tools with this specific key.

The **OK** button enables you to return to the start window.

2.2.4 Manage, New and Next

New forwards to semantic network generation.

Administrate forwards to the server administration.

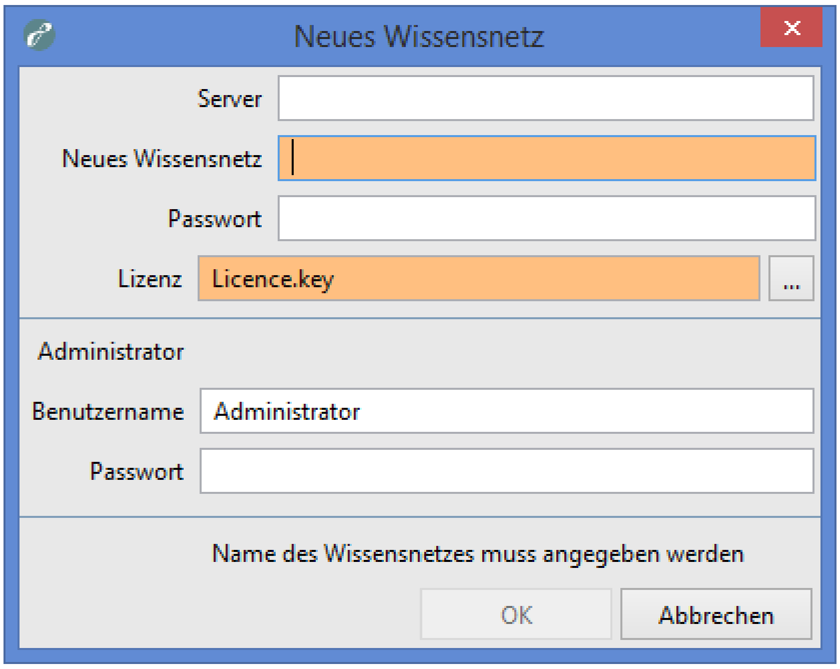
Next forwards to the individual network administration. The entries **user name** and **password** are used for this for logging in with an administrator account.

2.2.5 End

The **End** button closes the Admin tool.

2.3 Create a new knowledge network

A new semantic network is created via a separate **network creation window**. It can be reached via the **New** button on the **start screen**. Any inputs in the **Server** and **semantic network** free text fields are ignored.


Neues Wissensnetz

Server	
Neues Wissensnetz	
Passwort	
Lizenz	Licence.key

Administrator

Benutzername	Administrator
Passwort	

Name des Wissensnetzes muss angegeben werden

2.3.1 Server

The name or the IP address of the computer is specified in the free text field **Server** on which the mediator is running, and which should be used to create the new semantic network. If this cannot be reached using the default port, then a correct port number must also be named. The input form in this case is *[Computer name or IP address]:[Port number]*.

If the mediator that should be used to create the new semantic network is running on the same computer as the Admin tool, it can also be addressed using the computer name *local-host*.

If the field remains blank, the semantic network is generated in the *Volumes* subfolder in direct relation to the position of the Admin tool.

2.3.2 New knowledge network

The name of the semantic network is specified in the free text field **New semantic network**. The characters allowed for this purpose are specified by the file system of the operating system on which the semantic network is to be stored. To ensure that the data can also be stored in different file systems, the following applies:

- 64 characters maximum
- No blank spaces at the start or end
- Characters permitted: upper and lower scale Latin letters, numbers, spaces !@#$%&'()+-.[]∧_'{}∼Œœ and ASCII characters 160-255
- 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 semantic network or by changing file and directory names. If you make a change, keep in mind that the name of the semantic network might be used in initialization files and that the license might have been adapted to this.

2.3.3 Password (mediator)

The mediator supports authentication via a password. If a password has been set for the mediator that will be used to create the new semantic network, that password must be entered in the **Password** free text field, which is located between the **New semantic network** and **License** fields. If no password has been assigned, the free text field must remain empty.

2.3.4 License

A semantic network must have a valid license so that Knowledge Builder and other software components (with the exception of the Admin tool) can work with it. You can 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 semantic network 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 has administrator rights.

2.3.6 Password (user)

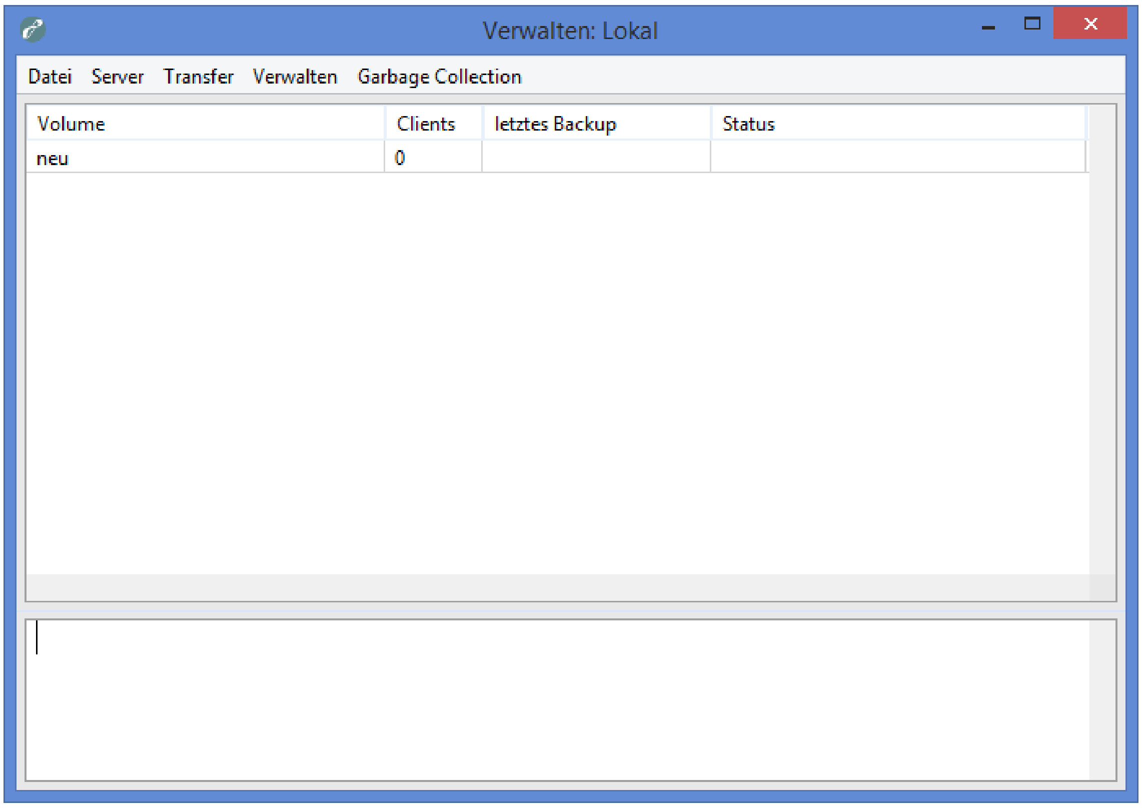
In the **Password** free text field, you can enter a password for the first user registered in the semantic network. This password will be required later on when this user attempts to log in to the Knowledge Builder or the Admin tool for the semantic network.

2.3.7 Ok and abort

The **OK** button generates the semantic network, 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 network administration allows the administration of all semantic networks 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 of the **start screen** is necessary for this. Any entries in the **semantic network** of the **start screen** are ignored. If the semantic networks to be administrated are addressed using a mediator, the correct mediator password must also be specified in a separate window.



Volume	Clients	letztes Backup	Status
neu	0		

The **overall network administration window** is comprised of a **network overview** in the form of a table, a **message field** and a **menu line**.

2.4.1 Network overview

The **network overview** in the form of a table provides details about

- the name (*volume*)
- the number of users currently active (*clients*),
- the date and time of the last backup (*last backup*) and
- the last status message (*status*) of the respective network.

The individual columns can be sorted by clicking on the head of the column.

The data are only updated when triggering operations, and are therefore not always up-to-date. A manual update can be forced at any time using the menu item **Server** –> **Update**.

2.4.2 Message field

The **Message field** outputs all status reports for all networks. Status reports are created when activities are triggered in the Admin tool. They are lost when the Admin tool is closed. To prevent this, they can be exported via the menu option **File** –> **Administration log**. The **Message field** can be edited, but changes are ignored during export.

2.4.3 Menu line

The **menu line** consists of the following menu tabs:

2.4.3.1 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.

Log off closes server administration and opens the log-in window again.

Exit closes server administration

2.4.3.2 Server

Update reloads the data collected in the **network overview** in the **whole network administration window**.

Re-import ini file makes the server import its ini file again. Here, not all options can be updated during operation. The server outputs a message about updated options.

Download log generates a copy of the mediator log file usually stored in the folder of the connected mediator (default file name: *mediator.log*). 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 mediator s activities from its first commissioning.

Server connections shows the numbers and IP addresses of all software components (except blob service) currently registered in semantic networks via the mediator in the **message field** and groups them according to semantic networks. The number is generated sequentially by the mediator and re-assigned whenever a new software component registers.

2.4.3.3 Transfer

Download volume creates a copy of the semantic network selected in the **network overview** and saves it locally in the *volumes* subfolder that is located relative to the position of the Admin tool. A new name can be assigned to this copy in a separately appearing free text field.

Copy volume creates a copy of the semantic network selected in the **network overview** and saves it in the same folder as the original network. A new name must be assigned to this copy in a separately appearing free text field.

Upload volume creates a copy of a selected local semantic network and saves it in the *volumes* subfolder in the location relative to the connected mediator. A new name can be assigned to this copy in a separately appearing free text field. The local semantic network, 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.

Replace volume creates a copy of the selected local semantic network and uses it to overwrite the semantic network selected in the **network overview**. In the process, the copy is given the name of the semantic network it has replaced. The local semantic network, 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 semantic network 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.

2.4.3.4 Administrate

Open Admin tool logs on to the selected volume with the Admin tool. No authentication in the volume is required - mediator authentication is sufficient.

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 semantic network selected in the **network overview** and saves it in the *backup* folder, which lies in a parallel position relative to the position of this semantic network. There a separate subfolder is created for each backup; its name contains the time, precise to the second, at which this copy was created.Every backup is a full copy of the original network.

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.

Restore backup creates a copy of a selected backup and saves it in the same folder as the semantic networks shown in the **network overview**. A new name must be assigned to this copy in a separately appearing free text field. To select the backup, which must be stored in a subfolder of the *backup* folder, which in turn is parallel to the position of the semantic networks displayed in the **network overview**, two separate selection windows must be navigated: in the first, the semantic network must be selected; in the second, the version must be selected from a list sorted by creation date.

Delete backup deletes a selected backup. To select this backup, which must be stored in a subfolder of the *backup* folder, which in turn is parallel to the position of the semantic networks displayed in the **network overview**, two separate selection windows must be navigated: in the first, the semantic network must be selected; in the second, the version must be selected from a list sorted by creation date.

The block assignment of clusters and blobs within the original semantic network is not modified when a semantic network copy is created. The copy process initiated by the backup operations therefore creates no compression effect.

Delete volume deletes the semantic network selected in the **network overview**.

Compress volume reduces the amount of space required by the semantic network selected in the **network overview**. This is done by removing unused interior blocks. 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 storage updates the version of the block file system of the semantic network selected in the **network overview**. If the semantic network is addressed via a mediator, the version it contains is used; otherwise, the version supplied in the Admin tool is used. The update makes it possible to save index structures more quickly. It is possible for semantic networks whose i-views core component is older than 4.2.

2.4.3.5 Garbage Collection

Garbage collection is a procedure that deletes objects that are no longer referenced (according to a programming terminology reading) from the semantic network and thereby minimizes the memory usage of the semantic network. Use of the garbage collection requires that the semantic network that is to be cleaned up is activated via a mediator.

Start launches a new garbage collection for the semantic network selected in the **network overview** or continues a paused garbage collection. No confirmation is sent when the process is completed. You can determine its progress via the **Status** menu option.

Pause interrupts the execution of the active garbage collection for the semantic network selected in the **network overview**.

Stop terminates the execution of the active garbage collection for the semantic network selected in the **network overview**.

Status writes the current status of the garbage collection for the semantic network selected in the **network overview** to the status column of the **network overview** and to the **message field**. If garbage collection is active, feedback on its progress is provided in percent.

2.5 Individual network administration

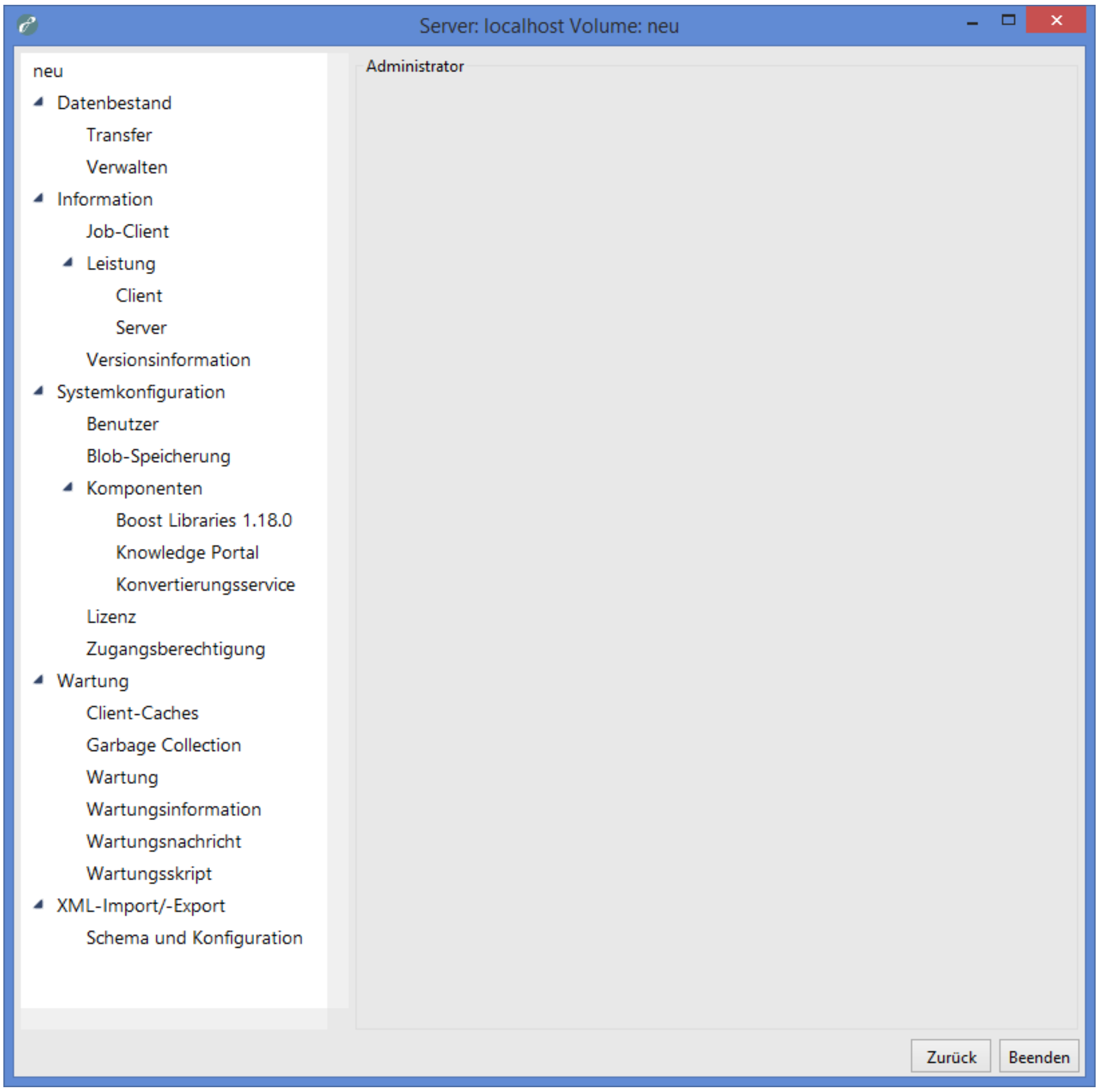
Individual network administration allows you to manage an individual semantic network. It can be reached via the **Start** button on the start screen. This requires the corresponding entries in the fields **Server**, **Semantic network**, **User** and **Password** of the start screen.

2.5.1 User authentication

To access the **network administration window** the user needs to log on with administrator rights.

If you no longer have access to the semantic network, you can access the semantic network through authentication on the server by logging on to the **server administration**.

2.5.2 Individual network administration window



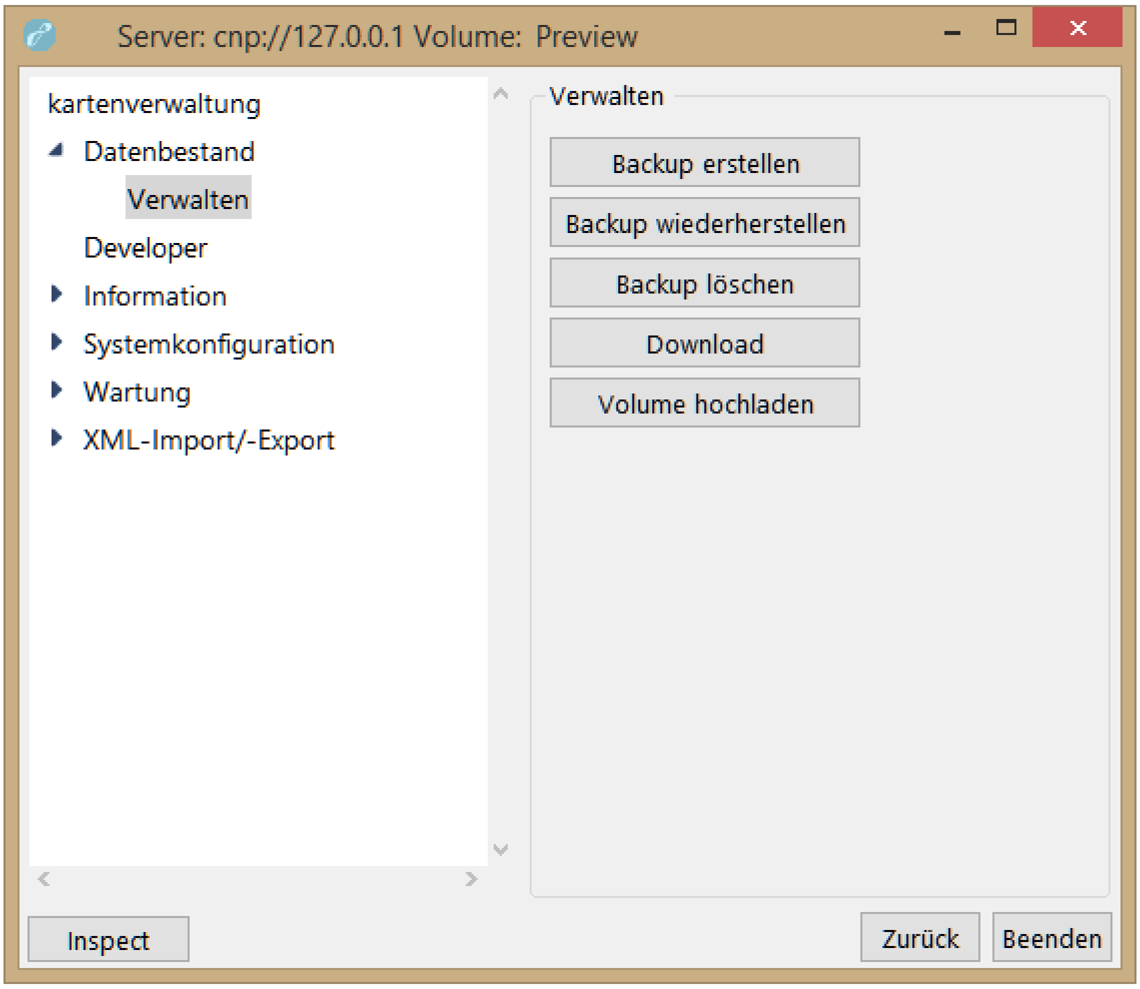
The **network administration window** has a menu list with a multilevel structure on the left, and an operation window on the right. The content of the operation window depends on the menu option selected in the menu list.

The **Back** button returns you to the start window.

The **Exit** button closes the Admin tool.

If the semantic network to be administrated is addressed without a mediator, other users cannot access the semantic network via the Knowledge Builder or another instance of the Admin tool for as long as the **network administration window** is open.

2.5.2.1 Manage the data



Create backup creates a backup of the semantic network and saves it (on the server) in the *backup* folder, which lies in a parallel position relative to the position of this semantic network. There a separate subfolder is created for each backup; its name contains the time, precise to the second, at which this copy was created.Every backup is a full copy of the original network.

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.

Restore backup replaces the current semantic network with a backup (afterwards you are logged off automatically). The backup is selected according to the time of the relevant backup.

Delete backup deletes an individual backup of this semantic network.

The block assignment of clusters and blobs within the original semantic network is not modified when a semantic network copy is created. The copy process initiated by the backup operations therefore creates no compression effect.

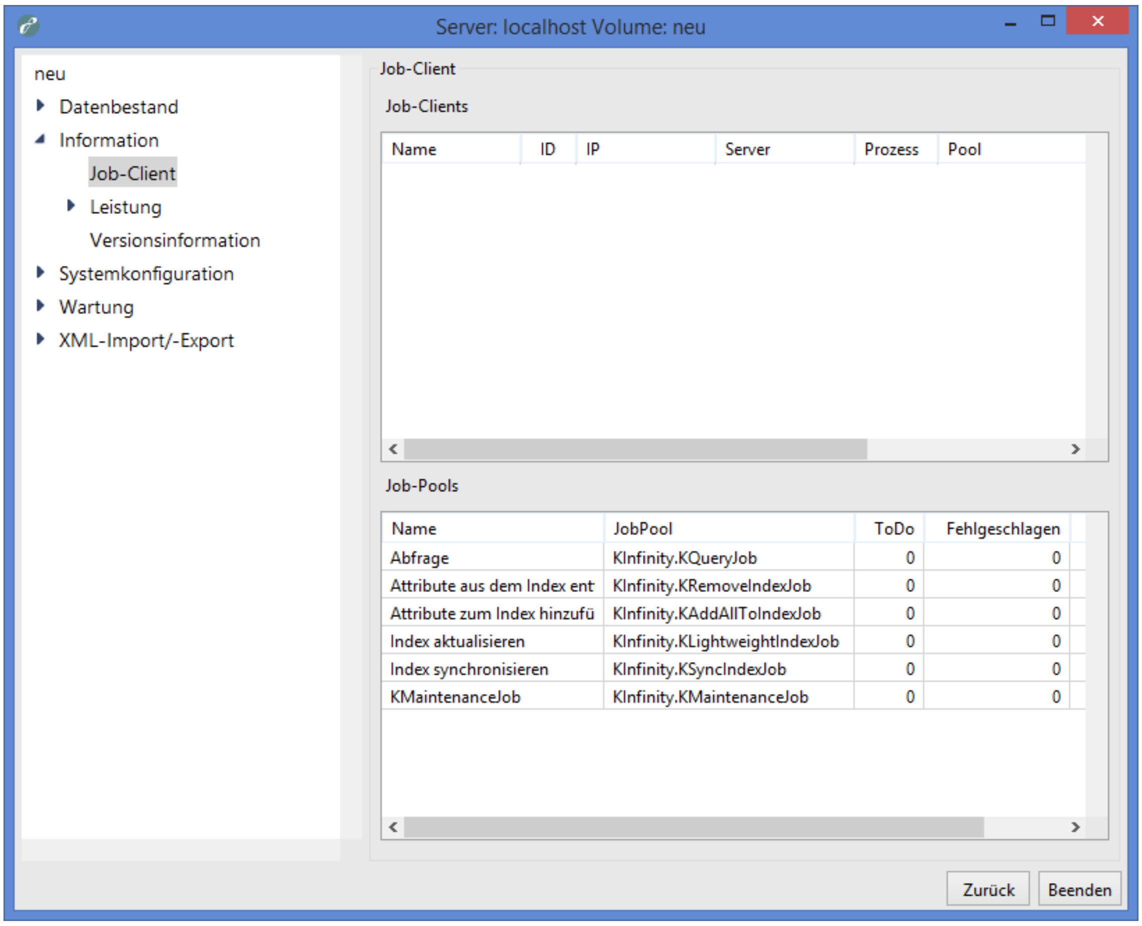
Download creates a copy of the semantic network and saves it locally in the *volumes* subfolder that is located relative to the position of the Admin tool. A new name can be assigned to this copy in a separately appearing free text field.

Upload volume transfers a locally stored network and replaces the current semantic network with this network (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, and querying semantic networks and executing scripts, some of these processes can be optionally performed by Job-Clients while others are exclusively performed as jobs by Job-Clients (a software service). To do so, the user interface of the Knowledge Builder or a script must be used to trigger a job, or the conditions for triggering it must be defined. Moreover, at least one Job-Client must be configured and started which can perform jobs of this job type (job pool). 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.



Name	ID	IP	Server	Prozess	Pool

Name	JobPool	ToDo	Fehlgeschlagen
Abfrage	KInfinity.KQueryJob	0	0
Attribute aus dem Index ent	KInfinity.KRemoveIndexJob	0	0
Attribute zum Index hinzufü	KInfinity.KAddAllToIndexJob	0	0
Index aktualisieren	KInfinity.KLightweightIndexJob	0	0
Index synchronisieren	KInfinity.KSyncIndexJob	0	0
KMaintenanceJob	KInfinity.KMaintenanceJob	0	0

The **Job-Clients overview** table shows the following for each job-client that is currently running:

- its name in the format [Job-Client-Name]@[Mediator-Name] (*name*),
- its job-client number (*ID*),

- its IP address (*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 (*completed*).

The Job-Client number is generated sequentially by the mediator and a new number is assigned with each new log-in. 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** in the 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,

- 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 it (*Job-Clients*)

are named.

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:

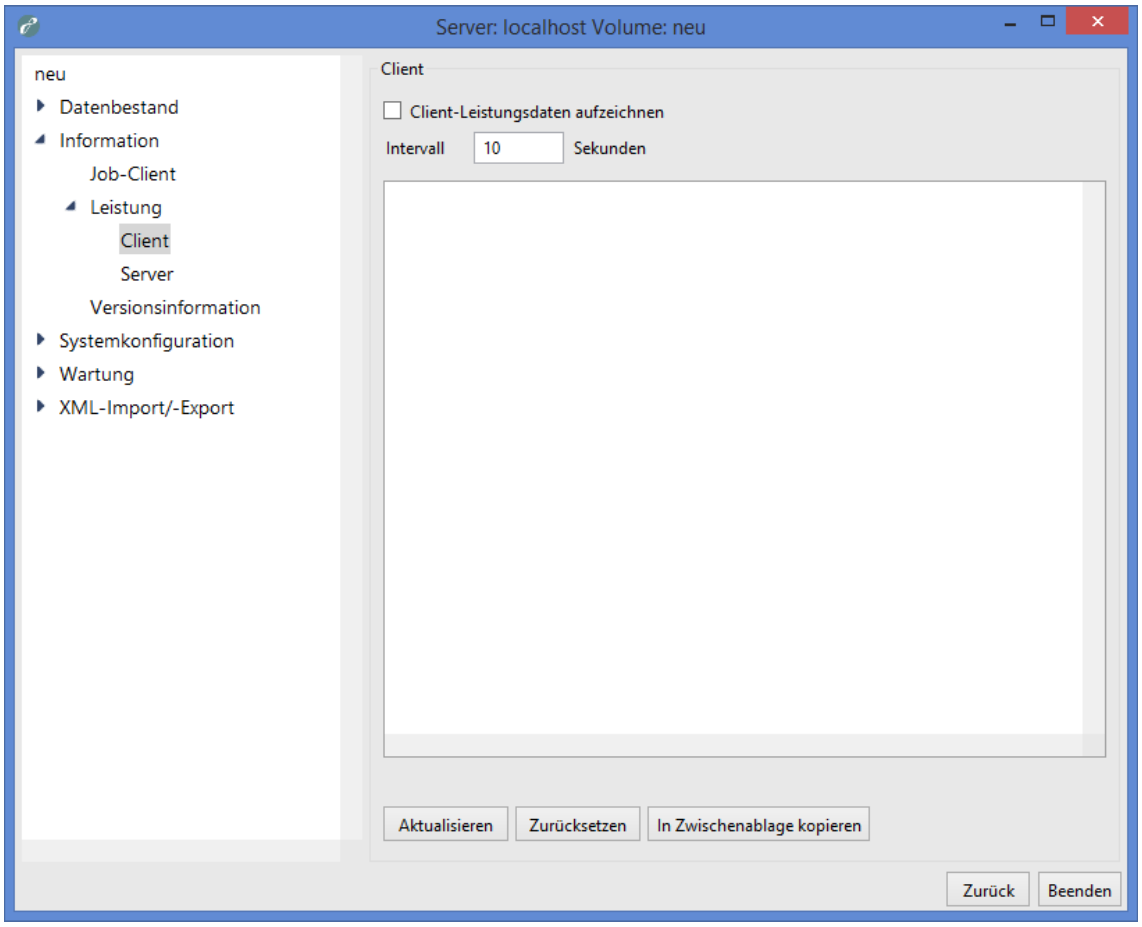
- **Empty job pool** deletes all uncompleted and failed jobs of the job type selected in the

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


Server: localhost Volume: neu

neu
- Datenbestand
- Information
  - Job-Client
  - Leistung
    - Client
    - Server
  - Versionsinformation
- Systemkonfiguration
- Wartung
- XML-Import/-Export

Client

☐ Client-Leistungsdaten aufzeichnen

Intervall 10 Sekunden

Record client performance data starts and ends the collection of diverse key performance indicators that are coupled to activities by the software components connected to the semantic network. These key performance indicators can be used for the performance analysis.

Interval sets the required time period in seconds until a software component sends another data packet with key performance indicators to the Admin tool. It cannot be changed after recording starts. The preset is 10 seconds.

The key performance indicators are output in nested list items 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 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.

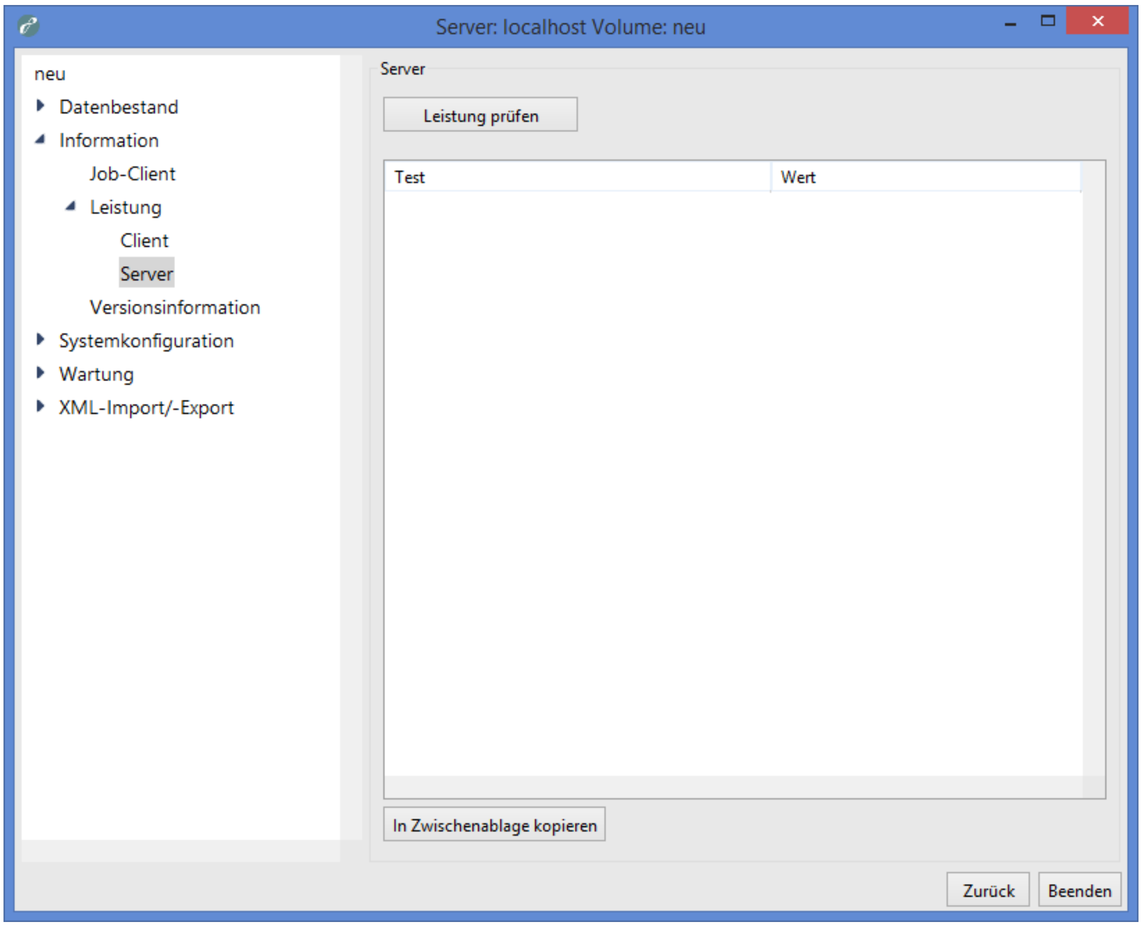
Double-clicking on a list item allows all key performance indicators stored below it to be shown at a glance in a separate window. There, they can be copied to the clipboard of the operating system (**Copy to clipboard** button) or be exported to any location as a text file that can be given any name using a saving dialog (**Save** button).

Update refreshes the key performance indicators shown in the **key performance indicator overview**.

Reset deletes the key 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.

Server



Check performance starts a test process that evaluates the performance of the mediator connected. This sends four requests to the mediator, and the responses sent to the Admin tool are evaluated. Measurements are taken of

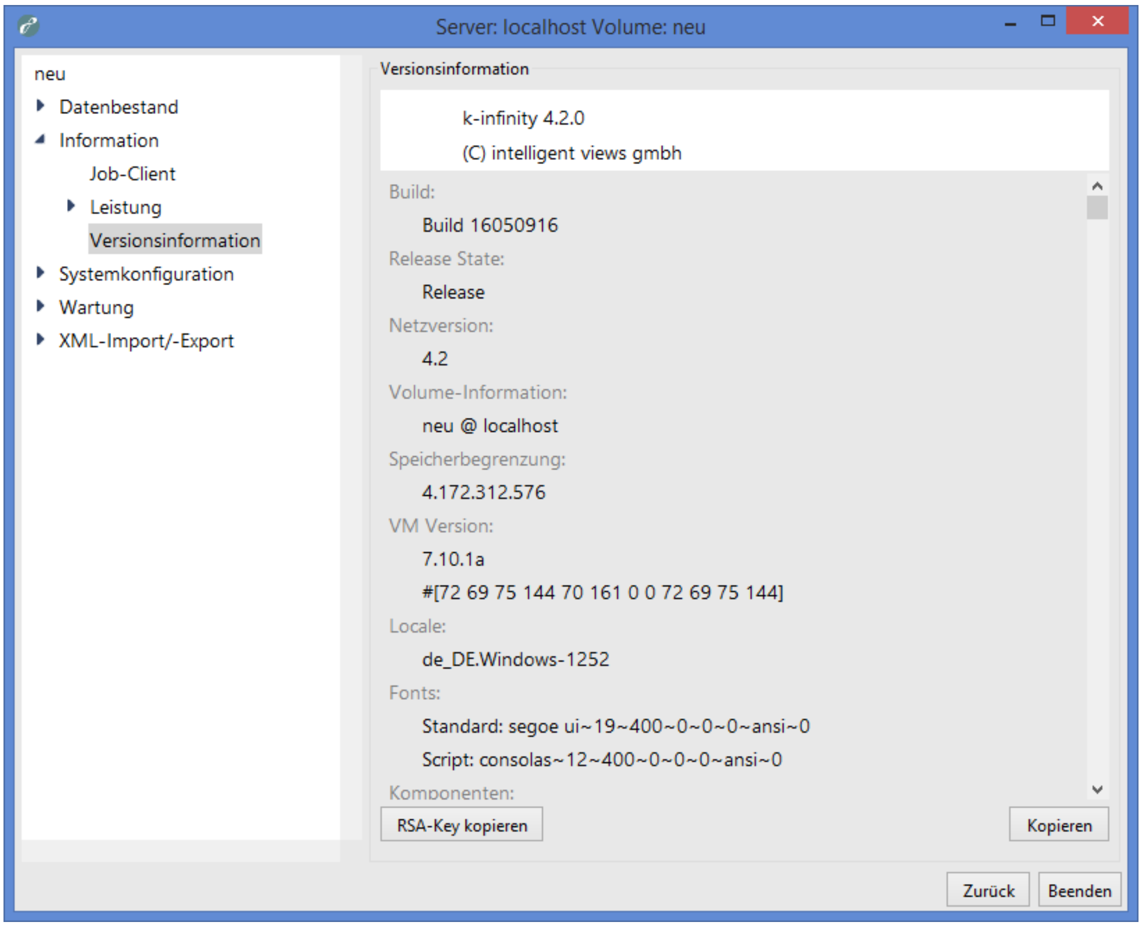
- the times until a small file is send back (*roundtrip: Blob*) and
- the result of an index search request (*roundtrip: RPC*) and
- the average transmission rate when sending several 1 MB files (*throughput: Blob (1.0 MB)*) and
- the average transmission rate when sending several 100 KB files (*throughput: Blob (100.0 KB)*).

The test results are written to the **results list** provided. The individual columns of the table can be sorted by clicking on the head of the column.

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 semantic network and Admin tool.



k-infinity 4.2.0
(C) intelligent views gmbh

Build:
Build 16050916
Release State:
Release
Netzversion:
4.2
Volume-Information:
neu @ localhost
Speicherbegrenzung:
4.172.312.576
VM Version:
7.10.1a
#[72 69 75 144 70 161 0 0 72 69 75 144]
Locale:
de_DE.Windows-1252
Fonts:
Standard: segoe ui~19~400~0~0~0~ansi~0
Script: consolas~12~400~0~0~0~ansi~0
Komponenten:

Specifically, you can retrieve:

- The version number of the Admin tool (*Build*),
- The publication status of the Admin tool (*Release*),
- The version number of the semantic network (the network version), the name of the semantic network and the mediator used (*volume information*),
- The maximum system memory in bytes that can be used by the Admin tool (*Memory limit*),
- The version number and the digital finger print of the execution environment used by the Admin tool (*VM version*),
- The language setting active in the operating system (*Locale*),
- The fonts provided in the Admin tool (*Fonts*),
- The semantic network components installed in the semantic network and their version numbers (*software components*) and
- The small talk packages including version number used in the Admin tool (*Packages*).

The information is output in an invisible text field, which 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.

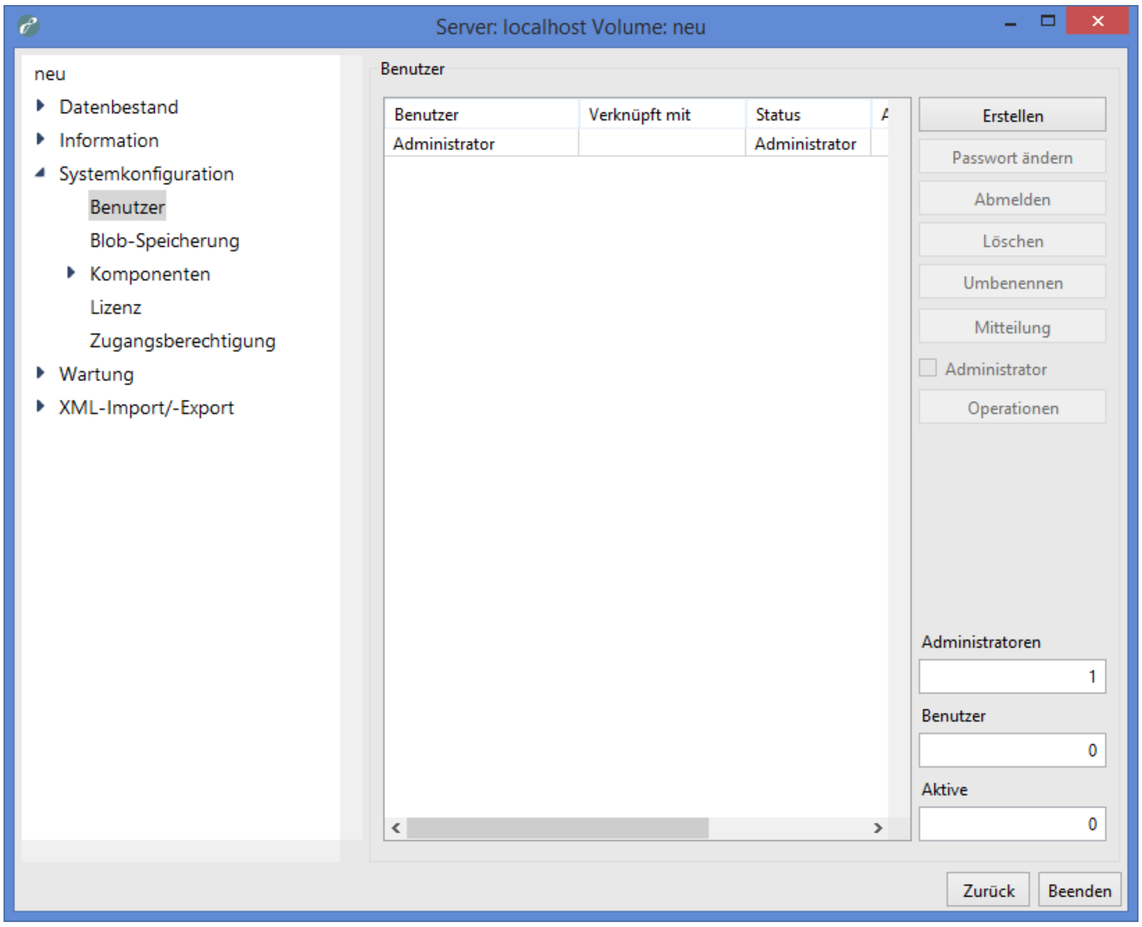
The **Copy** button copies all information to the clipboard of the operating system.

The **Copy RSA key** button copies the unique key for each compiled Admin tool to the clipboard of the operating system. This key can be entered into the initialization file of a mediator (default file name: *mediator.ini*) and thus restricts this mediator s access via an Admin tool to Admin tools with this specific key.

2.5.2.3 System configuration

2.5.2.3.1 User

The user administration compares the ones in the Knowledge Builder, with the exception that no links between users and objects of the user-generated subnet can be processed.



Benutzer	Verknüpft mit	Status
Administrator		Administrator

Administratoren: 1
Benutzer: 0
Aktive: 0

The **user overview** in the form of a table shows, for every user registered in the semantic network,

- the user name (*user*),

- the object of the user-generated subnet the user is linked to (*linked to*),
- which status the user currently has (*status*),
- on which date and at which time the user logged into the semantic network using the Knowledge Builder(*log-in date*) if the user is still logged in, and
- which method was used to encrypt the password (*password type*).

The individual columns of the table can be sorted by clicking on the head of the column.

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 semantic network 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.

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.

Log out logs out the user selected in the **user overview** from the semantic network following a security confirmation. To ensure this operation has its effect, this user must be currently logged into the semantic network using the Knowledge Builder.

Delete deletes the user 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.

Notification uses a free text field in a separate window to send a message to the user selected in the **user overview**. The message is buffered in the semantic network and appears to the user addressed in a separate window in the Knowledge Builder as soon as the user uses it to log into the semantic network. The user cannot reply to this message.

Administrator assigns the administrator rights to the user selected in the **user overview** , or takes them away. A user must have a password to obtain administrator rights. Once the user has administrator rights, deleting the password is then possible. At least one user must have administrator rights.

Operations opens a new window in which the user selected in the **user overview** can, from a list of operations, these being

- Create backup,
- Delete backup,
- Restore backup,
- Garbage collection,
- Copy,
- Download log,
- Download volume,
- Upload volume,
- Delete volume,

select those operations that this user may execute within the scope of individual network administration in future, without input of the mediator password. To confirm the selection, the correct mediator password must be entered in the free text field **Server password for operations**.

The **Operations** operation can only be selected by a user with administrator rights. Its use also requires that a mediator password has been set.

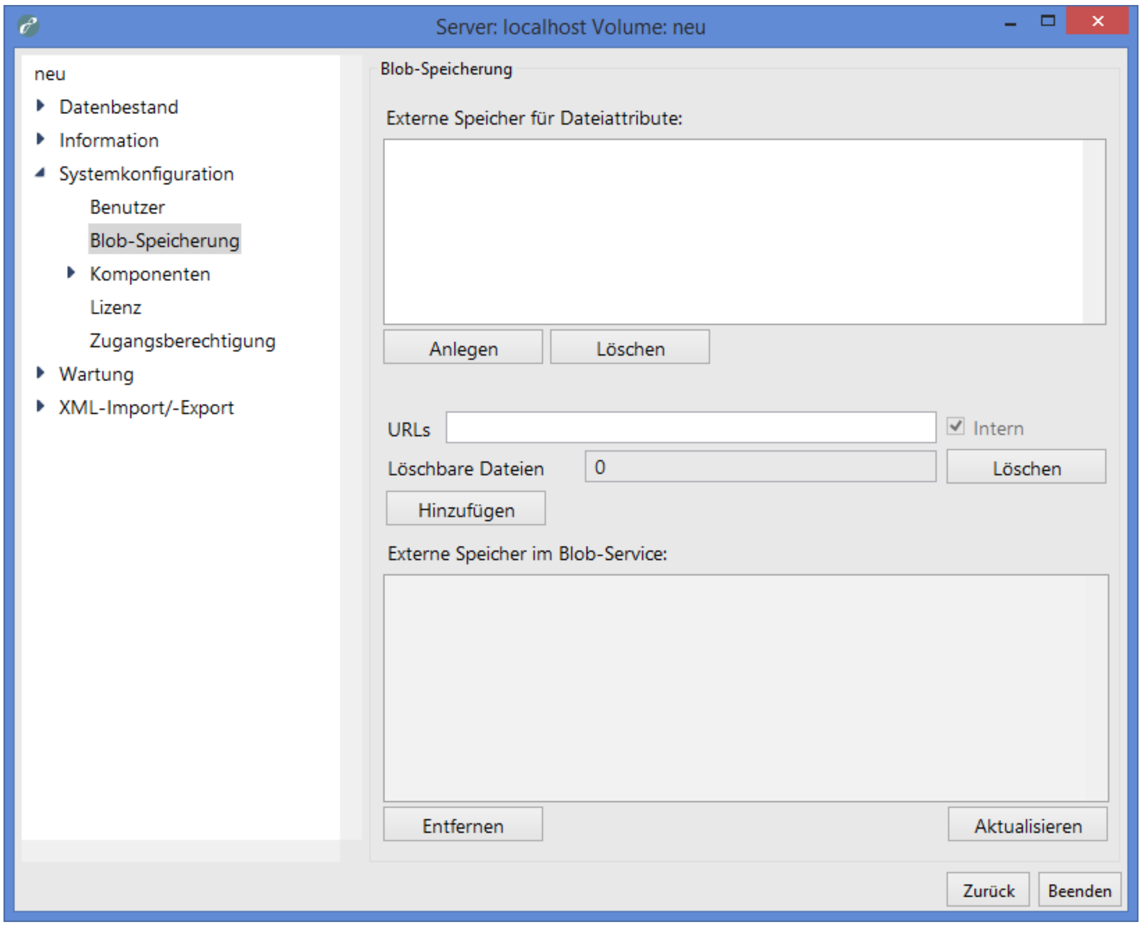
The field **Administrators** specifies the number of all users with administrator rights registered in the semantic network.

The field **Users** specifies the number of all users without administrator rights registered in the semantic network.

The field **Active** specifies the number of all users currently logged into the semantic network using the Knowledge Builder.

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 semantic network. The advantage of this is that they can be managed independently of the semantic network 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).



Create generates a new blob store. Using the name format *[semantic network 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 semantic network 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 *[semantic network 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**.

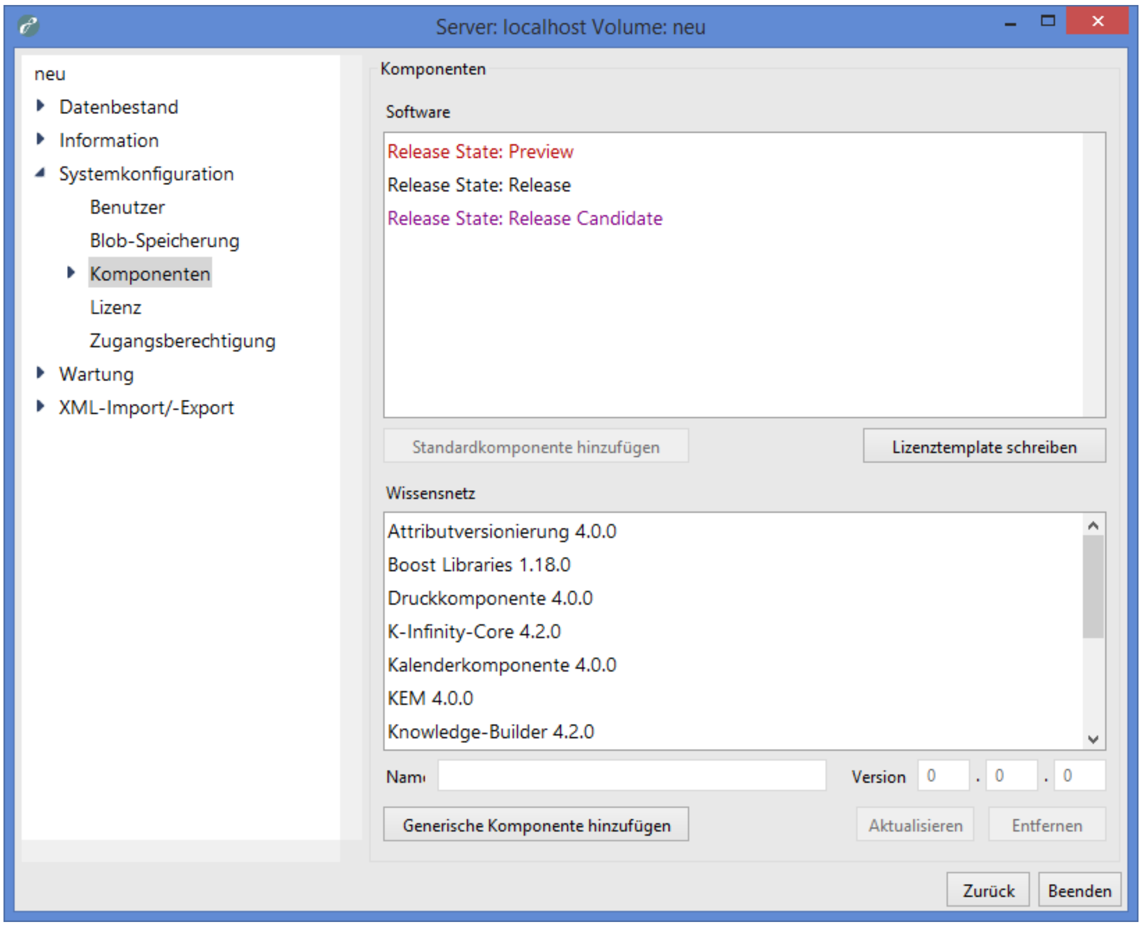
Remove interrupts the connection of the blob store selected in the **overview of registered blob stores** to the blob service and removes the blob store from the overview. In doing so, all blobs stored in the blob store irrevocably lose their internal references to the respective attributes in the semantic network and can no longer be retrieved in the semantic network. 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 semantic network is established using an SQLite database.

2.5.2.3.3 Components

Semantic networks consist of semantic network components. In addition to the basic functions, they basically provide the semantic network 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 semantic network components. If such a component is installed in the semantic network, only software components with suitable publication statuses are able to access the semantic network.



Release State: Preview
Release State: Release
Release State: Release Candidate

Attributversionierung 4.0.0
Boost Libraries 1.18.0
Druckkomponente 4.0.0
K-Infinity-Core 4.2.0
Kalenderkomponente 4.0.0
KEM 4.0.0
Knowledge-Builder 4.2.0

The **Software list** provides an alphabetical list of all semantic network 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 semantic network. Publication status components do not have a version number.

If you right-click on a semantic network 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 semantic network component selected in the **software list** in the semantic network. A separate window informs of the installation status. Some semantic network components require other semantic network components installed in the semantic network. Most installed semantic network 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: *[semantic network].componentLicenseTemplate.ini*). Irrespective of the configuration of the semantic network just administered, configuration placeholders are specified for the components *KEM*, *i-views core* and *Knowledge Builder*. The version number of the respective semantic network component supplied in the Admin tool is pre-entered in every configuration placeholder.

The **semantic network list** alphabetically lists all semantic network components installed in the semantic network with their respective version numbers. An installed semantic network 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 semantic network by default.

The text fields **Name** and **Version** show the name and the three-digit version number of the installed semantic network component selected in the **semantic network list**.

Add generic component adds a generic model component or a generic software component to the **semantic network list**. The component type is selected in a separate window. Generic components allow bundling of project-specifically created semantic network extensions and simplify their installation (removal) and version monitoring via the Admin tool. The name and version number of a generic semantic network component installed in the semantic network can be freely assigned in the corresponding text fields.

Update (the name changes to **Renew**, if it can be deactivated) updates the installed semantic network component selected in the **semantic network** 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 semantic network component was originally installed in the semantic network, identifiers of all elements and element types of this semantic network component are also updated. Depending on the semantic network 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.

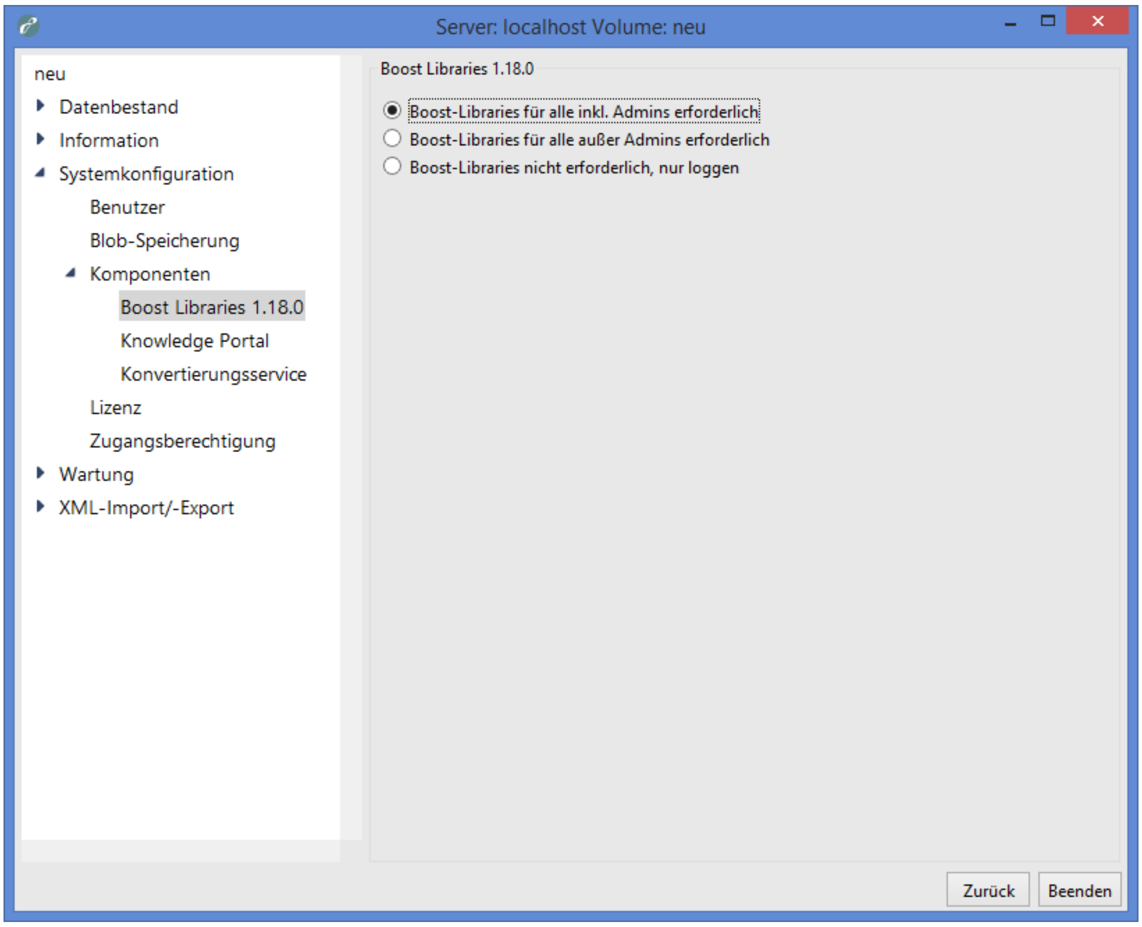
Remove removes the installed semantic network component selected in the **semantic network list**. If semantic network components in the installed status in Knowledge Builder have an entry in the *Technical* category, they leave their own subnet after they have been removed, which has to be removed manually. Semantic network components can only be removed if no other semantic network components that depend on the semantic network component to be removed are installed. The two semantic network 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* semantic network 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* semantic network component makes it possible to ensure that access to the Boost.Regex library is possible.


Server: localhost Volume: neu

neu
- Datenbestand
- Information
- Systemkonfiguration
  - Benutzer
  - Blob-Speicherung
  - Komponenten
    - Boost Libraries 1.18.0
    - Knowledge Portal
    - Konvertierungsservice
  - Lizenz
  - Zugangsberechtigung
- Wartung
- XML-Import/-Export

Boost Libraries 1.18.0

[✓ Boost-Libraries für alle inkl. Admins erforderlich]
Boost-Libraries für alle außer Admins erforderlich
Boost-Libraries nicht erforderlich, nur loggen

If the **Boost libraries required for all incl. Admins** option is selected, all software components apart from the Admin tool can only access the semantic network 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 semantic network if they can access the Boost.Regex library. The only ones excepted from this access lock are users with administrator rights who access the semantic network via the Knowledge Builder.

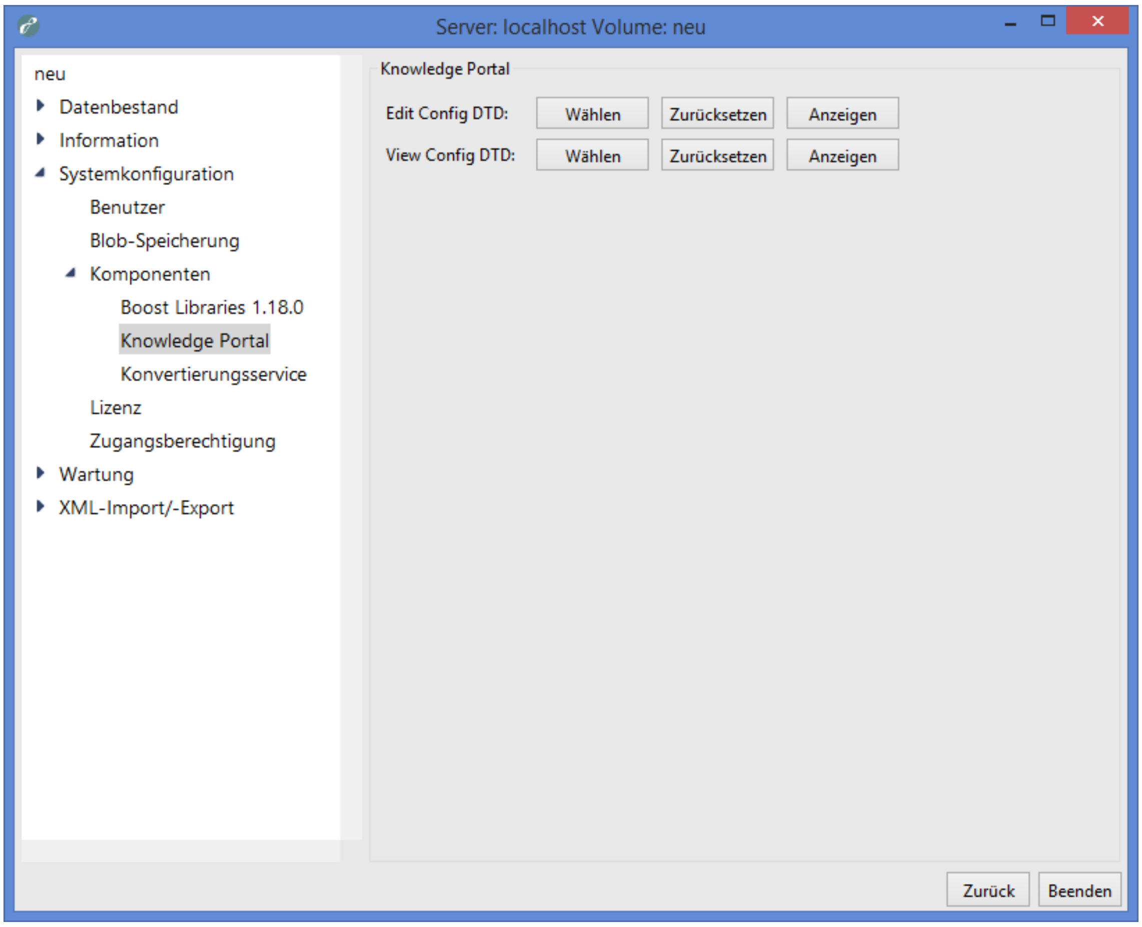
If the **Boost libraries not required, logging only** option is selected, each software component enters a corresponding warning in its respective log file, if available, if it cannot access the Boost.Regex library during start-up. Access to the semantic network remains possible regardless.

Knowledge portal

This configuration menu appears only if the *Knowledge portal* semantic network component is installed.

The *Knowledge portal* semantic network component enables a semantic network to operate a knowledge portal (of a front-end that can be displayed via browser). The configuration of the display and control elements of this front-end is performed in the Knowledge Builder on the relevant element types via an editor specially provided by the semantic network component for that purpose and with the help of the XML markup language. To make maintenance easier, and for the logical regulation of XML documents, it is possible to install schemas in the DTD format, on the basis of which the XML documents can be validated.

In the front-end, a distinction is made between an edit view and a presentation view, each of which have exclusive display and control elements. Separate DTD schemas are maintained for both views. Each of the control elements explained below exists for every view.



The **Select** button can be used to access the file system of the operating system in order to load a DTD schema file for the relevant view and install it in the semantic network. The default file name for edit view DTDs is *editConfig.dtd*, and the default file name for presentation view DTDs is *viewConfig.dtd*.

Reset deletes the DTD schema installed for the relevant view from the semantic network.

Display shows the DTD schema installed for the relevant view in a separate window. There it can be copied to the clipboard of the operating system (**Copy to clipboard** button) or exported to any location via a saving dialog as a text file with a name of your choice (**Save** button). 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.

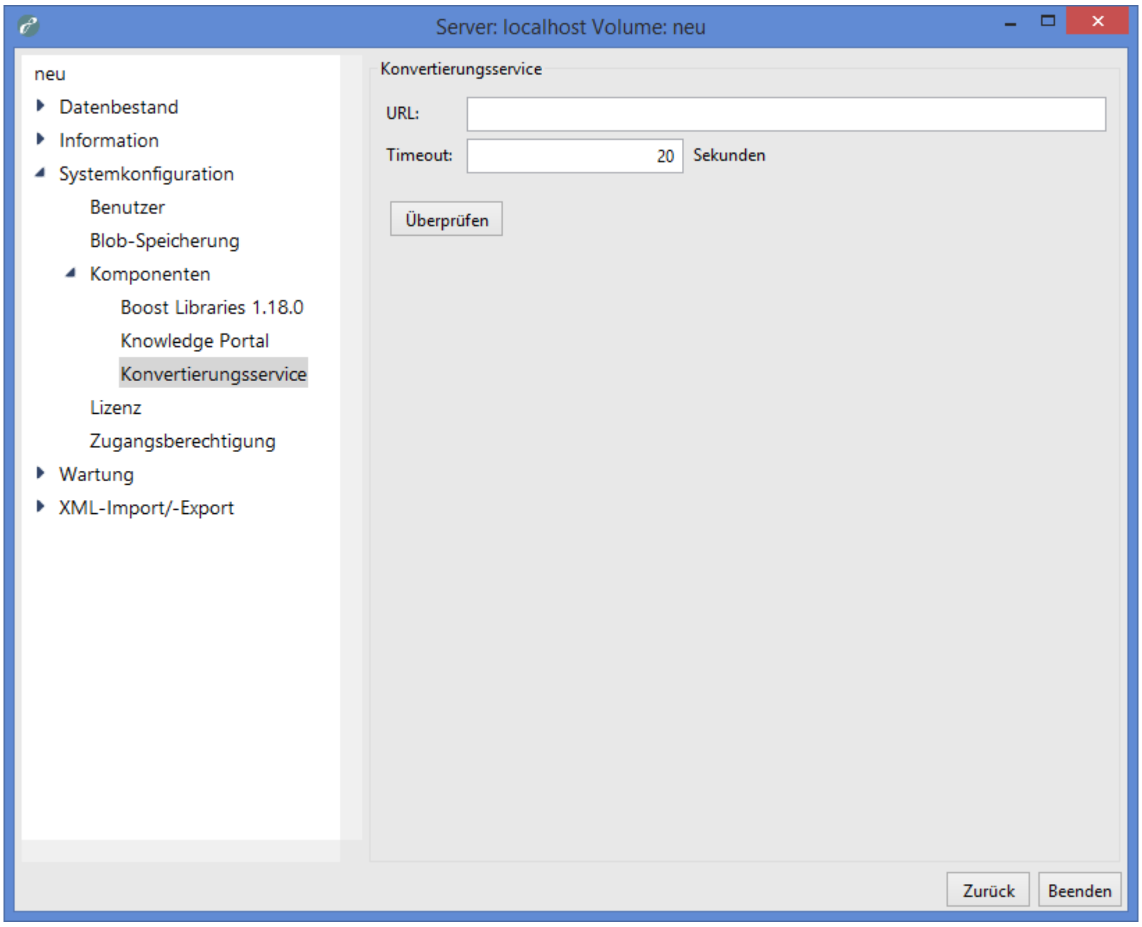
Conversion service

This configuration menu appears only if the *print component* semantic network component is installed.

The *print component* allows selected semantic 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 semantic network using the Knowledge Builder and be linked to the semantic 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 semantic network.

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 Sekunden

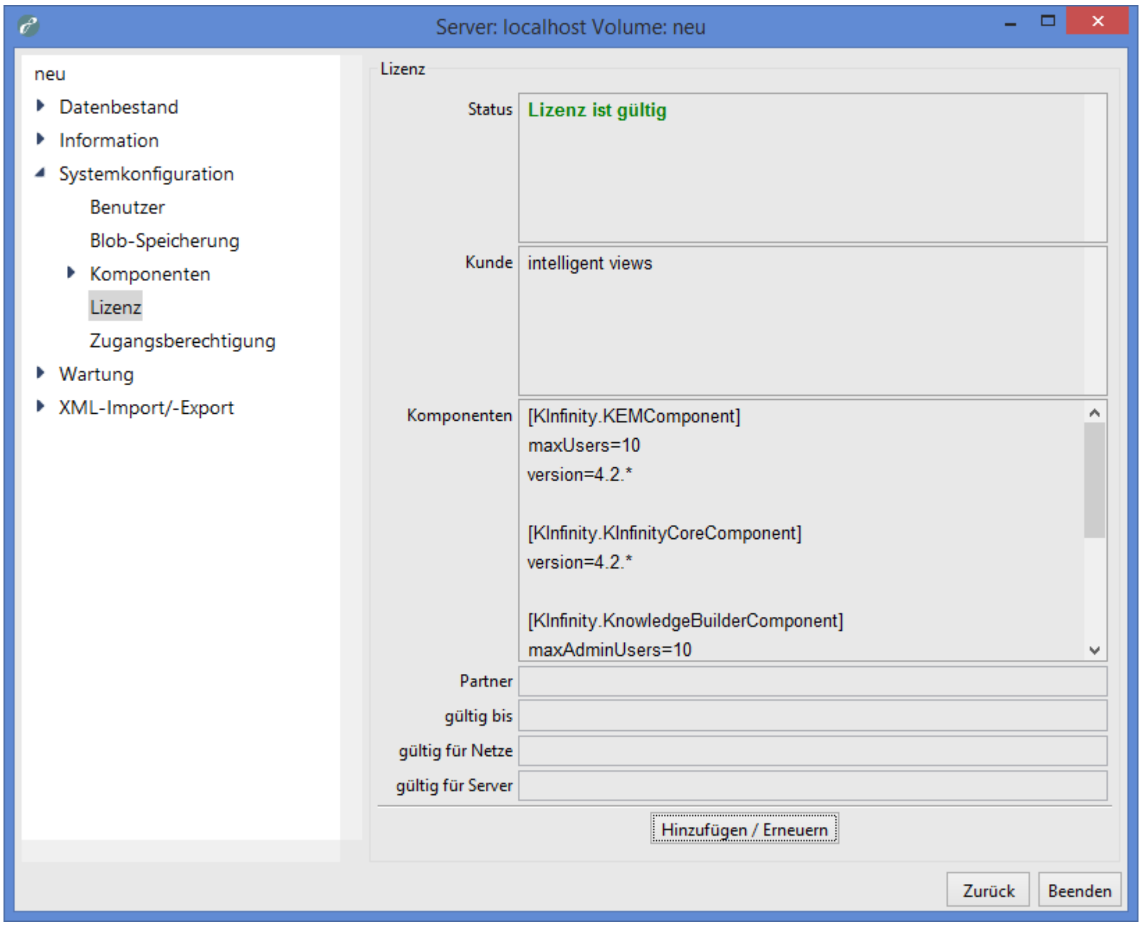
The bridge is linked to the print component using the free text field **URL**. The network address of the bridge is entered there in the format *http://[Bridge-IP-Number]:[Bridge-Port]/jodService/jodconverter/service*. The path section */jodService/jodconverter/service* has historical reasons and activates 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 semantic network 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: Lizenz ist gültig

Kunde: intelligent views

Komponenten:
```
[KInfinity.KEMComponent]
maxUsers=10
version=4.2.*

[KInfinity.KInfinityCoreComponent]
version=4.2.*

[KInfinity.KnowledgeBuilderComponent]
maxAdminUsers=10
```

Partner:

gültig bis:

gültig für Netze:

gültig für Server:

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 *[semantic network].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 networks** field contains a list of names of all networks 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 semantic network.

The fields **Partner**, **Valid to**, **Valid for networks** and **Valid for servers** can be left blank.

All fields have a context menu that can be activated by right-clicking.

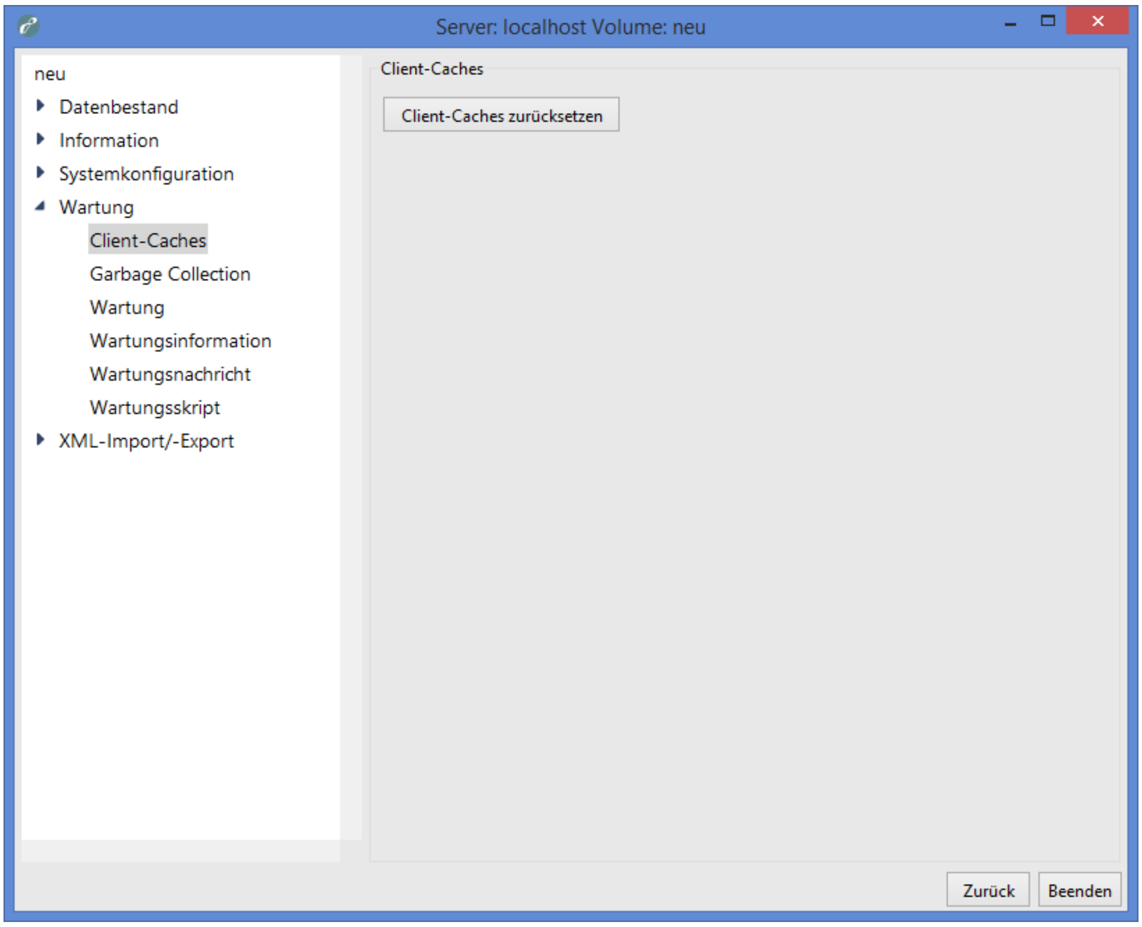
- **Select All** selects all the text. Alternatively, the mouse pointer can be used to mark any text segment.
- **Copy** copies the selected text area to the clipboard of the operating system.
- **Find Again** searches for the selected text area and finds its next occurrence in according to the read 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

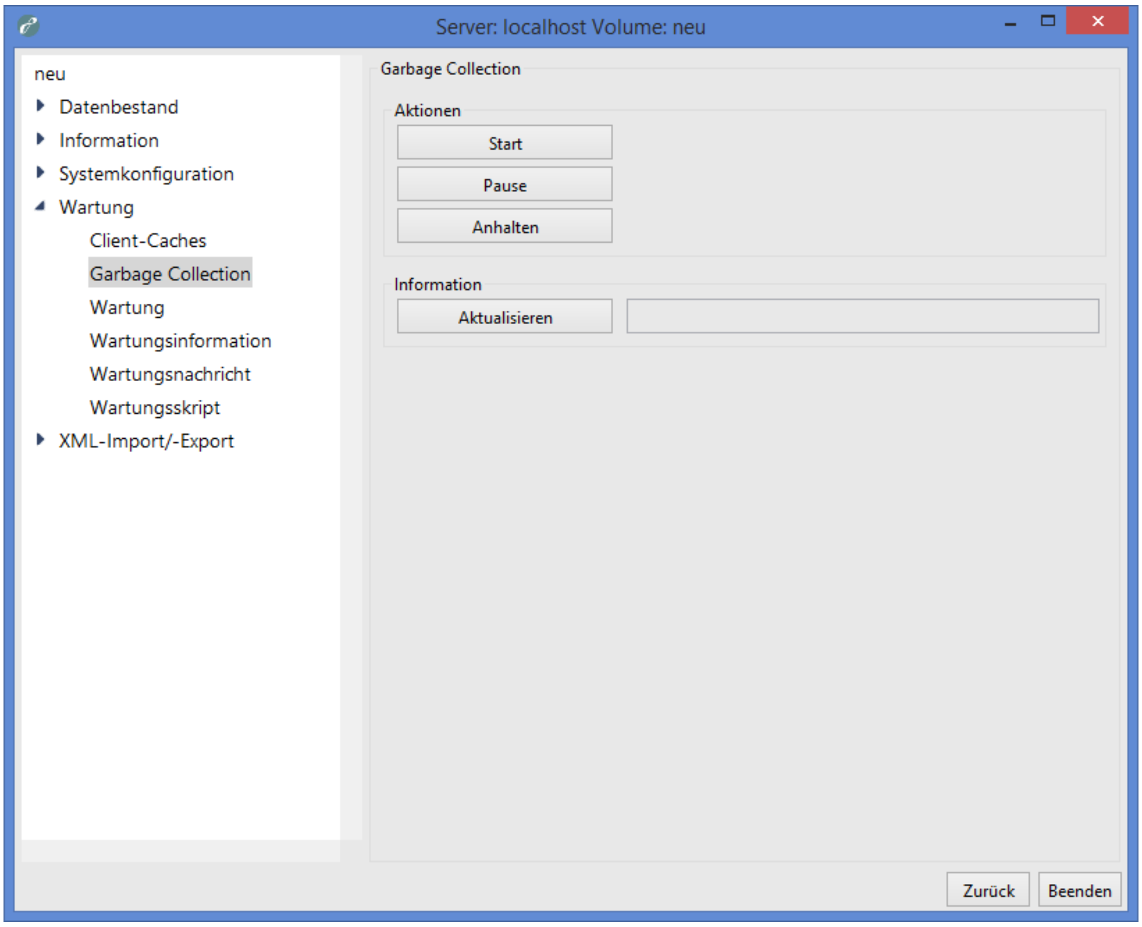
To improve performance, software components accessing the semantic network often fall back on their own buffers (cache). These buffer the schema and configuration data of the semantic network so they can access them more quickly if they need to use them later on.



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 semantic network 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 semantic network and thereby minimizes the memory usage of the semantic network. Use of the garbage collection requires that the semantic network that is to be cleaned up is activated via a mediator.



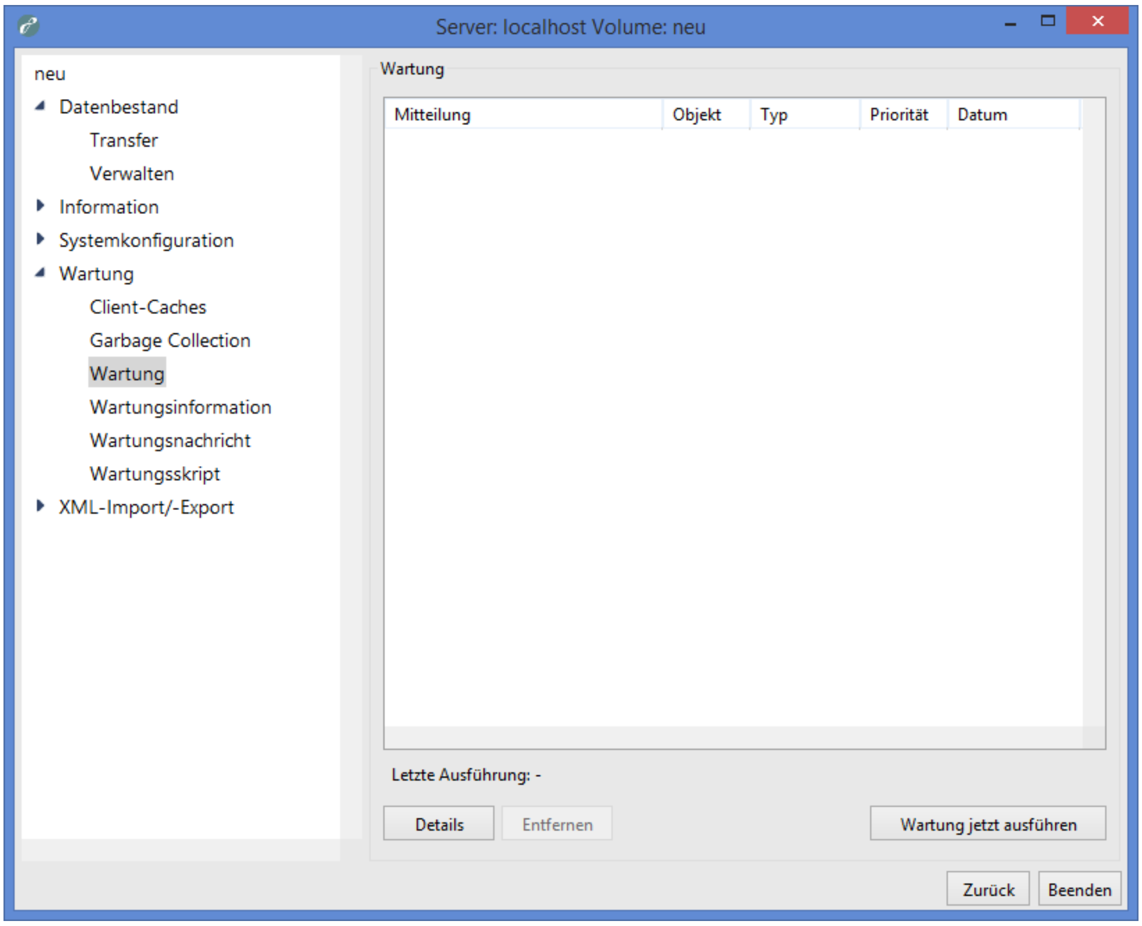
Start launches a new garbage collection for the semantic network or continues a paused garbage collection. No confirmation is sent when the process is completed. You can determine its progress via the **Refresh** menu option.

Pause interrupts the execution of the active garbage collection for the semantic network.

Stop cancels the execution of the active garbage collection for the semantic network.

Refresh writes the current state of the garbage collection for the semantic network 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 semantic network 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

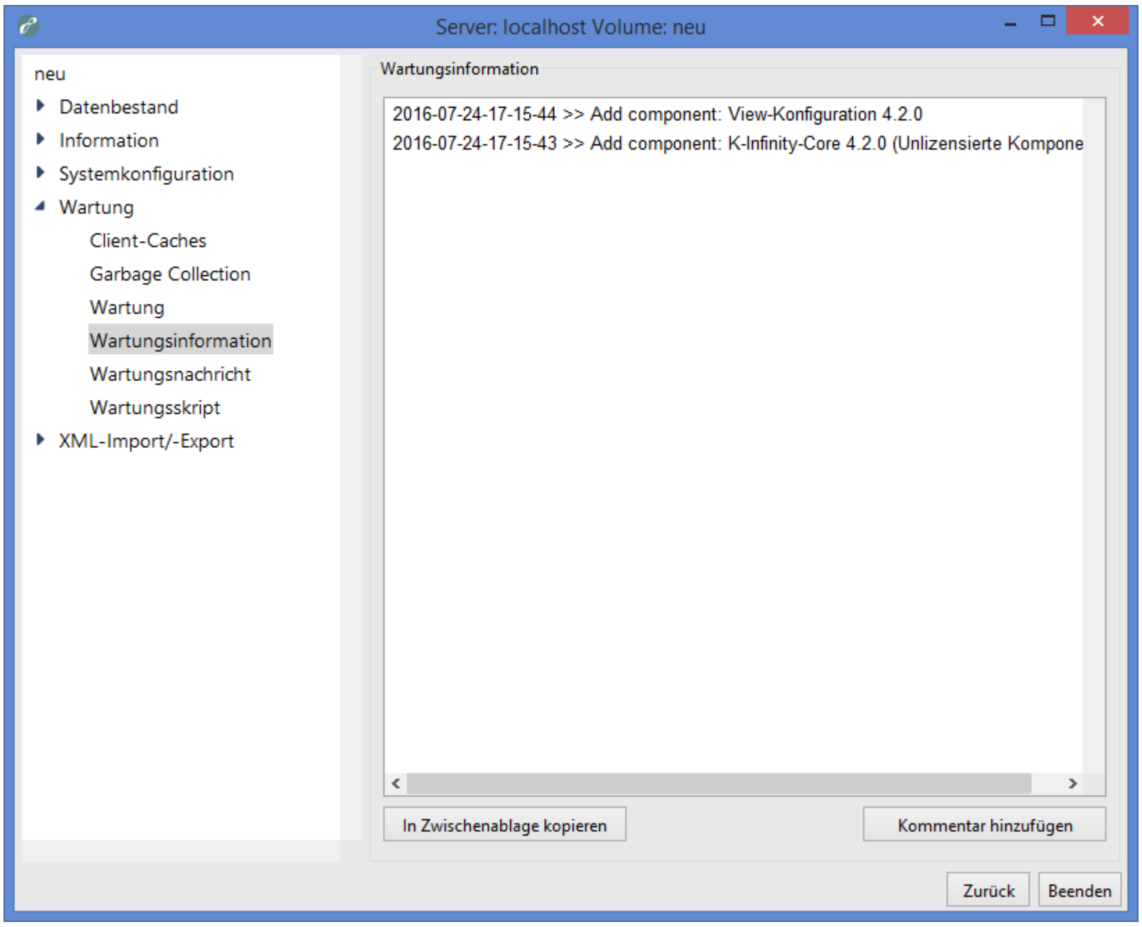
- the first point in time at which it was identified in the form of a date (*date*)

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



```
2016-07-24-17-15-44 >> Add component: View-Konfiguration 4.2.0
2016-07-24-17-15-43 >> Add component: K-Infinity-Core 4.2.0 (Unlizensierte Kompone
```

This menu option can be used to call up a chronologically ordered **maintenance history** of all essential administration processes in the semantic network 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:

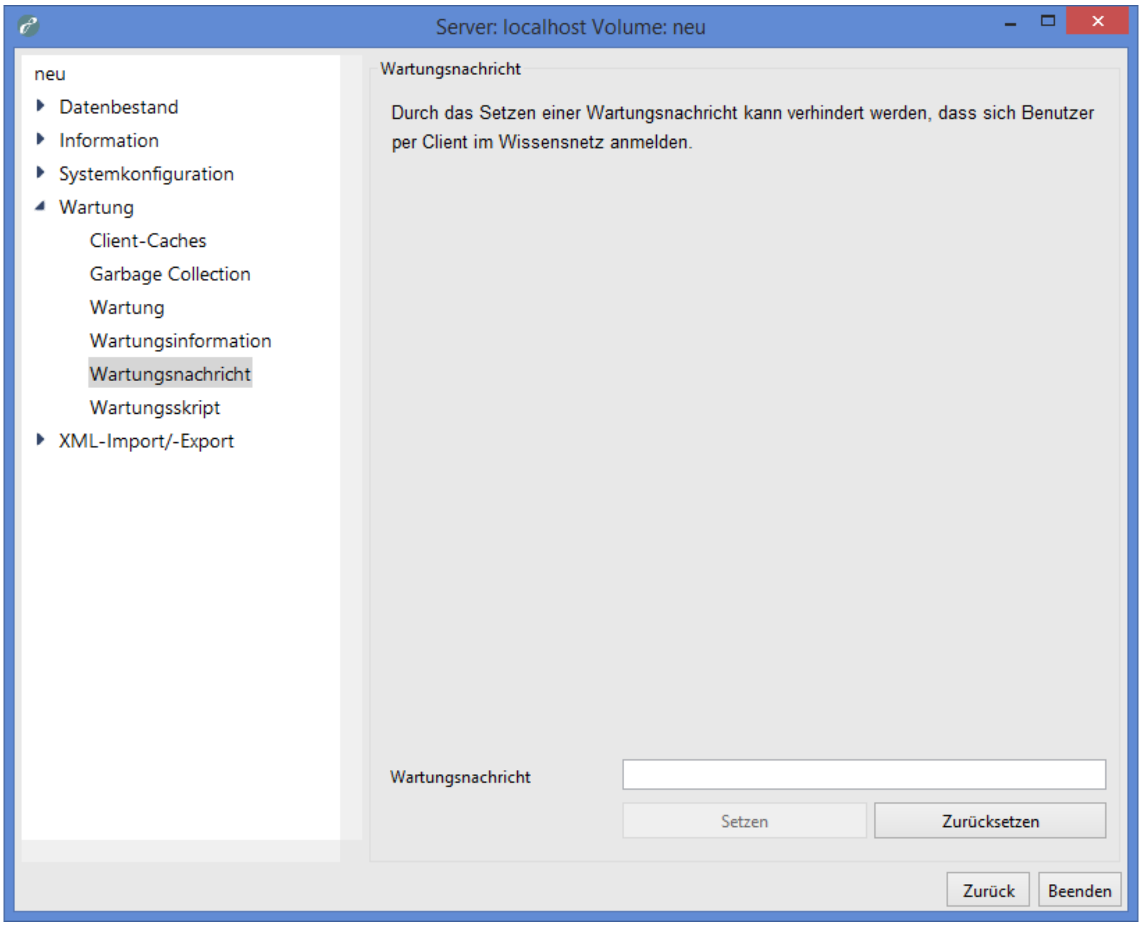
- **Select All** selects all the text. Alternatively, the mouse pointer can be used to mark any text segment.

- **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



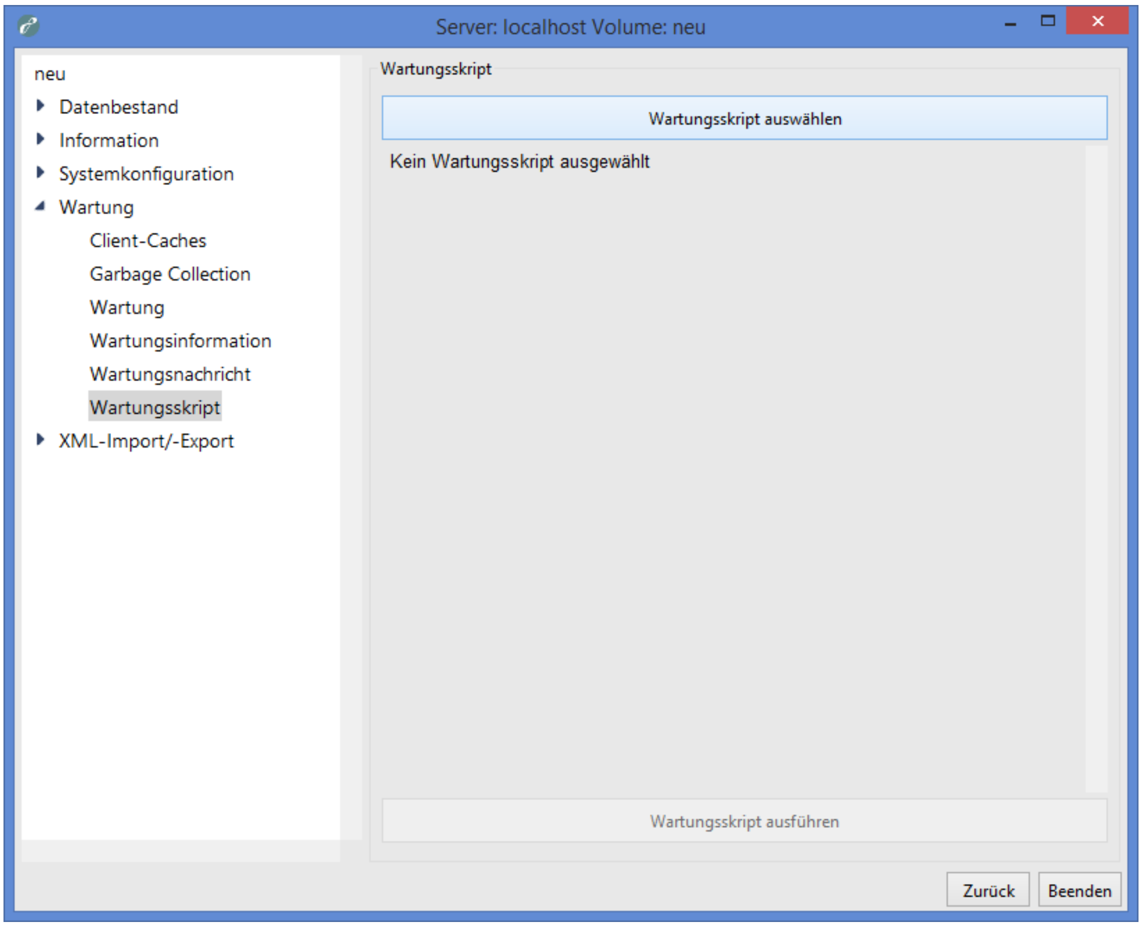
Durch das Setzen einer Wartungsnachricht kann verhindert werden, dass sich Benutzer per Client im Wissensnetz anmelden.

The **Set** button activates a maintenance block that prevents all users from accessing the semantic network 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 semantic network 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



Kein Wartungsskript ausgewählt

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.

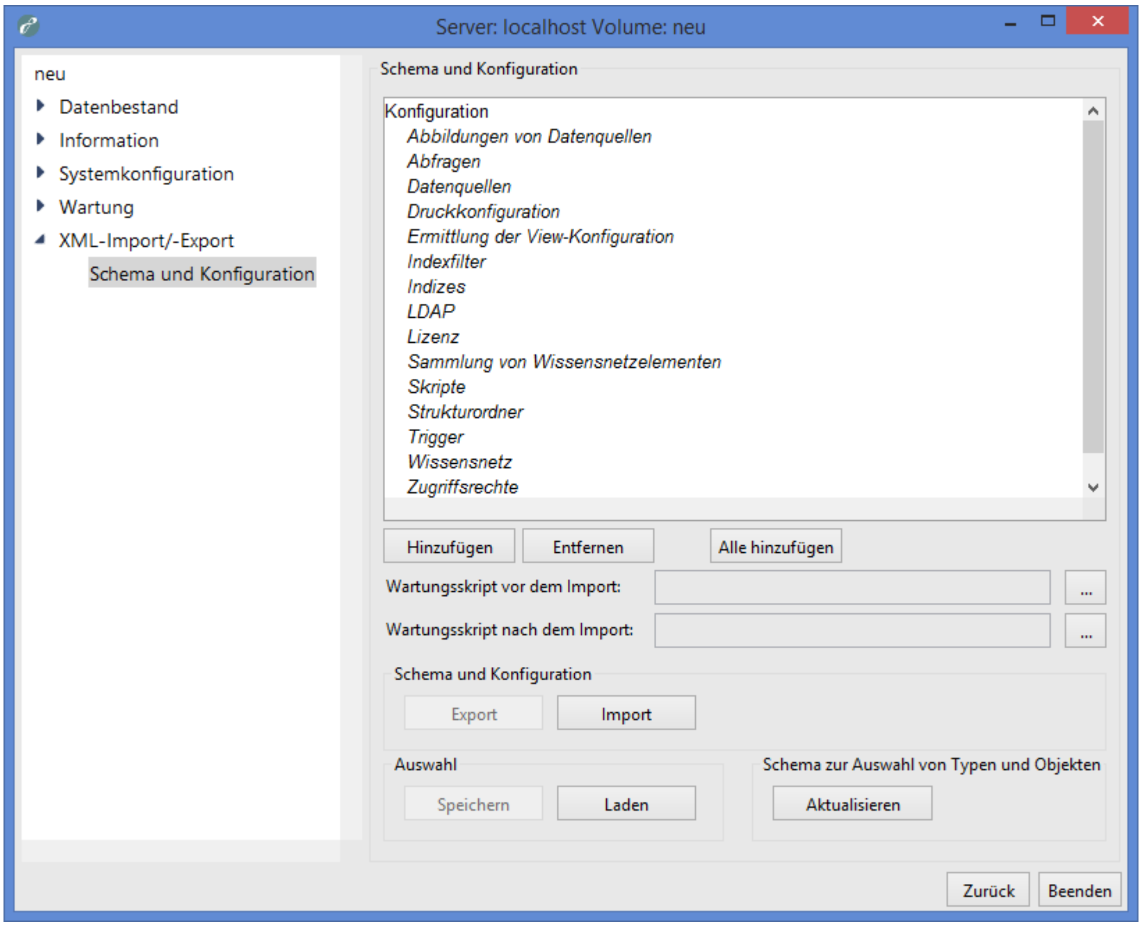
Execute maintenance script starts the maintenance script. A separate window tells you when the maintenance script was executed and, depending on the script, offers additional execution information or permits script-specific execution options.

2.5.2.5 XML import/ -export

2.5.2.5.1 Schema and configuration

Along with subnets generated by the user and imported using components (schemas with useful data), a semantic network, by extension, is also comprised of diverse other modules (configurations) that extend, configure or work with this subnet in functional terms. Schemas and configurations are referred to jointly as configurations within the context of this menu item.

Numerous configurations of a semantic network can be systematically exported and imported.



The **configuration overview** is a list providing an overview of all configuration types in a semantic network 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 semantic network (*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 subnets (*semantic network*) and
- the set of all modules defined within the rights category (*access rights*).

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.

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.

Add adds a configuration of the configuration type selected there to the **configuration overview**. If more than one configuration exists in the semantic network 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 semantic network to the **configuration overview** and distributes them among the respective suitable configuration types.

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 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: *[Semantic network].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 semantic network 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: *[Semantic network].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 semantic network 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 semantic network from an XML file (default file name: *instruction.xml*).

Update adds the attribute types equipped with the Boolean attribute value type

- *XML: Export all objects*
- *XML: Export direct objects*
- *XML: Do not export type and all subtypes* and
- *XML: Do not export subtypes*

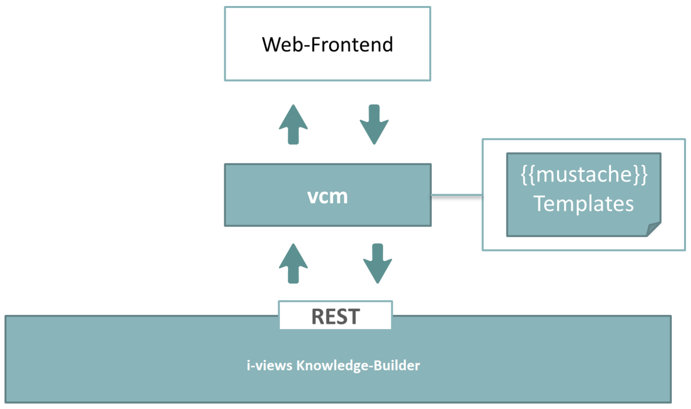
to the knowledge network if they do not yet exist in it. These attribute types are required to select which elements and element types found in this configuration should be exported or not exported respectively in the event of exporting a configuration of the configuration type semantic network. To do so, these attribute types are attached to suitable object types using the Knowledge Builder and are given suitable attribute values.

If nothing else has been configured using these attribute values, then this applies for every object type that the object type itself is exported, not, however, 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.

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 Configuration

The usual procedure involves activation of the ViewConfiguration Mapper components in the semantic network 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.

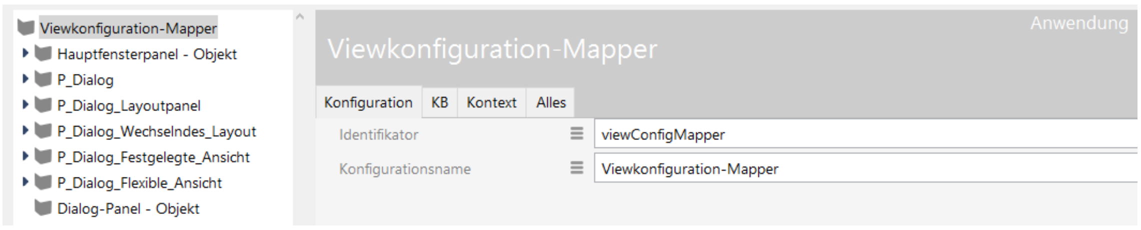
Grunt (gruntjs.com/) is used as the TaskRunner, and as a Package Manager Bower (bower.io/). More detailed information and a list of the Grunt tasks is available in the README.md in the project.

3.2.1 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.

3.2.1.1 Panel confoguration

If the web application is supposed to be based on a panel configuration, the application must be linked to the panel configuration.



Viewkonfiguration-Mapper
- Hauptfensterpanel - Objekt
- P_Dialog
- P_Dialog_Layoutpanel
- P_Dialog_Wechselndes_Layout
- P_Dialog_Festgelegte_Ansicht
- P_Dialog_Flexible_Ansicht
- Dialog-Panel - Objekt

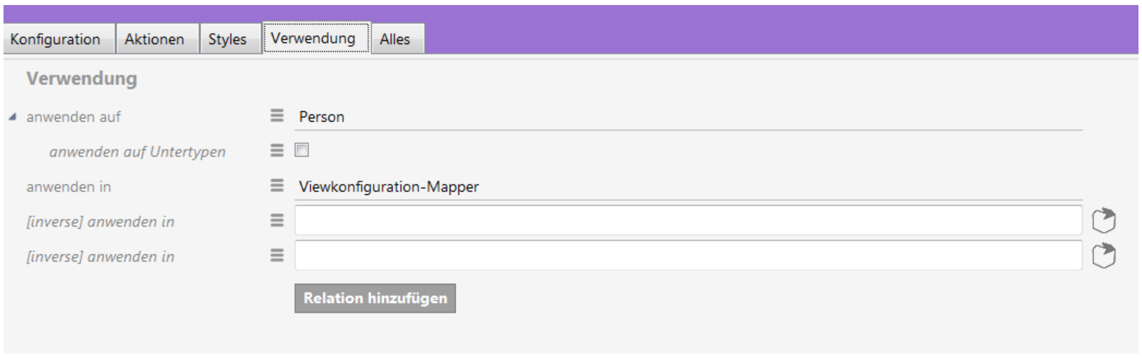
Viewkonfiguration-Mapper

Identifikator	viewConfigMapper
Konfigurationsname	Viewkonfiguration-Mapper

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.

3.2.1.2 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.



Verwendung

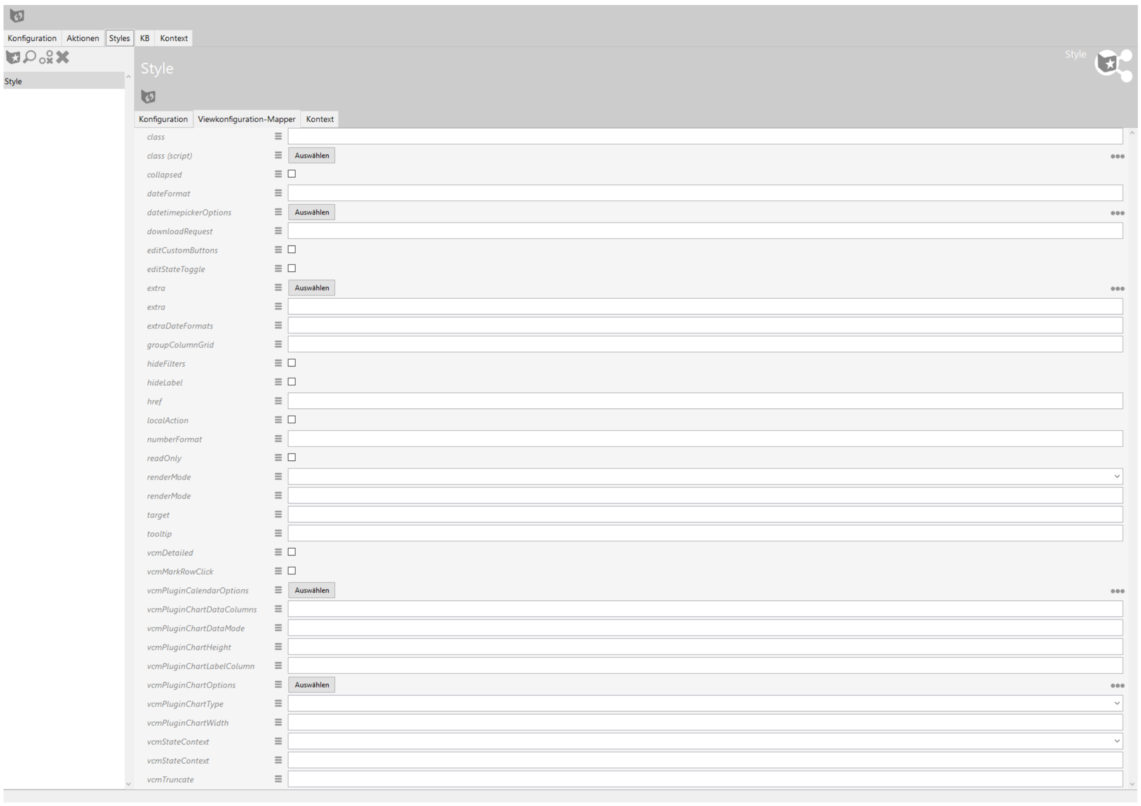
anwenden auf	Person
anwenden auf Untertypen	☐
anwenden in	Viewkonfiguration-Mapper
[inverse] anwenden in	
[inverse] anwenden in	

3.2.1.3 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.

In total there are multiple setting options for the styles (see figure):



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 in the CSS of the ViewConfiguration Mapper or in the "view-configmapper.config.GET" script

class (script)	Reference to script		Definition of CSS styling in the form of a script return value
collapsed	Boolean		
dateFormat	String		
datetimepickerOp-tions	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 "ex-ternal" renderMode and with a parameter specifi-cation in the "href" line to define an external web link for the action of a but-ton.
extra	String		
extraDateFormats	String		
groupColumnGrid	String	Group	The expected input is a string of figures sep-arated by a space or comma. Each figure defines the quantity of columns if the maximum is 12 columns.
hideFilters	Boolean		Hides the table query fil-ters in the table header
hideLabel	Boolean		Hides the label of a view configuration element (la-bel on the tab of an alter-native 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.
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		
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	

vcmPluginChart-DataMode	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"
vcmPlugin-ChartHeight	String	VCM plugin	Absolute height of a chart in pixels (e.g.: "300px")
vcmPluginChartLa-belColumn	String	VCM plugin	
vcmPluginChartOp-tions	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.
vcmPluginChart-Type	Selection	VCM plugin	Selection options for the "chart" RenderMode (ap-plicable for tables): • bar • doughnut • line • pie • pole • radar
vcmPlugin-ChartWidth	String	VCM plugin	Absolute width of a chart in pixels (e.g.: "380px")
vcmStateContext	Selection		Selection options: • global • page • none
vcmStateContext	String		
vcmTruncate	String		

Beachten Sie, dass es zu einzelnen View-Konfigurationselementen eigene Style-Möglichkeiten gibt, die entsprechend als Unterkapitel an den jeweiligen View-Konfigurationselementen erklärt werden. So ist zum Beispiel die Möglichkeit gegeben, bei Eigenschafts-Darstellungen die Aufteilung eines Labels und seines Wertes anzupassen.

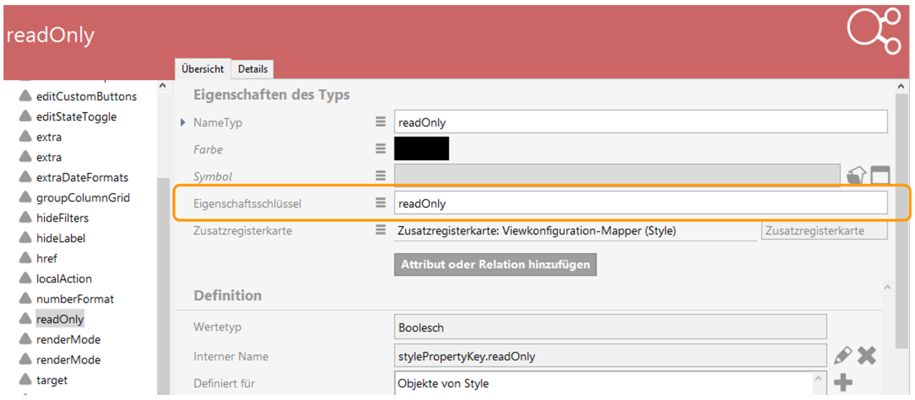
3.2.1.3.1 Definition of style attributes

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

Übersicht | Details

editCustomButtons
editStateToggle
extra
extra
extraDateFormats
groupColumnGrid
hideFilters
hideLabel
href
localAction
numberFormat
readOnly
renderMode
renderMode
target

Eigenschaften des Typs

NameTyp	readOnly
Farbe	■
Symbol	
Eigenschaftsschlüssel	readOnly
Zusatzregisterkarte	Zusatzregisterkarte: Viewkonfiguration-Mapper (Style)

Attribut oder Relation hinzufügen

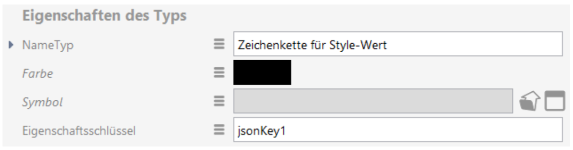
Definition

Wertetyp	Boolesch
Interner Name	stylePropertyKey.readOnly
Definiert für	Objekte von Style

In the JSON output, the key and value pairs (*StylePropertyKey* -> Style property) are output as an array under *additionalConfig*.

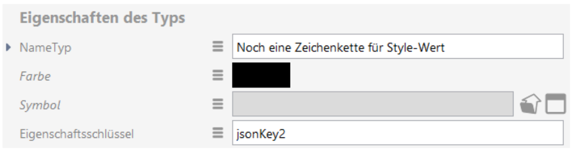
Example

Configuration of the type *String for style value*


Eigenschaften des Typs

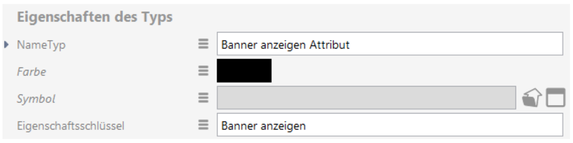
NameTyp	Zeichenkette für Style-Wert
Farbe	■
Symbol	
Eigenschaftsschlüssel	jsonKey1

Configuration of the type *Additional string for style value*


Eigenschaften des Typs

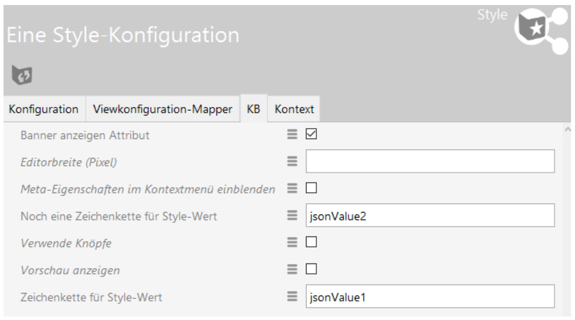
NameTyp	Noch eine Zeichenkette für Style-Wert
Farbe	■
Symbol	
Eigenschaftsschlüssel	jsonKey2

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

Style

Konfiguration | Viewkonfiguration-Mapper | KB | Kontext

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": { ... }
}
```

3.2.1.3.2 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.

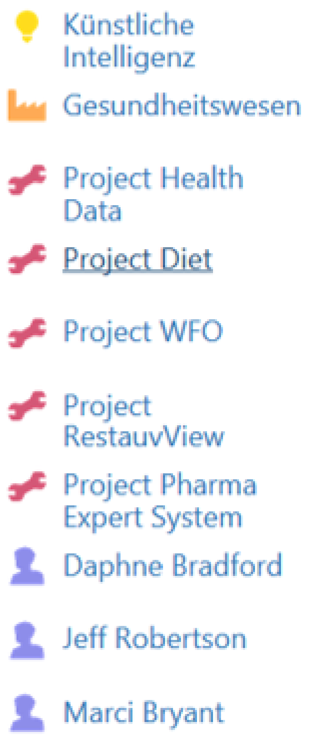
The following renderModes are available in the drop-down menu:

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

edit	Subordinate properties can be edited	Group

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

grid	In combination with the *groupColumnGrid* property, the layout can be divided according to a preconfigured grid with 12 units. Depending on the quantity of elements, it is possible to define the relative distribution by specifying the available units. Example: 5 3 4	Group
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

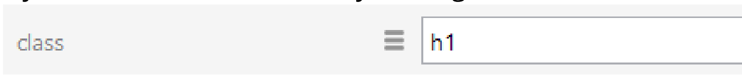
medialist	Displays the table entries as an HTML text link; displays the element with their icons  Künstliche Intelligenz Gesundheitswesen Project Health Data Project Diet Project WFO Project RestauvView Project Pharma Expert System Daphne Bradford Jeff Robertson Marci Bryant	Table
multiline	Necessary 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
panel	Has the effect of displaying as an expandable group	Group
pre	Displays the string as a pre-formatted and scrollable text	String property
table	Table view	Group
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)	Property

The renderModes available in the input line are related to Bootstrap. They include the following renderModes, for example:

render-Mode	Explanation	Applicability
email	Creates a link to the email address	String property
image	Displays an icon on the action	Action
jumbotron	Highlighted display. See getbootstrap.com/docs/4.1/components/jumbotron/	Group
well	Creates a box with a compressed effect. See getbootstrap.com/docs/3.3/components/#wells	Group

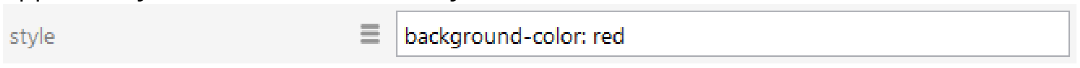
3.2.1.3.3 Usage of CSS

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). 

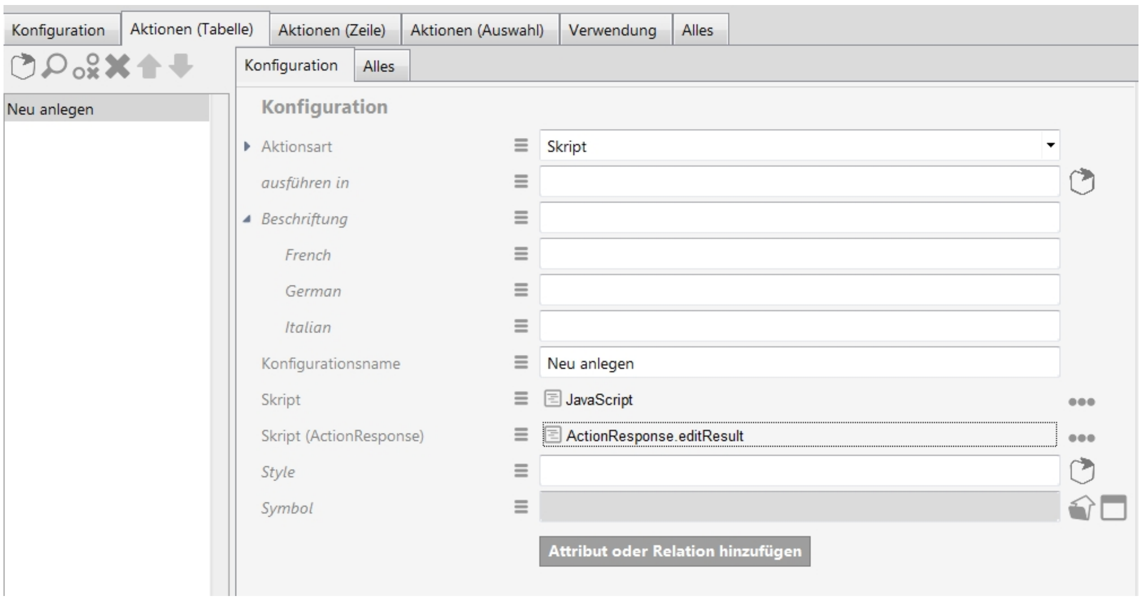
class ≡ h1

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.

3.2.1.4 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."



Konfiguration

Aktionsart	Skript
ausführen in	
Beschriftung	
French	
German	
Italian	
Konfigurationsname	Neu anlegen
Skript	JavaScript
Skript (ActionResponse)	ActionResponse.editResult
Style	
Symbol	

3.2.2 Login configuration

3.2.2.1 JWT authentication

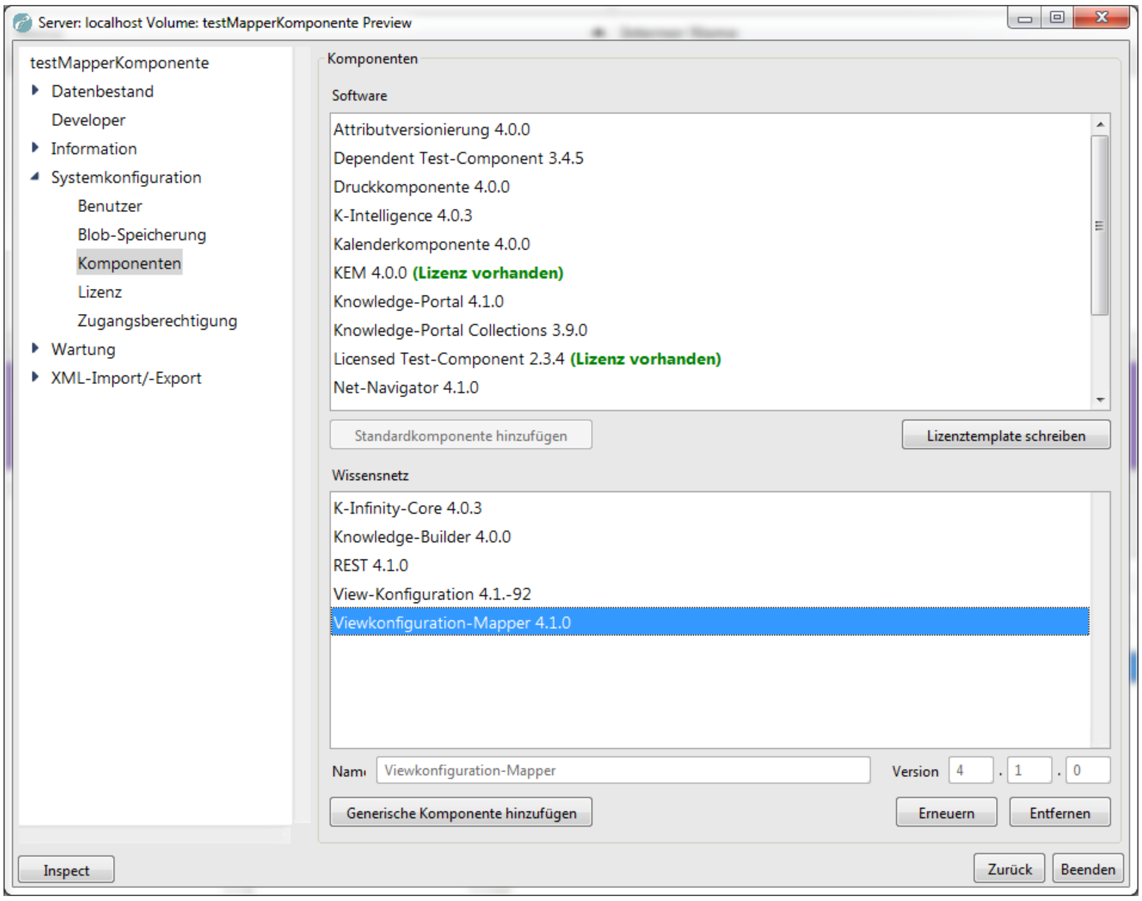
3.2.2.1.1 Modify the login form

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

3.2.3 The View Configuration Mapper component

To use the ViewConfiguration Mapper, activation of the corresponding components first in the Admin tool is a prerequisite.


Server: localhost Volume: testMapperKomponente Preview

testMapperKomponente
- Datenbestand
- Developer
- Information
- Systemkonfiguration
  - Benutzer
  - Blob-Speicherung
  - Komponenten
  - Lizenz
  - Zugangsberechtigung
- Wartung
- XML-Import/-Export

Komponenten

Software

Attributversionierung 4.0.0
Dependent Test-Component 3.4.5
Druckkomponente 4.0.0
K-Intelligence 4.0.3
Kalenderkomponente 4.0.0
KEM 4.0.0 **(Lizenz vorhanden)**
Knowledge-Portal 4.1.0
Knowledge-Portal Collections 3.9.0
Licensed Test-Component 2.3.4 **(Lizenz vorhanden)**
Net-Navigator 4.1.0

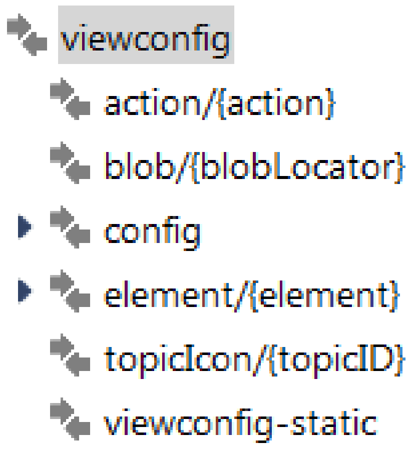
Wissensnetz

K-Infinity-Core 4.0.3
Knowledge-Builder 4.0.0
REST 4.1.0
View-Konfiguration 4.1.-92
Viewkonfiguration-Mapper 4.1.0

Name: Viewkonfiguration-Mapper
Version: 4 . 1 . 0

Standardkomponente hinzufügen
Lizenztemplate schreiben
Generische Komponente hinzufügen
Erneuern
Entfernen
Inspect
Zurück
Beenden

The component ensures the specific properties required are created in the view configuration and also creates all REST services that the vcm requires. (Please 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 network. “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 semantic network 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.

3.2.4 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.

3.2.5 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.

3.2.6 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.

3.3 Actions

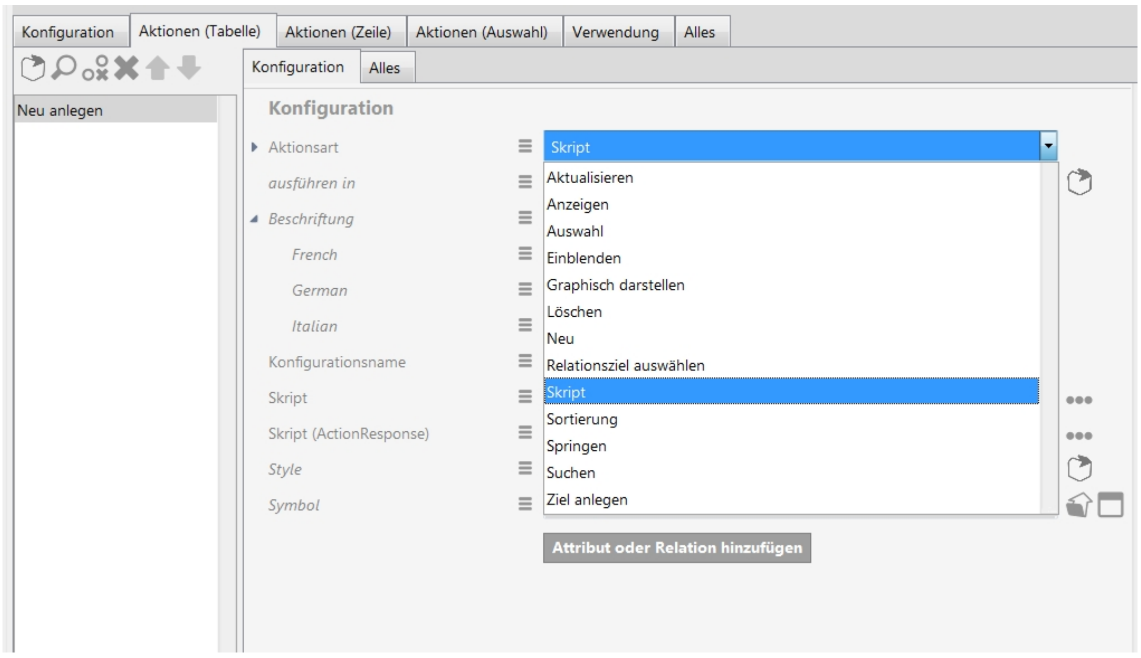
This chapter describes all VCM-specific options for setting actions and script parameters (context, actionResponse).

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, which can then also be executed in VCM.

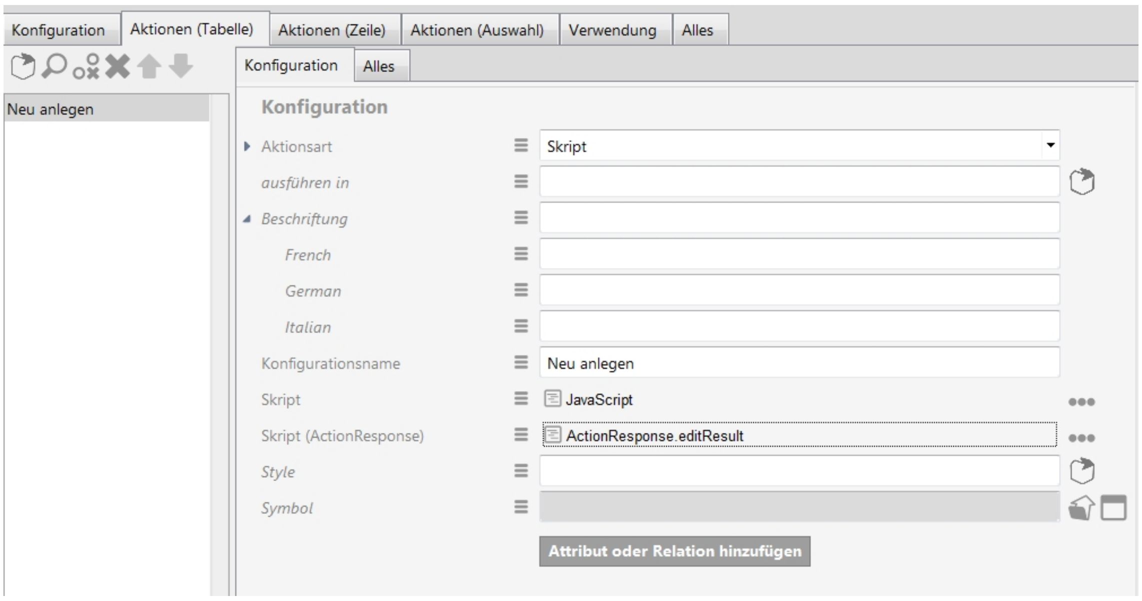
There are two types of user-defined actions:

- Standard action with user-defined return value
- Script action

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” entry in the list.



Konfiguration

Aktionsart	Skript
ausführen in	
Beschriftung	
French	
German	
Italian	
Konfigurationsname	Neu anlegen
Skript	JavaScript
Skript (ActionResponse)	ActionResponse.editResult
Style	
Symbol	

To determine the return value of an action, a script can be created for both action types under the entry “Script (Action Response).”

Example of a typical return script:

```
function actionResponse(element, context, actionResult){
 var actionResponse = new $k.ActionResponse();
 actionResponse.setFollowup("show");
 actionResponse.setData({
                elementId: actionResult.idString(),
                viewMode: "edit"
            });
 return actionResponse;
}
```

Possible follow-ups:

- show - The transferred contents are loaded and displayed on the entire page.
- refresh - All components update their contents.
- reload - The page is reloaded.
- update - The transferred contents are loaded and displayed in the view specified in “execute in.” An error occurs if “execute in” is not set.

You can also define your own follow-ups. However, in that case the interaction with the front-end must be adapted as VCM is not designed for this by default. On the other hand, however, it is also possible to overwrite the current reactions to a follow-up in your own adaptation project.

Additional possible return values:

- elementId - ID of the element to be displayed
- viewMode - At present only read and write mode are distinguished, whereby read mode is assumed and write mode is only displayed for viewMode: “edit."

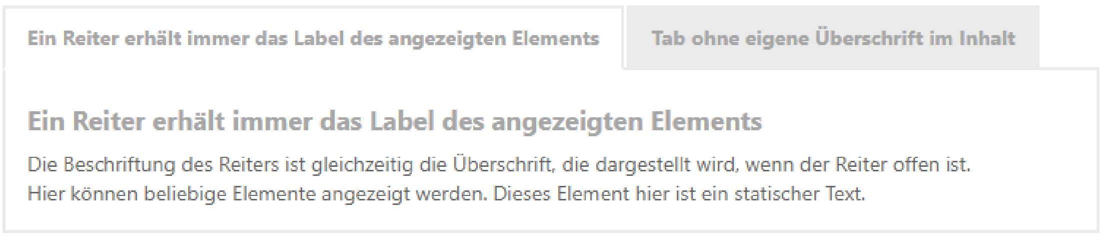
You can also define your own values here and adapt the front-end accordingly.

3.4 Viewconfig elements

3.4.1 General

3.4.2 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 group view, the summarized views are not shown simultaneously, but instead in alternating order (e.g. via tabs).

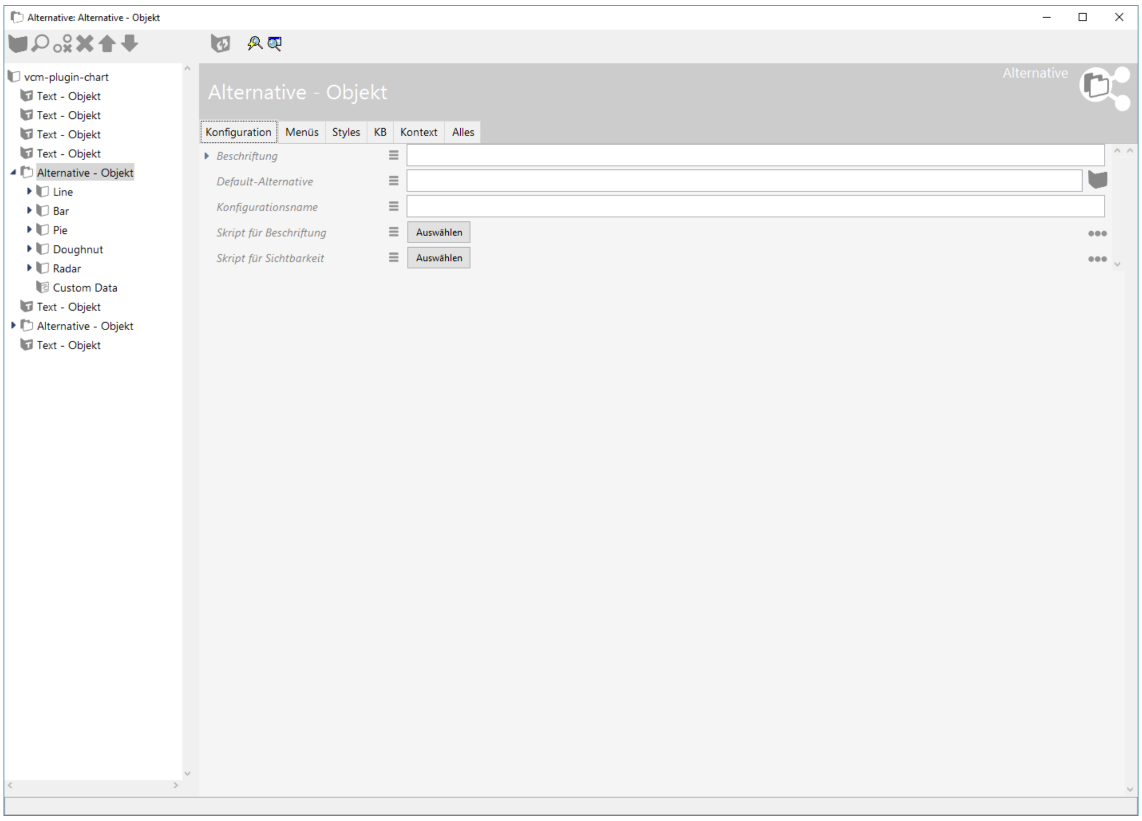


Ein Reiter erhält immer das Label des angezeigten Elements | Tab ohne eigene Überschrift im Inhalt

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.



Alternative: Alternative - Objekt

- vcm-plugin-chart
  - Text - Objekt
  - Text - Objekt
  - Text - Objekt
  - Text - Objekt
  - Alternative - Objekt
    - Line
    - Bar
    - Pie
    - Doughnut
    - Radar
    - Custom Data
  - Text - Objekt
  - Alternative - Objekt
  - Text - Objekt

Alternative - Objekt (Alternative)

Konfiguration | Menüs | Styles | KB | Kontext | Alles

Beschriftung	
Default-Alternative	
Konfigurationsname	
Skript für Beschriftung	Auswählen
Skript für Sichtbarkeit	Auswählen

The “Configuration” tab features options for specifying the general display of the list:

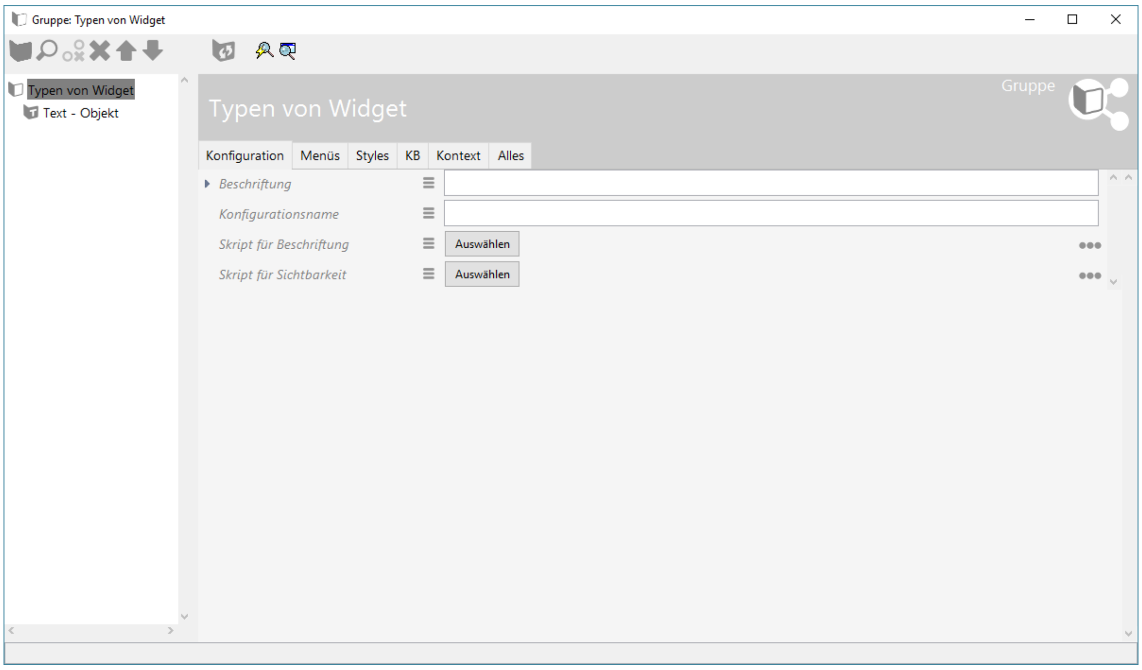
Label	The value entered here appears as the heading of the alternative
Default alternative	By default the first attached view is displayed. If you prefer the view on the third tab to be displayed first, for example, you can specify this view here. The front-end remembers the last displayed view within 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.
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 alternative can be determined in a script.
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.

3.4.3 Group

A group 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 group view as subviews. Their position decides the order in which they are displayed. Hence, the arrow buttons can be used to change their positions.



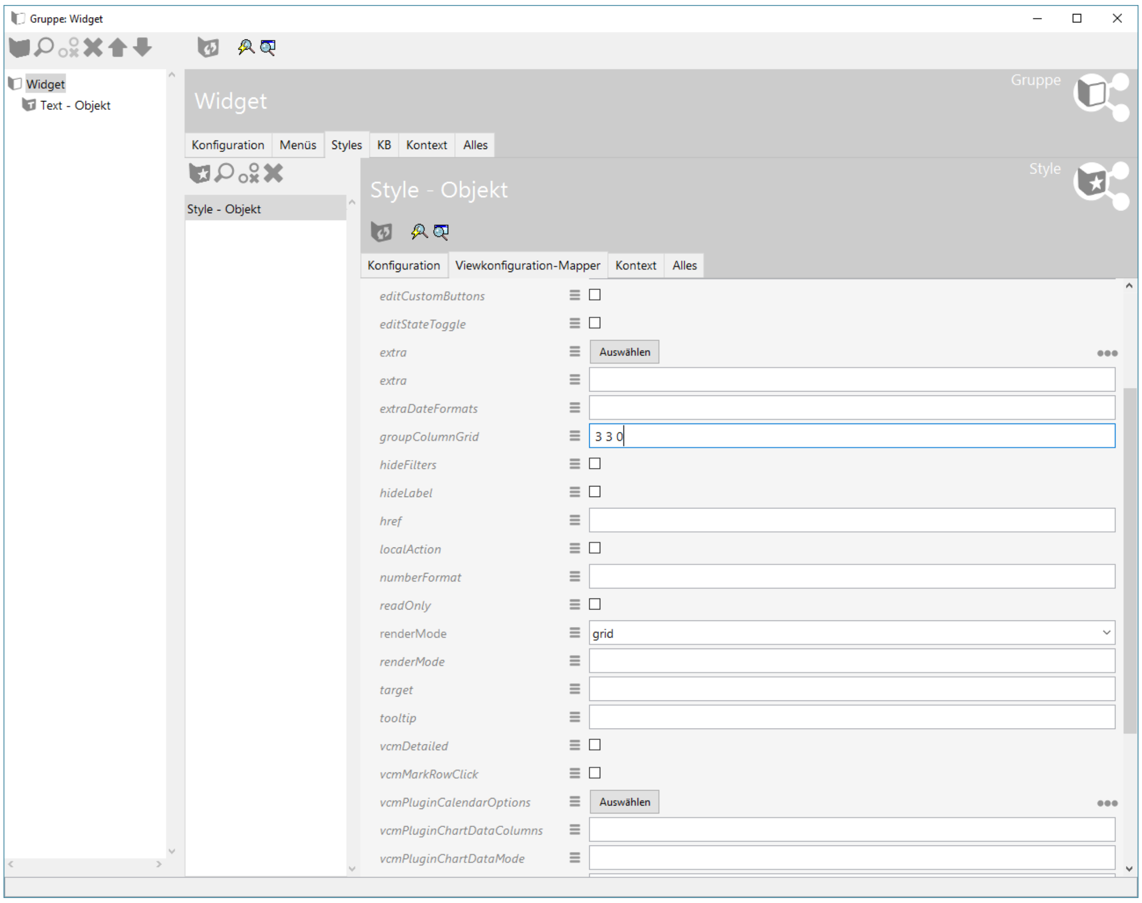
The "Configuration" tab features options for specifying the general display of the list:

Label	The value entered here appears as the header of the group
Configuration name	The configuration name can be used to identify views and panels.
Script for labeling	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.

The "Menus" tab lets you configure actions for groups, 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 group view is to be used and in which application contexts.

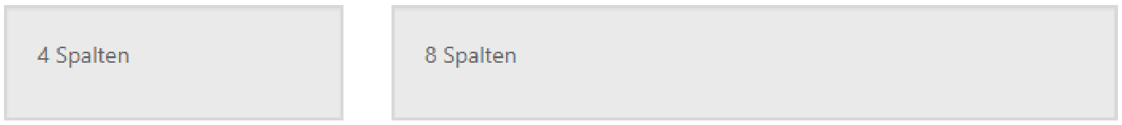
A group view is to be used when several views, which are based on the data of an object or type, are to be displayed simultaneously and grouped. In contrast to this, there is the alternative that displays the contained views for an object alternatingly (e.g. as tabs).

In the web front-end, there are different options for displaying grouped views. If not configured differently, the views are arranged vertically. You can use a style to enable horizontal or grid arrangement:



editCustomButtons	☐
editStateToggle	☐
extra	Auswählen
extra	
extraDateFormats	
groupColumnGrid	3 3 0
hideFilters	☐
hideLabel	☐
href	
localAction	☐
numberFormat	
readOnly	☐
renderMode	grid
renderMode	
target	
tooltip	
vcmDetailed	☐
vcmMarkRowClick	☐
vcmPluginCalendarOptions	Auswählen
vcmPluginChartDataColumns	
vcmPluginChartDataMode	

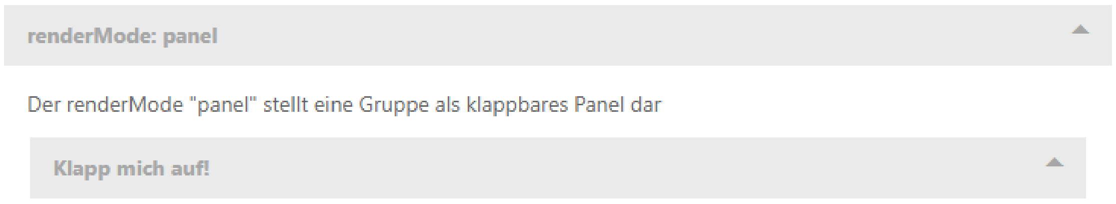
To do this, the style object is created on the group view and the value “grid” is selected as the “renderMode” and the desired grid configuration is entered under “groupColumnGrid”.


4 Spalten

8 Spalten

The example view has the grid "4 8 0." The total of summands must always be twelve.

If you select “panel” as the “rendering mode,” you get an expandable group.


renderMode: panel

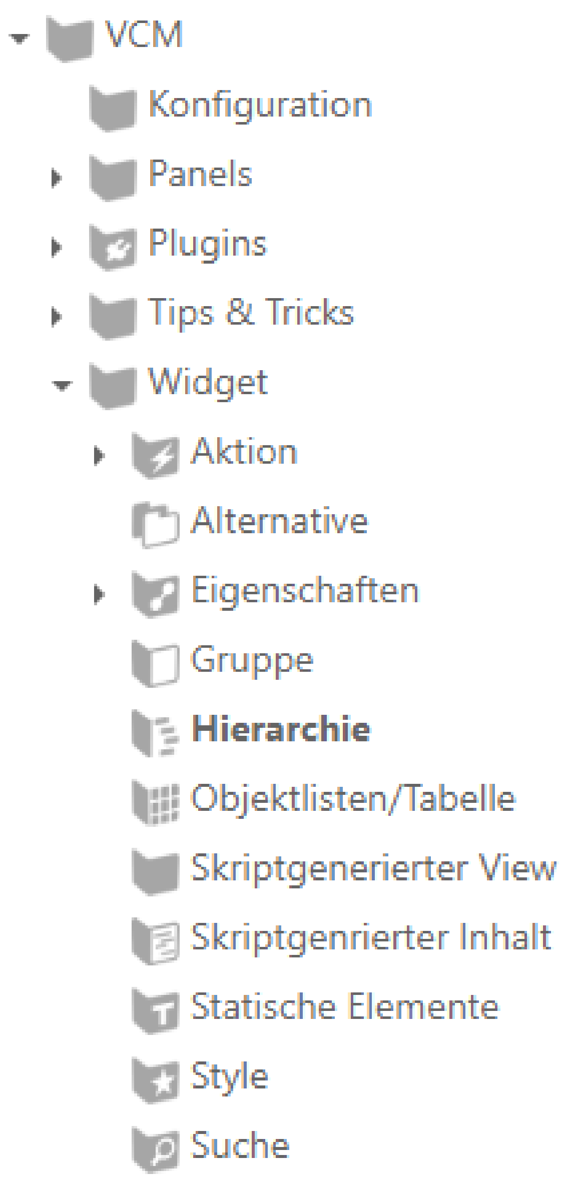
Der renderMode "panel" stellt eine Gruppe als klappbares Panel dar

Klapp mich auf!

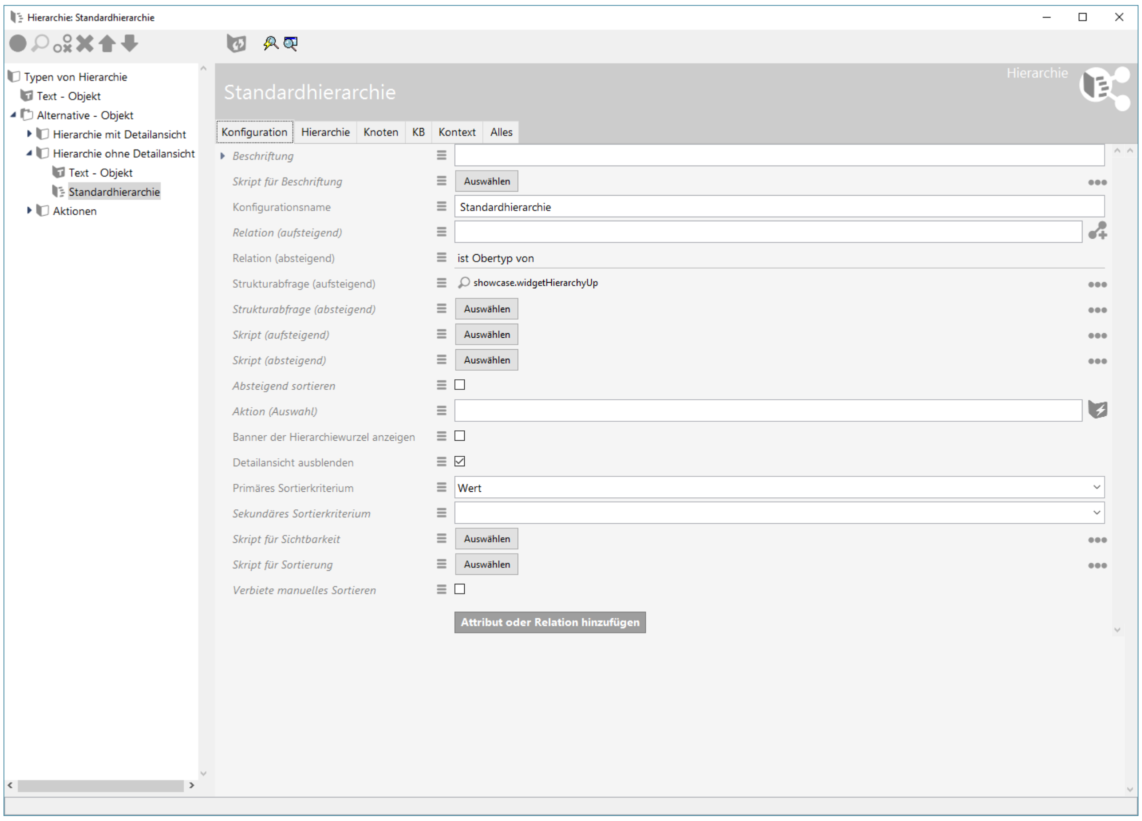
The popular Bootstrap rendering mode values "jumbotron" and "well” are also supported for the group.

3.4.4 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.


Hierarchie: Standardhierarchie

Typen von Hierarchie
- Text - Objekt
- Alternative - Objekt
  - Hierarchie mit Detailansicht
  - Hierarchie ohne Detailansicht
    - Text - Objekt
    - Standardhierarchie
  - Aktionen

Standardhierarchie

Beschriftung	
Skript für Beschriftung	Auswählen
Konfigurationsname	Standardhierarchie
Relation (aufsteigend)	
Relation (absteigend)	ist Obertyp von
Strukturabfrage (aufsteigend)	showcase.widgetHierarchyUp
Strukturabfrage (absteigend)	Auswählen
Skript (aufsteigend)	Auswählen
Skript (absteigend)	Auswählen
Absteigend sortieren	☐
Aktion (Auswahl)	
Banner der Hierarchiewurzel anzeigen	☐
Detailansicht ausblenden	☑
Primäres Sortierkriterium	Wert
Sekundäres Sortierkriterium	
Skript für Sichtbarkeit	Auswählen
Skript für Sortierung	Auswählen
Verbiete manuelles Sortieren	☐

The “Configuration” tab provides options for determining the general display of the hierarchy:

Label	The value entered here appears as the heading of the hierarchy
Script for label	As an alternative to “Label,” the title of the hierarchy can be determined in a script.
Config-uration name	The configuration name can be used to identify views and panels.

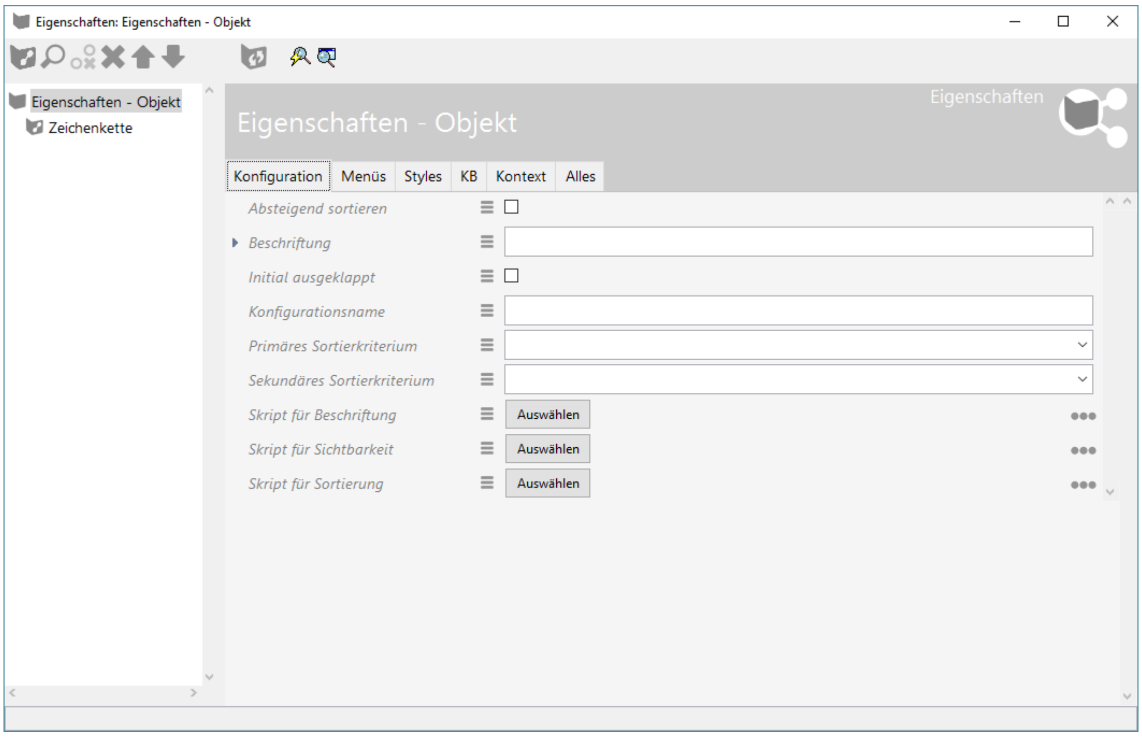
Relation (ascending) Relation (descending) Structured query (ascending) Structured query (descending) Script (ascending) Script (descending)	The hierarchy view starts with an **object as the basis**. This object is passed to the hierarchy either by the context element on the higher-level panel or by 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 network 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.
Sort downward	The hierarchy is sorted in ascending order by default. Activating the checkbox reverses this sort order.
Action (selection)	The action that should occur when a user selects an element in the hierarchy can be configured at this point.
Display the hierarchy root banner	This checkbox is used to define whether the root element banner should be displayed in the hierarchy view. This configuration is only taken allowed for in the Knowledge Builder.
Hide details view	By default, a details view is displayed for a selected node (also see Action (selection)) and can be deactivated using this option.
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 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.
Deny 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.

3.4.5 Properties

A Properties view is a list of the attributes and relations of an object.



Eigenschaften - Objekt

Absteigend sortieren	☐
Beschriftung	
Initial ausgeklappt	☐
Konfigurationsname	
Primäres Sortierkriterium	
Sekundäres Sortierkriterium	
Skript für Beschriftung	Auswählen
Skript für Sichtbarkeit	Auswählen
Skript für Sortierung	Auswählen

The “Configuration” tab features options for specifying the general display of the list:

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.
Label	The value entered here appears as the heading of the list
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.
Configuration name	The configuration name can be used to identify views and panels.
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 label	As an alternative to the “Label,” the title of the list can be determined in a script.
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.

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 group 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.

3.4.5.1 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.

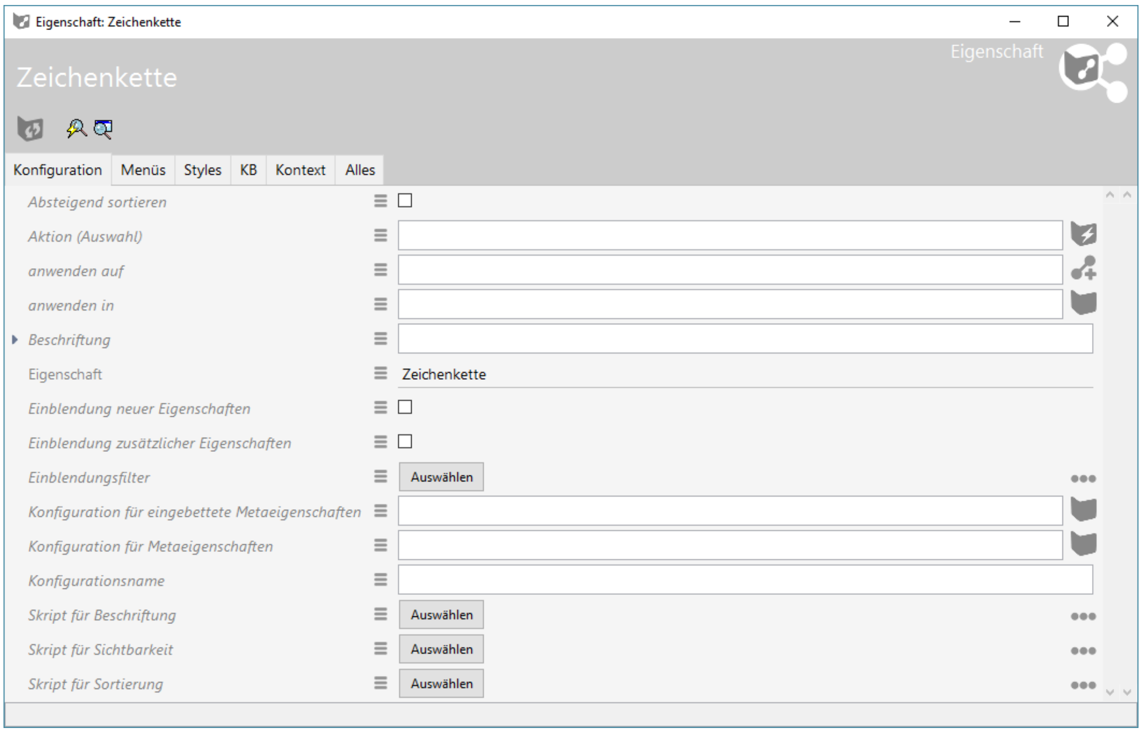
Für den "class"-Eintrag gibt es die Klasse "list", die die Aufteilung zwischen Label und darzustellendem Eigenschaftswert bestimmt. Voreingestellter Wert ist "list-5-6". Die Eigenschaften-View ist in ein gedachtes Raster von zwölf Einheiten unterteilt, wobei die letzte Einheit für die Aktion an einer 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.

3.4.6 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.



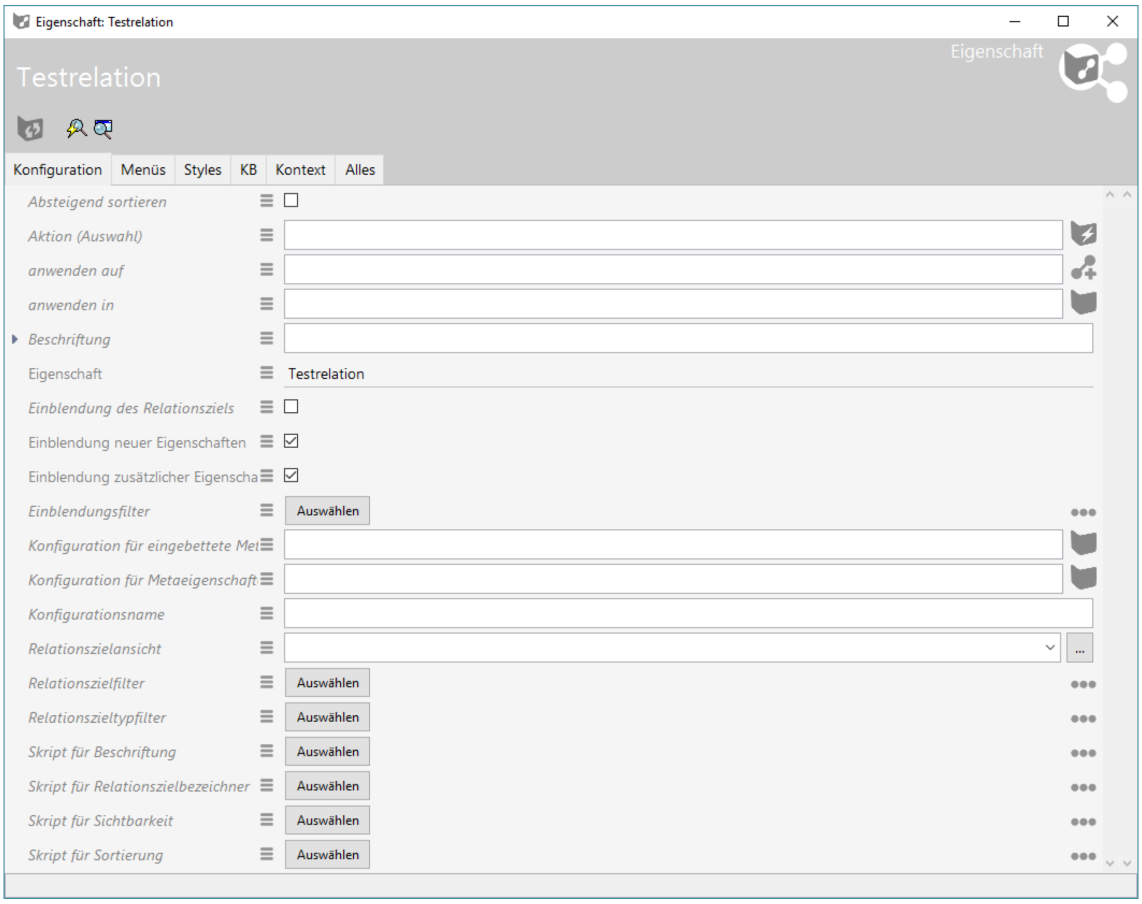
Absteigend sortieren	☐
Aktion (Auswahl)	
anwenden auf	
anwenden in	
Beschriftung	
Eigenschaft	Zeichenkette
Einblendung neuer Eigenschaften	☐
Einblendung zusätzlicher Eigenschaften	☐
Einblendungsfilter	Auswählen
Konfiguration für eingebettete Metaeigenschaften	
Konfiguration für Metaeigenschaften	
Konfigurationsname	
Skript für Beschriftung	Auswählen
Skript für Sichtbarkeit	Auswählen
Skript für Sortierung	Auswählen

In the “Configuration” tab, there are options for determining the general display of the attributes/relations:

Sort downward	Since there can be several attributes/relations of a type for object, these are sorted in ascending order of property value if there are several. This option reverses the sort order. For “Property,” you can also select a higher-level attribute type (e.g. user relation). The sort order of the subordinate attribute types, however, is determined via the sorting configuration of the corresponding Properties view. The Property view can only be used to influence the sorting within the same attribute.

Action (selection)	The action configured here is executed if the property value is selected in the front-end (e.g. by clicking).
Apply to/apply in	At this point, you can specify the context in more detail so that different configurations of a property can be used in a Properties view.
Label	A property is displayed as a name and value pair on the front-ends. The name of the configured property is used by default. You can use “Label” to display a value other than the name.
Property	The attribute/relation type of the displayed object is determined here. It is possible to specify a higher-level attribute type. For example, if “User relation” is selected, this displayed list shows all relation values for this object that are subordinate to this user relation. The sort order of these properties can be configured in the Properties view.
Show new properties	If no attribute of this type exists on the displayed object yet, the attribute would not be listed in the default view. You can use this option to configure that a placeholder with an input field for the value of the new attribute is displayed if no attribute value exists yet.
Show additional properties	If the configured attribute may occur several time, you can use this option to specify that a new placeholder for an additional attribute value is displayed automatically.
Display filter	This script can be used to filter the existing attributes of this types that only specific ones of these are taken into account in the display.
Configuration for embedded meta properties/configuration for metaproperties	If an attribute contains a meta property, a view can be placed here to determine how these meta properties are displayed. The only difference with the embedded meta properties is that they appear indented under the attribute.
Configuration name	The configuration name can be used to identify views and panels.
Script for label	Instead of using the “Label,” the displayed attribute name can be determined in a script.
Script for visibility	This script can be used to specify whether this property is supposed to be displayed.
Script for sorting	This script can be used if there are several values of a property for an object.

There are additional options for relations:


Eigenschaft: Testrelation

Testrelation

Eigenschaft

Konfiguration | Menüs | Styles | KB | Kontext | Alles

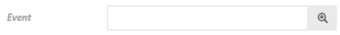
Absteigend sortieren	☐
Aktion (Auswahl)	
anwenden auf	
anwenden in	
Beschriftung	
Eigenschaft	Testrelation
Einblendung des Relationsziels	☐
Einblendung neuer Eigenschaften	☑
Einblendung zusätzlicher Eigenscha	☑
Einblendungsfilter	Auswählen
Konfiguration für eingebettete Me	
Konfiguration für Metaeigenschaft	
Konfigurationsname	
Relationszielansicht	
Relationszielfilter	Auswählen
Relationszieltypfilter	Auswählen
Skript für Beschriftung	Auswählen
Skript für Relationszielbezeichner	Auswählen
Skript für Sichtbarkeit	Auswählen
Skript für Sortierung	Auswählen

Display relation target	By default, the attribute value of the name of the relation target object is displayed as a link. This option is used to display the configured properties of the relation target. The view used for this purpose can be determined via its context, i.e. for “apply in” the corresponding view, the property view should be specified.
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.

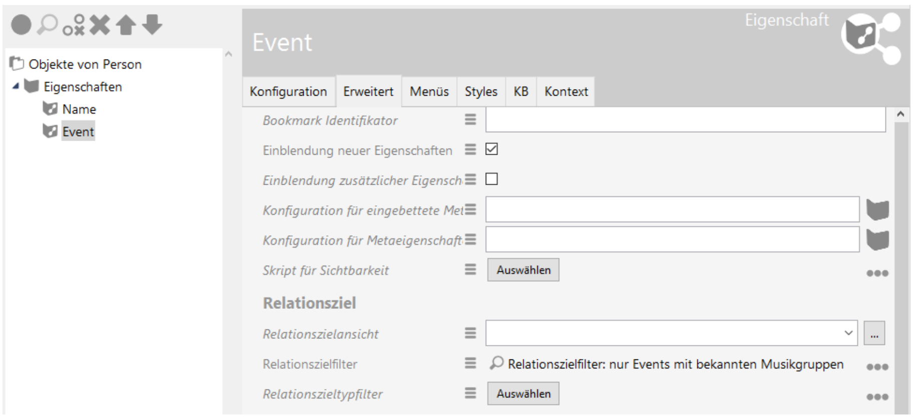
In the “Menus” tab, you can configure additional actions for the property, while the “Styles” tab lets you select certain display options. The “KB” tab features options that only apply to the Knowledge Builder and are not used in the web front-end. You can use the “Context” tab to trace in which view the Property view is used.

3.4.6.1 Relation target filter

Um den Benutzer in seiner Auswahl eines geeigneten Relationsziels zu unterstützen, kann bei Relationszielfilter eine Abfrage zur Filterung der möglichen Relationsziele angebracht werden. Klickt der Nutzer im Frontend auf die Lupe, so erhält er eine gefilterte Auswahl.

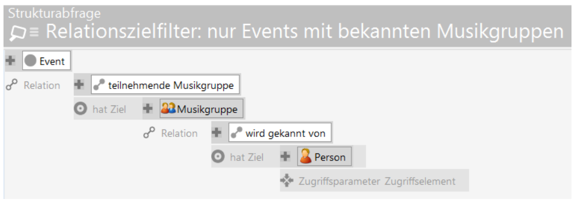


Soll zum Beispiel die Auswahl von Events, für die sich eine Person interessiert, nur auf die Events beschränkt werden, bei denen Musikgruppen auftreten, die die Person kennt, so muss die Abfrage zur Filterung der möglichen Relationsziele diese Bedingung enthalten.



Bookmark Identifikator
Einblendung neuer Eigenschaften ☑
Einblendung zusätzlicher Eigensch ☐
Konfiguration für eingebettete Met
Konfiguration für Metaeigenschaft
Skript für Sichtbarkeit: Auswählen
Relationsziel
Relationszielansicht
Relationszielfilter: Relationszielfilter: nur Events mit bekannten Musikgruppen
Relationszieltypfilter: Auswählen

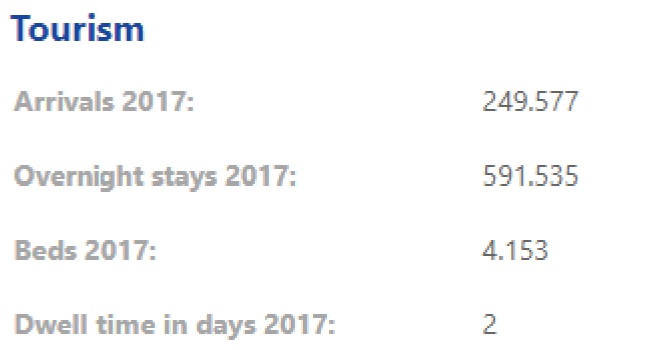
In der Abfrage kan wie gewohnt das Zugriffselement - in diesem Beispiel die Person - übergeben werden.



Standardmäßig werden Relationsziele als einfache Tabelle mit Name dargestellt. Ist eine definierte Tabelle gewünscht, kann diese erstellt und per Relation "anwenden in" mit der Eigenschaft - in diesem Beispiel "Event" - verknüpft werden.

3.4.6.2 Styling of a property view

A property in a properties-list is displayed by default as follows:



Tourism

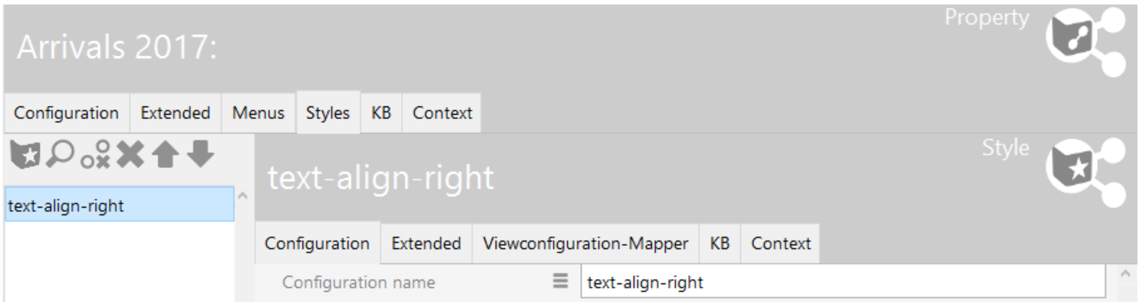
Arrivals 2017:	249.577
Overnight stays 2017:	591.535
Beds 2017:	4.153
Dwell time in days 2017:	2

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.

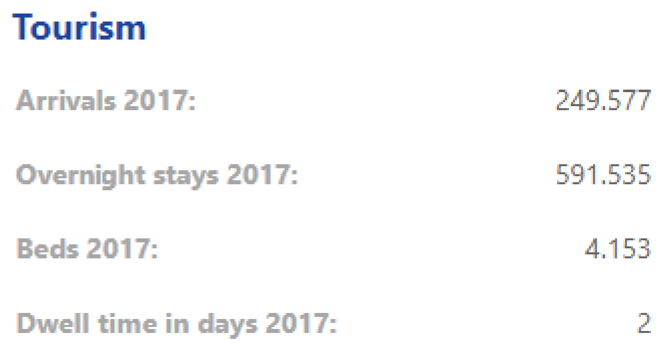
For example, if you want to display the values right-aligned, you must first create the appropriate css class:

```
.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:



Configuration name: text-align-right



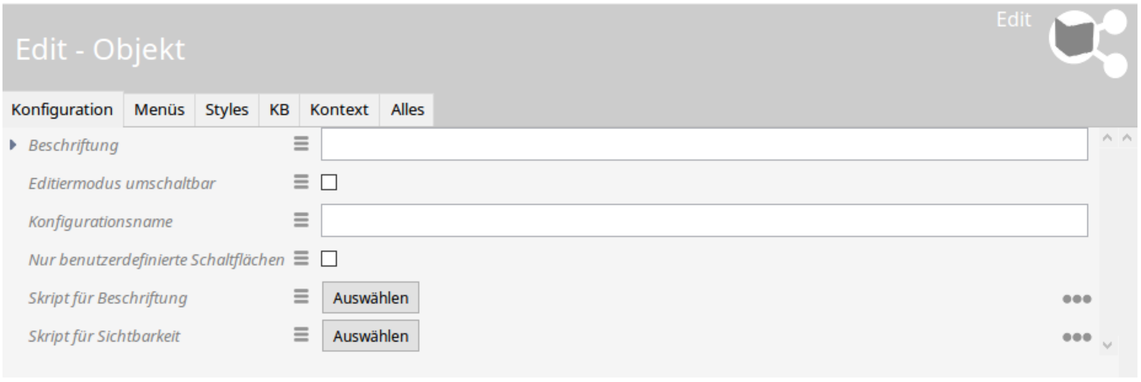
Tourism

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

3.4.7 Edit

And edit view is used to manage attributes or relations.


Edit - Objekt

Edit

Konfiguration | Menüs | Styles | KB | Kontext | Alles

Beschriftung

Editiermodus umschaltbar ☐

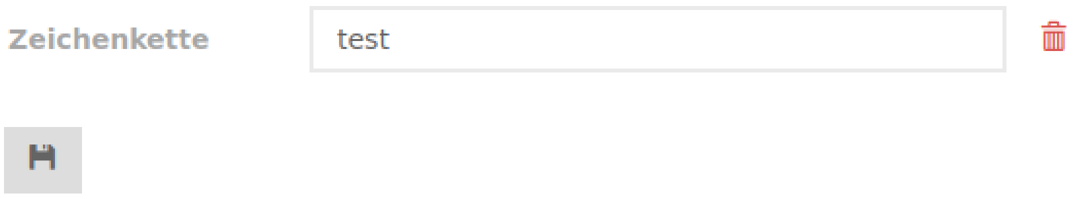
Konfigurationsname

Nur benutzerdefinierte Schaltflächen ☐

Skript für Beschriftung [Auswählen]

Skript für Sichtbarkeit [Auswählen]

In the process, all child configurations of the properties type are displayed as form fields. An edit view may contain exactly one child view. This should be a properties configuration. However, it is also possible to use a group view to use several Properties views. Changes can be synchronized with the semantic network by means of a Save button.


Zeichenkette test

The “Configuration” tab features options for specifying the general display of the edit view:

Edit mode switchable	This option makes the form mode “switchable”. That means Properties are first shown only in read mode. A Switch button can then be used to switch to edit mode.
Only custom buttons	This option can be used to configure custom buttons for switching over and saving. For this purpose, a menu with actions must be configured.

Custom buttons

If the option *Only custom buttons* is active, then users can configure their own buttons for switching and saving. When doing so, a separate ActionResponse with defined follow-ups must be configured on the respective action:

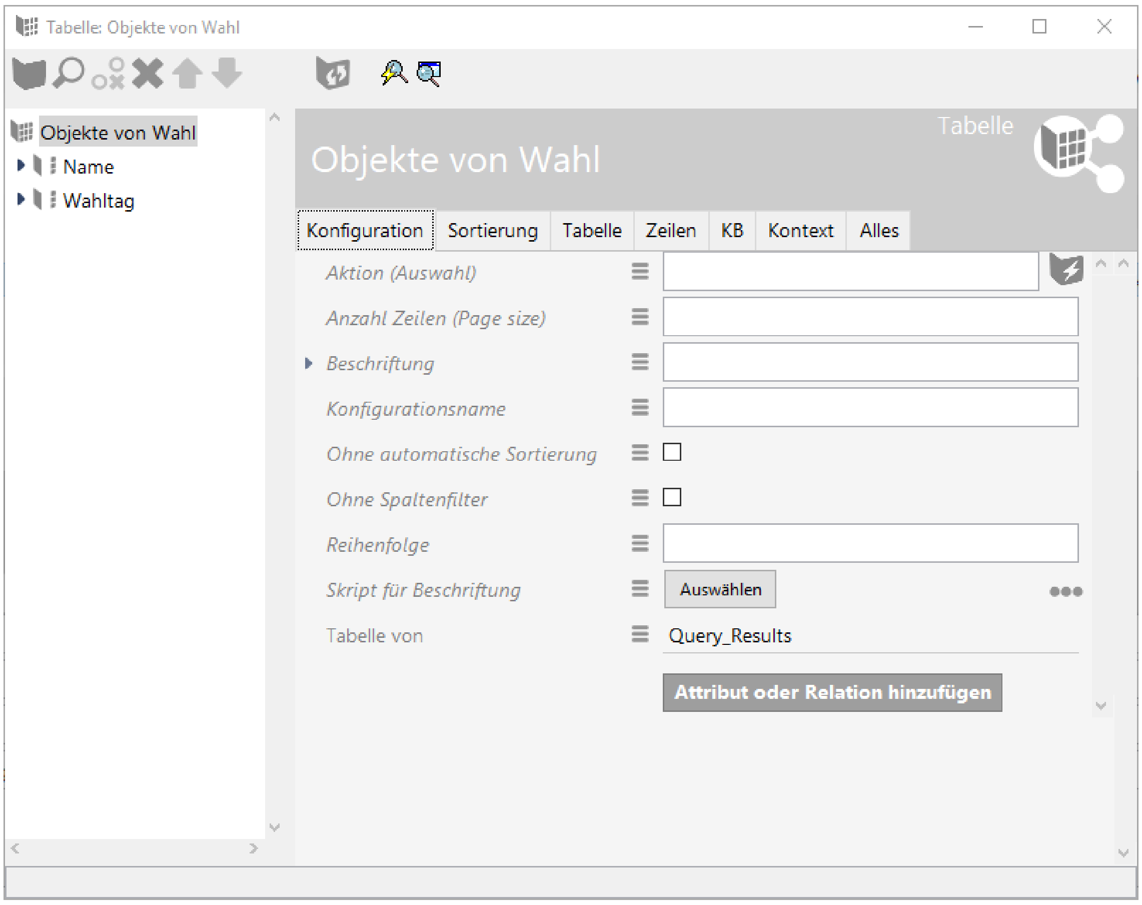
Action	**Follow-up**
Save	edit-save
Switch to read mode	edit-state-read
Switch to edit mode	edit-state-edit

Example of a user s own save button:

```
function actionResponse(element, context, actionResult) {
    var actionResponse = new $k.ActionResponse();
    actionResponse.setFollowup("edit-save");
    return actionResponse;
}
```

3.4.8 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.



Aktion (Auswahl)	
Anzahl Zeilen (Page size)	
Beschriftung	
Konfigurationsname	
Ohne automatische Sortierung	☐
Ohne Spaltenfilter	☐
Reihenfolge	
Skript für Beschriftung	Auswählen
Tabelle von	Query_Results

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.

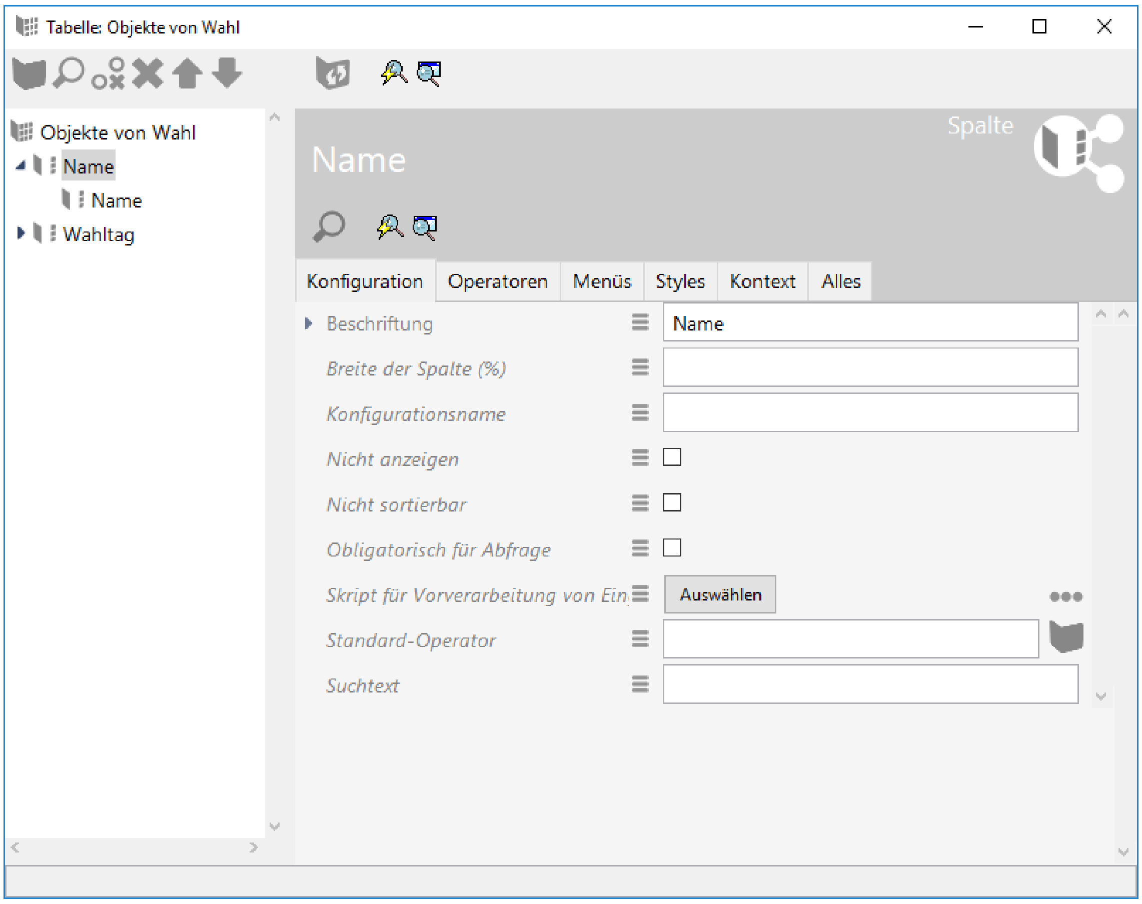


Tabelle: Objekte von Wahl

Objekte von Wahl
Name
Name
Wahltag

Spalte
Name

Konfiguration | Operatoren | Menüs | Styles | Kontext | Alles

Beschriftung: Name
Breite der Spalte (%)
Konfigurationsname
Nicht anzeigen ☐
Nicht sortierbar ☐
Obligatorisch für Abfrage ☐
Skript für Vorverarbeitung von Ein Auswählen
Standard-Operator
Suchtext

Label	Column name displayed
Column width (%)	Width of the column in percent of the width of the table
Configuration name	The configuration name can be used to identify views and panels.
Hide	This is used to hide a column. It is nonetheless calculated in the back-ground and can be used e.g. for sorting.
Cannot be sorted	In the default setting, the columns can be sorted by clicking on the header. This function can be deactivated here.
Script for label	Instead of using the "Label," the displayed attribute name can be de-termined in a script.
Script for pre-processing input fields	The text that was specified in the column filters can be influenced via a script here.
Standard opera-tor	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 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.

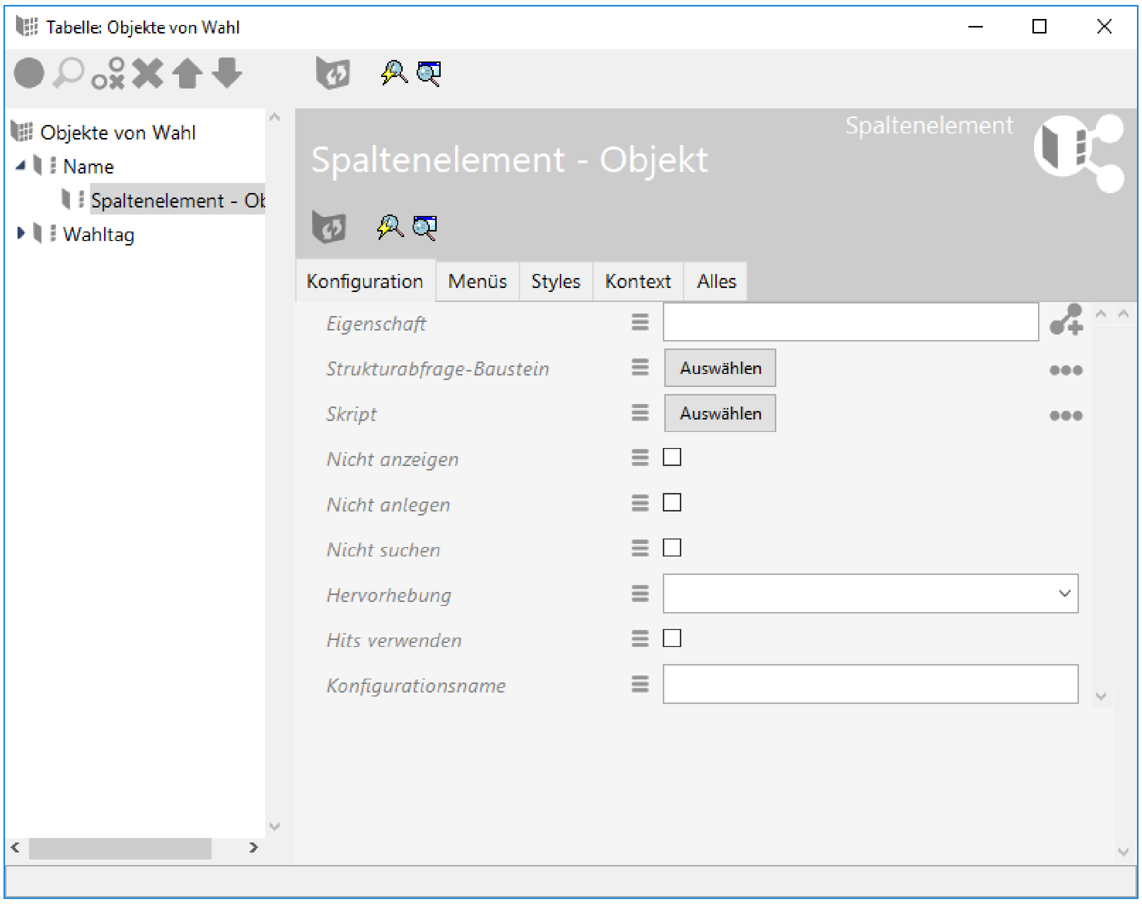


Tabelle: Objekte von Wahl

Objekte von Wahl
Name
Spaltenelement - Ob
Wahltag

Spaltenelement - Objekt
Spaltenelement

Konfiguration | Menüs | Styles | Kontext | Alles

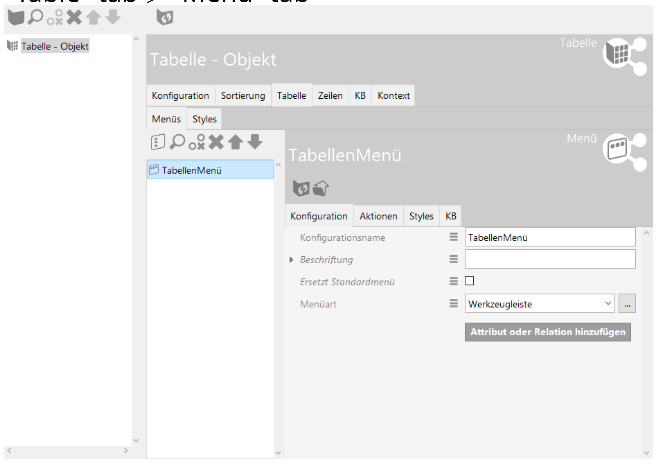
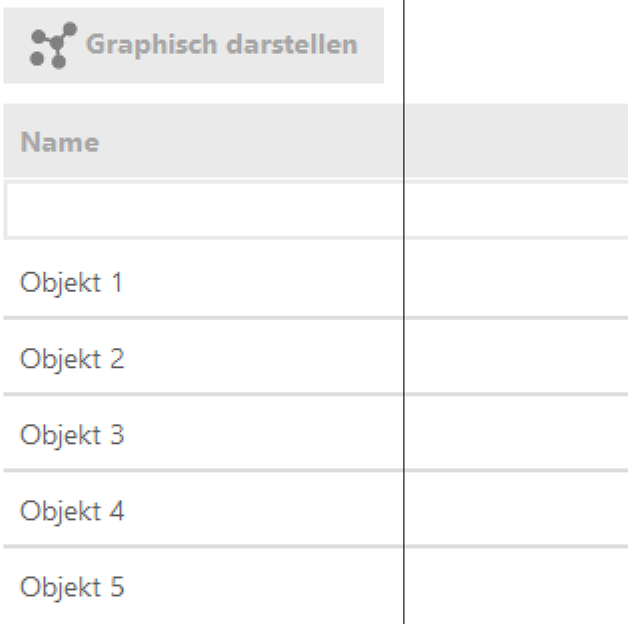
Eigenschaft
Strukturabfrage-Baustein: Auswählen
Skript: Auswählen
Nicht anzeigen: ☐
Nicht anlegen: ☐
Nicht suchen: ☐
Hervorhebung
Hits verwenden: ☐
Konfigurationsname

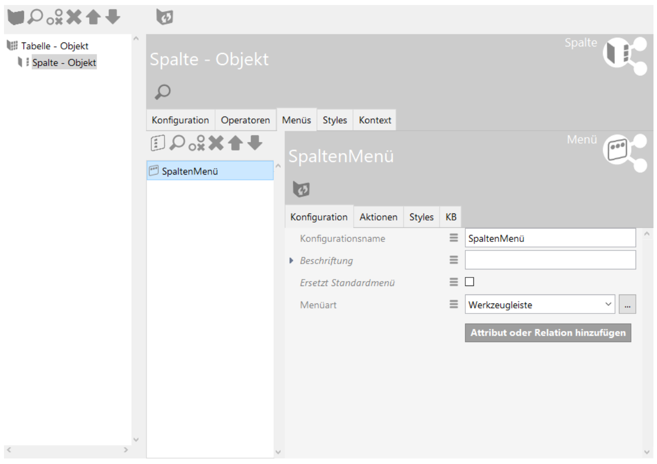
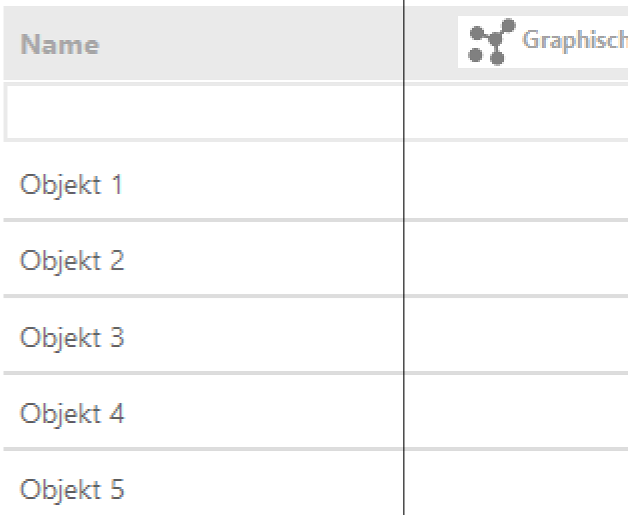
Prop-erty	The property of the element to be displayed in this column
Struc-tured query mod-ule	As an alternative to "Property," the content to be displayed can also be deter-mined using a structured query.
Script	As an alternative to the first two method, the content to be displayed can also be derived from the element via a script.
Hide	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.
High-light	This lets you choose if the content of the column element is to be highlighted by underlining it.

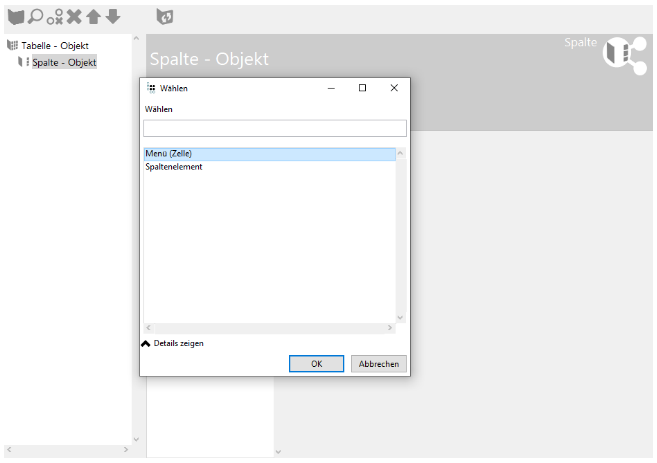
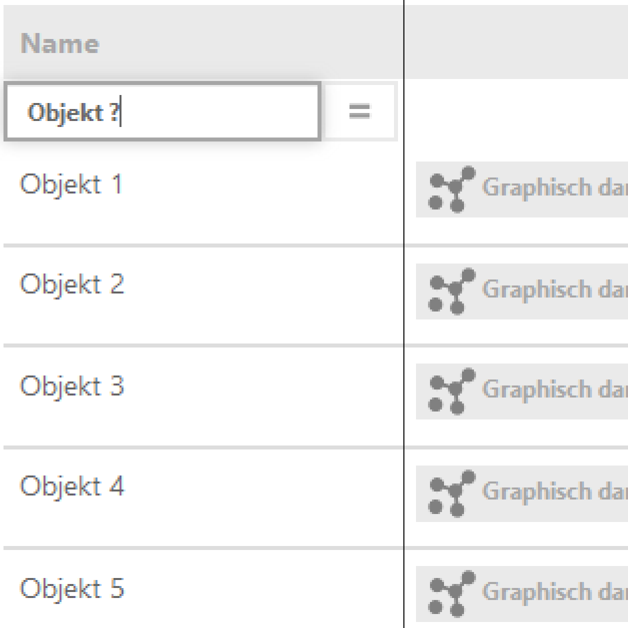
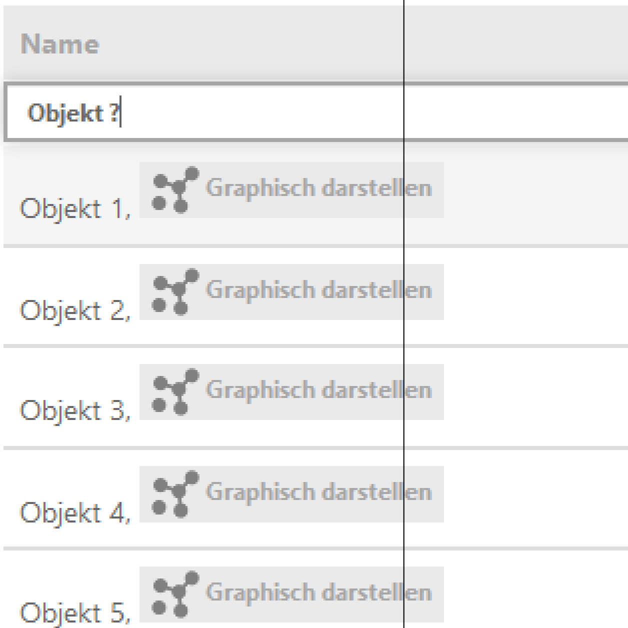
Use hits	Allows the use of all meta properties of a search result (“hit”), such as quality, cause etc. If the search results are processed further by a script, JavaScript object $k.SemanticElement or $k.Hit is forwarded.
Configuration name	The configuration name can be used to identify views and panels.

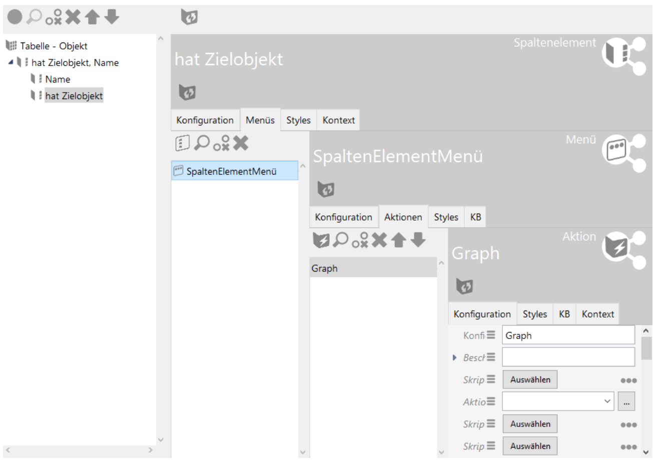
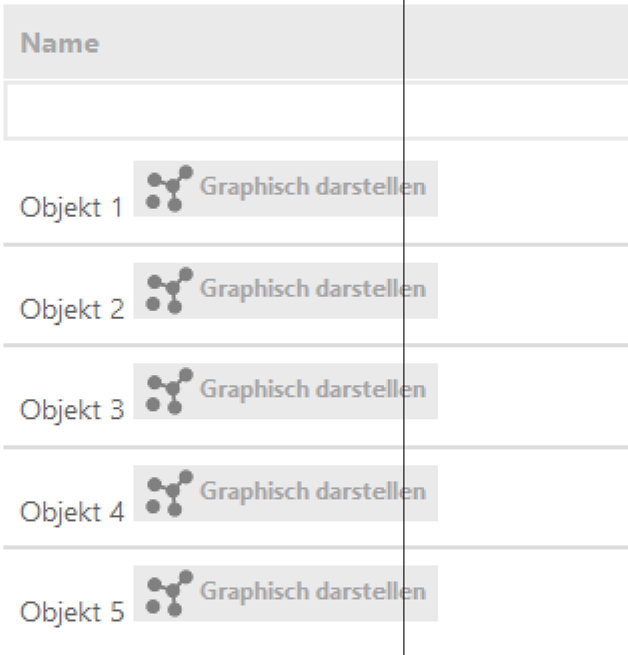
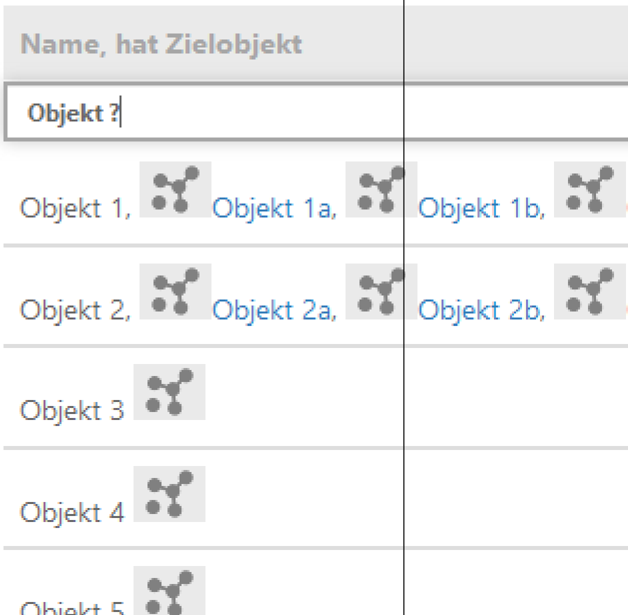
3.4.8.1 Menus in tables

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

Column: “Menus” tab  Tabelle - Objekt Spalte - Objekt Spalte - Objekt Spalte Konfiguration Operatoren Menüs Styles Kontext SpaltenMenü Menü SpaltenMenü Konfiguration Aktionen Styles KB Konfigurationsname: SpaltenMenü Beschriftung: Ersetzt Standardmenü: ☐ Menüart: Werkzeugleiste Attribut oder Relation hinzufügen	Actions are displayed in the *col-umn description* of a table:  Name | Graphisch Objekt 1 Objekt 2 Objekt 3 Objekt 4 Objekt 5

Column: Menu as a subelement of a column  Tabelle - Objekt Spalte - Objekt Spalte - Objekt Spalte Wählen Wählen Menü (Zelle) Spaltenelement Details zeigen OK Abbrechen	Actions are output *in every row* in a column: Menu in a separate column:  Name Objekt ? Objekt 1 Objekt 2 Objekt 3 Objekt 4 Objekt 5 Graphisch da Menu element in the same column as the column element to be displayed:  Name Objekt ? Objekt 1, Objekt 2, Objekt 3, Objekt 4, Objekt 5, Graphisch darstellen

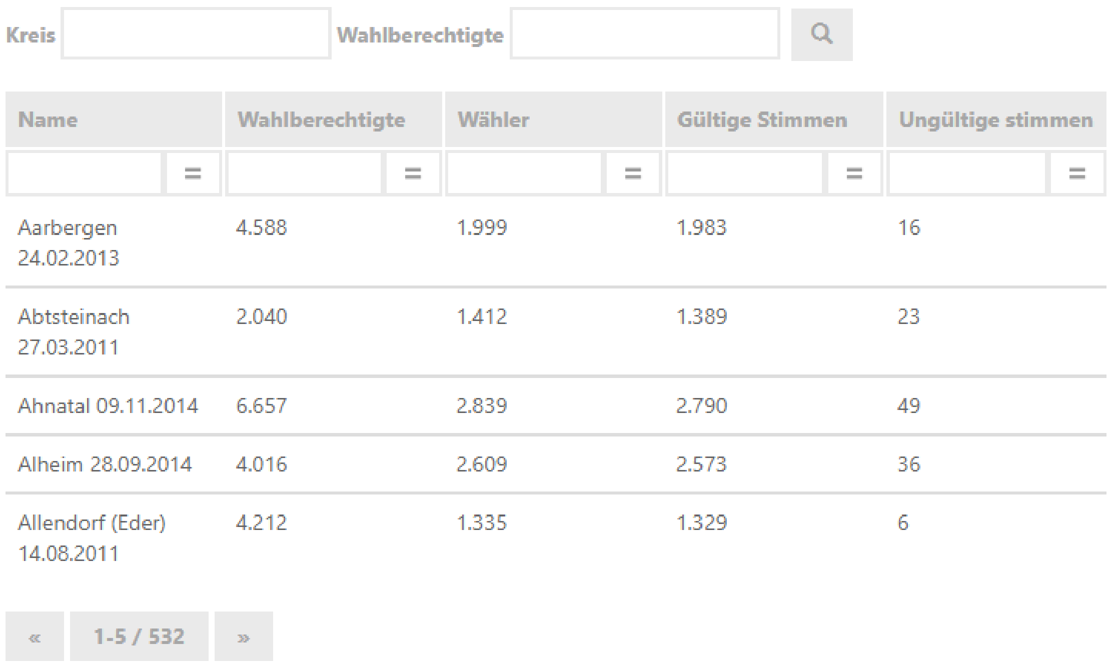
Column element: “Menus” tab  Tabelle - Objekt hat Zielobjekt, Name Name hat Zielobjekt Spaltenelement Konfiguration Menüs Styles Kontext SpaltenElementMenü Menü Aktionen KB Graph Aktion Auswählen	The action is output after every value: Output for one object per column element:  Name Objekt 1 Graphisch 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 2, Objekt 2a, Objekt 2b, Objekt 3 Objekt 4 Objekt 5

3.4.9 Search

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.

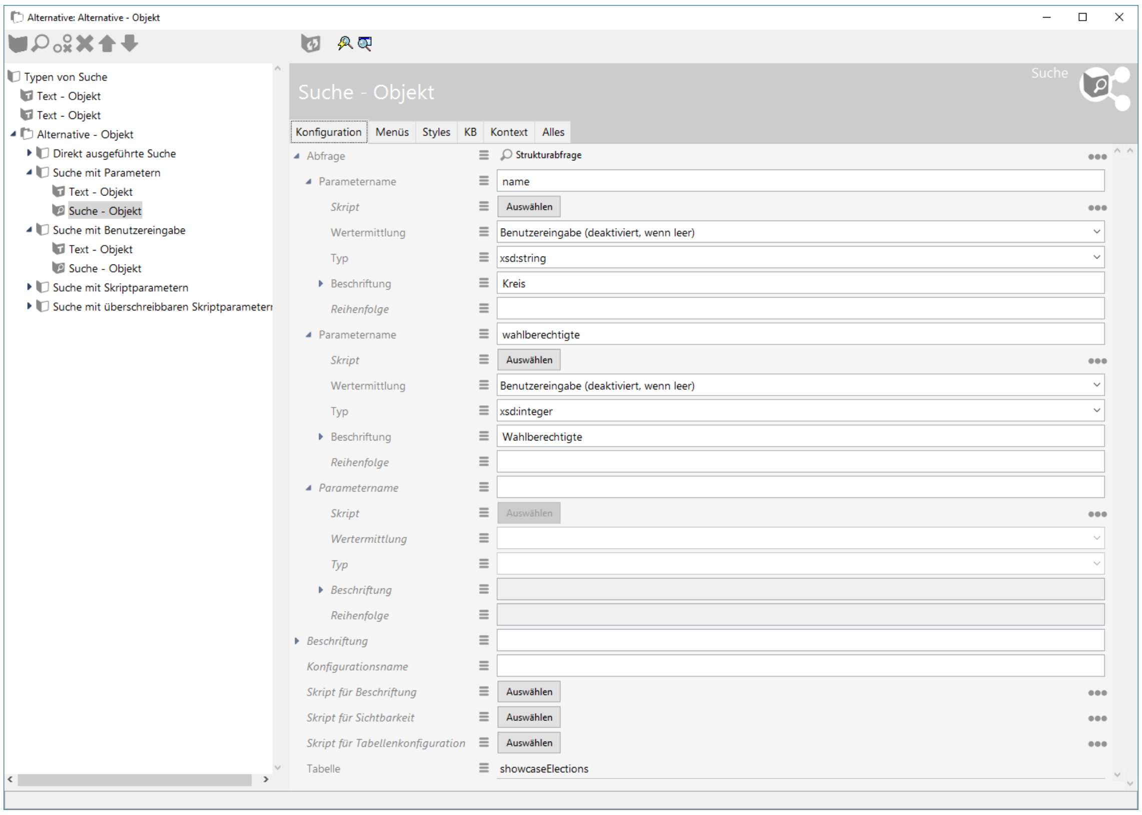
If, for example, only a search slot is supposed to be displayed in the title of a page and the results of this search then are then to be displayed further down the page, this can be achieved by configuring a search field element and search result aspect (view). Furthermore, facets can be created as well. These three configurations are described in more detail in the subsections of this chapter.

If a search with facets needs to be configured, the functional chain regarding panels influencing each other has to be obeyed: Search field aspect / action –> facet –> search result view.



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

A search view is created in the Knowledge Builder for a simple search page.


Alternative: Alternative - Objekt

- Typen von Suche
  - Text - Objekt
  - Text - Objekt
  - Alternative - Objekt
    - Direkt ausgeführte Suche
    - Suche mit Parametern
      - Text - Objekt
      - Suche - Objekt
    - Suche mit Benutzereingabe
      - Text - Objekt
      - Suche - Objekt
    - Suche mit Skriptparametern
    - Suche mit überschreibbaren Skriptparametern

Suche - Objekt
Suche

Konfiguration | Menüs | Styles | KB | Kontext | Alles

```
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
    Skript                           Auswählen
    Wertermittlung
    Typ
    Beschriftung
    Reihenfolge
Beschriftung
Konfigurationsname
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.

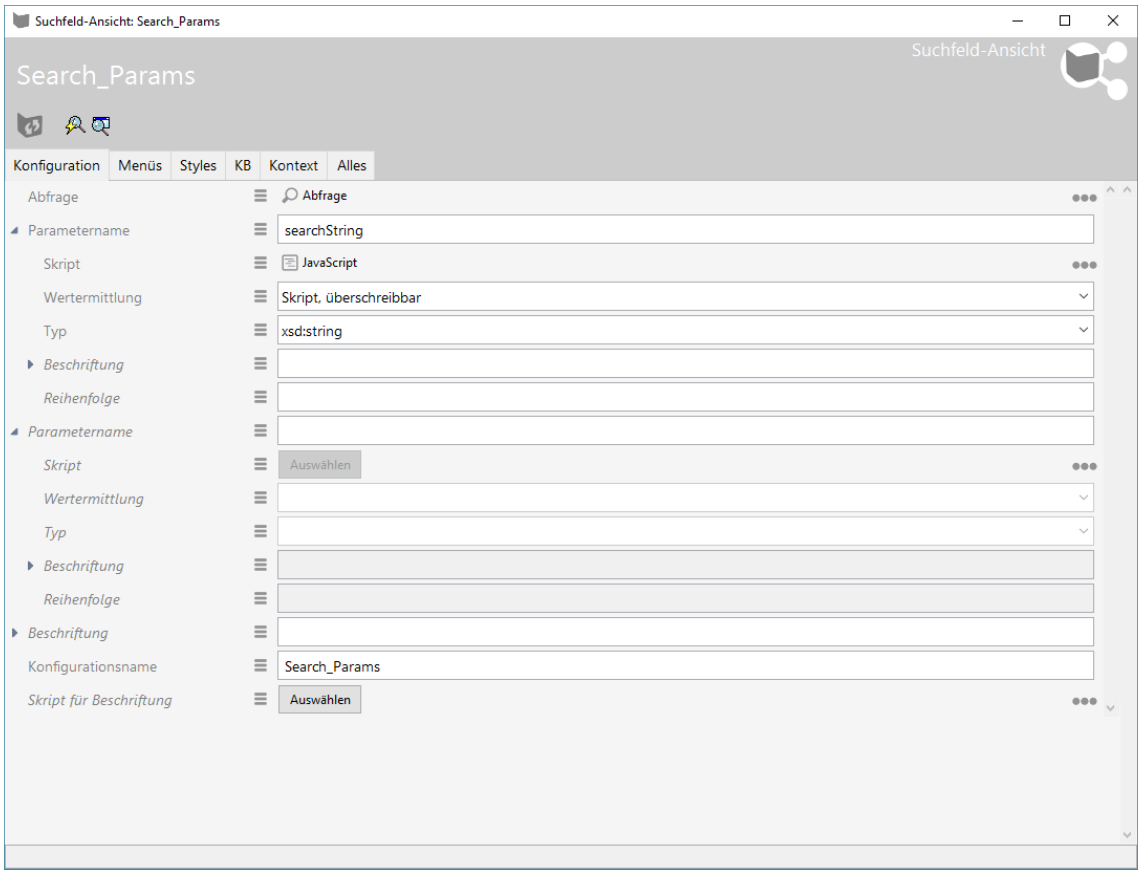
Value determination	Here you specify how the parameter value is to be determined. • “Script” (value determined via script) • “Script, can be overwritten” (the value is determined via script, but is overwritten by user input on the front-end) • “User input (optional)” (the parameter value is copied from 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	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 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.

3.4.9.1 Search field view

A search field element is used, for example, 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.


Suchfeld-Ansicht: Search_Params

Search_Params

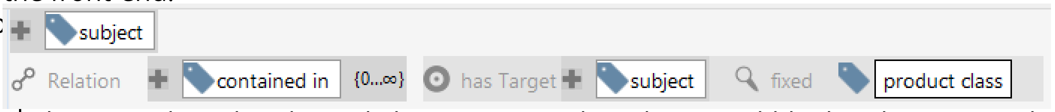
Suchfeld-Ansicht

Konfiguration | Menüs | Styles | KB | Kontext | Alles

Abfrage	Abfrage
Parametername	searchString
Skript	JavaScript
Wertermittlung	Skript, überschreibbar
Typ	xsd:string
Beschriftung	
Reihenfolge	
Parametername	
Skript	Auswählen
Wertermittlung	
Typ	
Beschriftung	
Reihenfolge	
Beschriftung	
Konfigurationsname	Search_Params
Skript für Beschriftung	Auswählen

The “Configuration” tab provides options for determining the general display of the search field:

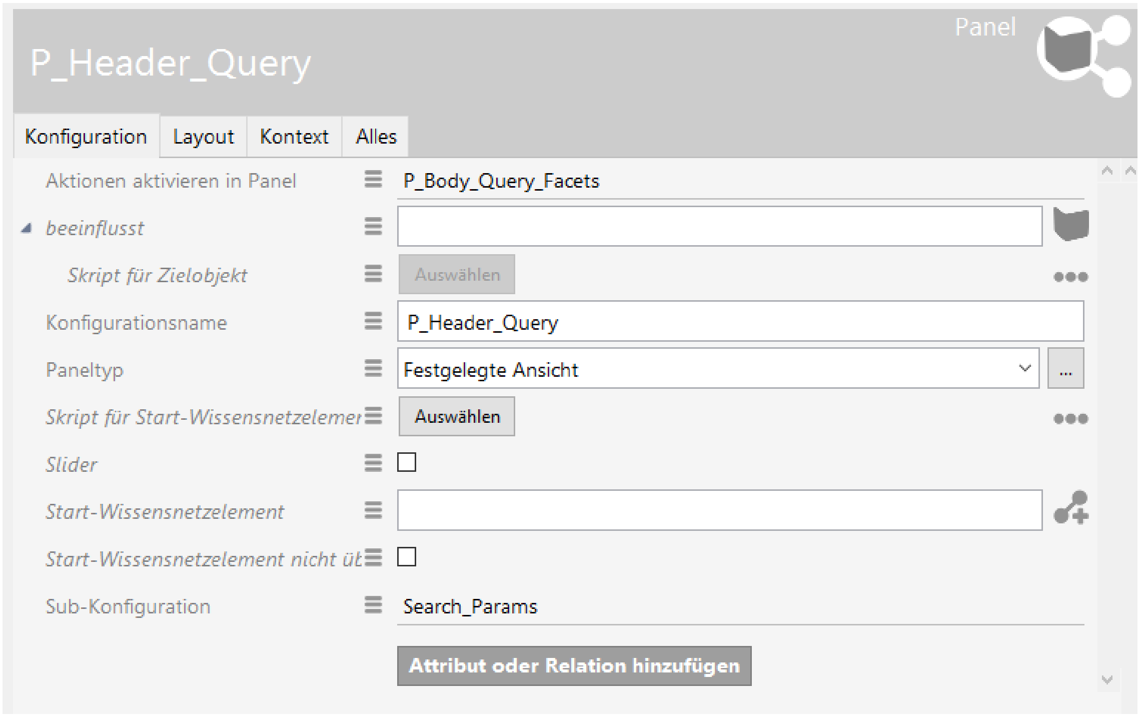
Query	This is where you configure the query that is to be executed when the query is executed.
Parameter name	Name of a search parameter. All parameters that are configured in the search must also be configured at this point to ensure no errors occur in the search.
Script	If the parameter value is to 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 semantic network) can also be used to deliver labels/strings as fixed values only (that is, without a mandatory reference to the semantic network). 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 group can be determined in a script.

In the "Menus" tab, you can configure actions for the search field element, 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 search field element is to be used and in which application contexts.

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

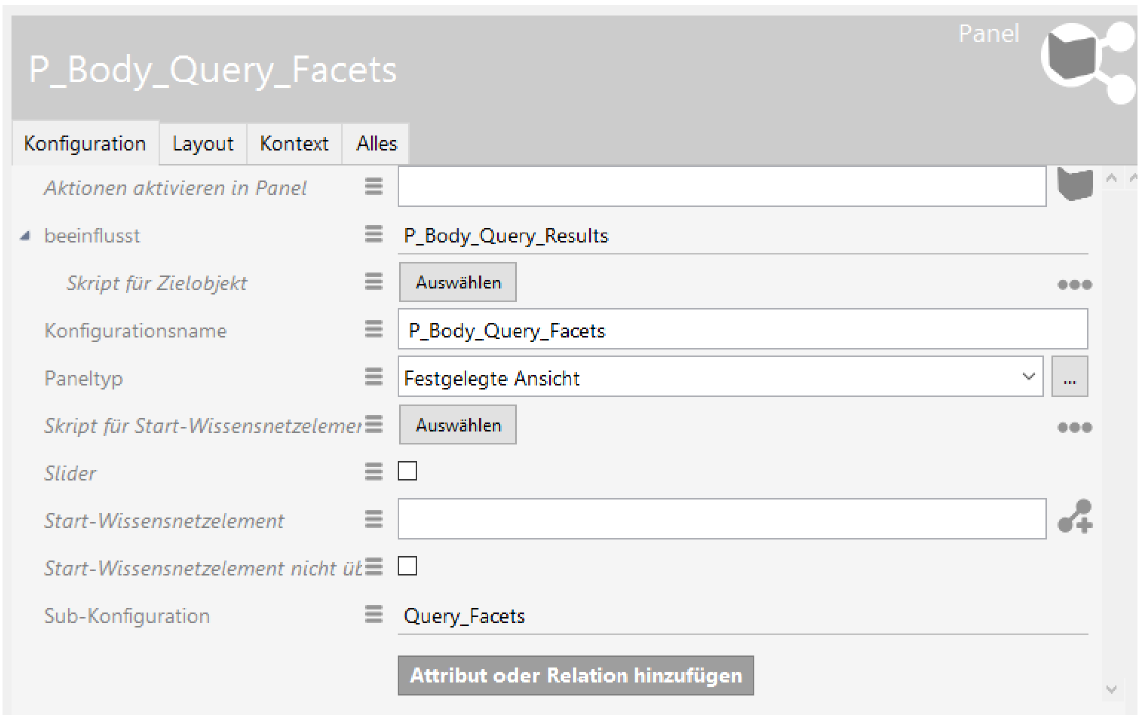
Panel

Konfiguration | Layout | Kontext | Alles

Aktionen aktivieren in Panel	P_Body_Query_Facets
beeinflusst	
Skript für Zielobjekt	Auswählen
Konfigurationsname	P_Header_Query
Paneltyp	Festgelegte Ansicht
Skript für Start-Wissensnetzelemer	Auswählen
Slider	☐
Start-Wissensnetzelement	
Start-Wissensnetzelement nicht üb	☐
Sub-Konfiguration	Search_Params

Attribut oder Relation hinzufügen

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

Panel

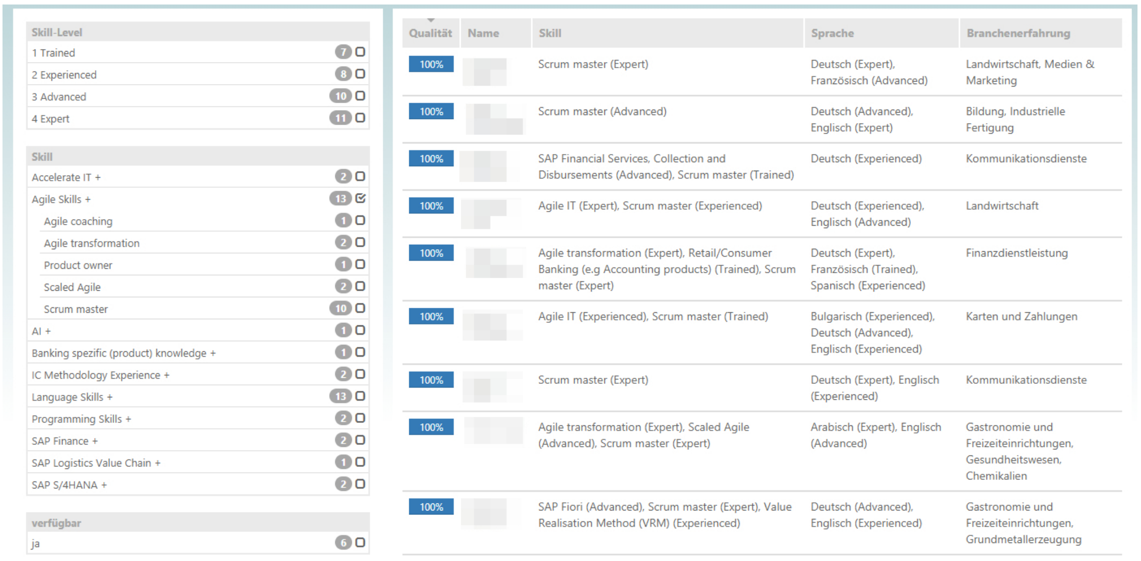
Konfiguration | Layout | Kontext | Alles

Aktionen aktivieren in Panel	
beeinflusst	P_Body_Query_Results
Skript für Zielobjekt	Auswählen
Konfigurationsname	P_Body_Query_Facets
Paneltyp	Festgelegte Ansicht
Skript für Start-Wissensnetzelemer	Auswählen
Slider	☐
Start-Wissensnetzelement	
Start-Wissensnetzelement nicht üt	☐
Sub-Konfiguration	Query_Facets

Attribut oder Relation hinzufügen

3.4.9.2 Facet view

Display



Skill-Level		
1 Trained	7	☐
2 Experienced	8	☐
3 Advanced	10	☐
4 Expert	11	☐

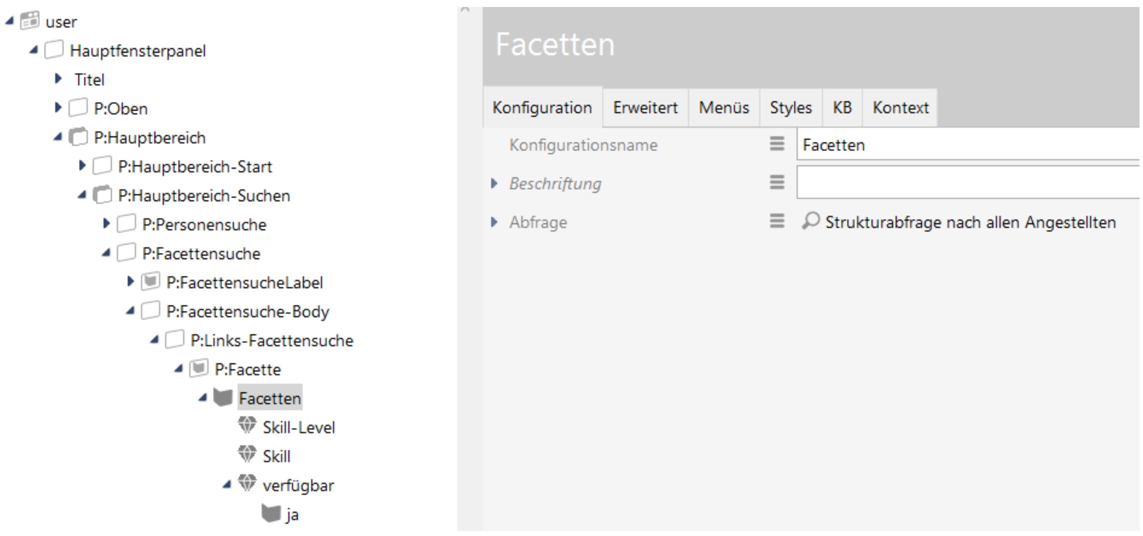
Skill		
Accelerate IT +	2	☐
Agile Skills +	13	✓
Agile coaching	1	☐
Agile transformation	2	☐
Product owner	1	☐
Scaled Agile	2	☐
Scrum master	10	☐
AI +	1	☐
Banking spezific (product) knowledge +	1	☐
IC Methodology Experience +	2	☐
Language Skills +	13	☐
Programming Skills +	2	☐
SAP Finance +	2	☐
SAP Logistics 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

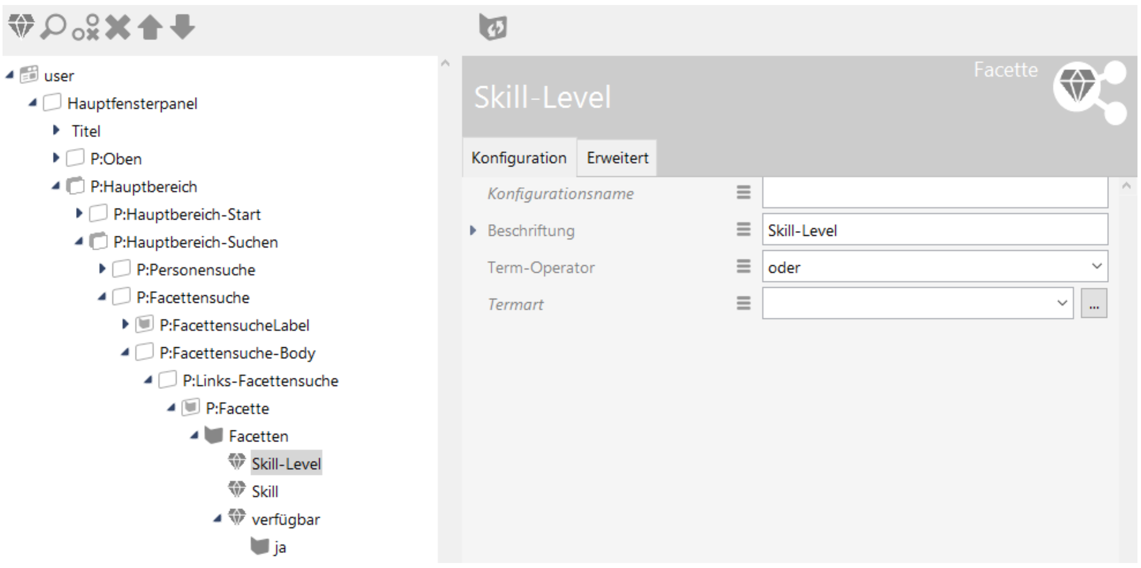
A facet view can be created as a sub configuration of a panel, but not within another view configuration elements. The panel of the facet view needs to influence the search result panel.



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.



- 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

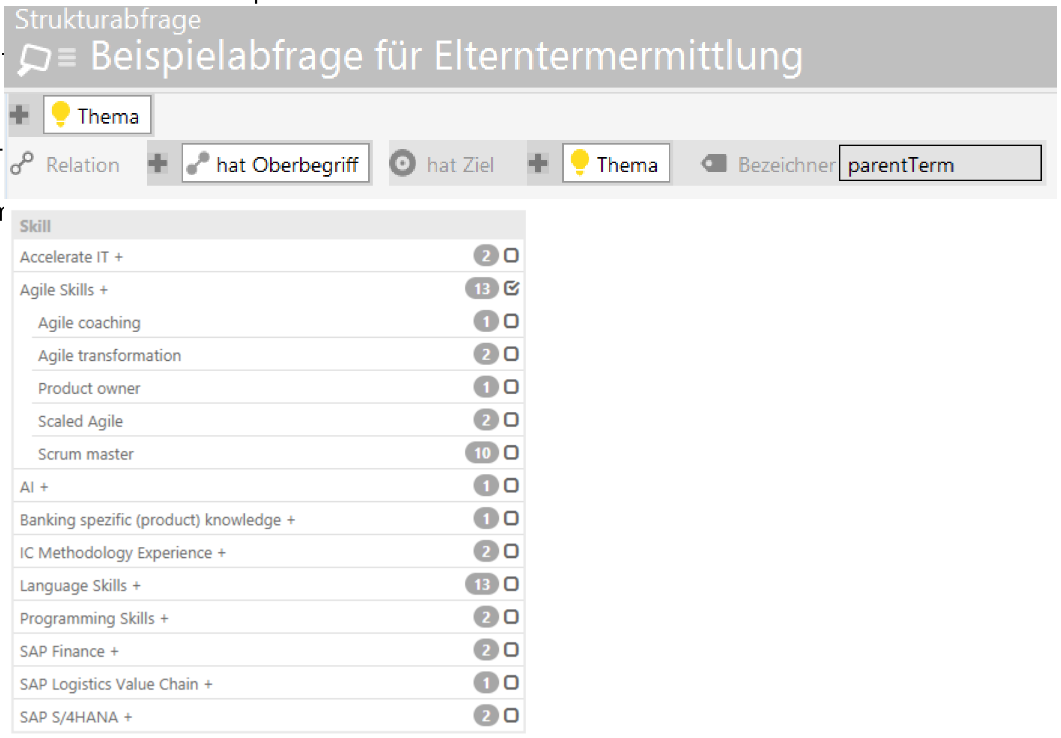
Skill-Level

Facette

Konfiguration | Erweitert

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 "parent-Term" identifies the parent element.



Strukturabfrage

Beispielabfrage für Elterntermermittlung

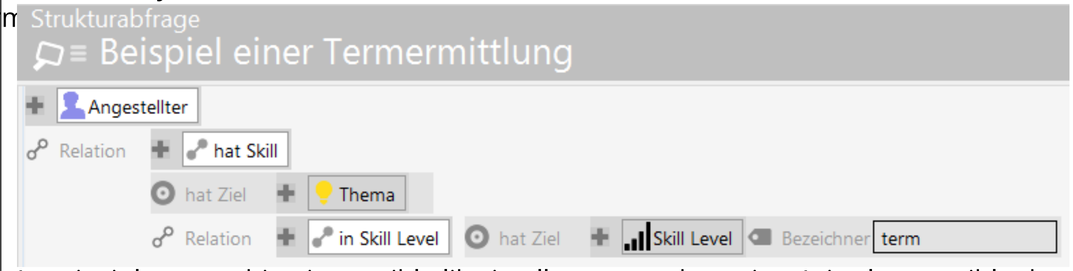
Thema

Relation | hat Oberbegriff | hat Ziel | Thema | Bezeichner parentTerm

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	☐

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.

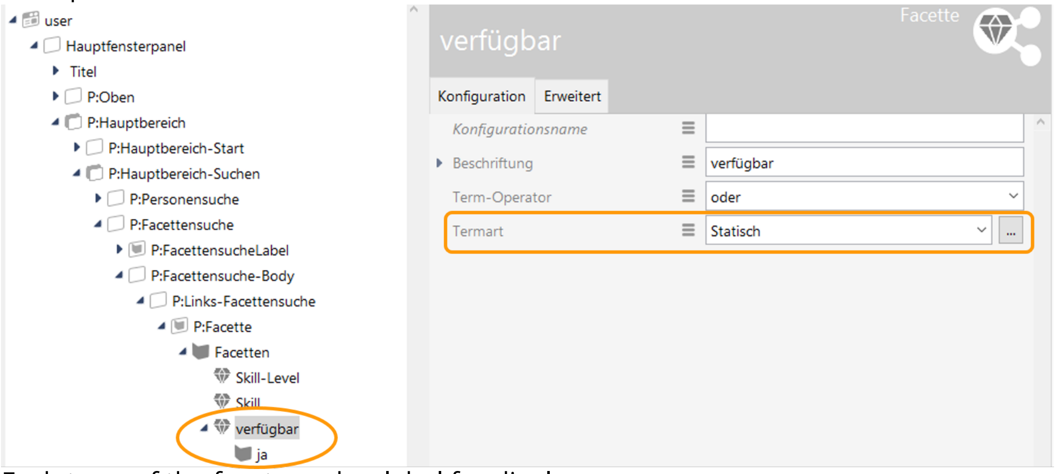
Maximum number of terms	Describes the maximum number of terms the facet can have. per default, all terms are displayed.
Do not display term number	In the front-end, the number of found terms is displayed right behind the facet title. This option deactivates this.
Term operator	At this point it is possible 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:

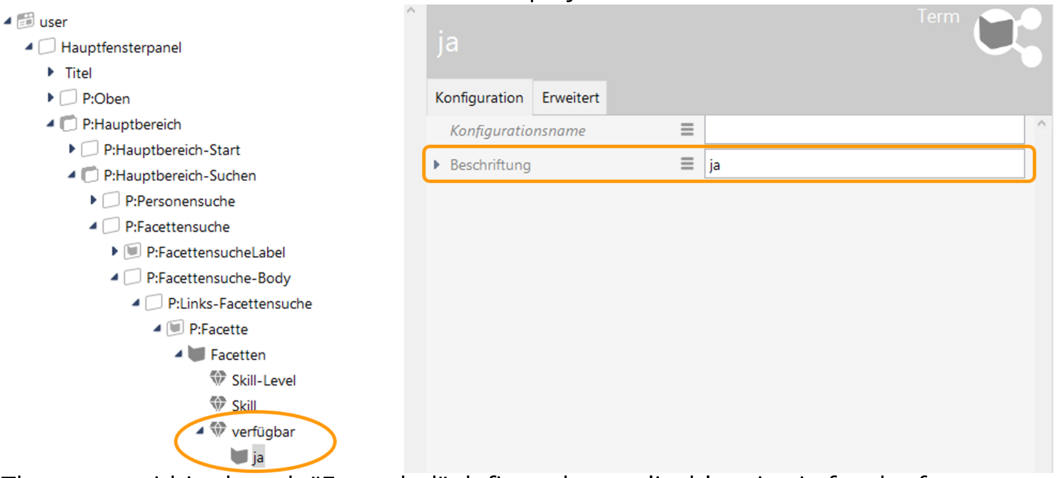


user
Hauptfensterpanel
Titel
P:Oben
P:Hauptbereich
P:Hauptbereich-Start
P:Hauptbereich-Suchen
P:Personensuche
P:Facettensuche
P:FacettensucheLabel
P:Facettensuche-Body
P:Links-Facettensuche
P:Facette
Facetten
Skill-Level
Skill
verfügbar
ja

Facette
verfügbar
Konfiguration | Erweitert

Konfigurationsname	
Beschriftung	verfügbar
Term-Operator	oder
Termart	Statisch

Each term of the facet needs a label for display:

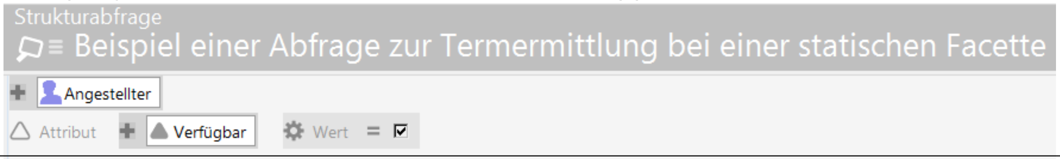


user
Hauptfensterpanel
Titel
P:Oben
P:Hauptbereich
P:Hauptbereich-Start
P:Hauptbereich-Suchen
P:Personensuche
P:Facettensuche
P:FacettensucheLabel
P:Facettensuche-Body
P:Links-Facettensuche
P:Facette
Facetten
Skill-Level
Skill
verfügbar
ja

Term
ja
Konfiguration | Erweitert

Konfigurationsname	
Beschriftung	ja

The query within the tab "Extended" defines the applicable criteria for the facet:



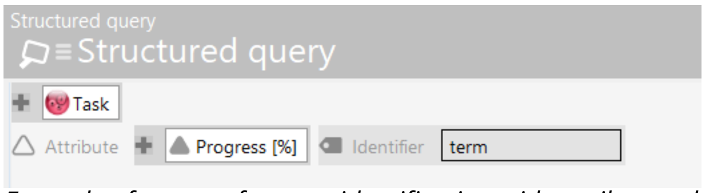
Strukturabfrage
Beispiel einer Abfrage zur Termermittlung bei einer statischen Facette
Angestellter
Attribut Verfügbar Wert = ☑

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

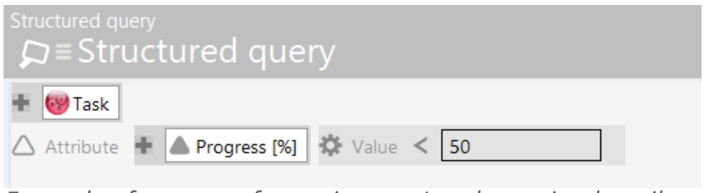
Search results can be faceted concerning predetermined attribute values, for which the term type "static" must be set. If the term type "static" is chosen, the terms must be added as a subconfiguration within a facette by clicking on the button "link new". For this purpose, the configuration is built 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:


Structured query
Structured query
Task
Attribute Progress [%] Identifier term

Example of a query for term identification with attribute values as terms

2. The facette itself has a further facette 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:


Structured query
Structured query
Task
Attribute Progress [%] Value < 50

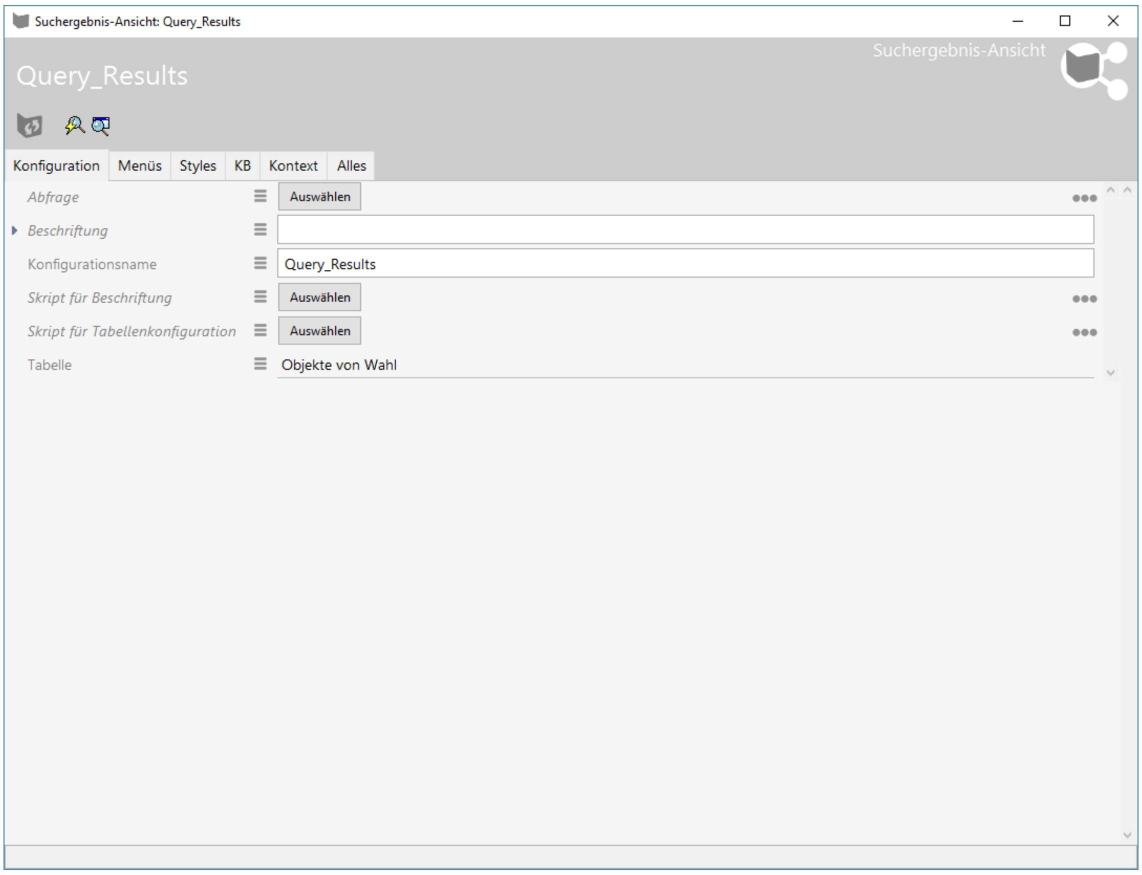
Example of a query of a static term (predetermined attribute value)

Note: The labeling of the facet term sub-configutaion is obligatory. If no label is set, the facet term will not be displayed.

3.4.9.3 Search result view

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

Konfiguration | Menüs | Styles | KB | Kontext | Alles

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 group can be determined in a script.
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.

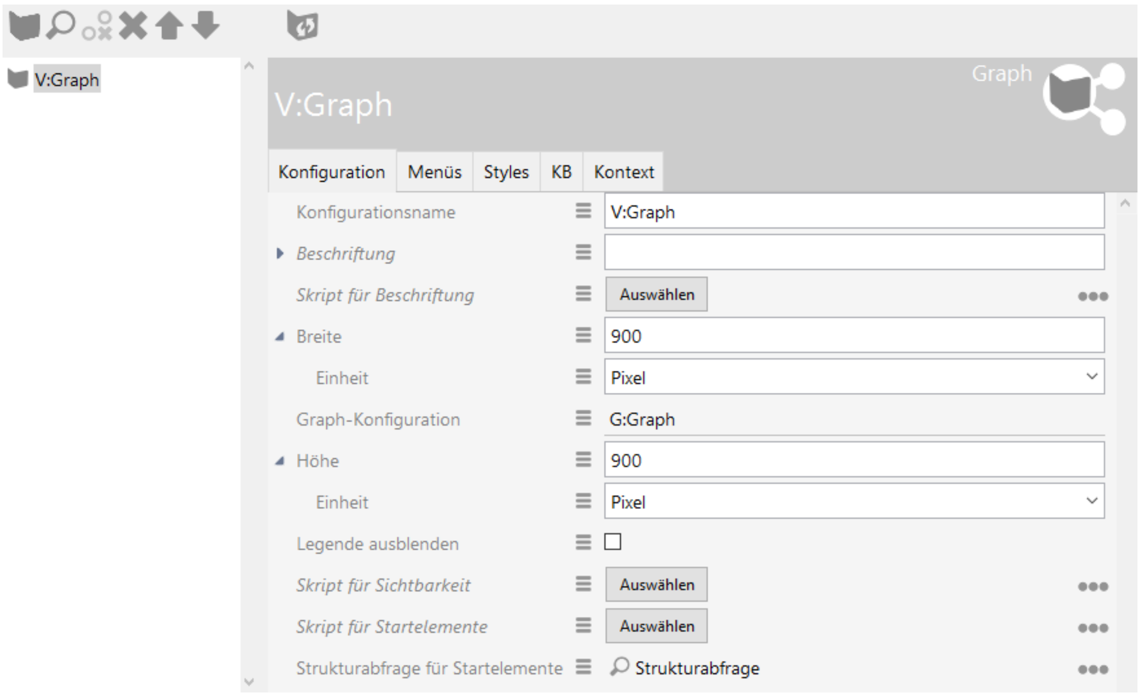
3.4.10 Graph configuration

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.*

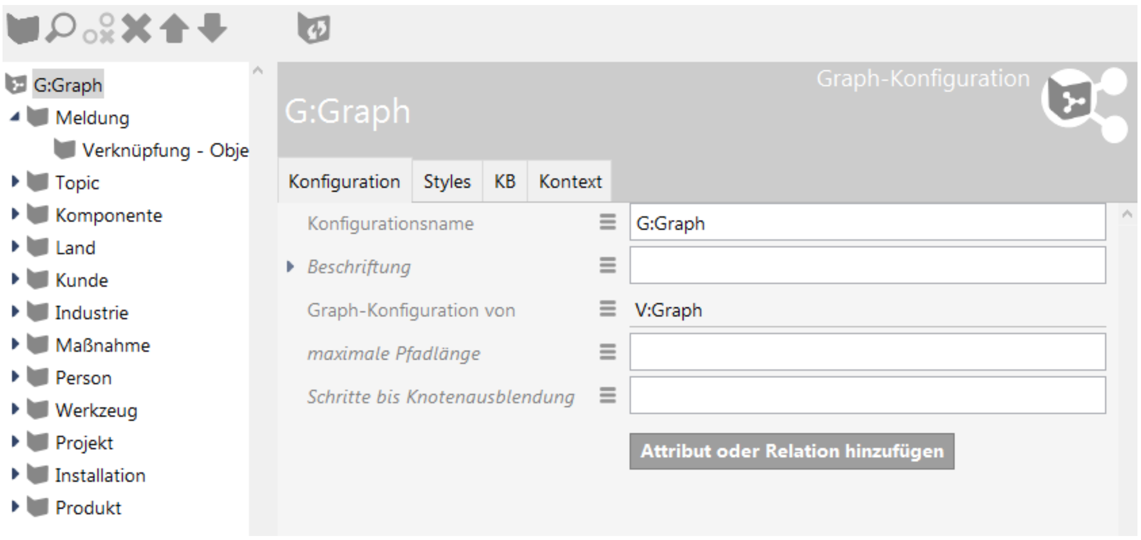
A **Graph view** and a **Graph configuration view** are required for display.
The panel in which the graph is to be displayed contains a graph view (“V:Graph”). Up to version 5.1, the context element (called start semantic element) was optional and displayed in the graph when the application started. 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.



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
Meldung
Verknüpfung - Obje
Topic
Komponente
Land
Kunde
Industrie
Maßnahme
Person
Werkzeug
Projekt
Installation
Produkt

Graph-Konfiguration
G:Graph

Konfiguration | Styles | KB | Kontext

Konfigurationsname	G:Graph
Beschriftung	
Graph-Konfiguration von	V:Graph
maximale Pfadlänge	
Schritte bis Knotenausblendung	

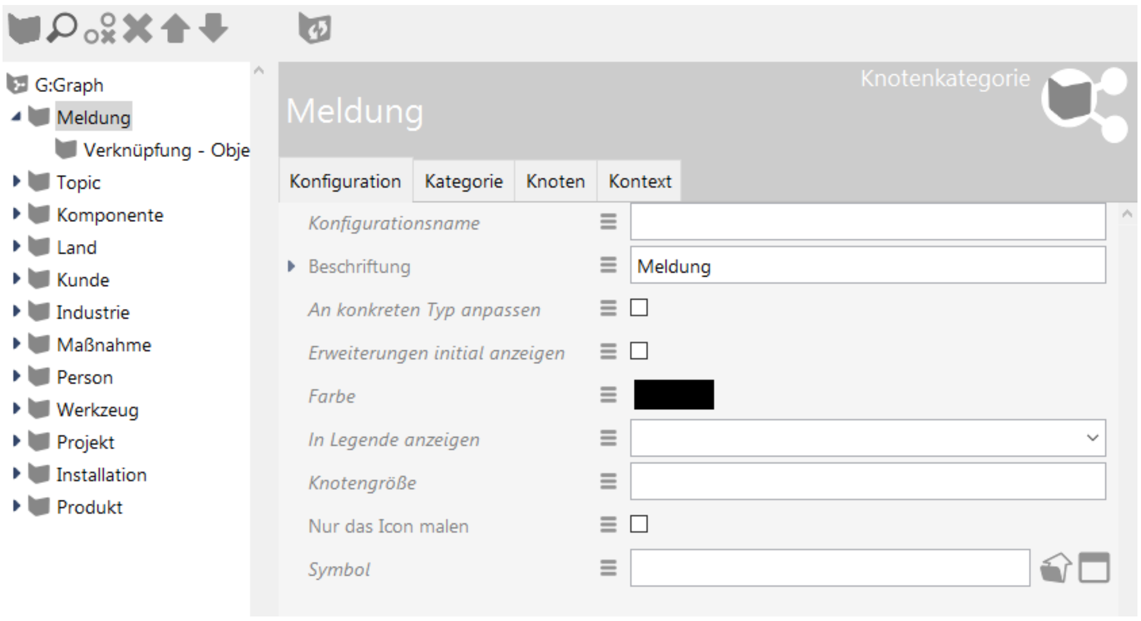
Attribut oder Relation hinzufügen

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).*

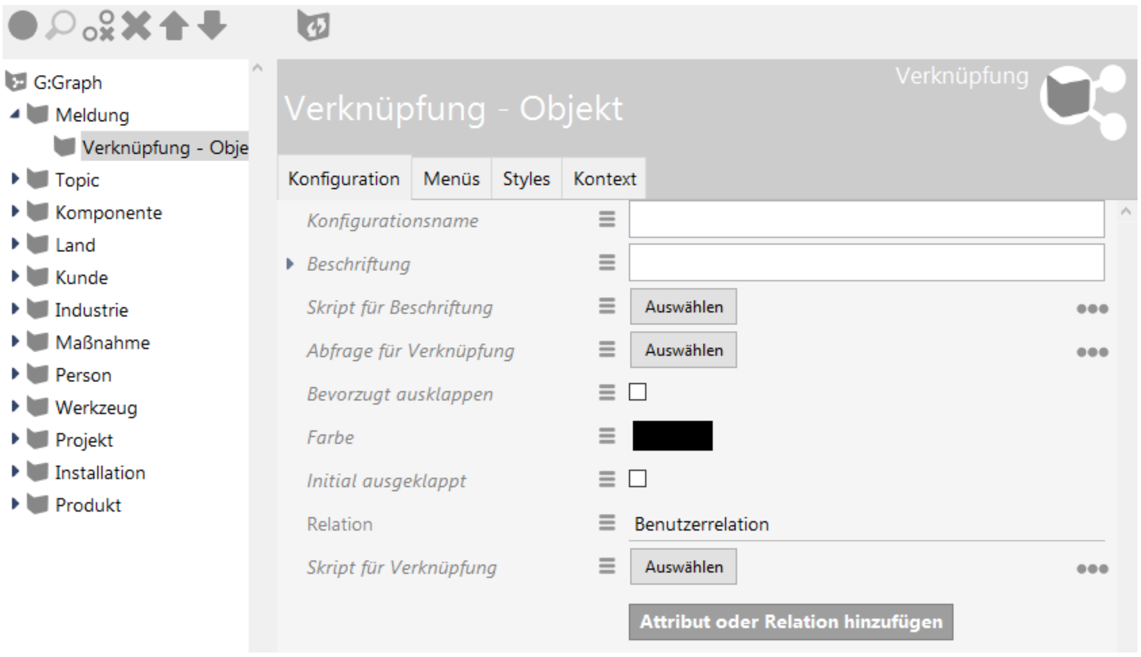

G:Graph
Meldung
Verknüpfung - Obje
Topic
Komponente
Land
Kunde
Industrie
Maßnahme
Person
Werkzeug
Projekt
Installation
Produkt

Knotenkategorie
Meldung

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

3.4.11 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 network" + $k.volume() + " generated text";
}
```

3.4.12 Image

Displays an image saved in the knowledge network 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

3.4.13 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");
}
```

3.4.14 Scriptgenerated 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.

view-Type	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
}
```

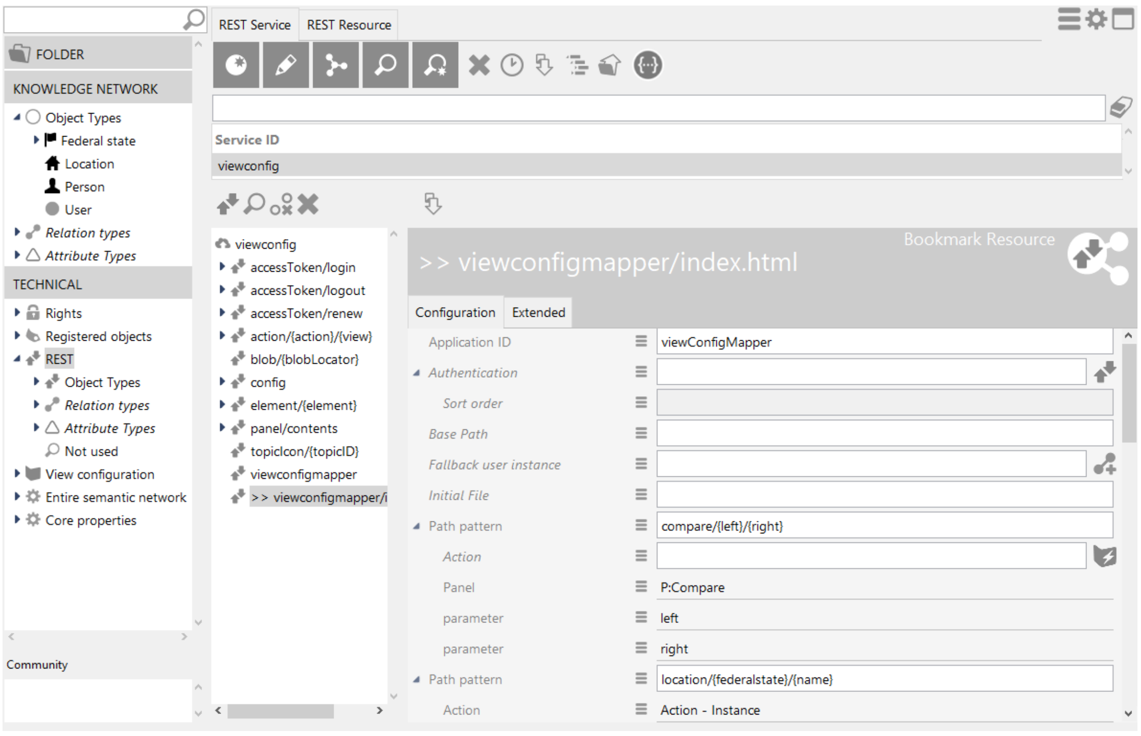
3.5 Bookmarks and history

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 and the usage of forward and backward navigation in the browser is possible. Furthermore, bookmarking improves indexability of the application by web search engines.

3.5.1 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 ressource 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.



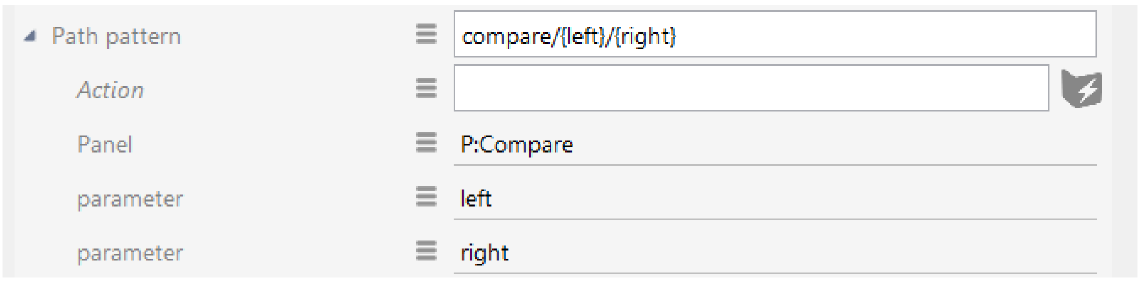
Service ID
viewconfig

>> 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:

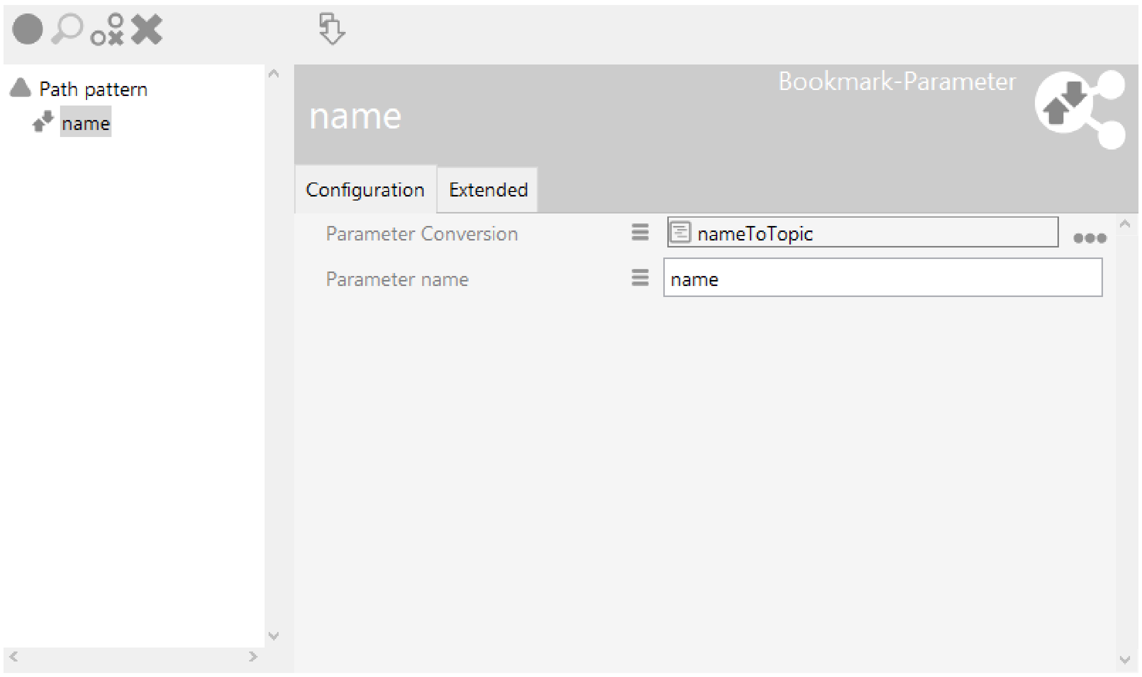
- help/{topic}
- performance/{company}/{year}

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 network 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:



name

Parameter Conversion: nameToTopic
Parameter name: name

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 retur
 * @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)
}
```

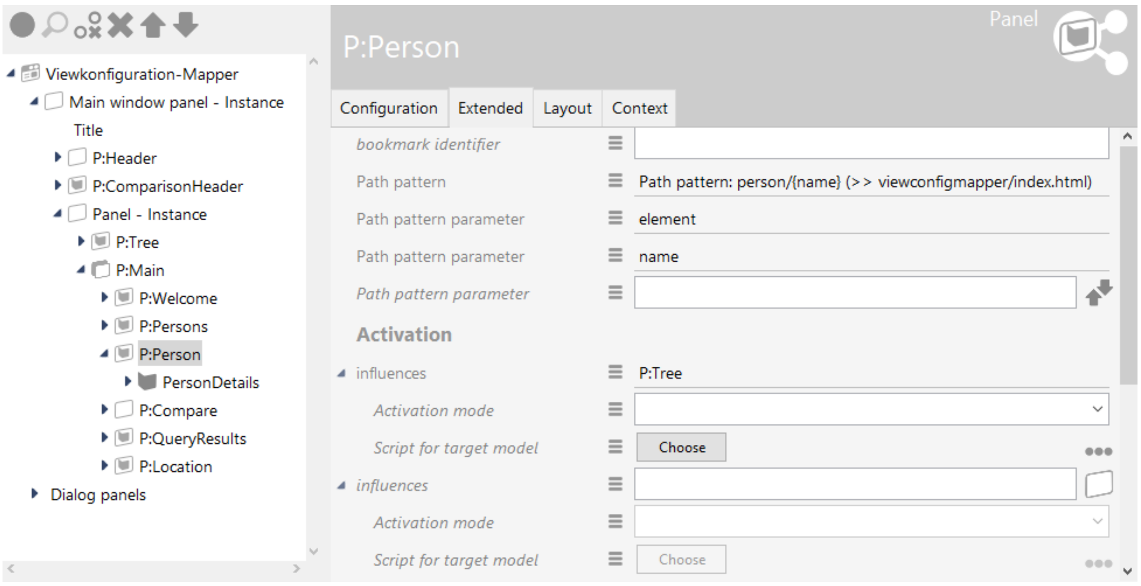
Composite parameters allow addressing of elements by means of structured descriptions (e. g. {chapter}/{version}). For each parameter fragment of the composite parameter there must be a corresponding Bookmark-Parameter object configured below the Composite-Parameter object. The Composite-Parameter object 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()".

3.5.2 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).



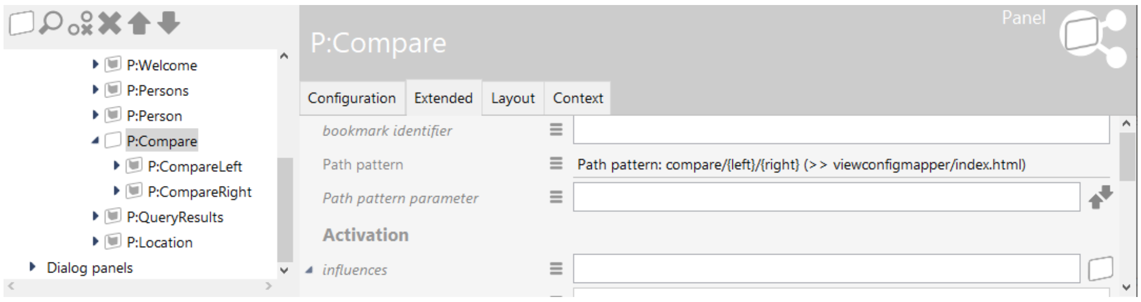
P:Person

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 ViewConfigMapper cannot decide which address pattern has to be used.

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

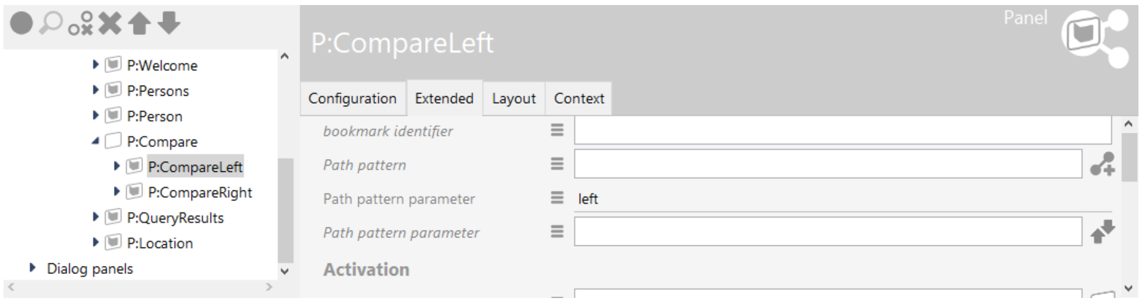
bookmark identifier

Path pattern: Path pattern: compare/{left}/{right} (>> viewconfigmapper/index.html)

Path pattern parameter

Activation

influences



P:CompareLeft

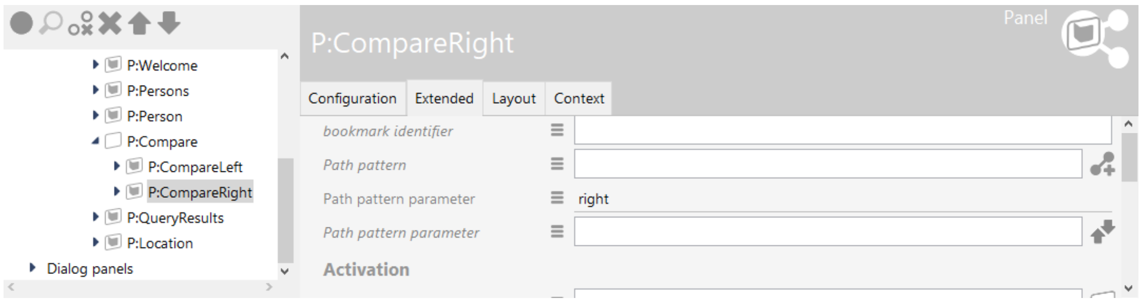
bookmark identifier

Path pattern

Path pattern parameter: left

Path pattern parameter

Activation



P:CompareRight

bookmark identifier

Path pattern

Path pattern parameter: right

Path pattern parameter

Activation

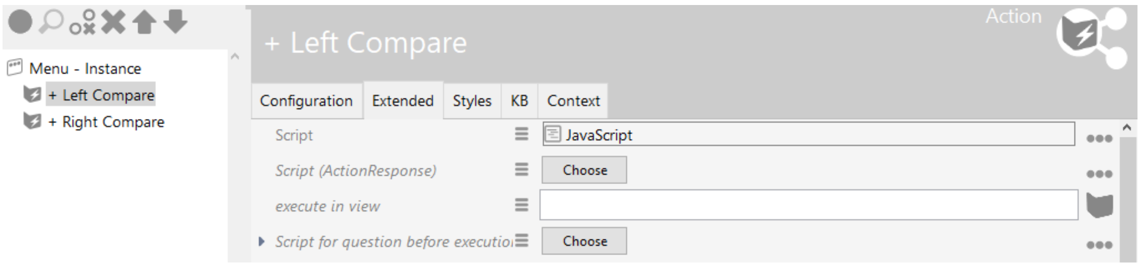
In order for the panel being able to show the needed content if activated via the path pattern, it requires a "Script for context element". The script reads the previously set session variable an gets the context element:

```
/**
 * Returns a semantic element
 * @function
 * @this $k.View
 * @param {$k.SemanticElement} element Existing semantic element
 * @returns {semantic element or undefined}
**/

function contextElement(element) {
        return $k.Session.main().getVariable('comparison.left')
}
```

Additionally, the option "Do not overwrite by external context element" must be activated.

In order to display the content in the previously mentioned panel, the session variables for the comparison action of a menu within a view needs to be set first. For this purpose, a script needs to be added:



Script	JavaScript
Script (ActionResponse)	Choose
execute in view	
Script for question before executio	Choose

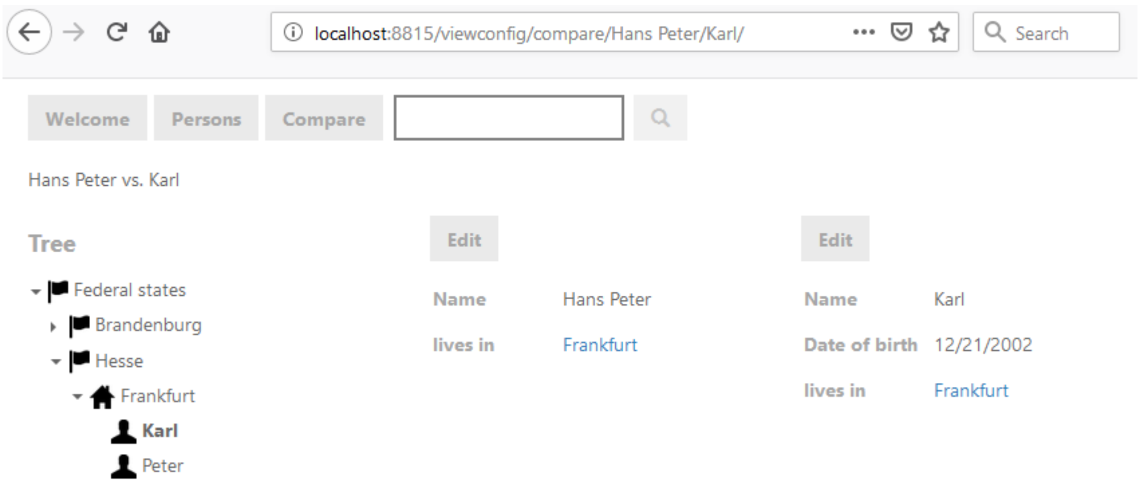
The action script for setting the session variable is shown in the following example:

```
/**
 * 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.



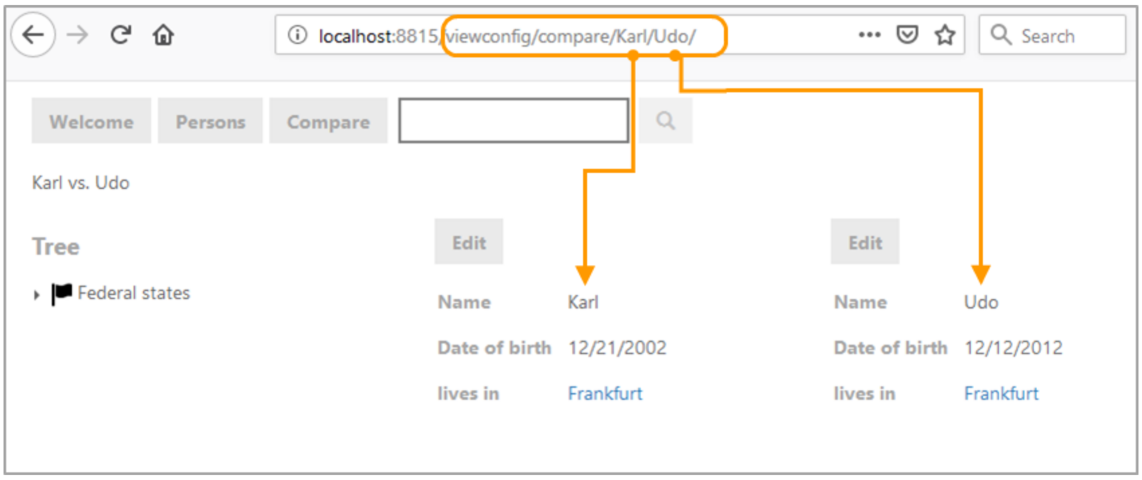
Hans Peter vs. Karl

Name	Hans Peter	Name	Karl
lives in	Frankfurt	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.



localhost:8815/viewconfig/compare/Karl/Udo/

Karl vs. Udo

Name	Karl	Name	Udo
Date of birth	12/21/2002	Date of birth	12/12/2012
lives in	Frankfurt	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.

3.5.3 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".

3.6 Plugins

In general, the following is to be documented:

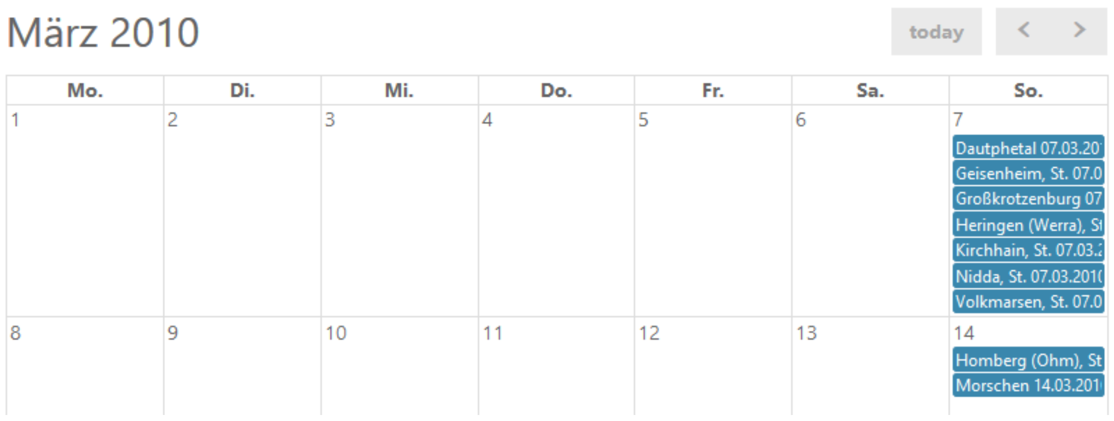
- What are plugins for
- Code example for embedding plugins in the view config mapper should **NOT** be documented here, but in the chapter on creating adjustment projects

The following should be documented for each individual plugin:

- General description of its mode of function and a screenshot
- View configuration elements to which a plugin is bound (e.g. vcm-plugin-calendar -> table view)
- Which plugin-specific styles are supported
- Plugin-specific actions (e.g. nn-expand for vcm-plugin-net-navigator)

3.6.1 vcm-plugin-calendar

The vcm-plugin-calendar can be used to display data in a calendar.



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.

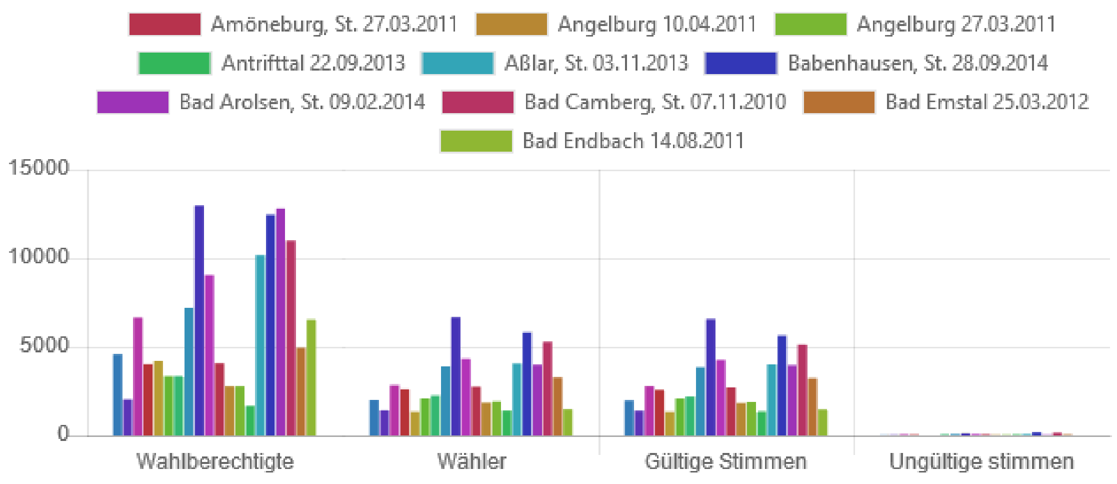
In addition, the vcmPluginCalendarOptions style attribute can be used to make additional configurations.

Further information on the plugin can be found at fullcalendar.io.

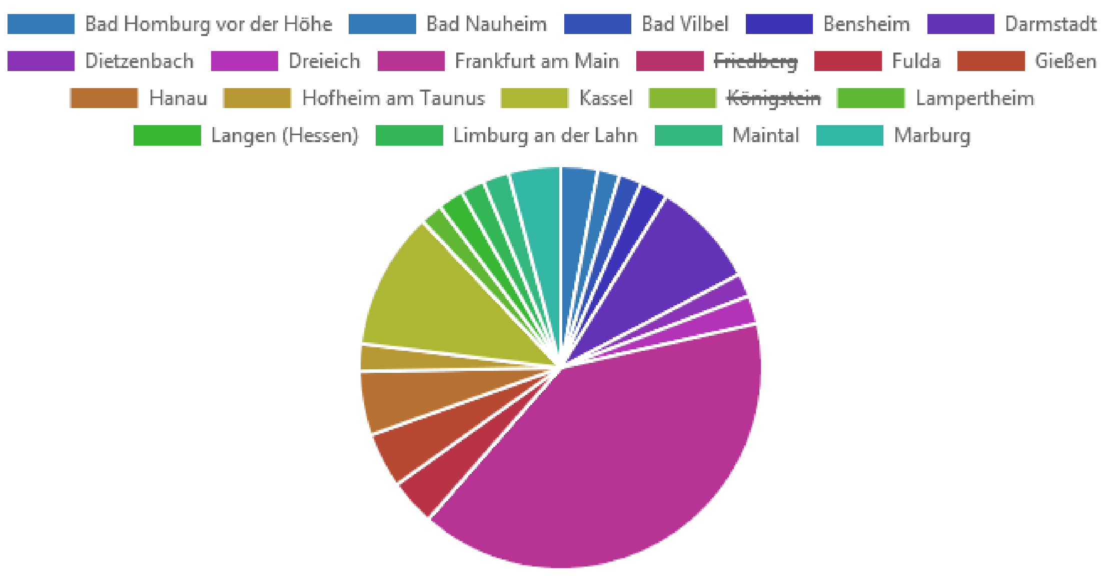
3.6.2 vcm-plugin-chart

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:



3.6.2.1 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 } }] }

   }

}
```

3.6.2.2 Configuration on basis of a scriptgenerated 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/

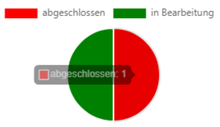
```
function customizeView(element, view) {
  var taskCount= $k.Registry.type(‘‘job").allInstances().reduce(function (result, job, index) {
        var status = job.attributeValueString(‘‘statusJob");
        result[status]= (result[status]||0)+1
        return result;
```

```
    }, {})

   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;
}
```

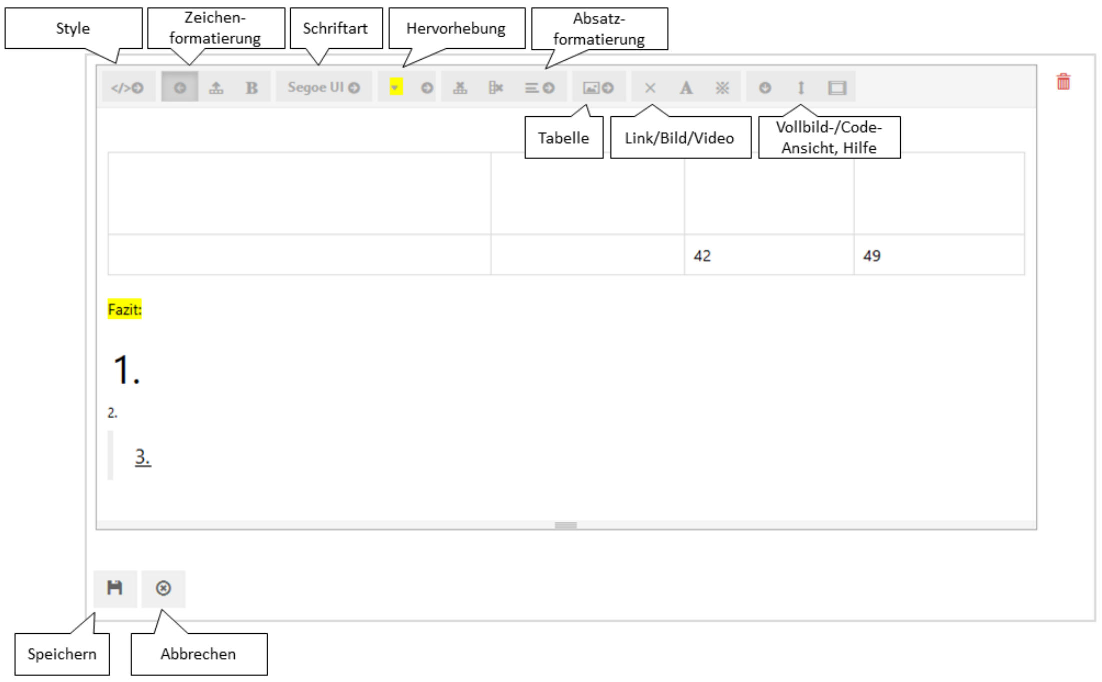


This pie chart was generated using a script that uses the chart.js plugin.

3.6.3 vcm-plugin-html-editor

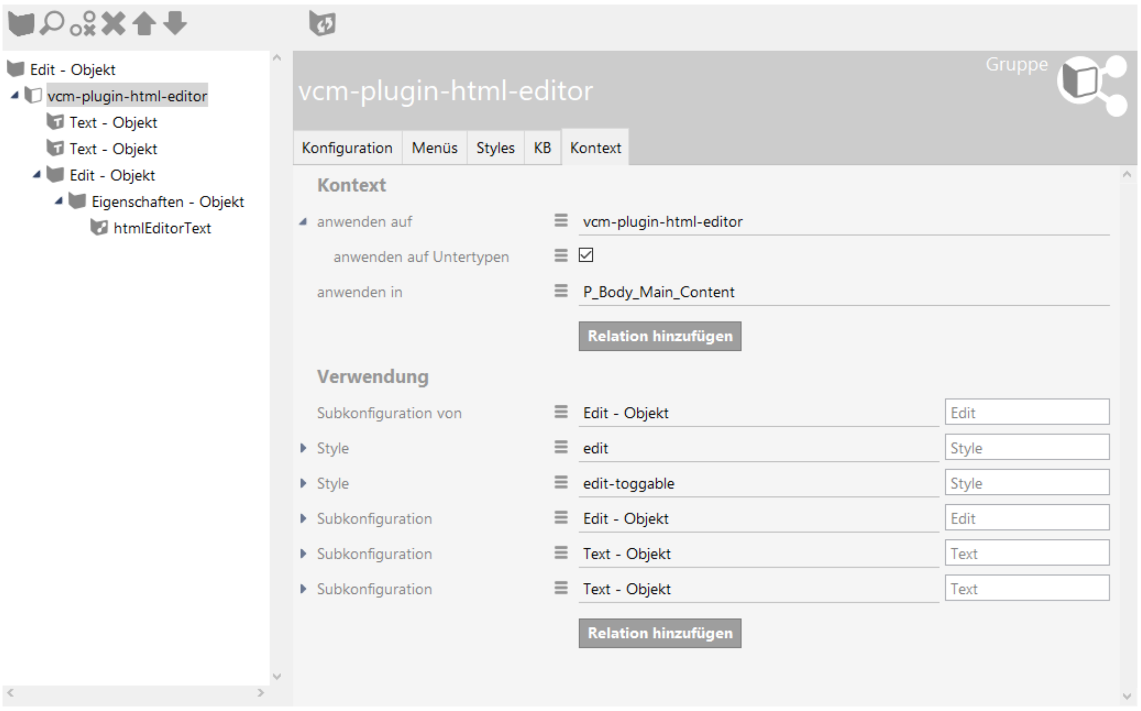
Web front-end

The vcm-plugin-html-editor makes it possible to edit HTML-formatted text. For this purpose it uses the summernote WYSIWYG editor.



Configuration

A group is required for the view configuration of the HTML editor: In the “Context” tab, the "vcm-plugin-html-editor" entry is required under “apply to”.



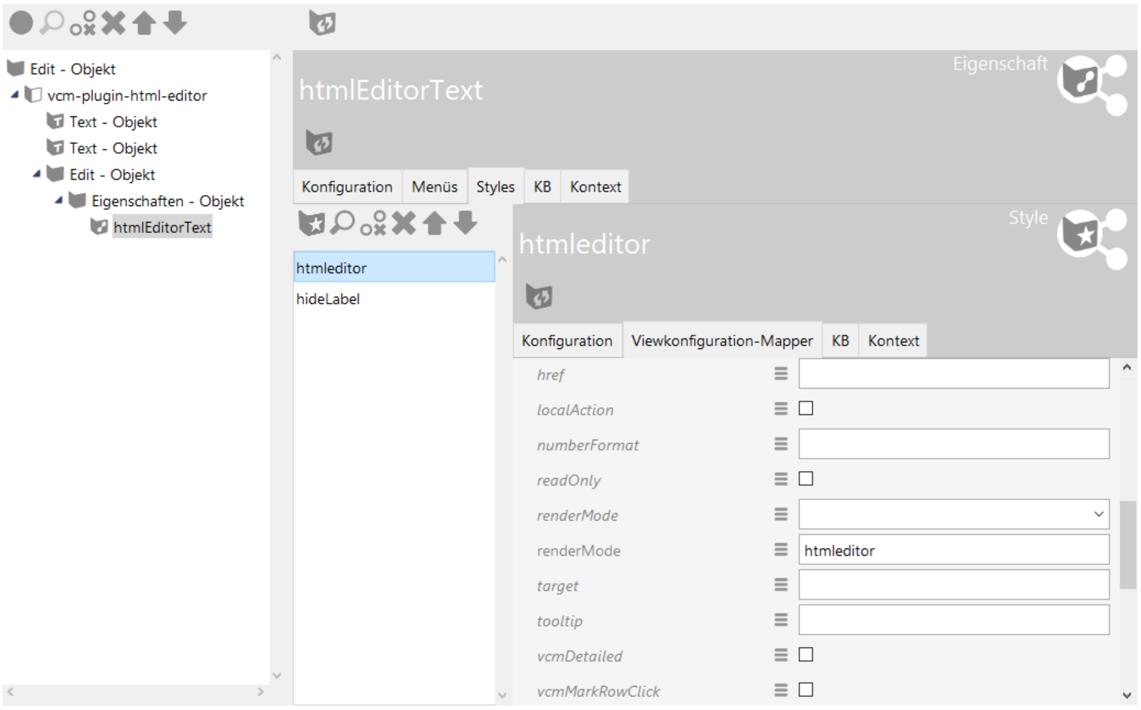
Kontext

anwenden auf	vcm-plugin-html-editor	
anwenden auf Untertypen	☑	
anwenden in	P_Body_Main_Content	

Verwendung

Subkonfiguration von	Edit - Objekt	Edit
Style	edit	Style
Style	edit-toggable	Style
Subkonfiguration	Edit - Objekt	Edit
Subkonfiguration	Text - Objekt	Text
Subkonfiguration	Text - Objekt	Text

After the configuration of the group, a property configuration must be created. It must be equipped with a style containing the “htmleditor” renderMode:



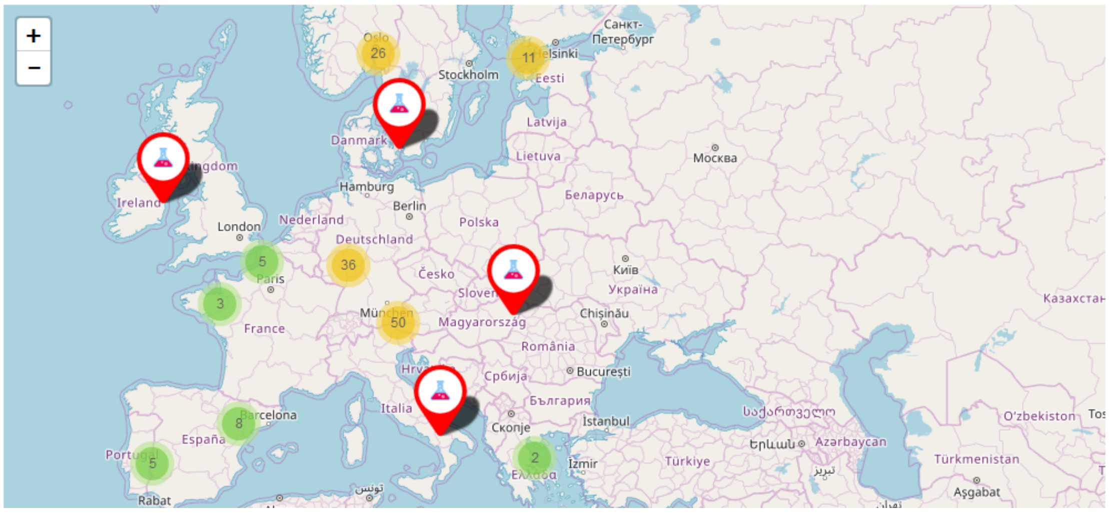
href	
localAction	☐
numberFormat	
readOnly	☐
renderMode	
renderMode	htmleditor
target	
tooltip	
vcmDetailed	☐
vcmMarkRowClick	☐

The property configuration also requires a string attribute that is supposed to contain the text.

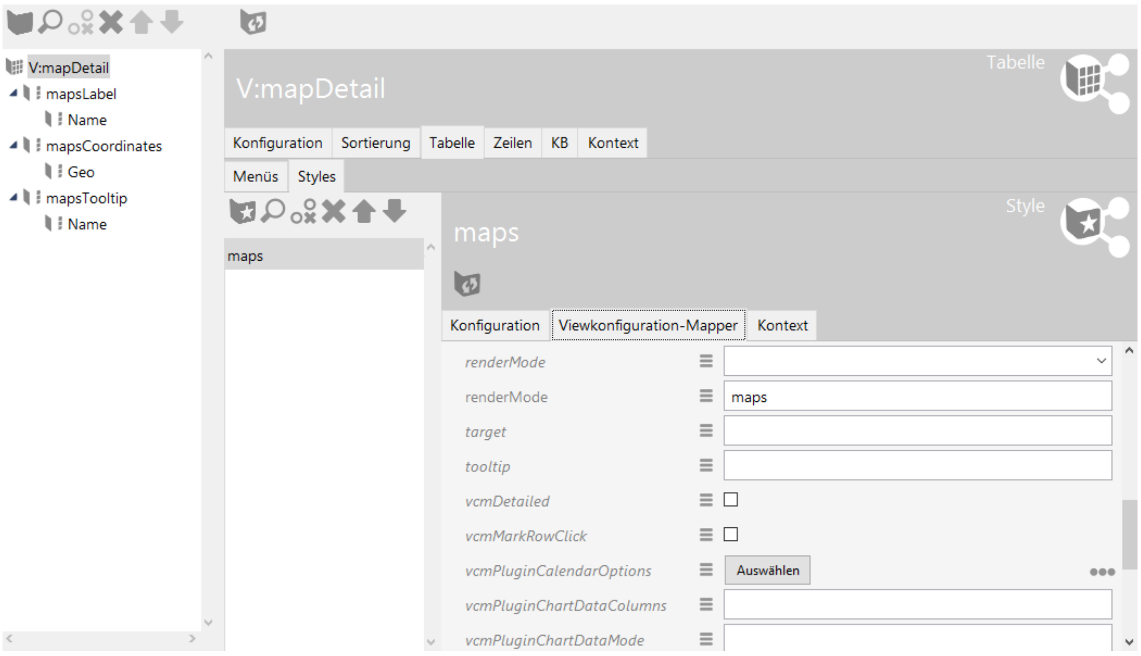
To ensure that the content can be edited from the web front-end, an additional style with the “edit” render mode must be created in the higher-level group of the property configuration. Alternatively it is possible to create a higher-level edit configuration for this purpose.

3.6.4 vcm-plugin-maps

The map plugin makes it possible to embed a map in the front-end. For this purpose the objects to be displayed must have an attribute of the “geographical position” type.



The map can be configured as a script-generated view or as an object list.
For use via object lists, a style with the “maps” renderMode is applied to the “Table” tab in a table view.



V:mapDetail
- mapsLabel
  - Name
- mapsCoordinates
  - Geo
- mapsTooltip
  - Name

V:mapDetail — Tabelle

Konfiguration | Sortierung | Tabelle | Zeilen | KB | Kontext

Menüs | Styles

maps

maps — Style

Konfiguration | Viewkonfiguration-Mapper | Kontext

renderMode	
renderMode	maps
target	
tooltip	
vcmDetailed	☐
vcmMarkRowClick	☐
vcmPluginCalendarOptions	Auswählen
vcmPluginChartDataColumns	
vcmPluginChartDataMode	

Columns are used to further configure the map. The columns with the labels “mapsLabel” (contains the name of the object) and “mapsCoordinates” (contains the attribute with the geographical coordinates) are obligatory because they are used to determine the objects for display and its coordinates. Please note that this exact label must be used.

Optional columns and functions:

- “mapsPopup” - ensures that a pop-up with the contents of this column is called up when the icon is clicked (accepts html). If a selection action is available, this column is deactivated.
- “mapsTooltip” - displays the configured property as a tooltip.
- “mapsColor” - determines the color of the marking element on the map.
- “mapsIconLocator” - by default the icon of the type is used to display the objects on the map. Here adjustments are possible by specifying a different icon location in the form of the ID of the corresponding file attribute.

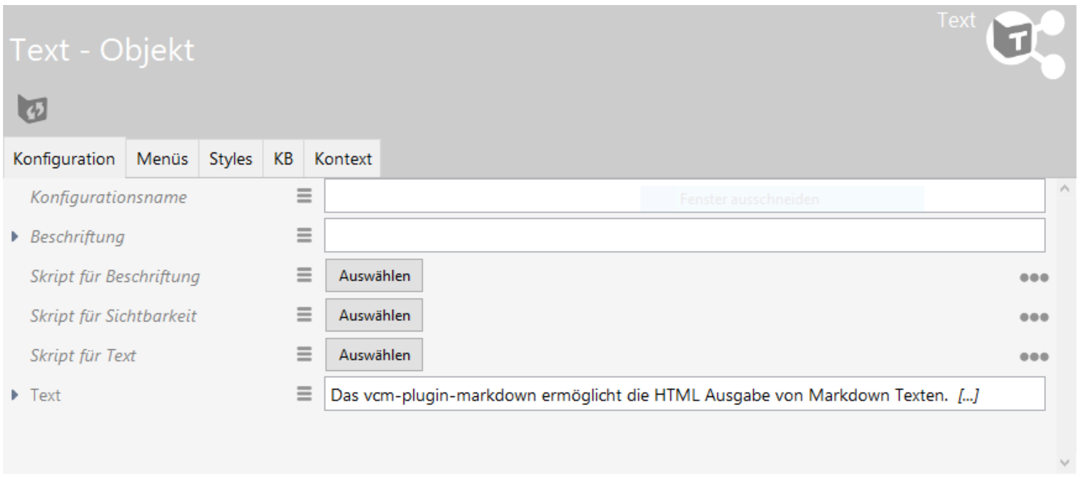
A selection action can be applied to the table; this action is activated when the marking element is clicked.

3.6.5 vcm-plugin-markdown

The HTML output enabled by the VCM Markdown plug-in makes it possible to output Markdown texts.

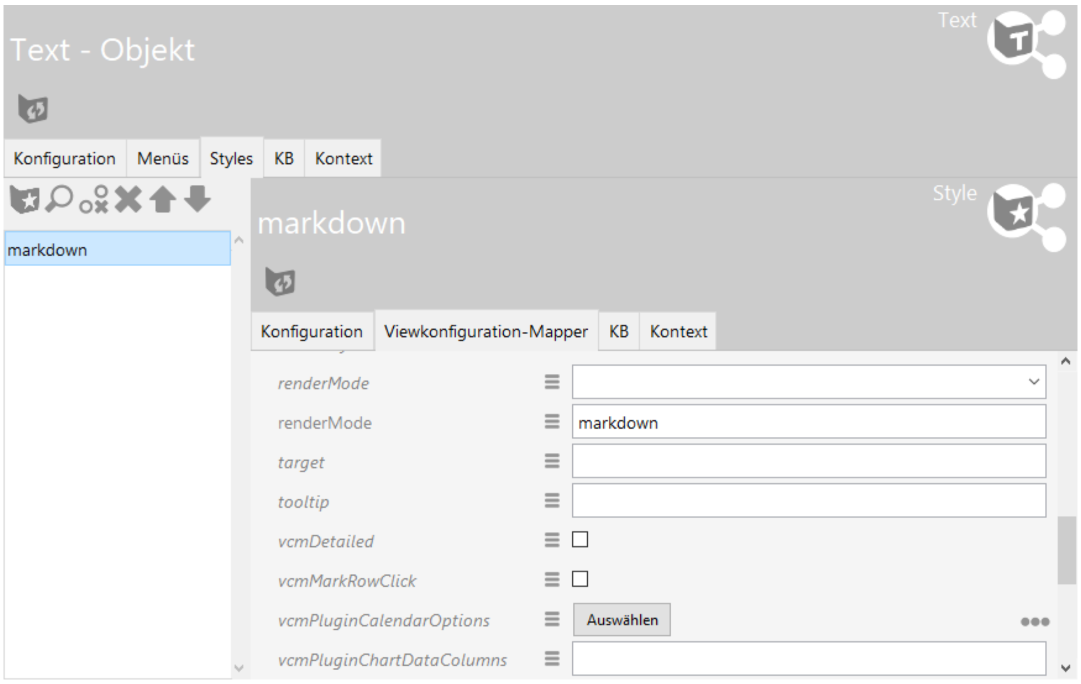
It can be used by adding a style with the render mode *markdown* to one of the following configuration elements:

- *Static text*: The Viewconfig property *text* of the configuration element is interpreted as markdown.



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:

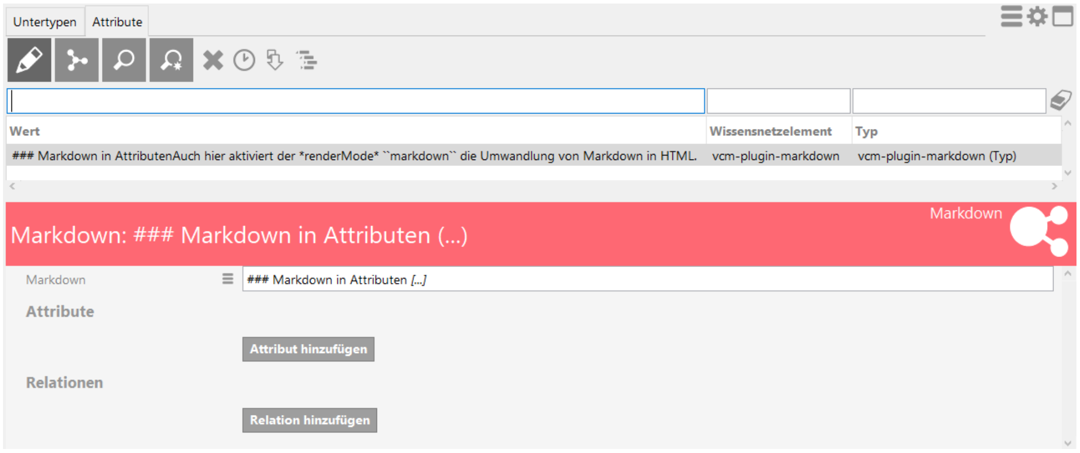


Text - Objekt

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)

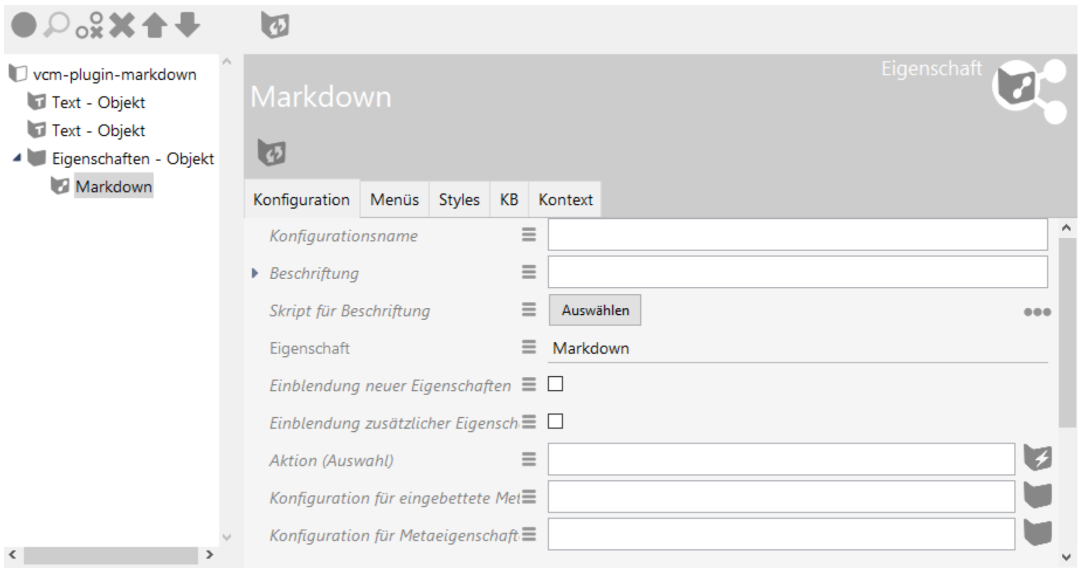
Markdown: ### Markdown in Attributen (...)

Markdown: ### Markdown in Attributen *[...]*

Attribute

Relationen

The view for the string attribute “Markdown” is configured using a property view:

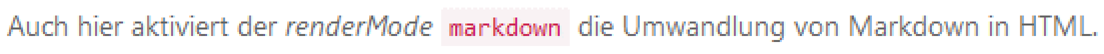


Markdown

Konfigurationsname	
Beschriftung	
Skript für Beschriftung	Auswählen
Eigenschaft	Markdown
Einblendung neuer Eigenschaften	☐
Einblendung zusätzlicher Eigensch	☐
Aktion (Auswahl)	
Konfiguration für eingebettete Met	
Konfiguration für Metaeigenschaft	

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.

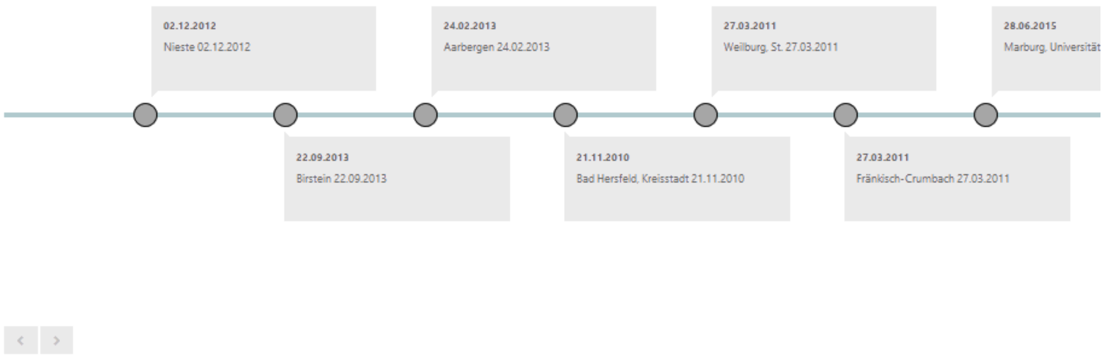
Further configuration of the plug-in is possible via the style attribute *vcmPluginMarkdownOptions*.

The plug-in uses the module markdown-it

3.6.6 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.



3.6.6.1 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.

3.6.6.2 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:

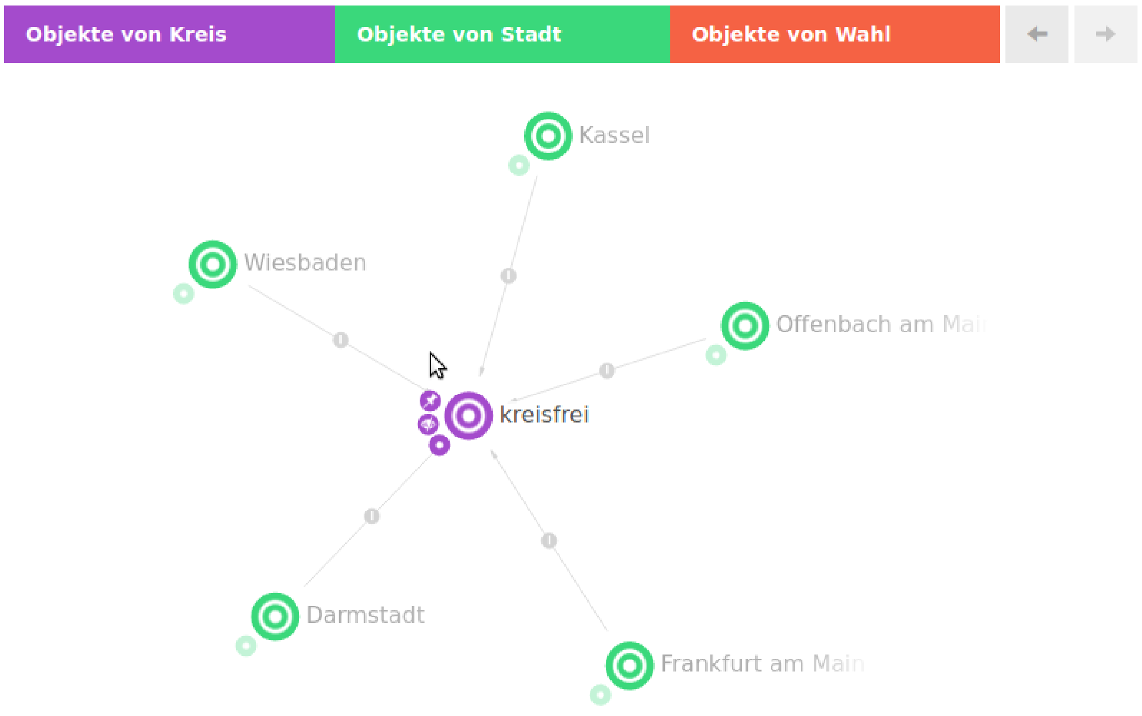
```
.timelineVertical ul li
.timelineHorizontal ul li
```

The mark points for the results can be modified using the following selector:

```
.timelineVertical ul li::after
.timelineHorizontal ul li::after
```

3.6.7 vcm-plugin-net-navigator

The vcm plugin Net-Navigator visualizes elements in a graph-like view.



3.6.7.1 Configuration

The plugin can be configured by means of styles.

Styles of the view

Style	Description
vcmPluginNetNavigatorOp-tions	A JSON object for the view options. See below for details
extra	Alternative to *vcmPluginNetNavigatorOptions*

Options

Option	Description
vcmPluginNet-NavigatorOp-tions.categories.hideLabel	Show/hide category labels

vcmPluginNet-NavigatorOp-tions.categories.embeddedActions	Configure where actions are to be displayed. For true, they are shown next to the categories
vcmPluginNet-NavigatorOp-tions.categories.compactActions	Combine actions in a menu
vcmPluginNet-NavigatorOp-tions.history.enabled	Activates/deactivates the navigation history
vcmPluginNetNaviga-torOptions.enableEditing	Activates/deactivates the option of creating new links be-tween elements in the graph
vcmPluginNetNaviga-torOptions.nnOptions	Options for the Net-Navigator component
vcmPluginNet-NavigatorOp-tions.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

ex-tra	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

ex-tra	A JSON object for the border options. See below for details

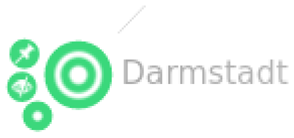
Border options

color	Overwrites the background color of the border

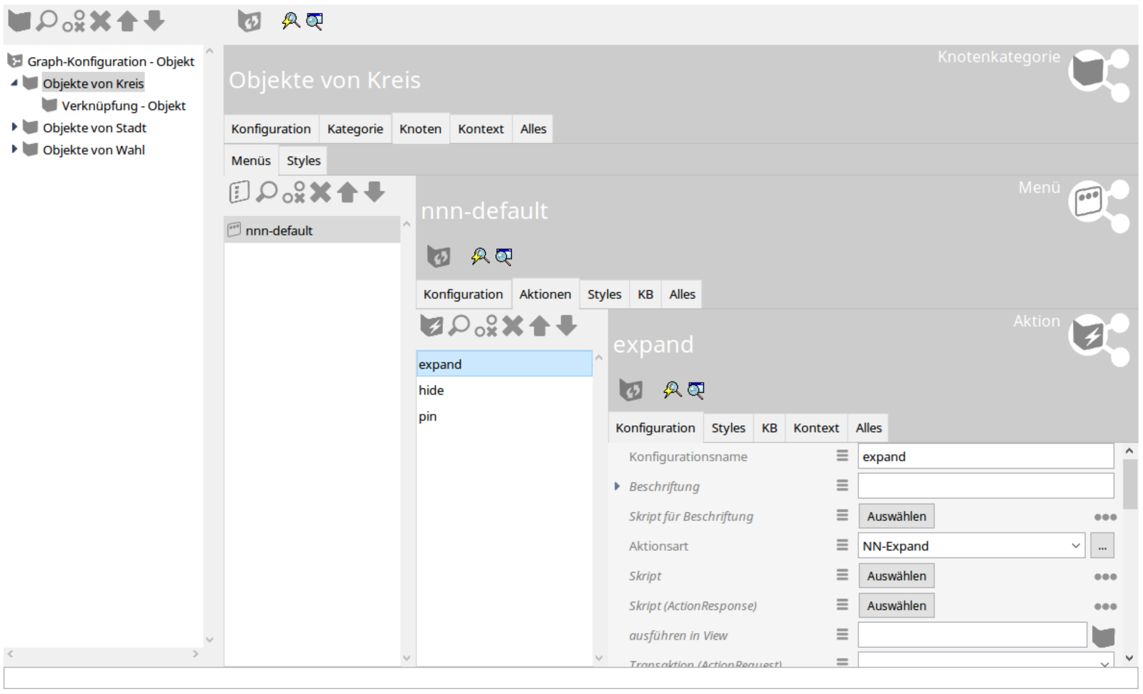
label	Overwrites the label of the border

3.6.7.2 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

3.6.7.3 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
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.ActionRespo

  actionResponse.setFollowup('graph-join')
  actionResponse.setData({
    elementId: [$k.rootType().idString()]
  })
  return actionResponse
}
```

3.7 Special configuration

This chapter covers specific application cases in the ViewConfiguration Mapper which require a combination of viewconfig element, search and/or script.

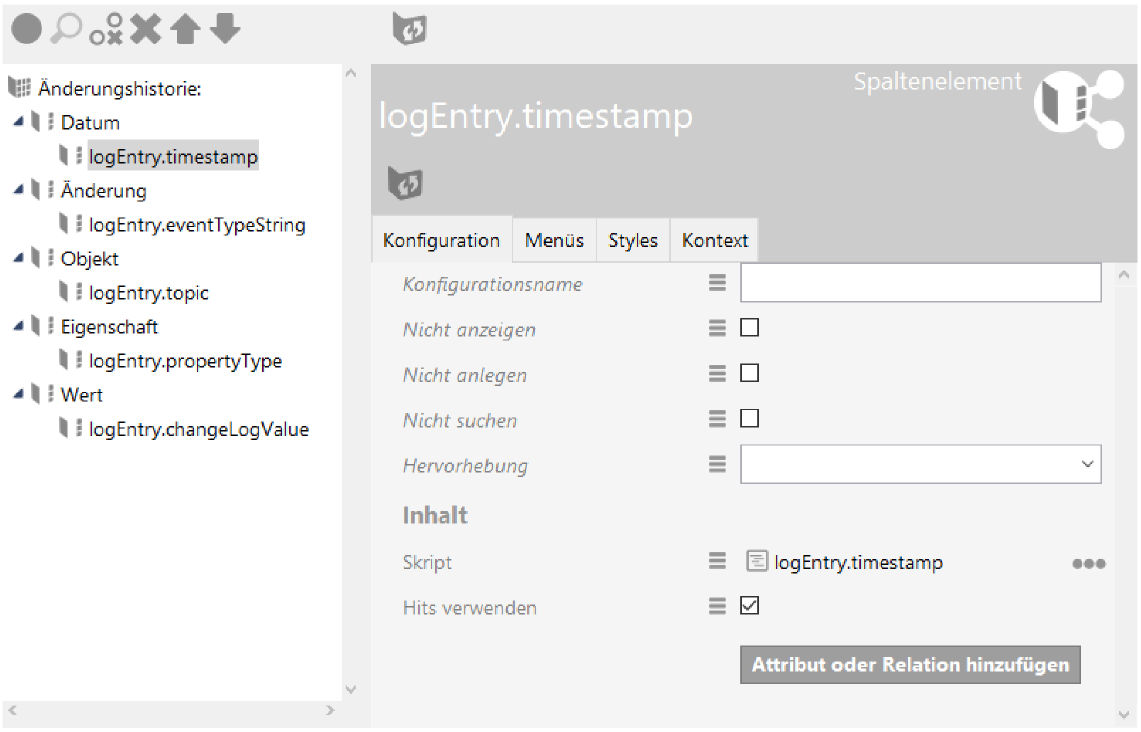
3.7.1 Display change history in a web frontend

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.

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

logEntry.timestamp (Spaltenelement)

Konfiguration | Menüs | Styles | Kontext

Konfigurationsname	
Nicht anzeigen	☐
Nicht anlegen	☐
Nicht suchen	☐
Hervorhebung	
Inhalt	
Skript	logEntry.timestamp
Hits verwenden	☑

Attribut oder Relation hinzufügen

From the change history it is possible to read out values such as date, change, affected element, modified properties etc.

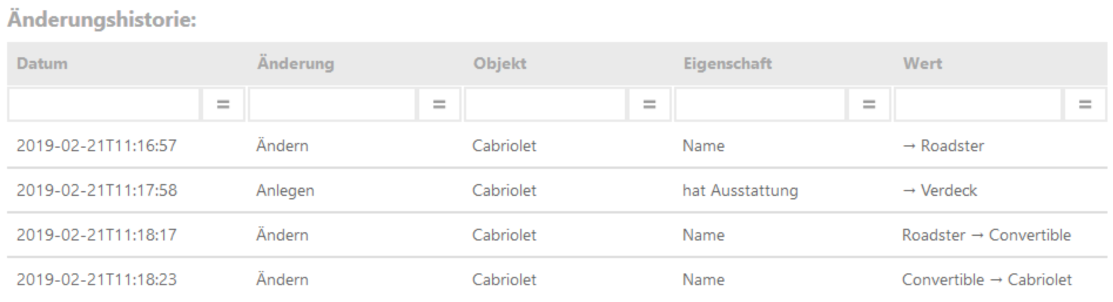
For each of these values, it is necessary to create a column configuration that contains a script as its column element. The script processes the entries as per the $k.HistoryChangeLogEntry class and returns the relevant value, filtered by value type, for the column element (for syntax, see JavaScript 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.

Please note:

1. 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).
2. 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) {
    return [ logEntry.propertyType().name() ]
}

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
}
```

3.8 Installation

- 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

3.9 Extension project

3.9.1 Development environment

- Node.js/Webpack/etc.

3.9.2 Technical details

- Diagram showing information flow in the case of actions
- Component state

4 i-views services

4.1 General

4.1.1 Command line parameter

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.

4.1.2 Configuration file

Some settings can be specified by means of a configuration file (*.ini) . The structure of the file is as follows:

```
[Default]
parameterName1=parameterValue1
parameterName2=parameterValue2
...
```

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.

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 data that are added for each log output:

- 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.

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. Please 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.

4.1.2.1 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
```

4.1.2.2 Macros

Within *.ini configuration files, further *.ini configuration 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/ivcontent-host.ini) $(include:../shared/ivcontent-volume.ini)
```

4.1.2.3 Caching

The following excerpt shows an *.ini file example for caching:

```
; pre-fill corresponding fields in the login window host=demo-server.empolis.com user=peter volume
```

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 semantic network.

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.

Technical data:

- Multi-platform executable based on the VisualWorks Smalltalk Virtual Machine (mediator.exe or mediator.im).
- Configurable TCP/IP server port for communication with the clients, standard in i-views 5.1 is 30064.

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. Other systems on request.

OS	**Version**	**Processor**	**Supported**	**64 Bit VM**
Windows	All versions, servers and clients currently supported by Windows	x86	Yes	Yes
Linux	RedHat, SLES, etc. (Kernel >= 2.4, glibc >= 2.5)	x86	Yes	Yes
	Kernel >= 2.6, glibc >= 2.5	PPC	No	
		ARM	No	
Mac	OSX 10.9+	x86	Yes	Yes

4.2.3 Operating modes

First, the following start parameters generally determine the mode in which the server is started. Without parameters, 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 (30064). The port of the stock is selected automatically. Authentication tokens between the two processes are generated automatically and do not have to be configured.

Please 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]

0.0.0.0interfaces=cnp://0.0.0.0:4998
```

This configuration ensures that the stock listens on port 4998 and communicates via the native Coast protocol.

The configuration file can contain the following entries:

```
[Default]
parameterName1=parameterValue1
parameterName2=parameterValue2
...
```

The following parameters can be used at this point:

`port=<port number>`

Starts the stock with port number <num>. Without this entry, port 30064 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.

`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 semantic networks are stored in this directory. Here, “volumes” is entered as the default value.

`backupDirectory=<Directory>`

Specifies the directory to which the semantic network 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.”

Note: 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=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

Changed

Logging settings:

For the configuration options for logging, see the logging settings in Chapter 11.1.2 Configuration file.

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.

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).

4.2.3.3 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 semantic networks 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.

Please 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 30061 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 semantic networks are in this directory. volumes is entered as the default value at this position.

`backupDirectory=<Directory>`

Specifies the directory to which the semantic network 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=190133293071522928001864719805591376361
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] rsa.n_1=93D516DF61395258AA21A91B33E8EE67 rsa.n_2=B07C6
```

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 semantic networks

Directory structure

The basic directory of the i-views server has the following structure:

```
volumes/
   knowledgenetworkName/
     knowledgenetworkName.cbf
     knowledgenetworkName.cdr
     knowledgenetworkName.cfl
     knowledgenetworkName.lock (if the semantic network is open)

backup/
  knowledgenetworkName/
    <ten-digit number>/
      knowledgenetworkName.cbf
      knowledgenetworkName.cdr
      knowledgenetworkName.cfl
```

Storage of semantic networks

Semantic networks 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 semantic network. A file with the .lock file extension indicates that a semantic network is currently in use.

Backup of semantic networks

The semantic network 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 semantic network 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 semantic network, this file can contain a category [name_of_network] with the following entries:

Example jobs.ini

```
[volume1]
;Backup of network ‘‘volume1’’

;Time the backup starts
backupTime=00:45

;Interval in days - daily in this case
backupInterval=1

;Keep the last 5 backups of this semantic network
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 network 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 semantic network 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 network, the Garbage Collection might require a lot of time and memory. When running the Garbage Collection in large networks, 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 network 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 network 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 using special call parameters of the i-views server:

-startGC <volume> -host <hostname>	Starts the Garbage Collection on the network named <volume> and on a potentially remote mediator on the computer <hostname> (optional incl. port information).
-stopGC <volume> -host <hostname>	Ends any Garbage Collection of the network named <volume> that might be running on the mediator <hostname>.
-infoGC <volume> -host <hostname>	Informs about the current status of the Garbage Collection.

These commands are transferred to another already running mediator by means of a mediator executable.

Another option is to start the Garbage Collection via the Admin tool.

To execute these demands, the parameter -password must be used to transfer the correct server password.

4.2.5.3.1 Status check of carbage collection via REST interface

For maintenance purposes, starting and status requesting both can be done via REST interface.

mediator.ini

The initialization file mediator.ini must include two interfaces for the [Default] block:

- cnp-Interface for communication with the job clients
- http-Interface for REST communication

The needed line will be as follows:

```
interfaces=cnp://0.0.0.0:67890,http://0.0.0.0:45678
```

The ports are freely selectable, but they must be adjusted for every other access.

Note: After changing the ini files, the mediator needs to be restarted.

Credentials

In order to prevent user and password from appearing within the process lists, a credentials file needs to be used. For this, we create a file called ".iviews_auth" containing the following lines:

```
machine localhost login admin password "<PASSWORD>"
```

The access rights need to be set to "0600" (read and write access) for the user who is intended for access via REST.

REST call per curl

Requesting status of garbage collection:

```
curl -netrc-file .iviews_auth http://localhost:45678/volumes/my-volume-name/garbageCollection
```

Status with json-prettyprint:

```
curl -s -netrc-file .iviews_auth http://localhost:45678/volumes/my-volume-name/garbageCollection |
```

Starting garbage collection:

```
curl -netrc-file .iviews_auth -g -d "action=start" http://localhost:45678/volumes/my-volume-name/g
```

Stopping garbage collection:

```
curl -netrc-file .iviews_auth -g -d "action=stop" http://localhost:45678/volumes/my-volume-name/ga
```

4.2.5.4 Operation in Unix

Under UNIX the server reacts to the following signals:

`SIGTERM/SIGHUP`

Shuts down the server

`SIGUSR2`

The server immediately begins to back up all semantic networks 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 semantic network constantly. If the part of the cluster on which the mediator is running fails, a new mediator that then manages access to the semantic network is started automatically

If the first mediator fails, it is possible that the mediator no longer has time to make the semantic network consistent and that the network 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 semantic network 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 semantic networks. When a locked semantic network is opened, this lock is detected and removed, if possible.

If the mediator detects an inconsistency, then the semantic network 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 bridge enables access to semantic networks on three types/operating modes:

- Via a RESTful services architecture (REST-Bridge). The interface is available as an HTTP or HTTPS version (KHTTPRestBridge)
- Via KEM-RPC (KEMBridge): Access via KEM If binary data is supposed to be stored in the semantic network, a REST bridge is required, which provides a REST service with a blob

resource handler.

- Operating mode “Load distributor for other bridges” (KLoadBalancer).

PLEASE NOTE: KLoadBalancer and KEMBridge/KHTTPRestBridge may not be activated in one bridge at the same time because they interfere with each other.

The bridge and all of the accesses to be activated in it can be configured via an ini file. Settings for accesses are bundled in sections. The most important of these parameters can also be specified via a command line. If that is the case, the values of the command line call take precedence over those in the ini file. The individual parameters are explained next.

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.

Examples of calling the bridge:

```
bridge -host server01:30000 -port KEMBridge 4713 -port KEMStreamingBridge 4714

bridge -ini bridge2.ini -port KMultiBridge 3030
```

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>.

`-stop <hostname>`

If you call the bridge with the parameter above, the current bridge is prompted to terminate on the specified host. All clients started in it are shut down and the bridge is terminated.

4.3.3 Configurationfile "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. At the moment, potential clients include:

- KEMBridge
- KHTTPRestBridge

In addition, the KLoadBalancer can be started as a client of the bridge, in which case the ini file only includes the section

- KLoadBalancer

host = <hostname:portnumber>

see command line parameter -host

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.

Logging settings:

For the configuration options for logging, see the logging settings in Chapter 11.1.2 Configuration file.

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 as an HTTP or HTTPS version.

The REST bridge runs inside the standard bridge of i-views (bridge.exe).

The interface is fully configured by configuration individuals in the semantic network. 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).

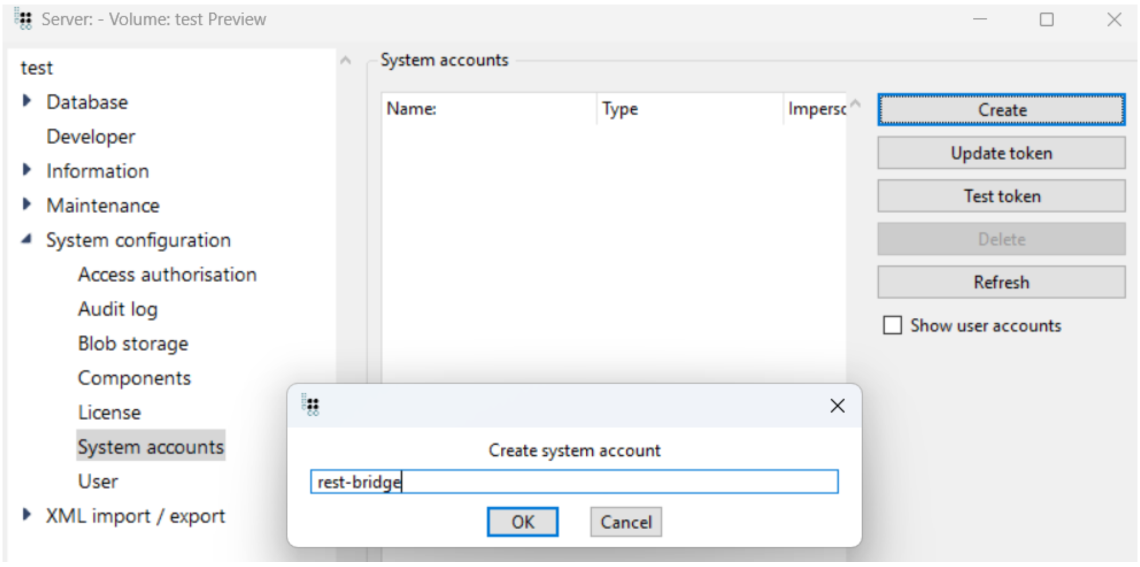
4.3.4.2 Installation

4.3.4.2.1 Prepare volume

I. Creating a system account for the bridge service

To allow a bridge service to access a knowledge network that is managed by a mediator service, a system account must be created for the bridge service in the knowledge network. 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:

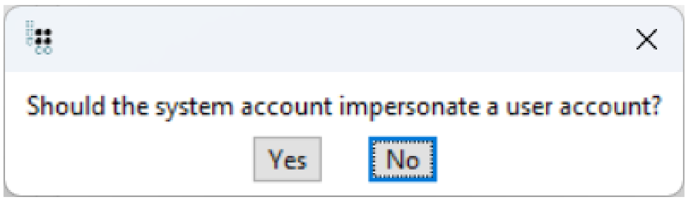


Create system account

rest-bridge

Admin tool - Create system account and enter account name

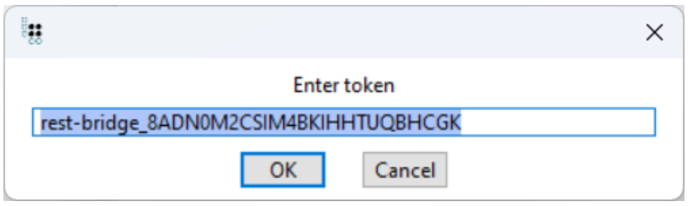
Step 2:



Should the system account impersonate a user account?

Click 'No'

Step 3:



Enter token

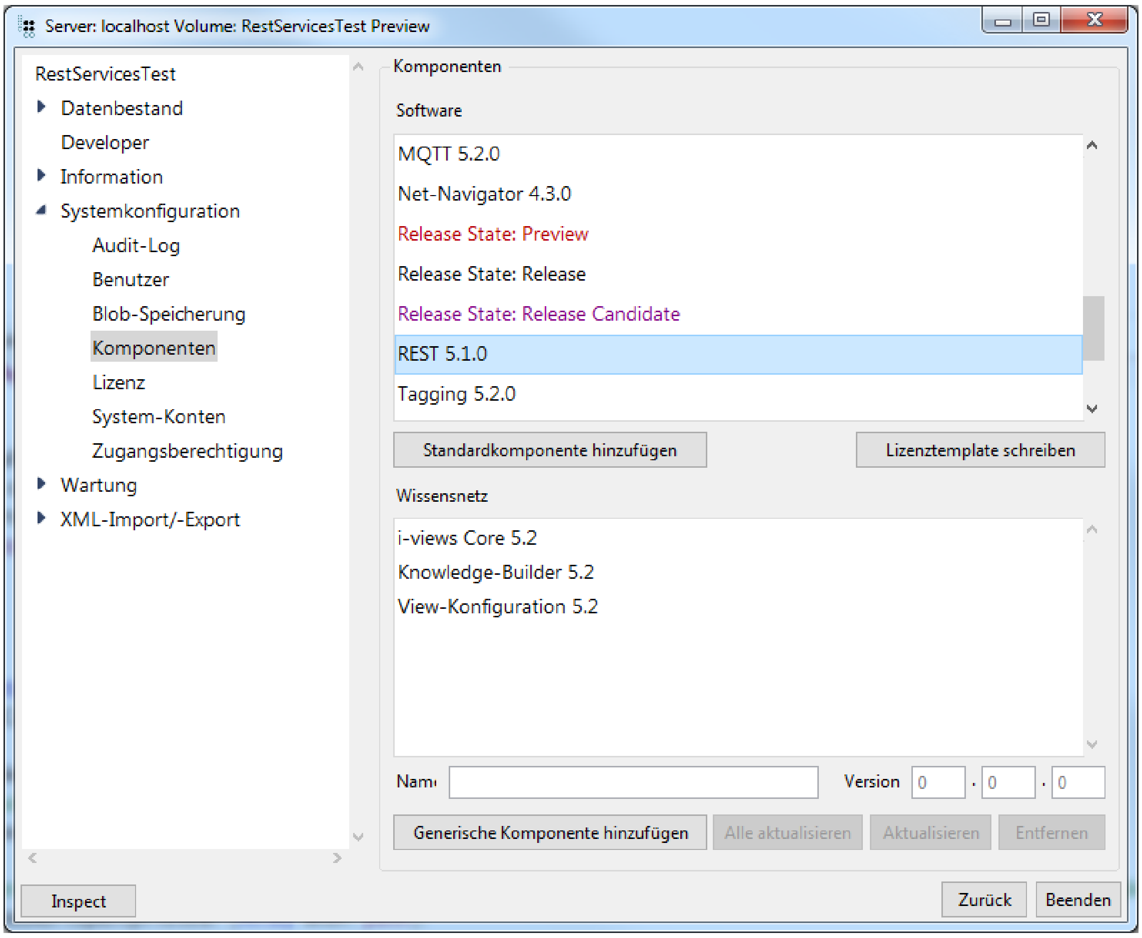
rest-bridge_8ADN0M2CSIM4BKIHHTUQBHCGK

Automatically generated login token is displayed

Important: 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.

II. Activating the REST component in the knowledge network

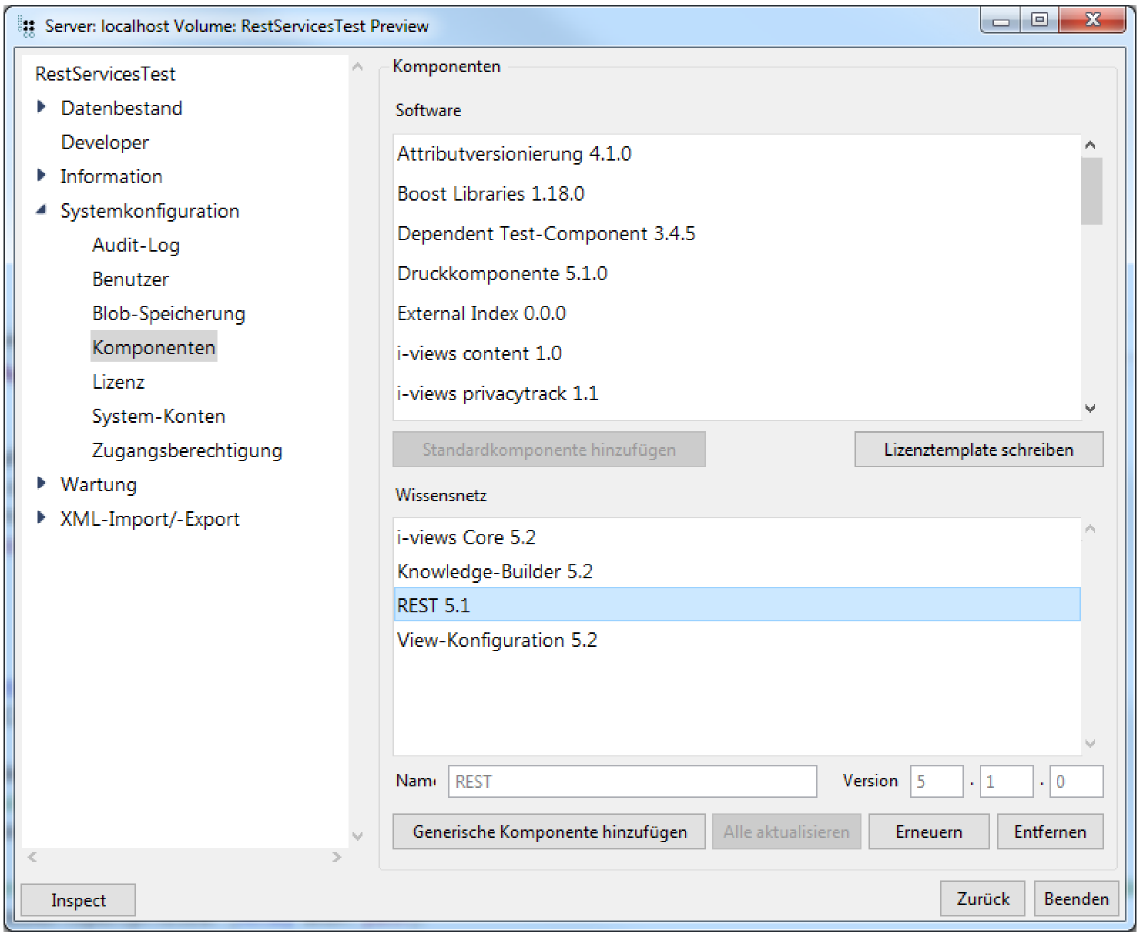
By adding the software component "REST" in the Admin tool, the required schema is created in the semantic network.



Software:
MQTT 5.2.0
Net-Navigator 4.3.0
Release State: Preview
Release State: Release
Release State: Release Candidate
REST 5.1.0
Tagging 5.2.0

Wissensnetz:
i-views Core 5.2
Knowledge-Builder 5.2
View-Konfiguration 5.2

Nam:
Version 0 . 0 . 0


Server: localhost Volume: RestServicesTest Preview

- RestServicesTest
  - Datenbestand
  - Developer
  - Information
  - Systemkonfiguration
    - Audit-Log
    - Benutzer
    - Blob-Speicherung
    - Komponenten
    - Lizenz
    - System-Konten
    - Zugangsberechtigung
  - Wartung
  - XML-Import/-Export

Komponenten

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

Standardkomponente hinzufügen | Lizenztemplate schreiben

Wissensnetz

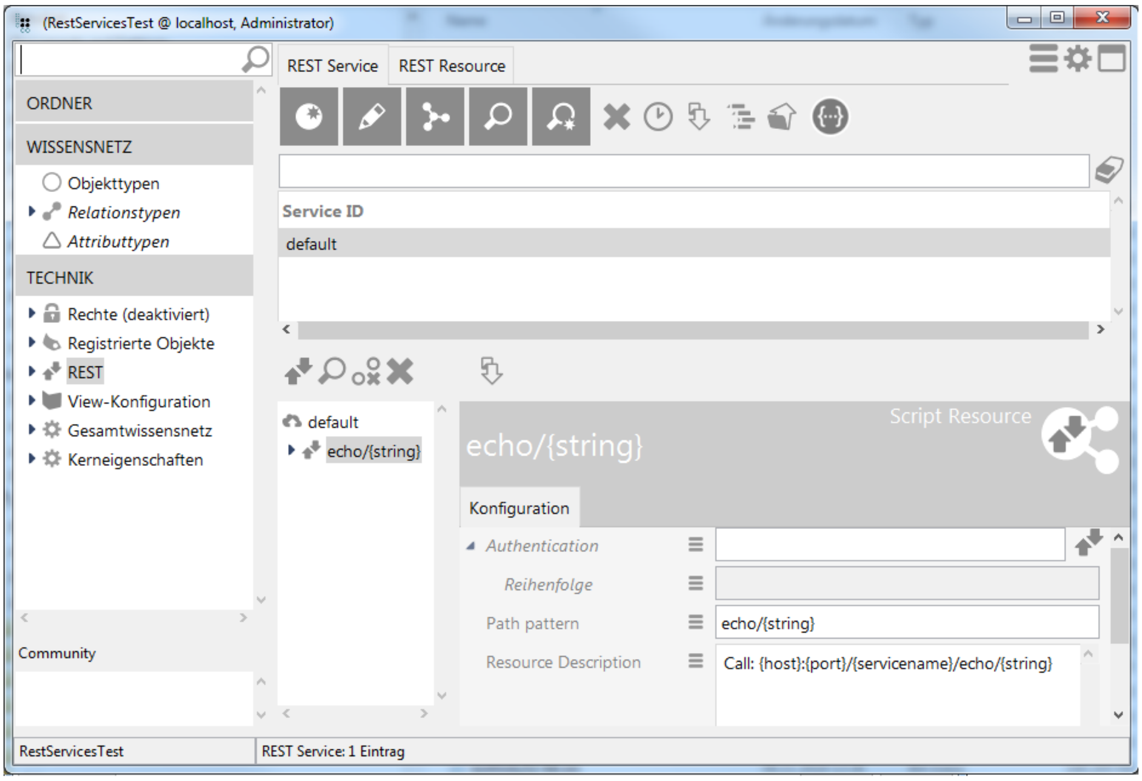
- i-views Core 5.2
- Knowledge-Builder 5.2
- REST 5.1
- View-Konfiguration 5.2

Name: REST Version: 5 . 1 . 0

Generische Komponente hinzufügen | Alle aktualisieren | Erneuern | Entfernen

Inspect | Zurück | Beenden

The schema is created as a subnet of the semantic network 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 **bridge.ini** contains an entry for the category **KHTTPRest-Bridge** or **KHTTPSRestBridge**:

```
[KHTTPRestBridge]
volume=name of the semantic network
port=port at which the service is to be reachable, the default is 8815
authentication=token value of the system account created for this bridge (see above)
services=list of REST service IDs to expose
```

For the HTTPS version, the file paths for the certificate and private key must also be specified in the configuration file.

```
[KHTTPSRestBridge]
volume=name of the semantic network
port=port at which the service is to be reachable, the default is 8815
authentication=token value of the system account created for this bridge (see above)
services=list of service IDs to expose
certificate=name of the .crt file
privateKey=name of the .key file
```

In the configuration section “KHTTPRestBridge” or “KHTTPSRestBridge” 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 KEM bridge

KEMBridge

Section name:

[KEMBridge]

port = <portnumber>

Specifies the port under which the KEMBridge reacts. If no entry is made, the default value of 4713 applies.

ldapHost = <hostname:portnumber>

Specifies the LDAP host to be contacted if authentication is to be performed via LDAP. If this parameter is specified, authentication must be handled via LDAP.

maxLoginCount = <number>

Maximum number of failed attempts to log in before the relevant user is locked out of the network. After that, login is only possible after they have been unlocked via the Knowledge Builder. If the value is not set, a user can make as many failed attempts to log in as they wish.

In order to allow a user to be locked out of the semantic network, a Boolean attribute with the internal name userlock and the default value false must have been defined for individuals of the person concept.

KEMrestrictToIPAddress = <IP address>

If this parameter is set, connections are only accepted from the host specified here.

trustedLoginEnabled = <true/false>

Makes it possible to log in without a password by means of the request “newAuthenticatedUser(username).”

preventSessionReplay=<true/false>

[default=false]

This parameter specifies that each writing session receives its own protected semantic network access, so that there is no longer any need for the usual mechanism of executing the actions of a deactivated session again during reactivation in order to restore the most recent editor state.

KEMStreamingBridge

Section name:

[KEMStreamingBridge]

port = <portnumber>

Specifies the port under which the KEMStreamingBridge reacts. If no entry is made, the default value of 4714 applies.

4.3.6 KLoadBalancer

The KLoadBalancer can be used to scale the services and availability of the KEMBridge and KEMStreamingBridge.

The following specification must be entered in the [KLoadBalancer] section in order to obtain the required operating mode:

- allowRemoteShutdown (default value false)
- autoRestart (default value true)
- directory (default value current working directory in which the KLoadBalancer was started)
- executable (default value 'bridge.exe')
- image (default value 'bridge.im')
- vm (default value 'visual')
- hostname (default value Localhost)
- **configNames (required value, not optional)**
- parameters (default value blank)

The parameter #configNames is used for continuing the configuration of the KEMBridges and KEMStreamingBridges to be started, with one bridge type controlled by each individual configuration. The configuration names must be separated by a comma.

Here is an example of a KLoadBalancer ini file:

```
[Default] [KLoadBalancer] hostname=ws01 port=30003 directory=C:\3.2\balancing executable=bridge.ex
```

Upon starting, KEMBridges and KEMStreamingBridges are started in accordance with both the configurations. Because the same software is used for operation as is used for operation of the KLoadBalancer, specifying the parameters #executable, #image and #vm (for operation in Linux), #hostname, #directory and #parameters are required.

executable / image, vm; directory: Specifications for how the individual bridges can be started. Specifying #executable and #directory is required under Windows, while specifying #image, #vm and #directory is required under Linux.

hostname / port: The host name which is used to refer to the bridges to be started, and the KLoadBalancer to be contacted for administration purposes. If nothing is specified here, then the name of the computer is determined and used. The port indicates the port used by the bridges to address the balancer, the default value is 4715.

Careful: The name of the respective mediator that the bridges contact to retrieve data must be entered in the respective ini files in accordance with the configuration section.

parameters: A field that is used to add additional specifications in the command line of the bridges to be started, and is the same for all bridges to be started.

allowRemoteShutdown: A parameter that specifies whether the KLoadBalancer can be ended by means of a shutdown request using remote access.

autoRestart: Parameter that specifies whether a stopped KEMBridge should be restarted after the shutdown, with a new ID.

Additional specifications must be entered in each configuration section:

- bridgeClientClassName (not optional, only one specification possible per section. Please observe the syntax described above!)
- inifile (ini file with settings for this type of bridge to be started)
- bridgeLogfile (sample of a log file name in which a placeholder is added, <id>, which is used to distinguish the log files for the individual bridges, and is replaced with the consecutive number of the bridge that was started)
- maxBridges (maximum number of bridges of the specified type to be started, not optional)
- sslEnabled (specifies whether the bridges of this type should use SSL to establish a connection, default value false)

Please note: The parameter #directory specifies the working directory in which the files specified in the configuration sections are searched for and, when applicable, created. Software and ini file for starting the KLoadBalancer may be located elsewhere.

The ini files for the respective bridges must have the usual structure. An example of the KEM-referenced ini file in the configuration section above is provided here:

```
[Default] host=mediator-hostname:30053 [KEMBridge] trustedLoginEnabled=true preventSessionReplay=t
```

For details, please refer to chapter 5, "Configuration file bridge.ini".

4.4 Jobclient

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.

One of its most important tasks is to execute all types of queries and deliver the search results to the clients (sorting, text formatting, rights filtering).

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.

Operation:
In the shared object space provided by the i-views mediator, the tasks of the clients for the services are stored in pools. All i-views Job-Clients are notified of new jobs and apply to process the new job, provided they are currently free. 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. Hence, the client logically commissions a Job-Client but the physical communication always goes through the i-views server. To the client it is transparent which Job-Client is executing its job just as the source of the job and how many parallel Job-Clients are currently active is transparent to the Job-Client. 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.

Technical data:
Multi-platform executable based on the VisualWorks Smalltalk Virtual Machine (jobclient.exe or jobclient.im)
Requires a TCP/IP connection to the i-views server
Automatic load distribution between services
Job-Clients can be connected or shut down at any time
Standby mode in case required resources are temporarily unavailable

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>`
volume	The name of the semantic network 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. Alternatively, you can also specify the category (e.g. “index”). In that case, all job pools of this category are selected.
	The possible types are presented in the subchapters.
		`jobPools=<job name1>` `[,<job name2>, ...]`
		Example:
		`jobPools=KScriptJob, query`

cacheDir	The description of the location at which the cache for the Job-Client is stored.	`cacheDir=<directory>`

volumeAccessor	Description of the storage type of the cache. Unless specified otherwise, CatBSBlockFileVolumeAccessor is used. This storage type is recommended especially for large networks as CatCSVolumeFileStorageAccessor would create a large number of files.	Example:
		`volumeAccessor=CatBSBlockFileVolumeAccessor`
		or
		`volumeAccessor=CatCSVolumeFileStorageAccessor`

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 networks.	`enableLowSpaceHandler=true/false`

useProxyValue-Holder	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`
loadIndexes	The loadIndexes=true option has been available since version 4.2. In that case, 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>, ...]`

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]`
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]`

Job configuration:

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.

Example:

```
[Job-Name1]
<Parameter>=<value>
...

[Job-Name2]
...
```

Logging settings:

For the configuration options for logging, see the logging settings in Chapter 11.1.2 Configuration file.

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-co
```

The directory *lucene-6.4.1* contains the Lucene binary files. The index is stored in the directory *lucene-index*.

4.4.2.1.2 Job specific parameters

In general:

Parameter:	**Description:**	**Syntax:**
jobPool	JobPool for executing the job.	`jobPool=<Job-Pool-Name>`
...?		

scheduledJobs:

Parameter:	Description:	Syntax:

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>`
unique	(?)	`unique=true/false`
user	(? only) Internal name of a user instance 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 ...]>`

4.4.2.2 JobPool types

The following types of job pools are available:

4.4.2.2.1 Index jobs

- Category (categories): **index**

If you specify **index** or the job classes displayed below, the indexing jobs are executed by the Job-Client. The indexing jobs should be performed by a single Job-Client. Instead of listing all the job classes individually in the job pool, you can also use the symbolic name **index**.

KAddAllToIndexJob

- Name: Add attributes to the index

KLightweightIndexJob

- Name: Update external index

  An external index is maintained via the KLightweightIndexJob.

KLuceneAdminJob

- Name: Lucene admin job

  The KLuceneIndexJob maintains an externally set-up Lucene index.

KRemoveIndexJob

- Name: Remove attributes from the index

KSyncIndexJob

- Name: Synchronize index

KAddAllToIndexJob, KRemoveIndexJob and KSyncIndexJob are required to maintain internal indexes.

4.4.2.2.2 KBrainbotJob

- Category/categories: <none>
- Name: KBrainbotJob

KBrainbotJob executes actions to maintain the Brainbot index.

Within the configuration in the Admin tool, if it is specified that maintenance actions are to be executed by a Job-Client (“Use Job-Client”), a Job-Client must be started to maintain the external index.

The KBrainbotJob has no additional configuration parameters the ini file because all the configuration takes place in the Admin tool.

4.4.2.2.3 KExternalCommandJob

- Category/categories: <none>

- Name: External call

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 inserted by a script call using KScript.

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" file`

```
        <Path path=“$image2$"></ResultAttribute>
```

Example 01:

Script:

```
<Script>     <ExternalCommandJob execution="local">         <Command>convert.bat</Command> <O
```

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> <
```

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
```

4.4.2.2.4 KExtractBlobTextJob

- Category/categories: <none>
- Name: Convert blob to a text attribute. Using the batch file specified on the “Index configuration -> External full text filter” tab in the Admin tool, the text content is extracted from the blob attribute and stored in a new attribute of the specified text attribute type. The other parameters available for the job are the topic in which the extract is to be created, and, if the specified text attribute is multilingual, the language of the attribute to be displayed. This job is inserted by a trigger, which should be set up to react to the creation and modification of blob attributes. The KScript rule to be specified in this process is “ExtractBlobText,” which permits the parameters described above to be specified.

4.4.2.2.5 KQueryJob

- Category(s): query
- Name: Search

Used to outsource the running of simple and expert queries to a Job-Client. Is equipped and executed to suit the needs of the examined search.

4.4.2.2.6 KScriptJob

- Category (categories): script
- Name: KScriptJob

You can use the KScriptJob to call KScripts from KScript so that they are executed on the Job-Client. Here, the job is generated by the KScript rule “ScriptJob” which is equipped with the script and the start objects calculated at this time as the starting point and enters the resulting KScriptJob into the job queue. In this way, work can be distributed asynchronously to Job-Clients. This is used, for example, to externalize activities that would block the calling client for too long in a sequential execution.

To do this, the parameter "scriptName" must refer to the registration key of a script stored in the network. The script is automatically encapsulated in a transaction.

4.4.2.3 Example for an ini file

```
volume=MyNetwork
host=localhost
```

```
jobPools=query, index
cacheDir=jobcache
logfile=jobclient01.log
maxMemory=400
name=jobclient01
```

4.4.2.4 Performance optimizations

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.
- **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.

Please note: For the entry *useProxyValueHolder* the value must be set to *false*. Otherwise, the Job-Client will attempt to send RPCs (requests to which the mediator can respond) to the mediator. The client however, should load the clusters itself and possibly retain them in its memory.

Note: To improve performance, it also helps to activate the hard drive 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 Blob service

4.5.1 Introduction

The blob service is used to store the data of large files outside the semantic network 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 semantic network only receives the semantic information that is based on files and remains easy to backup and transfer.

- Storage locations of the semantic network and file contents can be configured differently.
- Several blob services can be connected to one semantic network, so that one storage location can be provided for each attribute definition.

The following chapter explains how to set up and operate blob services.

4.5.2 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:

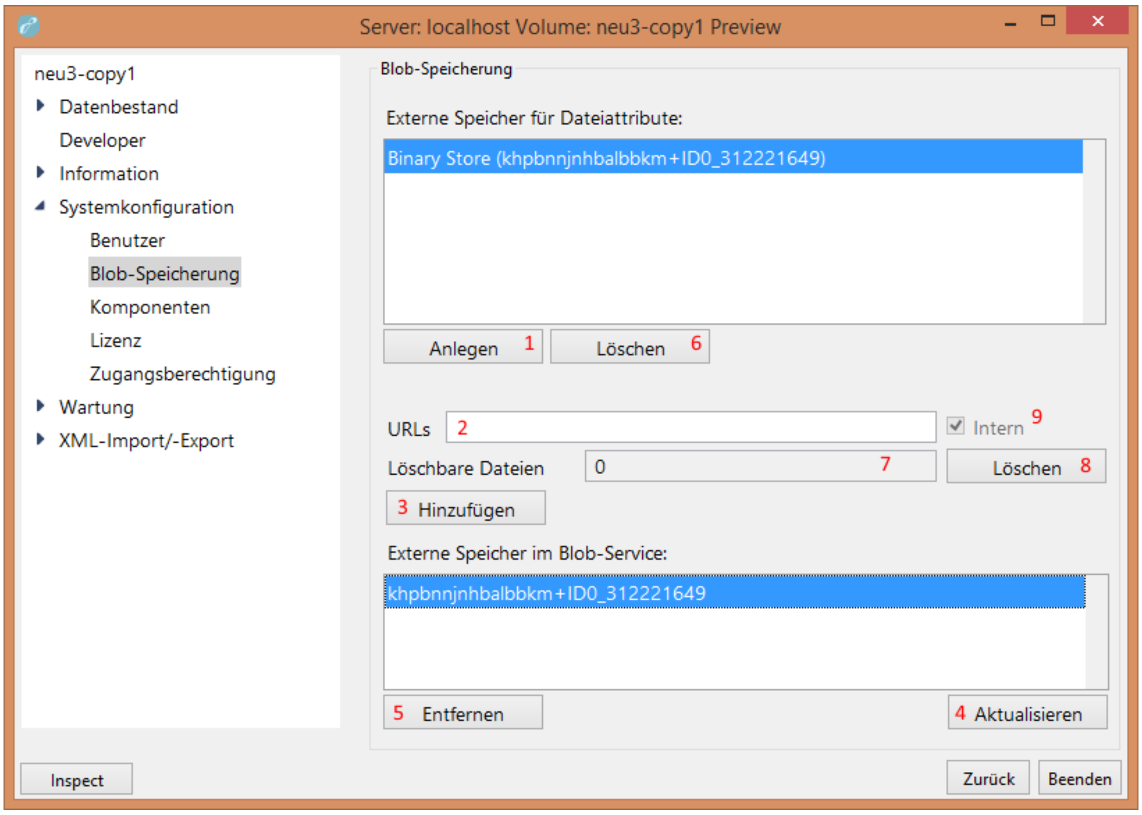
```
interfaces=https://0.0.0.0:30000
```

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 semantic graph database, the Admin tool offers the required tools under “System configuration - Blob storage:”



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 semantic network 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.

PLEASE NOTE!

By removing a blob store s link to a blob service, all files stored therein are lost!

4.5.3 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. Using OpenSSL, this is possible e.g. by means of "*openssl rsa -in input.key -out private/server.key -outform PEM*".

4.6 Install as an OS service

It is possible to set up the service programs as OS services in the various supported operating systems.

For Unix-type operating systems, it is necessary to use the mechanisms supported by the relevant platform; you will find several examples in the version-independent manual for i-views.

For MS-Windows, the services offer the parameters -installAsService NAME and -uninstallService NAME , which can be used to set up or remove a Windows-managed service from an administrative shell. During the installation, all the parameters specified after the service name are transferred to the installed service as command line parameters. Example:

```
bridge -installAsService iviews-bridge-rest -inifile bridge-rest.ini
```

sets up a service with the name “iviews-bridge-rest”, which is given

```
PFAD\bridge.exe bridge.exe -serviceName iviews-bridge-rest -ini bridge-rest.ini
```

as its call line.