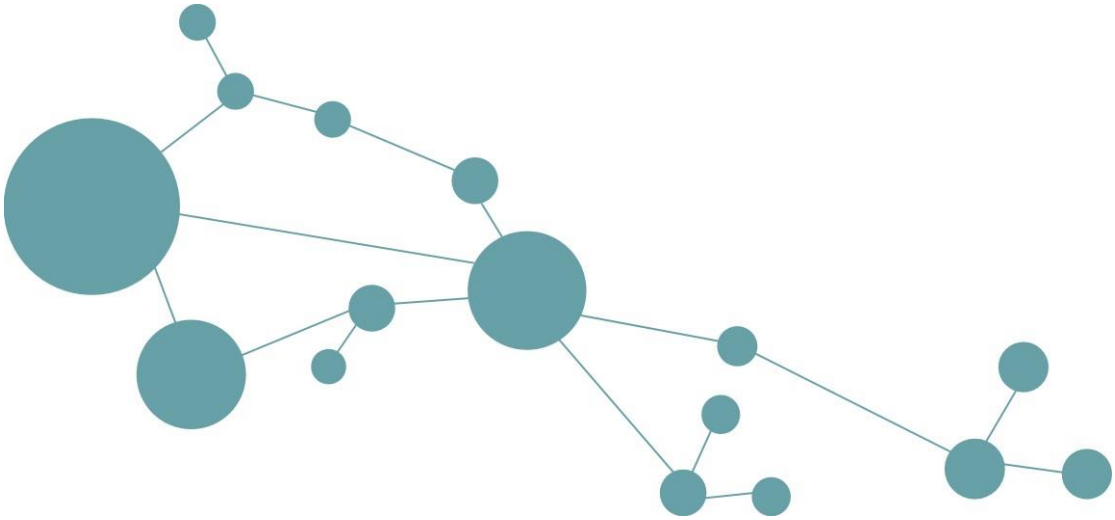


i-views 5.2

i·views

Contents

1 Knowledge Builder

1.1 Fundamentals

When using k-infinity, databases work the way people think: simple, agile and flexible. That is why in k-infinity 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 k-infinity 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 k-infinity has to offer.

Our central tool is the knowledge builder, one of the core components of k-infinity. 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

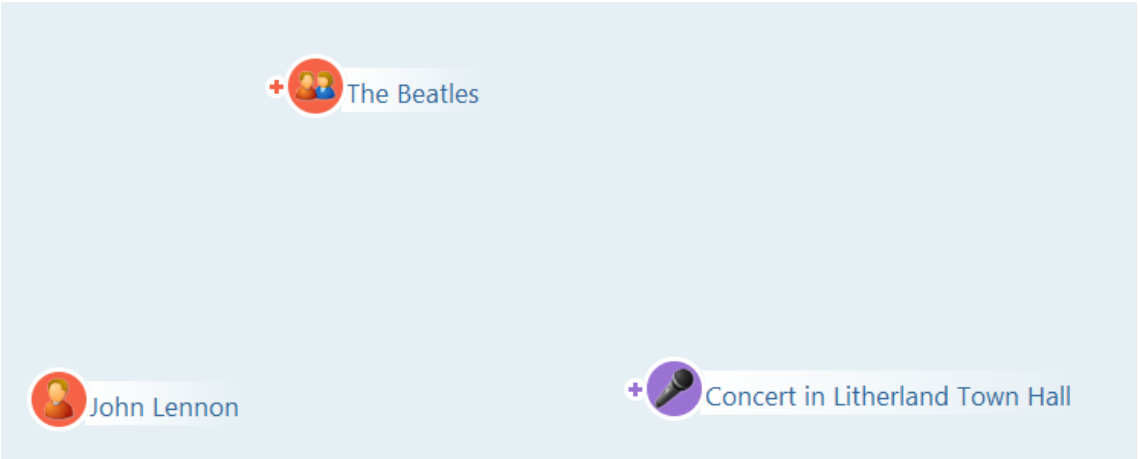
All these functions are the subject of this documentation. One continuous example is a semantic network surrounding music, bands, songs, etc.

1.1.1 Basic components

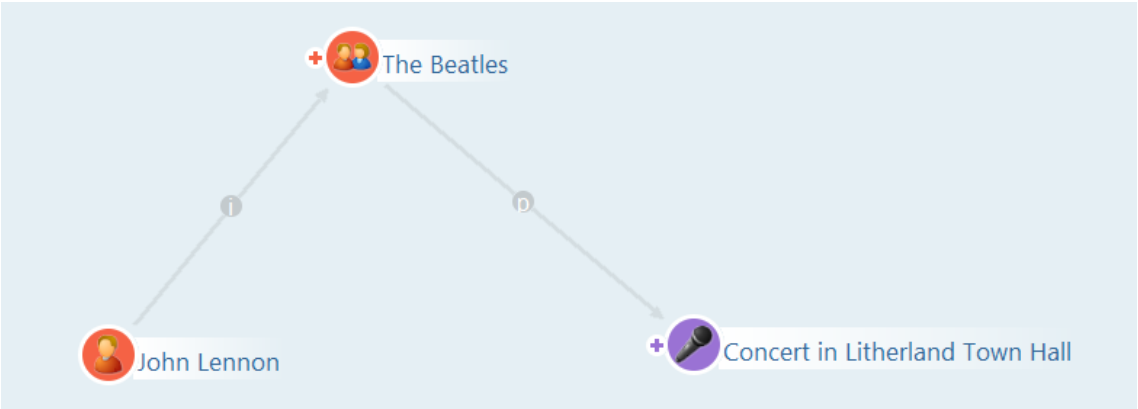
The basic components of modelling within k-infinity are:

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

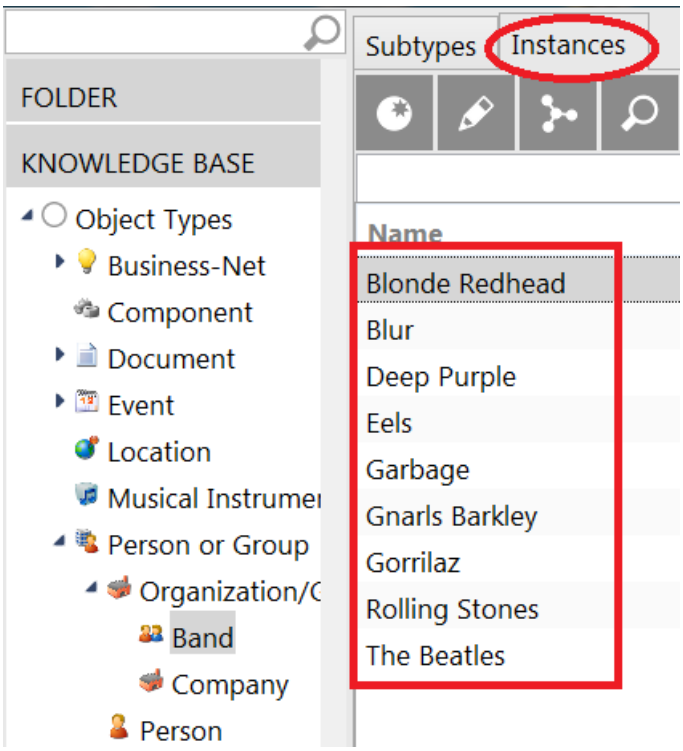
Examples for specific objects are John Lennon, the Beatles, Liverpool, the concert in Litherland Town Hall, the football world cup in Mexico in 1970, the leaning tower of Pisa, etc.:



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".

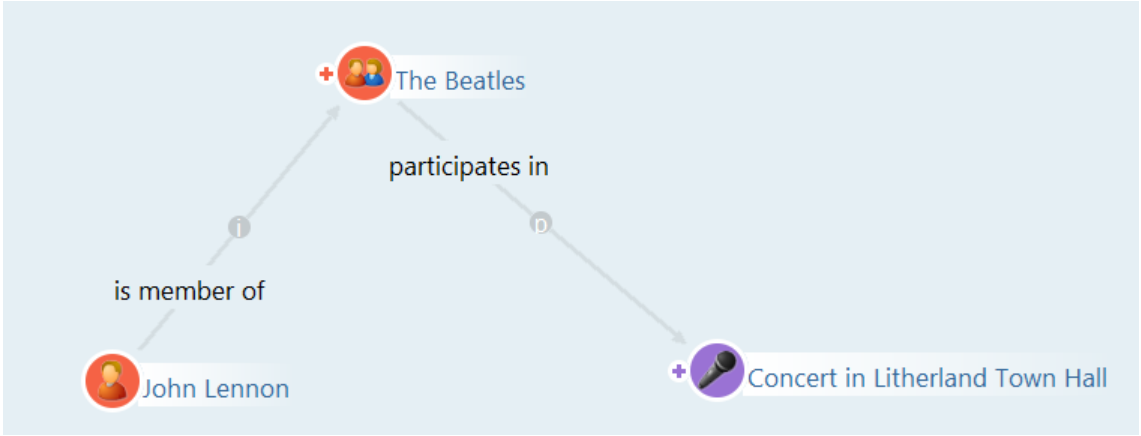


Additionally, we have introduced four types here: specific objects always have a type, e.g. the type of persons, type of the cities, the events or the bands - types which you may freely define in your data model.



The main window of k-infinity: 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 k-infinity 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.

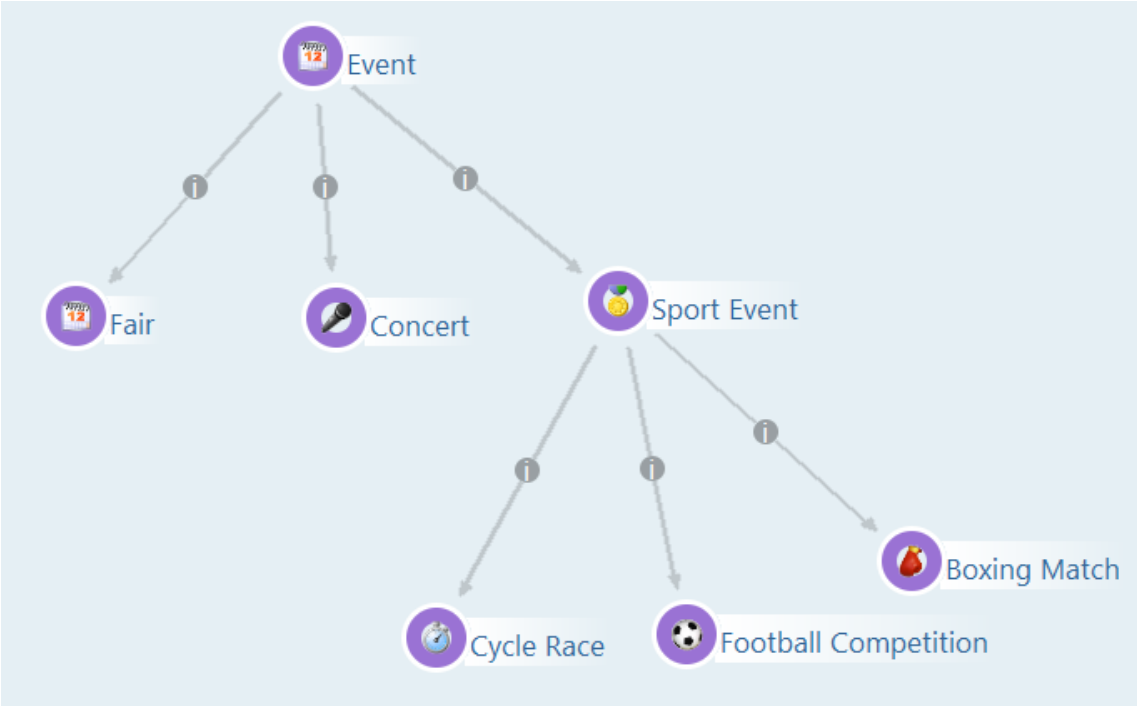


The same applies for attributes: in the case of a person these may be the name or the date of birth. Specific persons (objects of the type 'person') may then have name, date of birth, place of birth, address, colour of eyes, etc. Events may have a location and a time span. Attributes and relations are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

1.1.2 Type of 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).

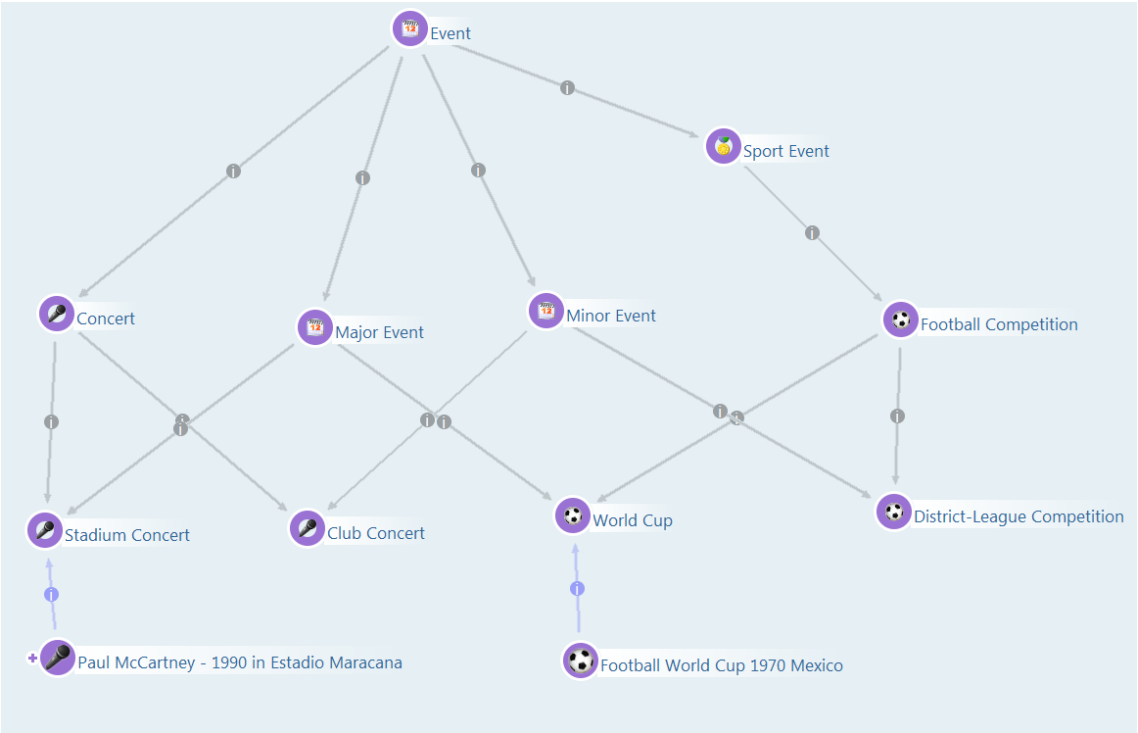
In this manner we obtain a hierarchy of supertypes and subtypes:



The hierarchy is transitive: when we ask k-infinity 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", k-infinity will reject a direct supertype / subtype relationship between event and boxing match - with a note that this connection is already known.

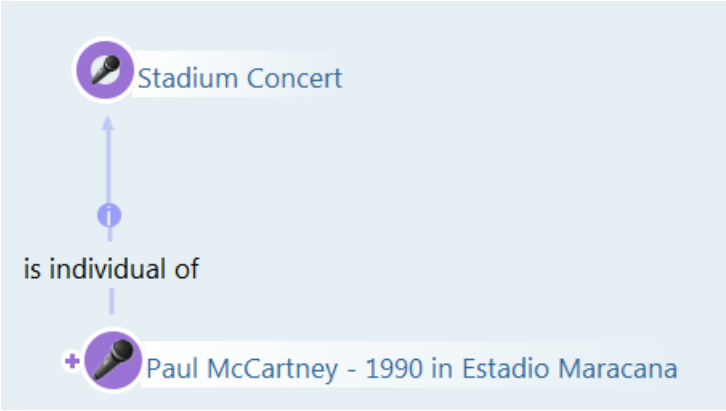
The hierarchical structure does not necessarily have to have the structure of a tree - a type of object may also have several upper types. However, an object may only have one type of object.

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:

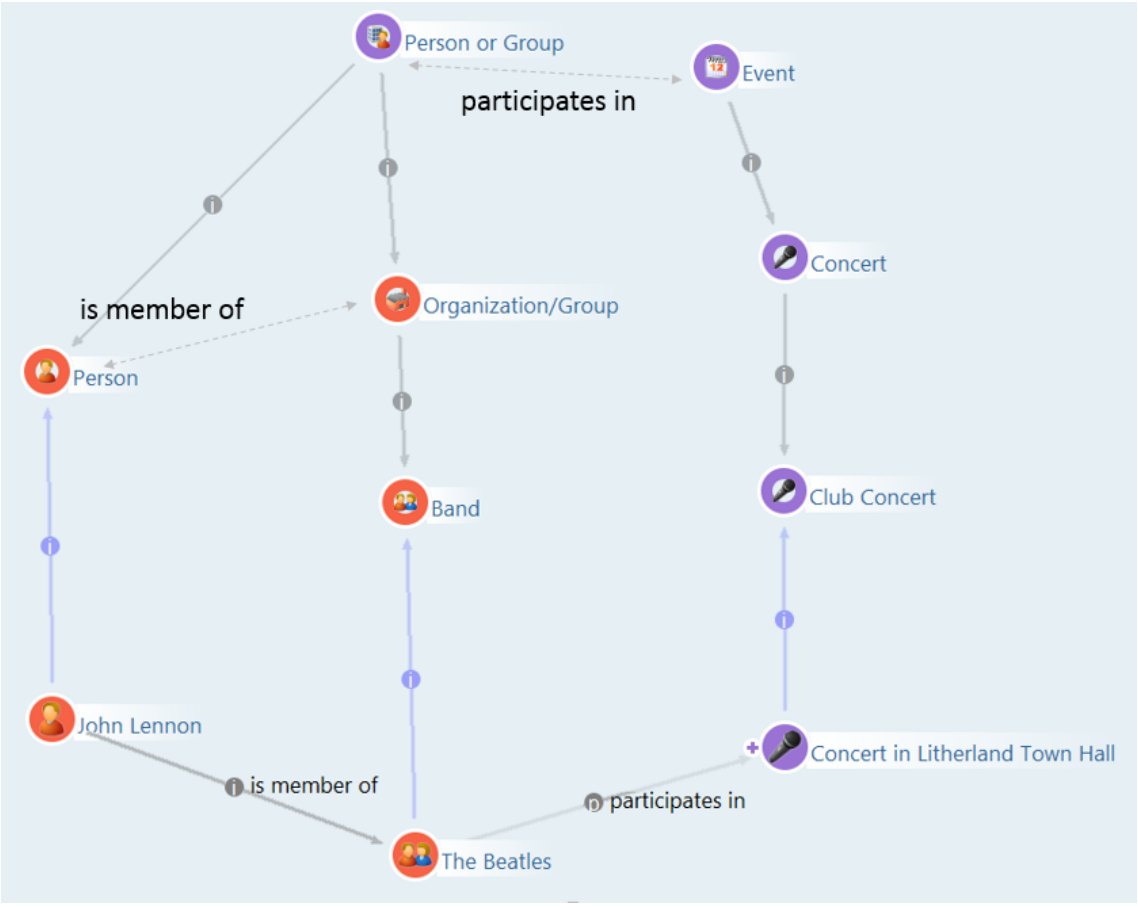


Type hierarchy with multiple inheritance

The affiliation of specific objects with a type of object is also expressed as a relation in k-infinity 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.

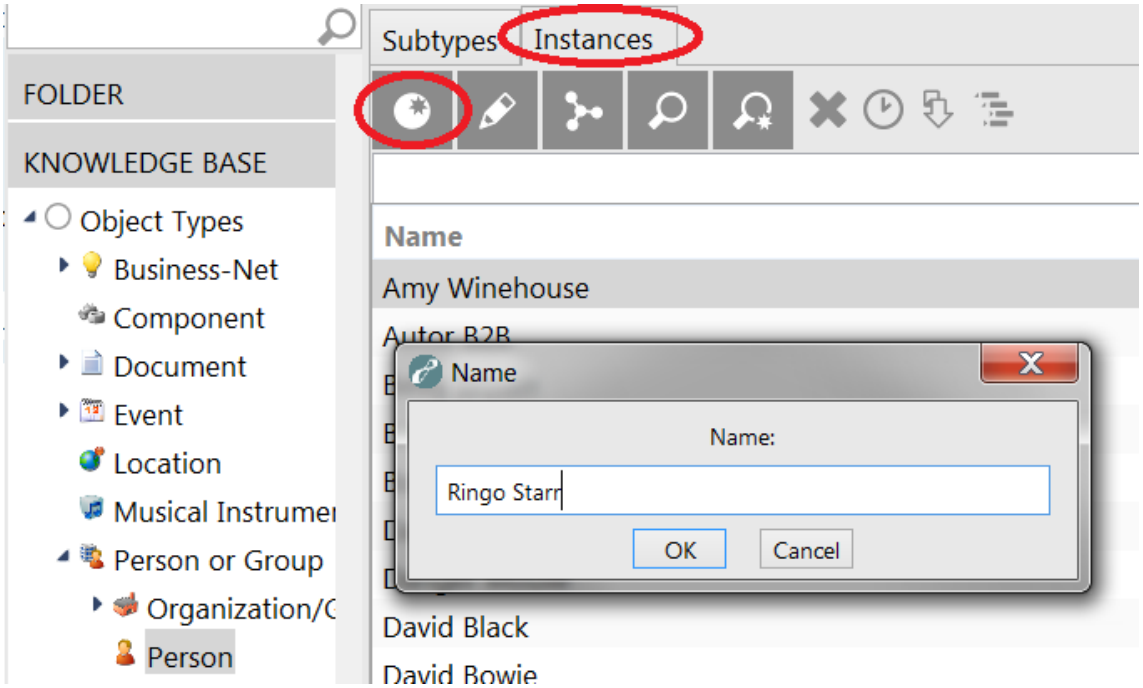


Inheritance makes it possible to define types of relations (and types of attributes) further up in the hierarchy of the object type and hence use them for different types of objects (e.g. for bands and other organisations.

1.1.3 Creating and editing objects

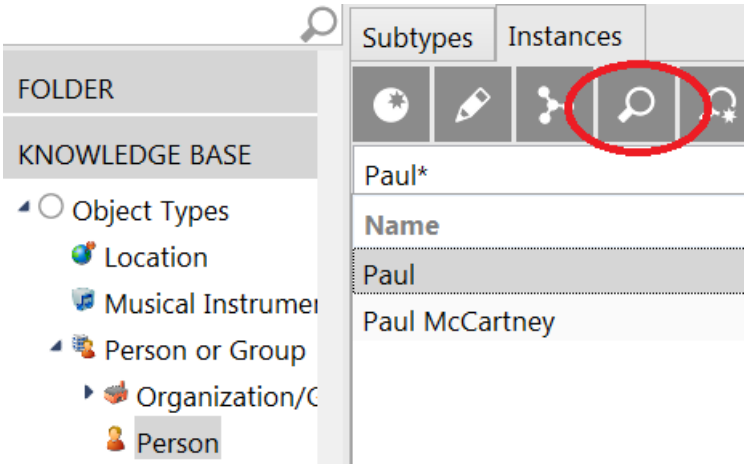
Creating specific objects

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.



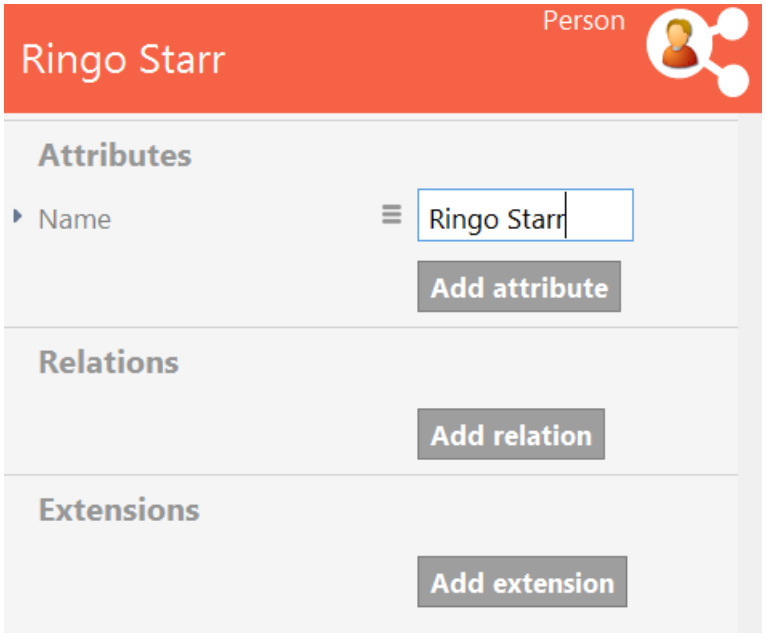
An object can be created by means of the button "new" and using the named entered

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 "?"):



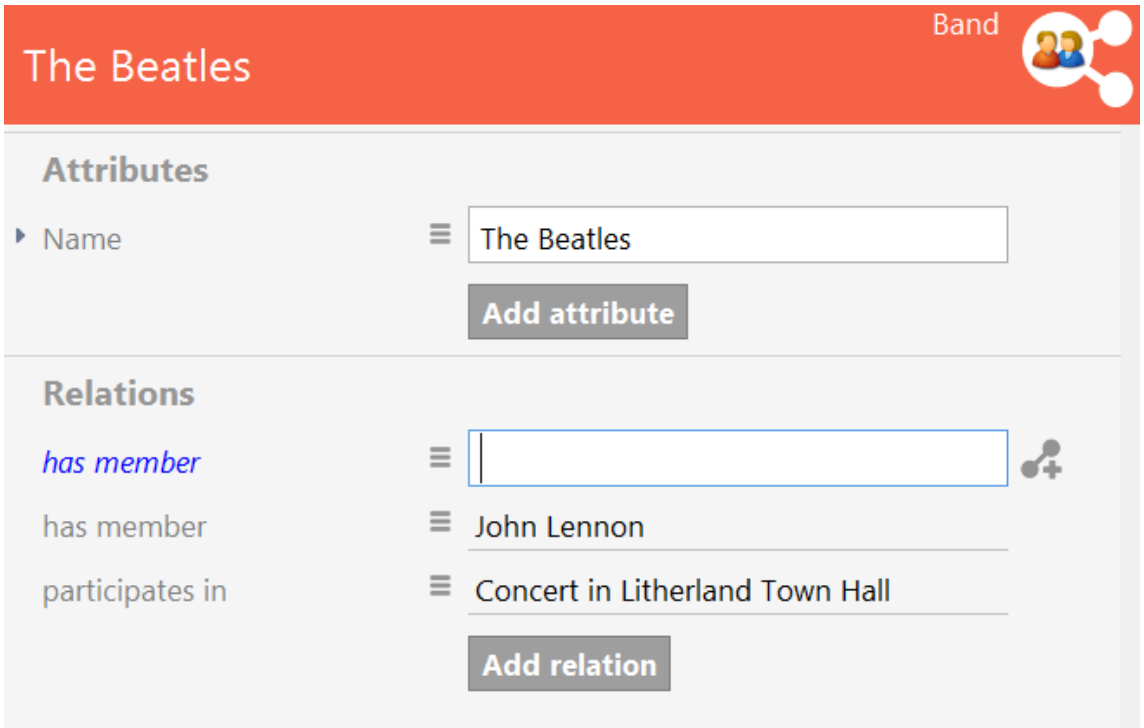
Editing objects

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.

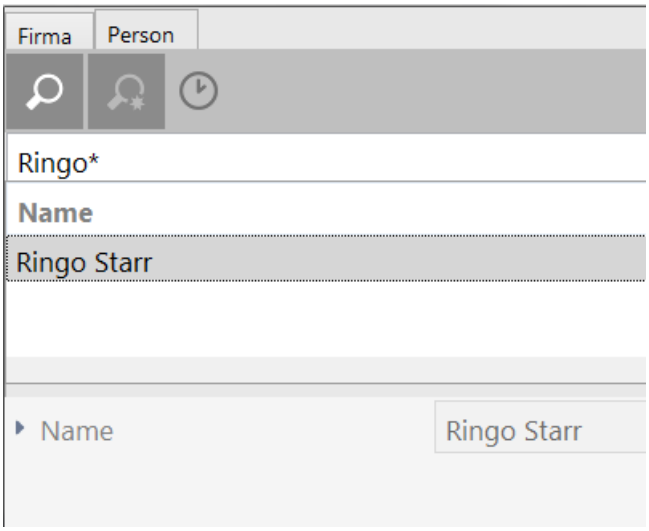


When editing an object we can, in addition to linking it to another object, also generate the target of the link if the object does not already exist.

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 k-infinity 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.

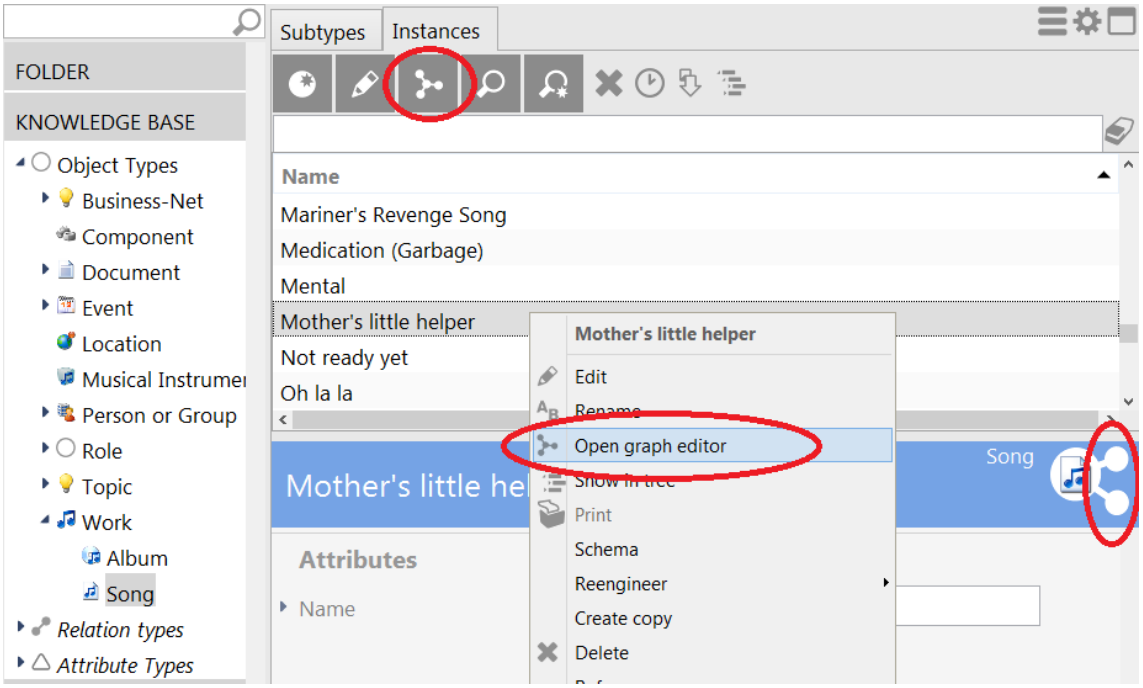
The target object of the relation itself will not be deleted as a result of this however. If an object has to be deleted this is done via the button in the main window or via the context menu directly on this object.

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

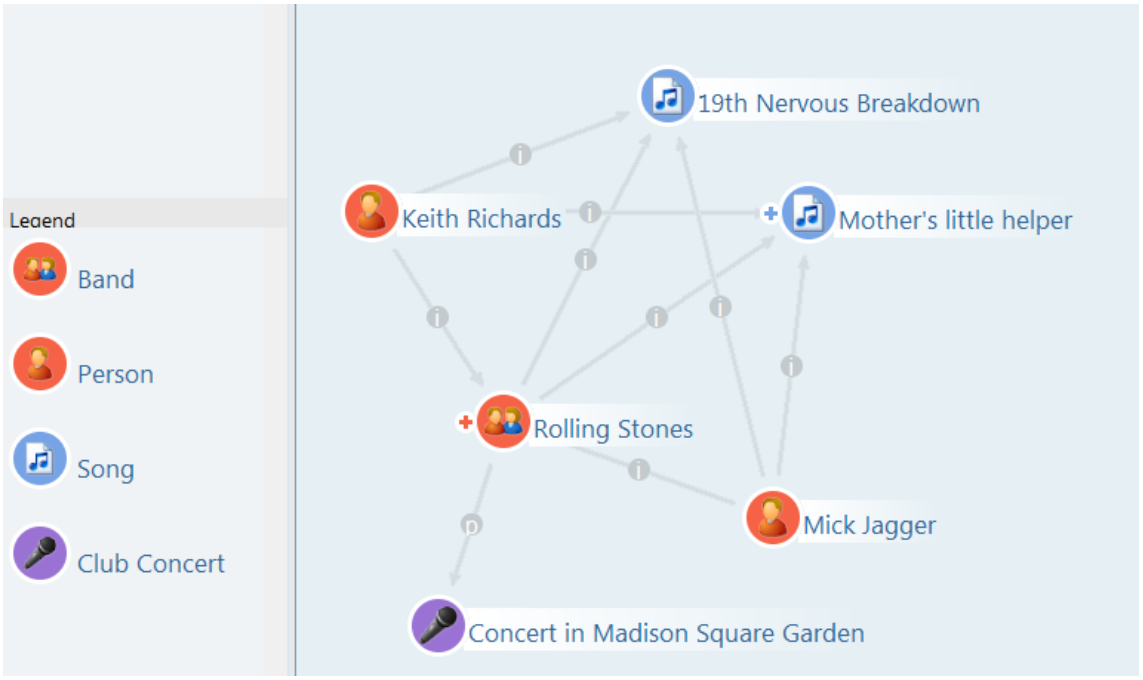
1.1.4 Graph editor

1.1.4.1 Introduction to 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.

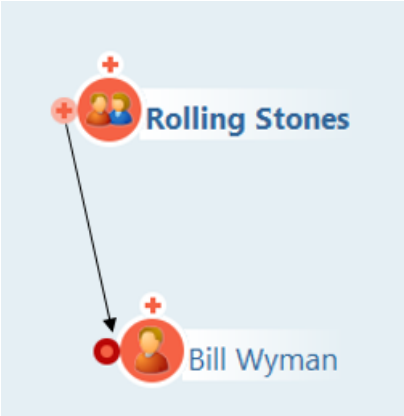


In the graph editor not only a section of the network may be displayed: objects and relations may be edited as well.

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.

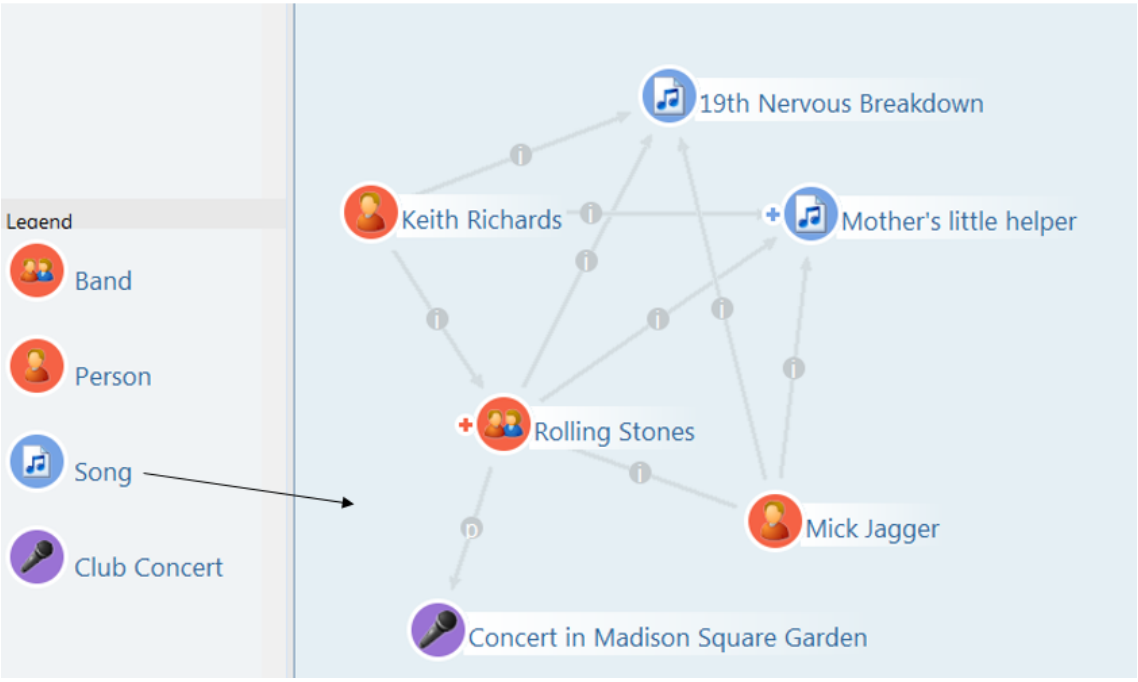


In order to display objects in the graph editor there are different options:

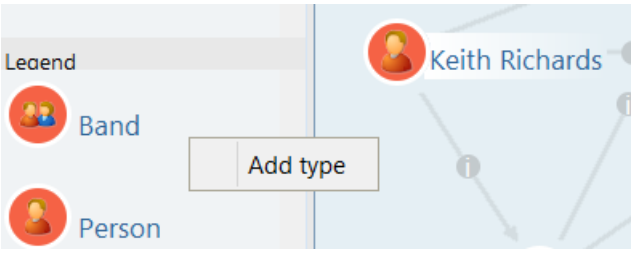
- Objects may be dragged from the hit list in the main window to the graph editor window using drag & drop.
- If the name of the object is known it can be selected via the context menu using the function "show individual".

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:



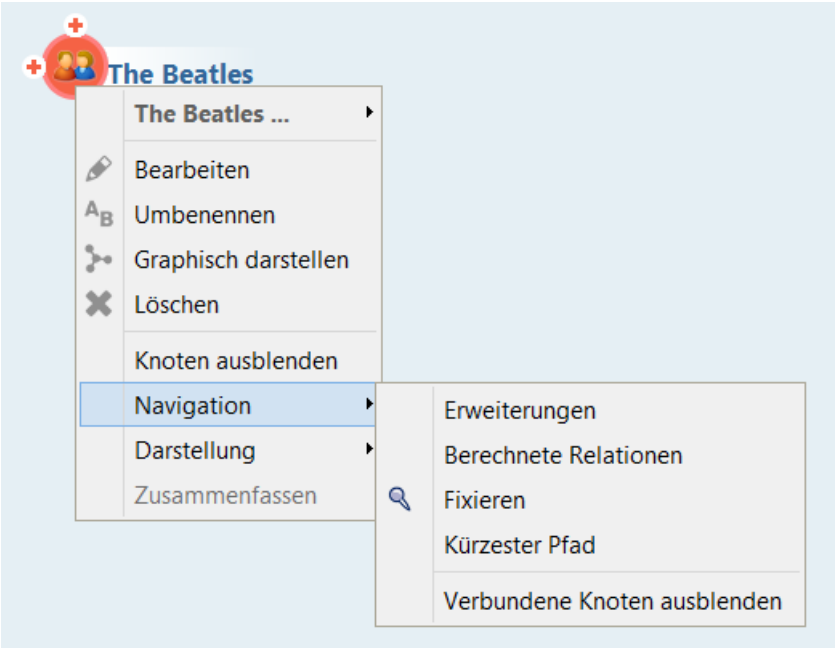
If there are no types of objects to be seen in the legend you can search for them using a right mouse click in the legend area. Following this, the name of the object will be given.



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 within the graph editor

The name can be changed later in the Admin Tool or in the Knowledge Builder. The user, who is created here, has automatic graph administrator rights. The context menu can be used to perform further operations by right-clicking on the object. For the most part, this context menu offers the same functions as the form editor, but also contains additional Graph Editor-specific components.

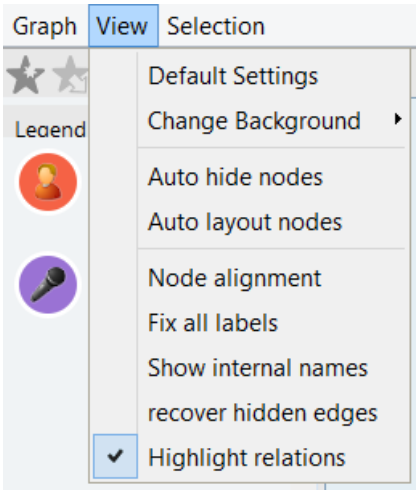


This context menu offers the following graph editor-specific functions:

Hide node: Here the node can be hidden.
Navigation - Extensions: Opens the extensions to an object.
Navigation - Computed Relations: Opens the computed relations to an object.
Navigation - Fix: Fixes the position of a node in the Graph Editor, so that it is not moved even when the layout is rebuilt. The fixation can be canceled again with the option Loosen
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

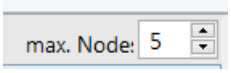
one step:

- **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:



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

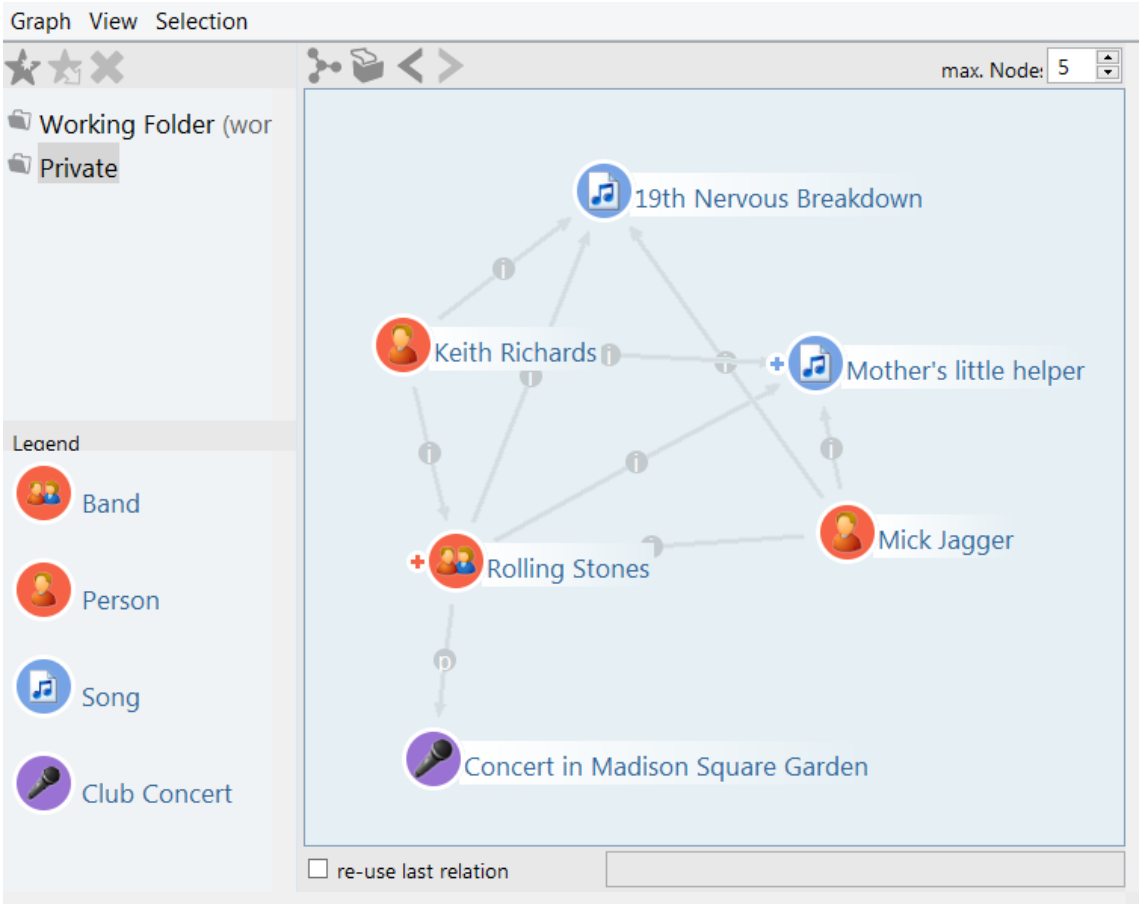
Fix all labels: using this option the names of all relations are always visible, not only when rolled over with the mouse. Alternatively, the description may be fixed directly in the context menu of a relation.

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.

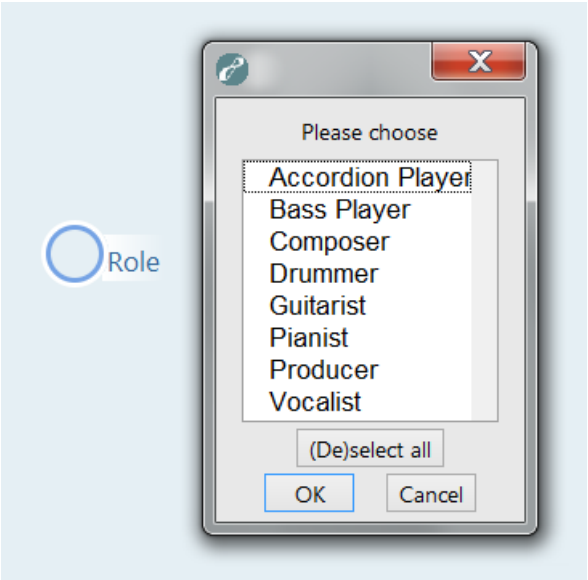


This legend shows the types of objects for the specific objects on the right-hand side.

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).

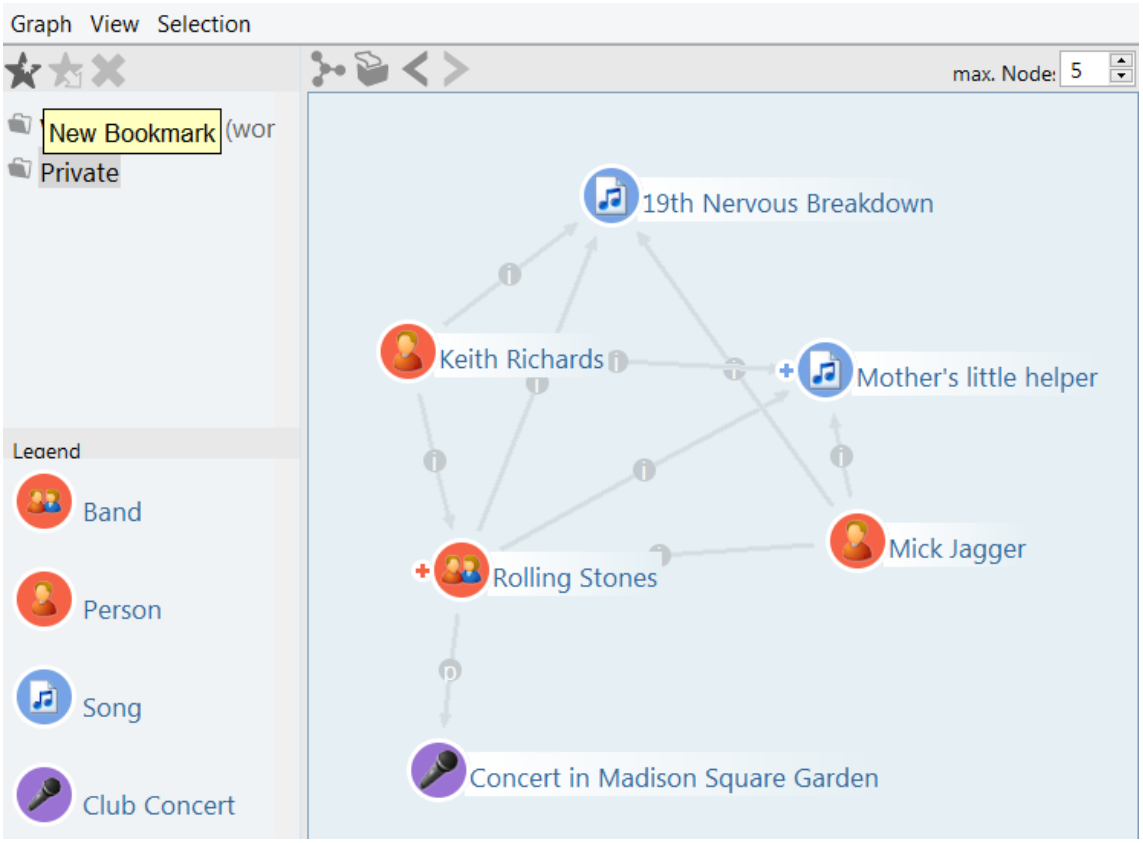
If the drag point has been clicked to show the adjacent objects a selection list will appear instead of the objects.



1.1.4.4 Bookmarks and history

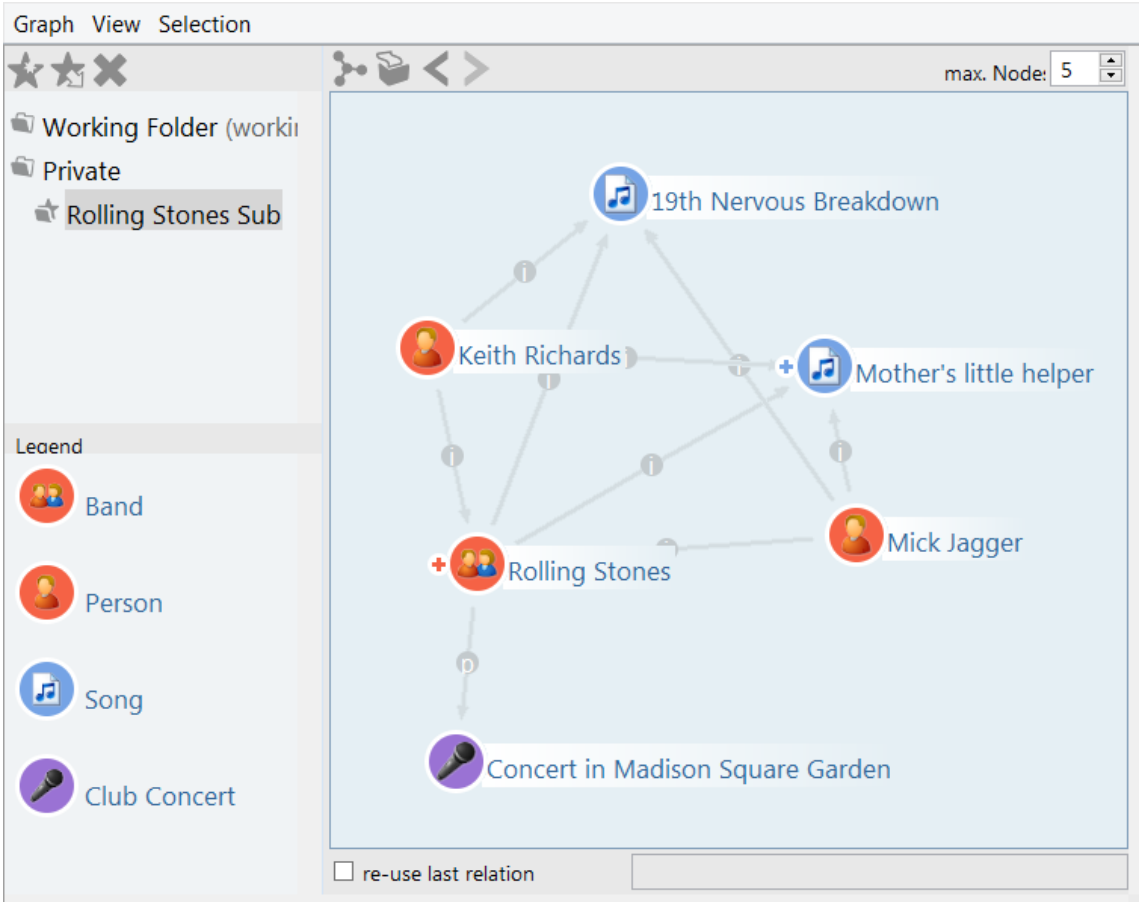
The menu *graph* contains more functions for the graph editor:

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.

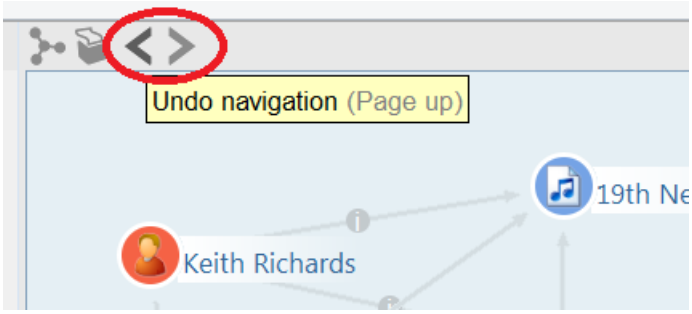


When a bookmark is created it may be given a name. All nodes contained in the bookmark are listed in the description of the bookmark.

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".

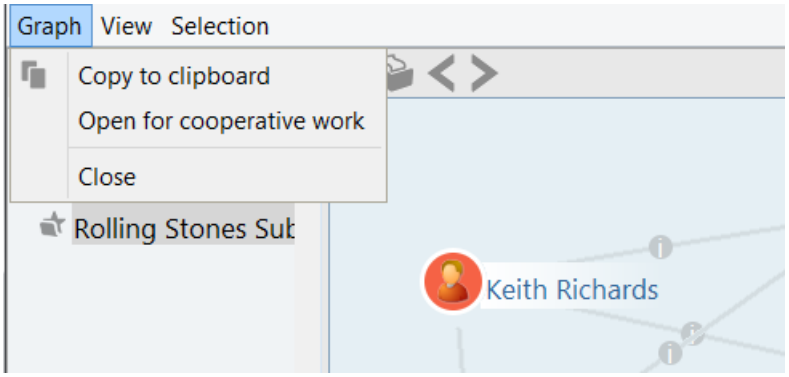


Layout: the layout function enables you to position nodes automatically when many nodes are not allowed to be positioned manually. When more nodes are displayed they will also be automatically positioned in the graph via the layout function.

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.

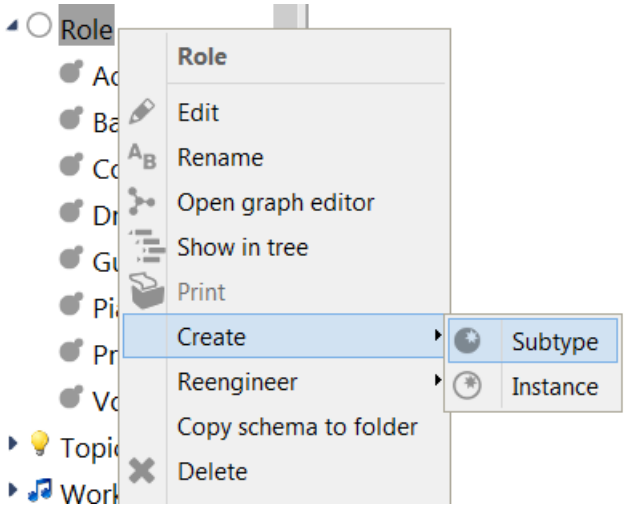
Cooperative work: this function enables other users to work on the graph mutually and simultaneously. All changes and selections of a user on the graph (layout, showing/hiding nodes, etc.) will then be shown to all other users synchronously.



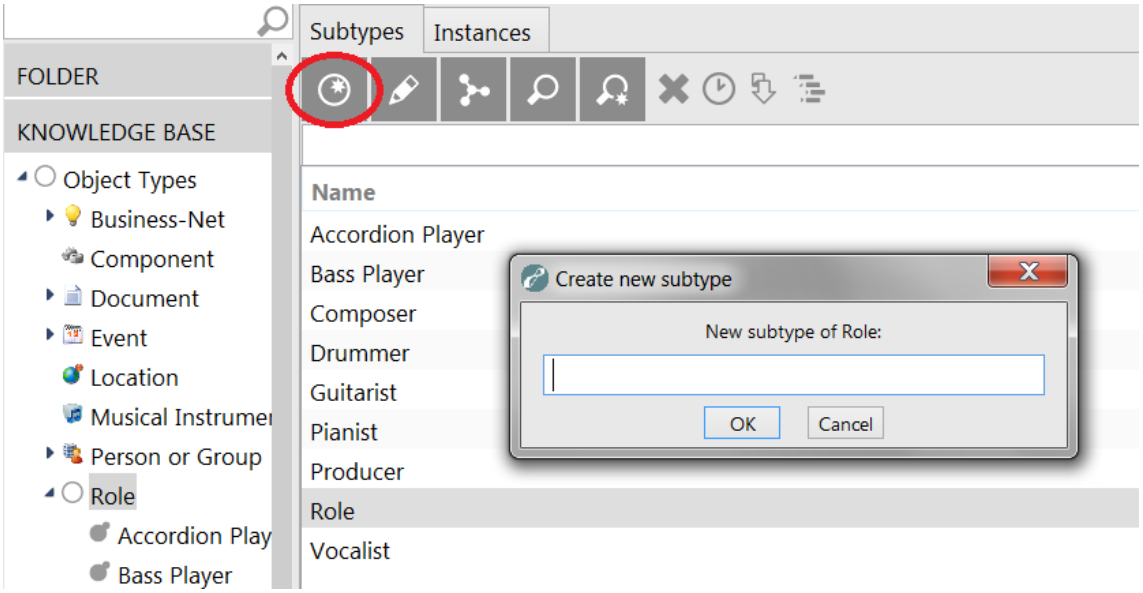
1.2 Schematic definition / model

1.2.1 Defining 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



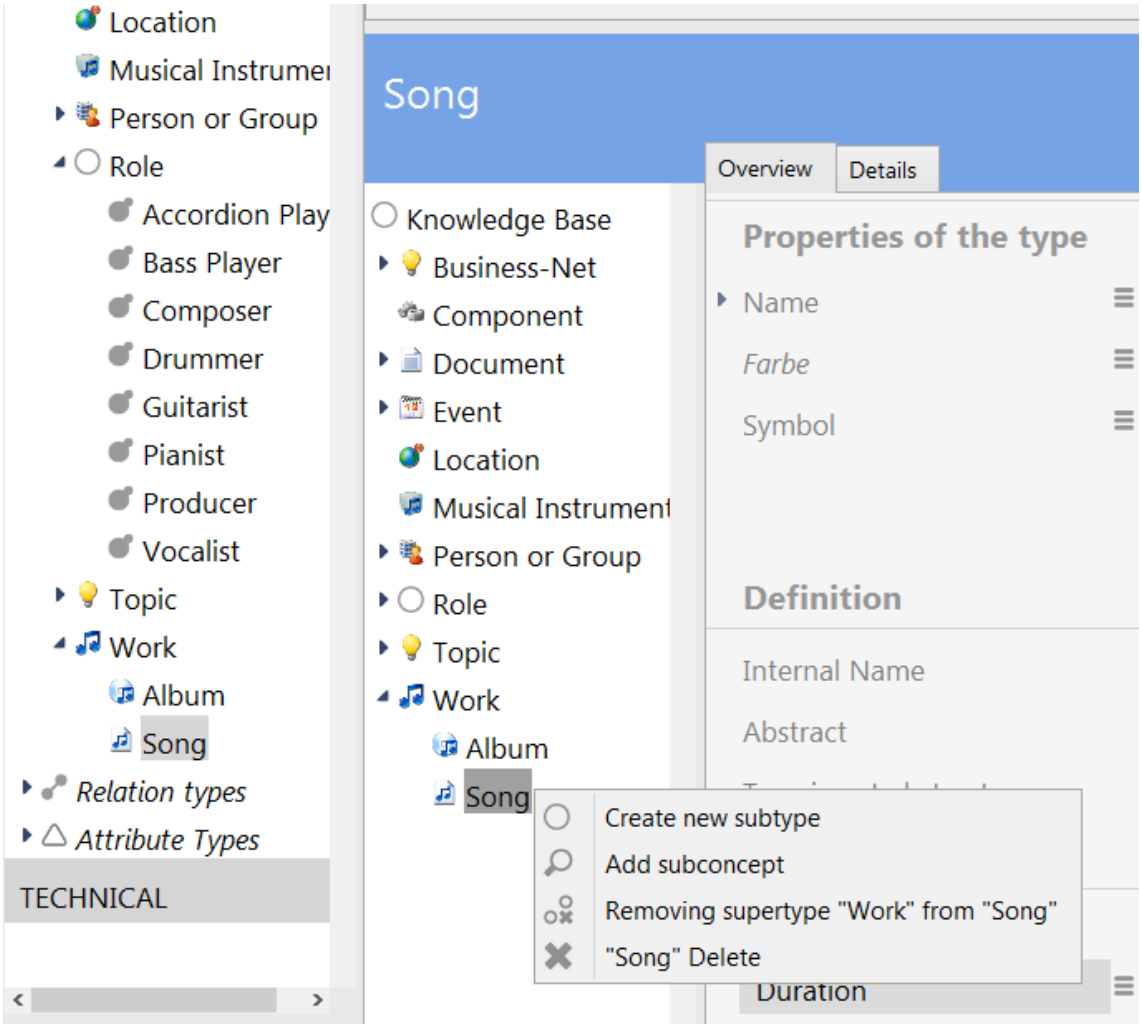
or in the main window using the tab "Subtypes" above the search field and the tab "new":



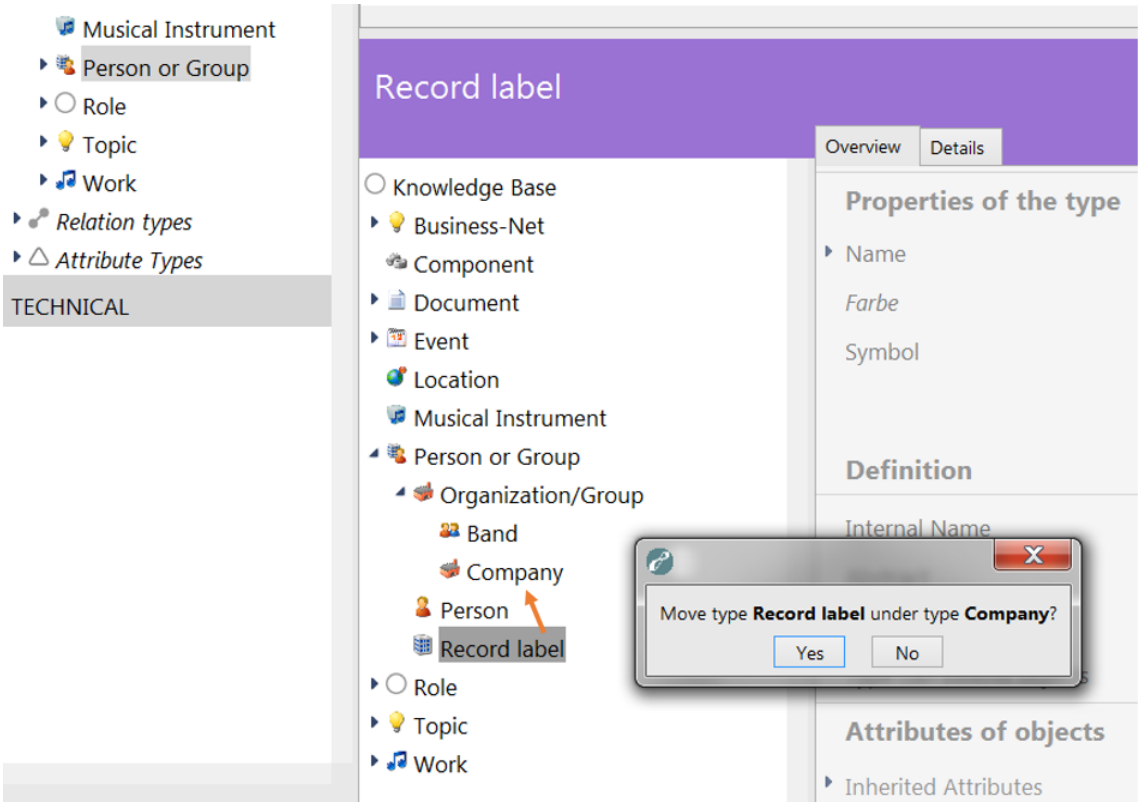
Changing the type hierarchy

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.



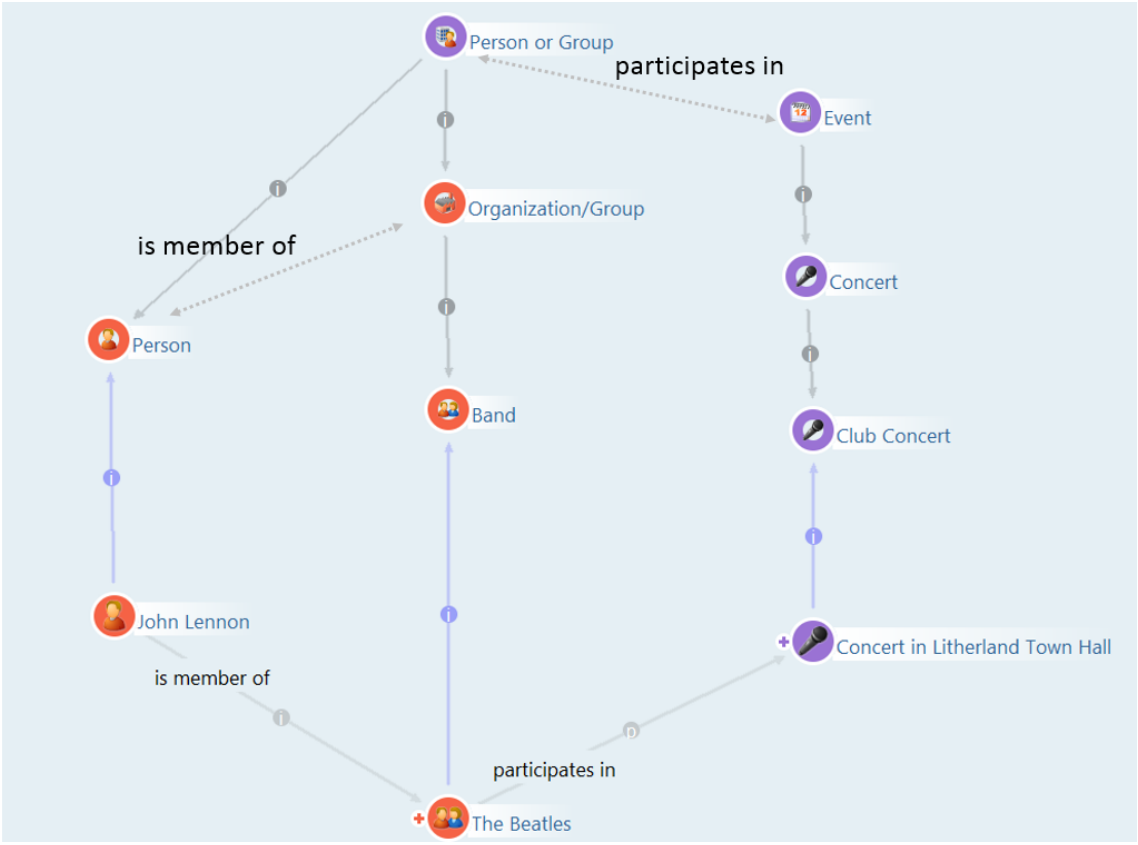
Using this option we can remove the currently selected object type from its position in the hierarchy of the object types and with drag & drop we can move an object type to another branch of the hierarchy. If we hold down the Ctrl key when using the drag & drop function the object type will not be moved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

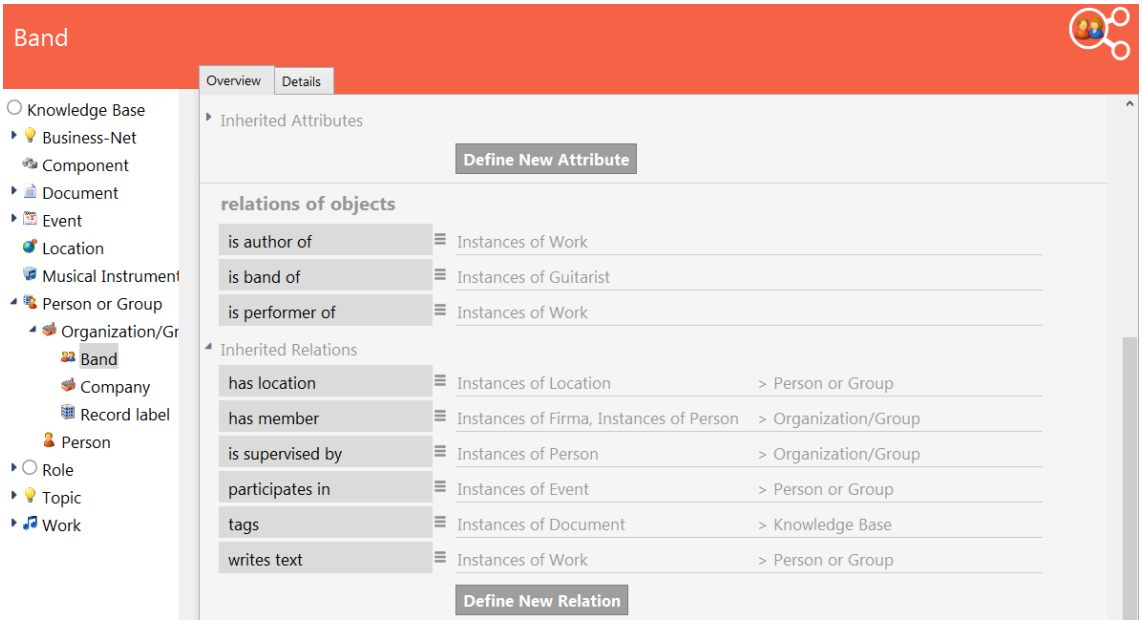


Configuring object types with properties

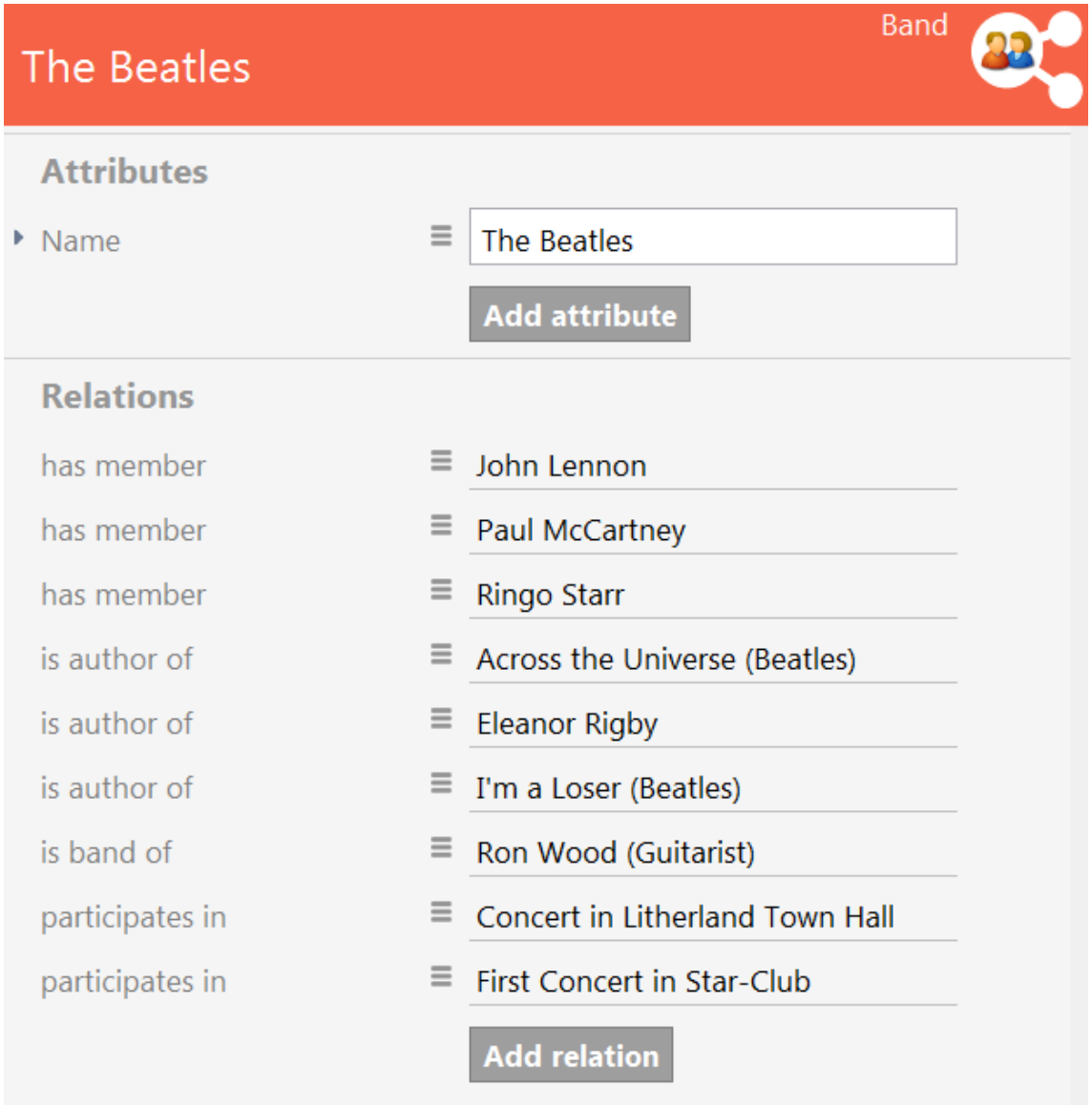
In the simplest case we define relations and attributes with an object type such as "band" or "person" and thus make them available for the specific objects of this type. (For example the year and location the band was established, date of birth and gender of people, location and date of events.)

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:





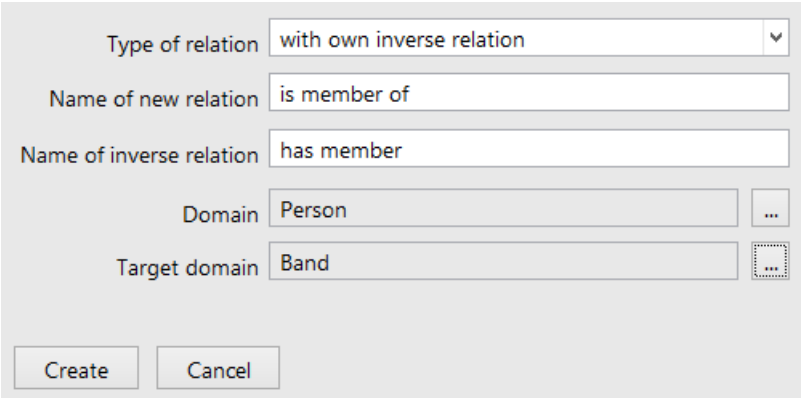
The editor for the object type "band" with directly defined and inherited relations there.



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 k-infinity: 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, k-infinity 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".



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

Defining attributes

When defining new attribute types, k-infinity 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

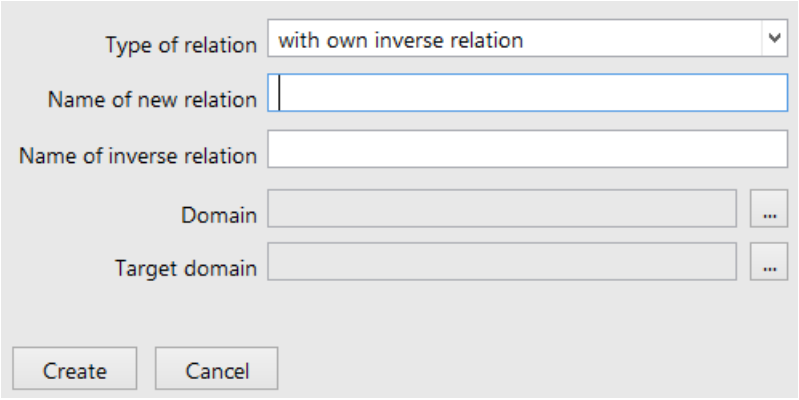
The intention of using these data types is not to define everything as character strings. Technical data types in a defined format later offer special feasibilities of inquiring and comparing. For example, numerical values may be compared to larger or smaller values within the structured queries and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

1.2.2 Details of the relation types and attribute types

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

1.2.2.1 Creating a new relation type

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



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 k-infinity 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:

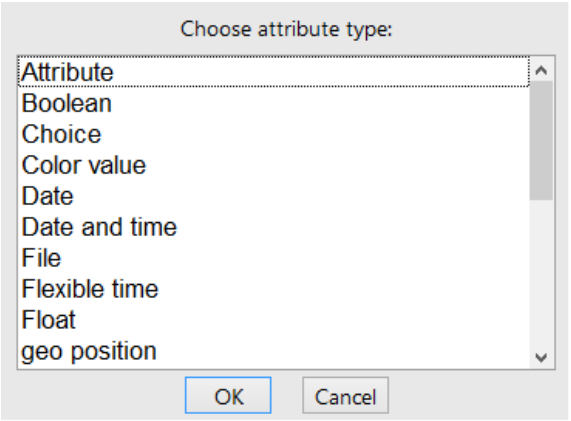
[John Lennon] [is a member of] [The Beatles]

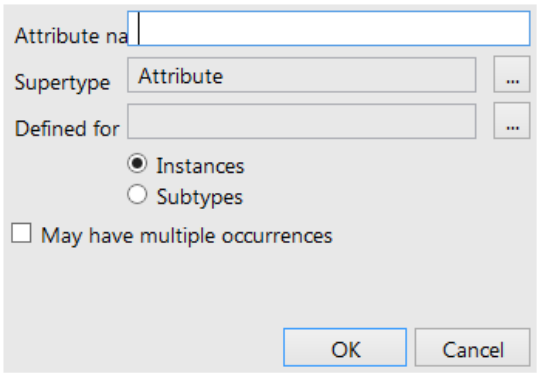
Furthermore it is helpful when the opposite direction (inverse relation) takes on the word selection of the main direction: "has a member / is a member of".

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 Creating a new attribute type

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





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

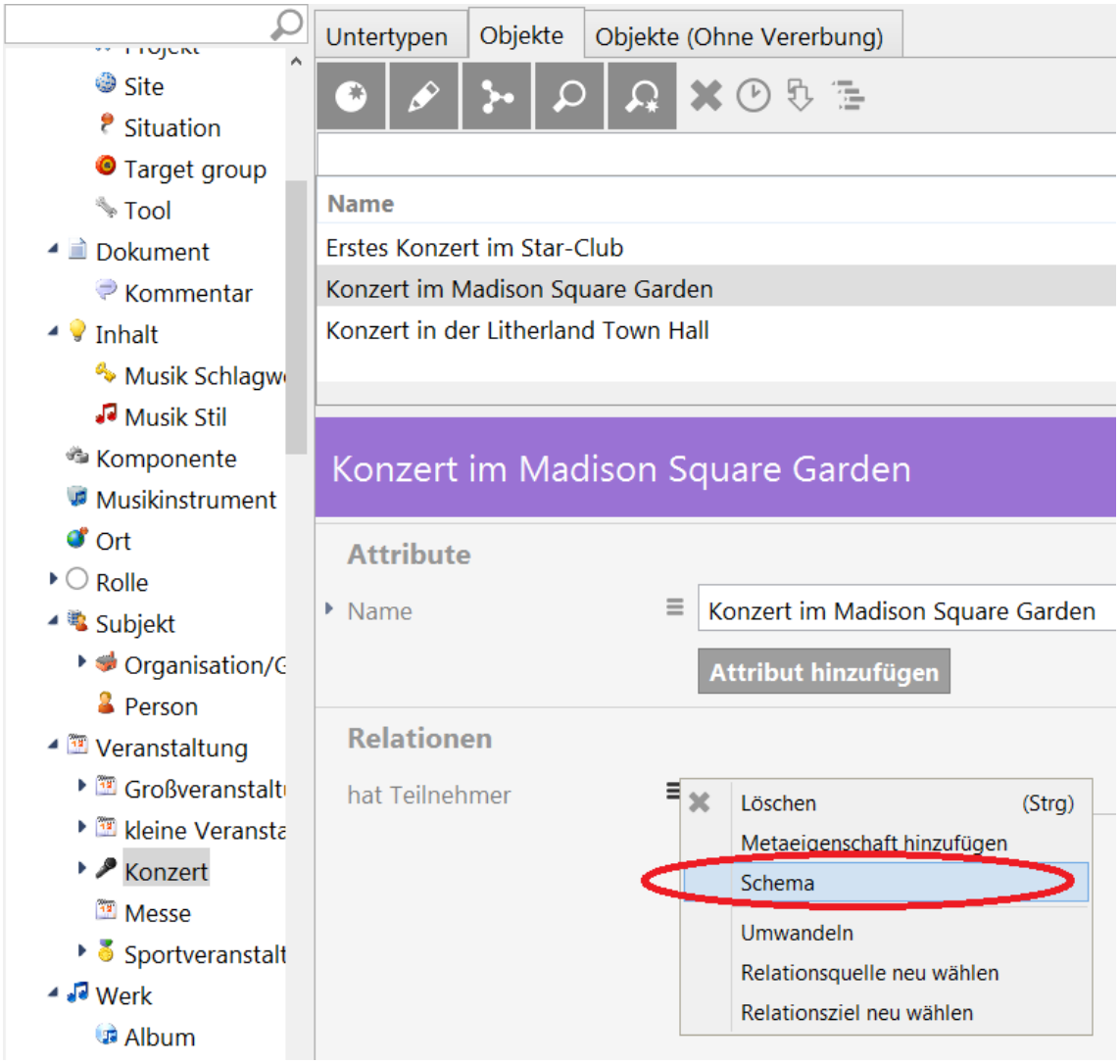
Supertype: here it is defined at what level in the hierarchy the attribute type should be placed.

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).

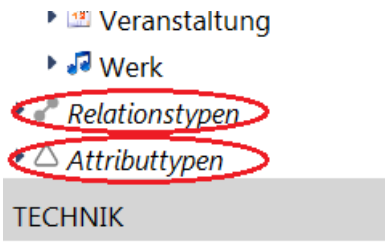
1.2.2.3 Modification of details

The dialogs for creating new attribute and relation types are reduced views of the attribute and relation type editors. To edit details of relations and attributes, editors with extended functionality must be started.

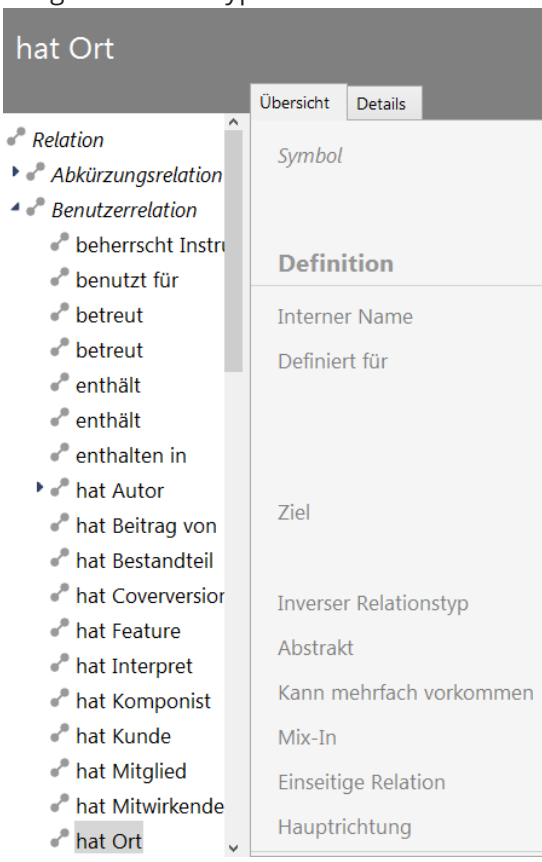
These two editors can be accessed via the list of relations and attributes in the "Schema" tab of the object editor:



Alternatively, it can be accessed via the hierarchy tree on the left in the main window. The object types include the hierarchies for relation and attribute types. The context menue can be opened with a right-click on the relation or attribute. To edit it select the option "edit".



In the following,we look at the details of the definition of properties using the relation type editor as an example (the attribute type definition is a subset thereof):



Defined for: Here we can subsequently change for which object types the relation can be created. Relations can be defined between multiple objects and thus they can have multiple sources and targets. With that we have the possibilitye.g. to allow in the scheme that both persons and bands can be authors of a song or assigned to a place - even if they do not have a common top type. With the "Add" button we can add more object types. With "Remove" we can deprive the selected object type and all its objects of the possibility to enter this relation. "Change" allows to exchange an object type. Existing relations are then deleted by the system. If there are relations to be deleted, a confirmation dialog appears before the change is made. **Goal**: Here you can change to which types of objects the relation can be dragged. To change the target object type, you have to change the inverse relation type: The button for changing has the name of the inverse relation type. After clicking the button, the inverse relation appears in the editor and can be edited in the same way as the previous

relation.

1.2.3 Model changes

In k-infinity 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).

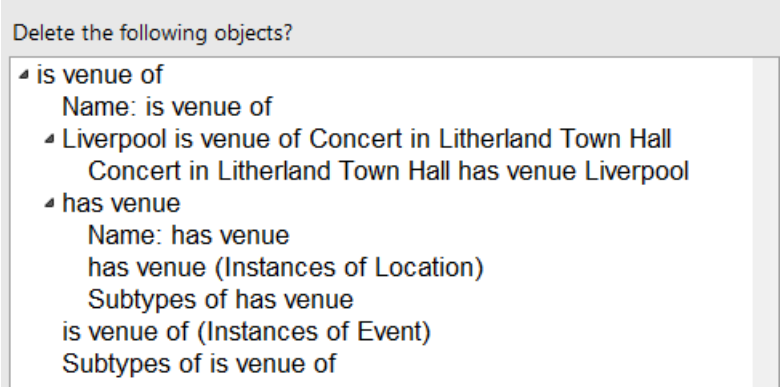
The system ensures consistency. When creating objects and properties the opposite direction of a relation is always included. Attribute values are checked as to whether they match the defined technical data type (for example, in a date field we cannot enter any random character string).

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.

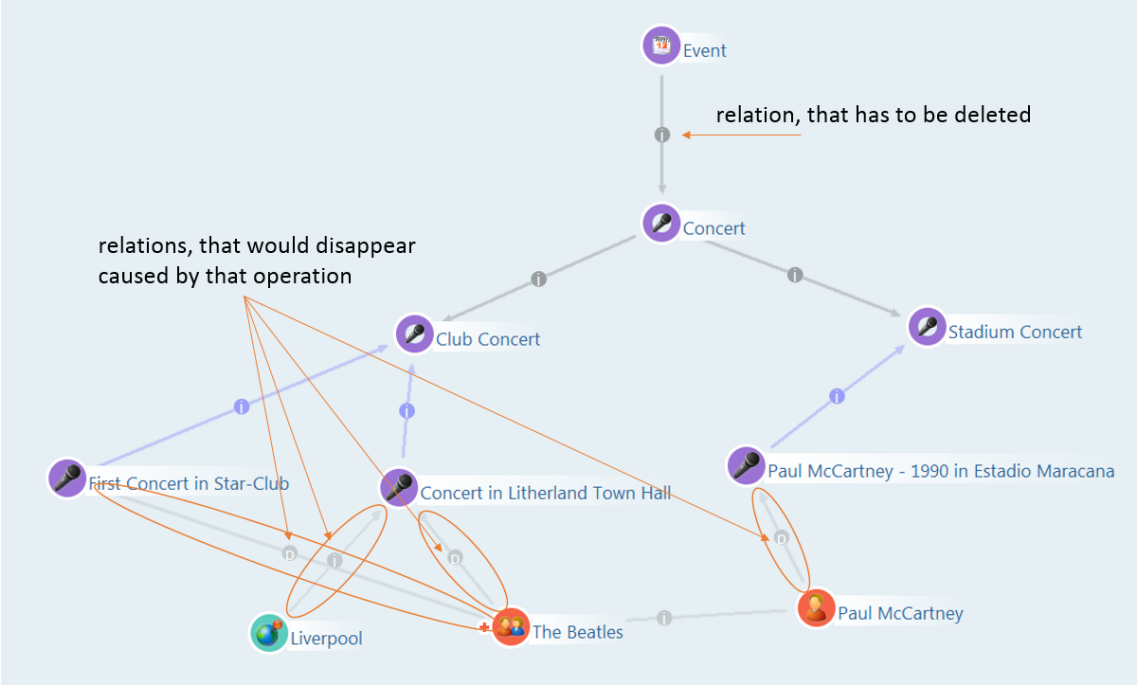
Since k-infinity always ensures that the objects and properties are in accordance with the model, deleting an object type or, where necessary, an operation has far-reaching consequences: when an object type is deleted, all its specific objects are also deleted - analogue to the relation and attribute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

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



k-infinity 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.



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, k-infinity 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.

k-infinity 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, k-infinity 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.

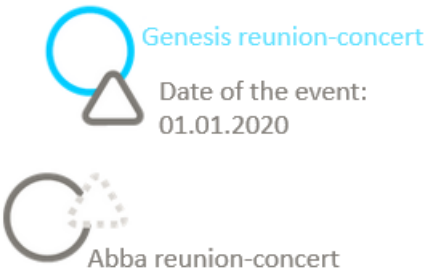
Special functions

Changing type: objects already in the knowledge network may be moved to objects of an-

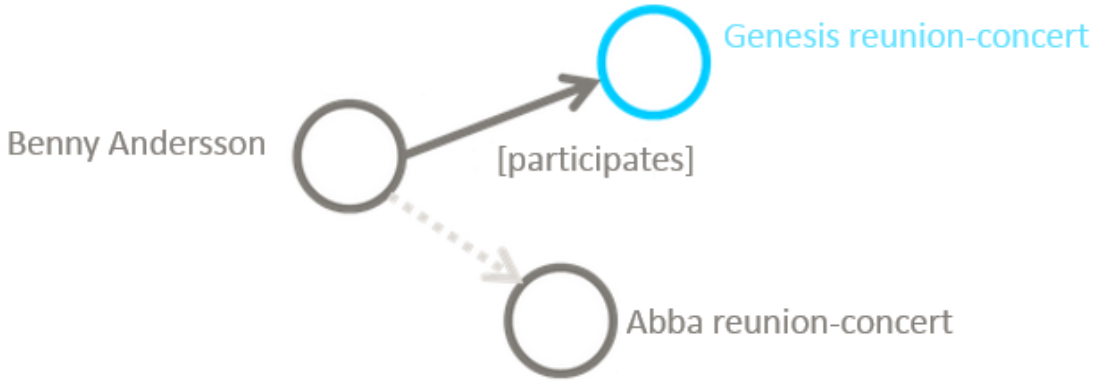
other type. For example, if the object type "event" differentiates to "sports event" and "concert". If there are already objects of the type sports event or concert in the knowledge network, they may be selected from the list in the main window and quite simply moved to a new, more suitable object type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



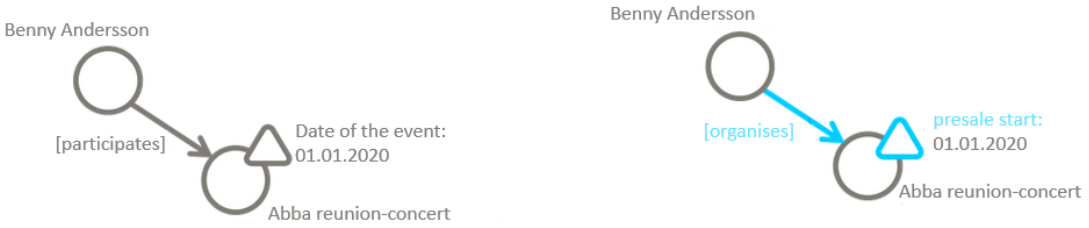
Reselect relation target: in relations this does not only apply to the source, but also the relation target.



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:



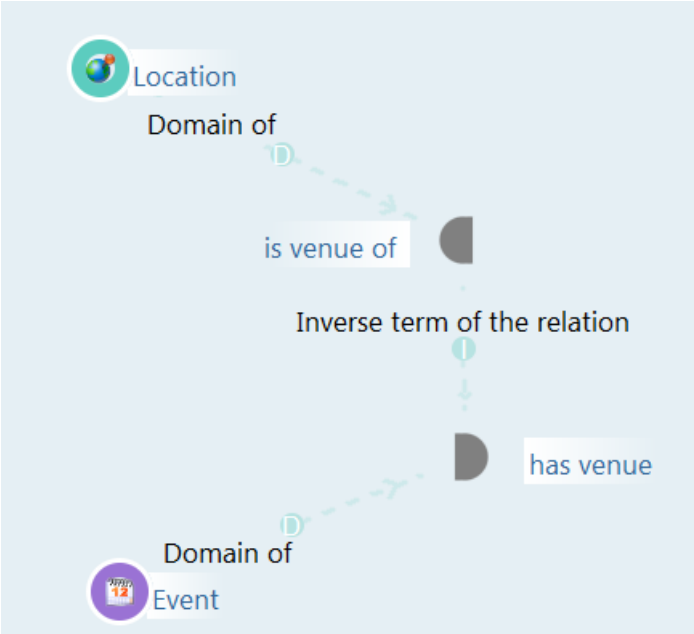
When converting the individual relations we are usually quicker when we delete these and replace them with another one. However, it may happen that meta properties are attached to the properties which we do not want to lose. On the other hand, the converting operations are also available for all properties 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 Presentation of diagrams in the graph editor

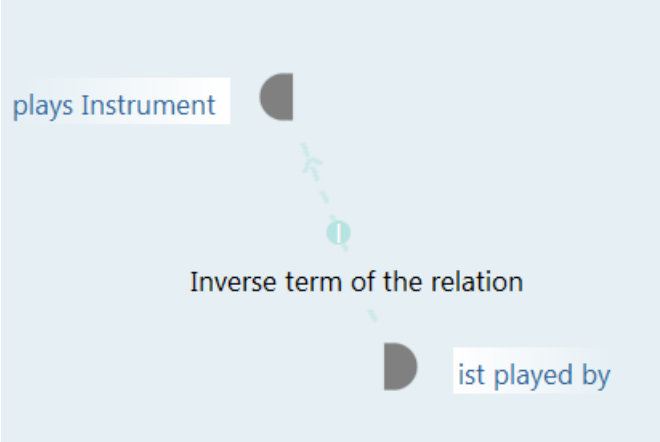
Until now we have mainly been dealing with linking of specific objects within the graph editor. Presenting such specific examples, discussing them with others and, where necessary, editing them is also the main function of the graph editor. We can, however, also present the model of the semantic network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

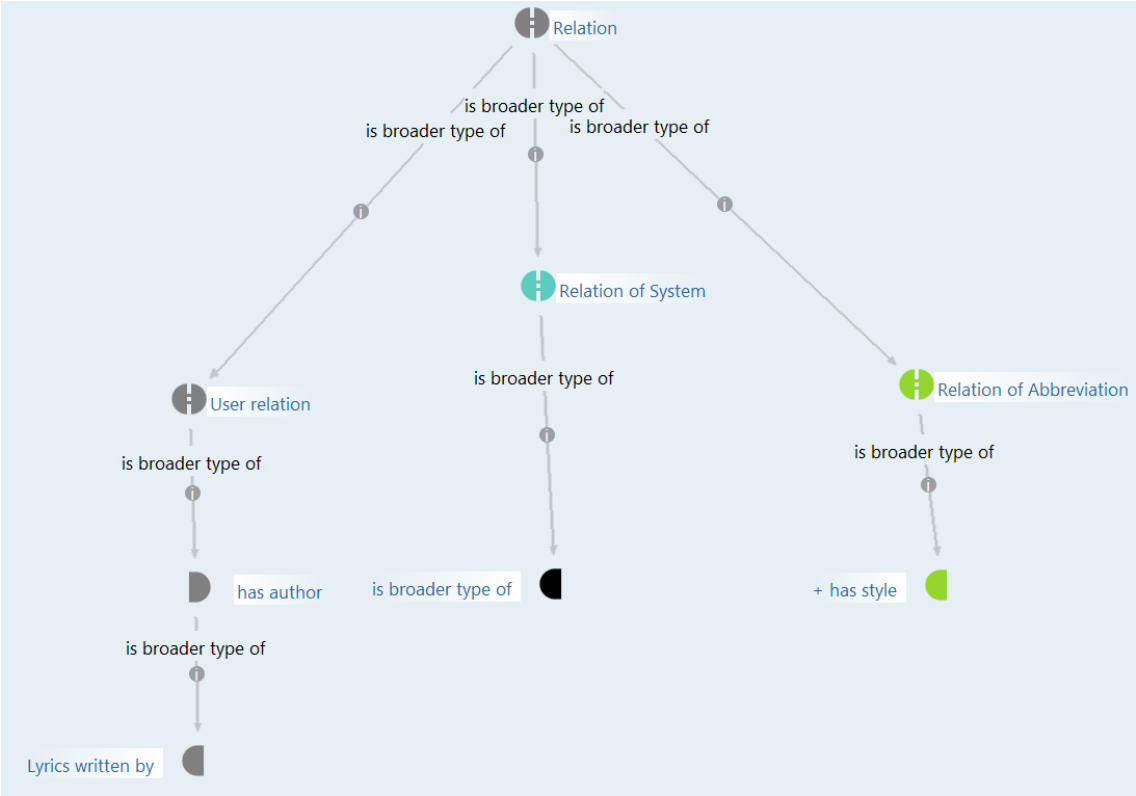


Relation types in the graph editor

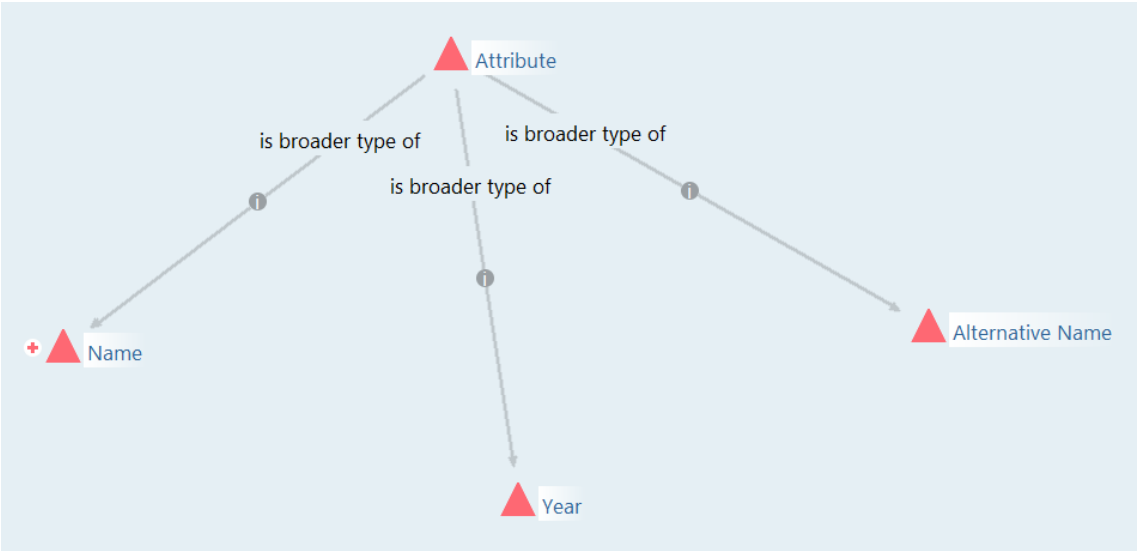
If until now we have been referring to relations in the graph editor, this concerned relation objects between specific objects of the knowledge network. Moreover, the general types of relations (hence the diagrams of the relations) may also be presented in the graph editor. A relation is depicted in 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.



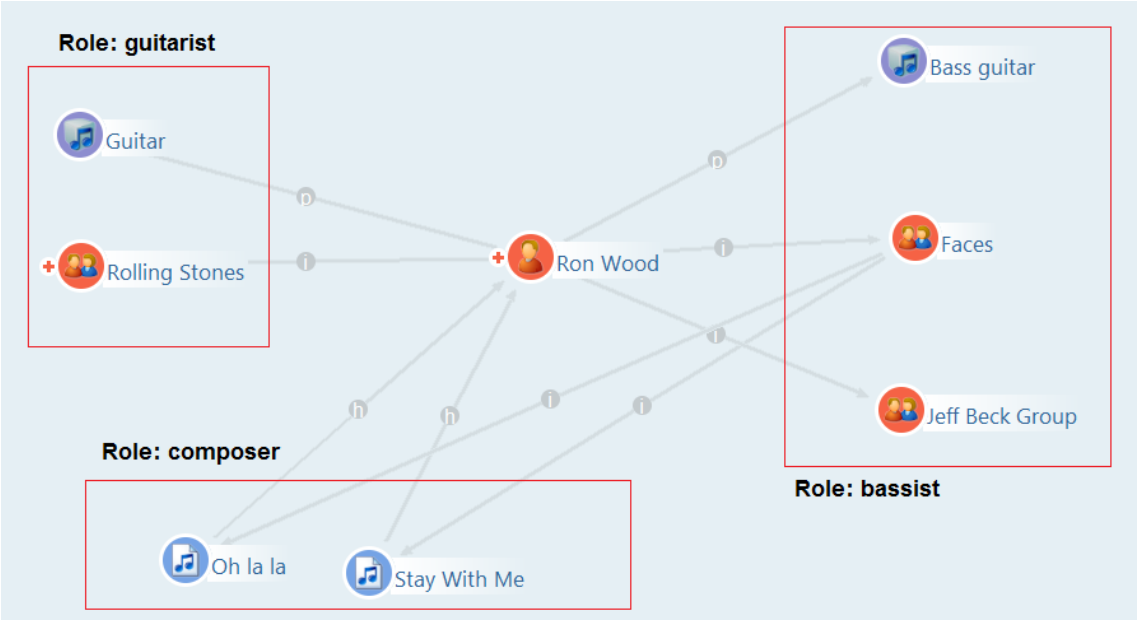
Analogue to the type of object hierarchy the hierarchy of the relations and attributes within the graph editor may be changed by deleting and dragging the supertype relation.

1.2.5 Meta model and advanced modelling options

1.2.5.1 Enhancements

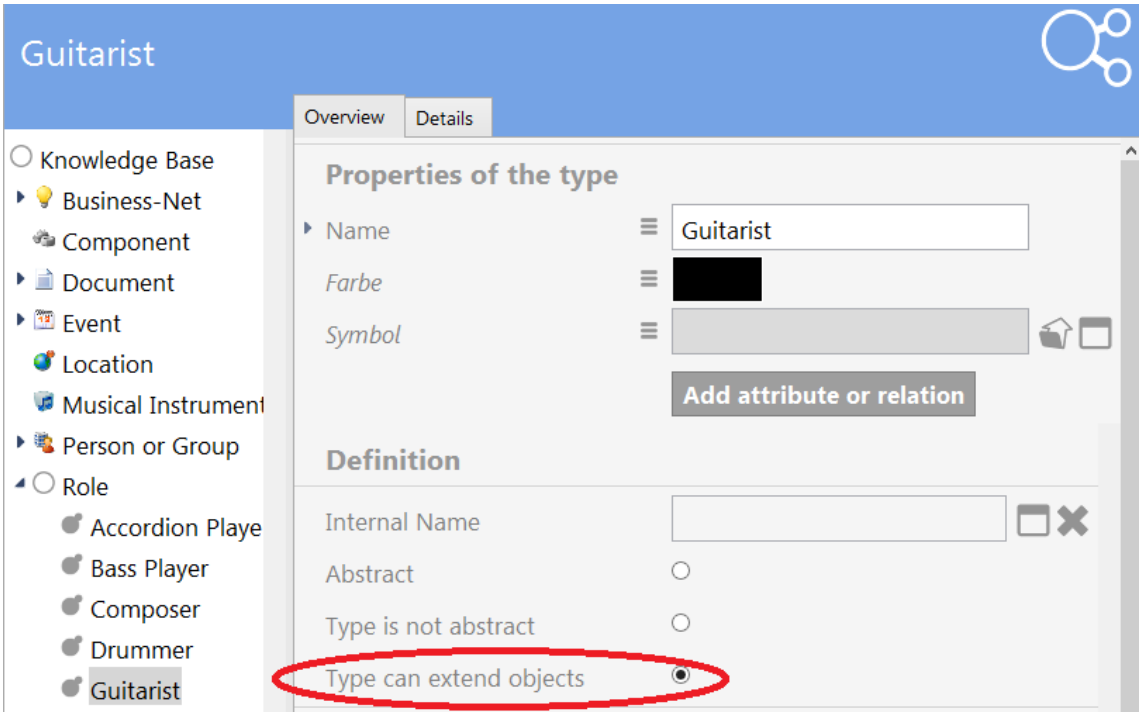
As a further means of modelling, k-infinity offers the possibility of enhancing objects.

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.

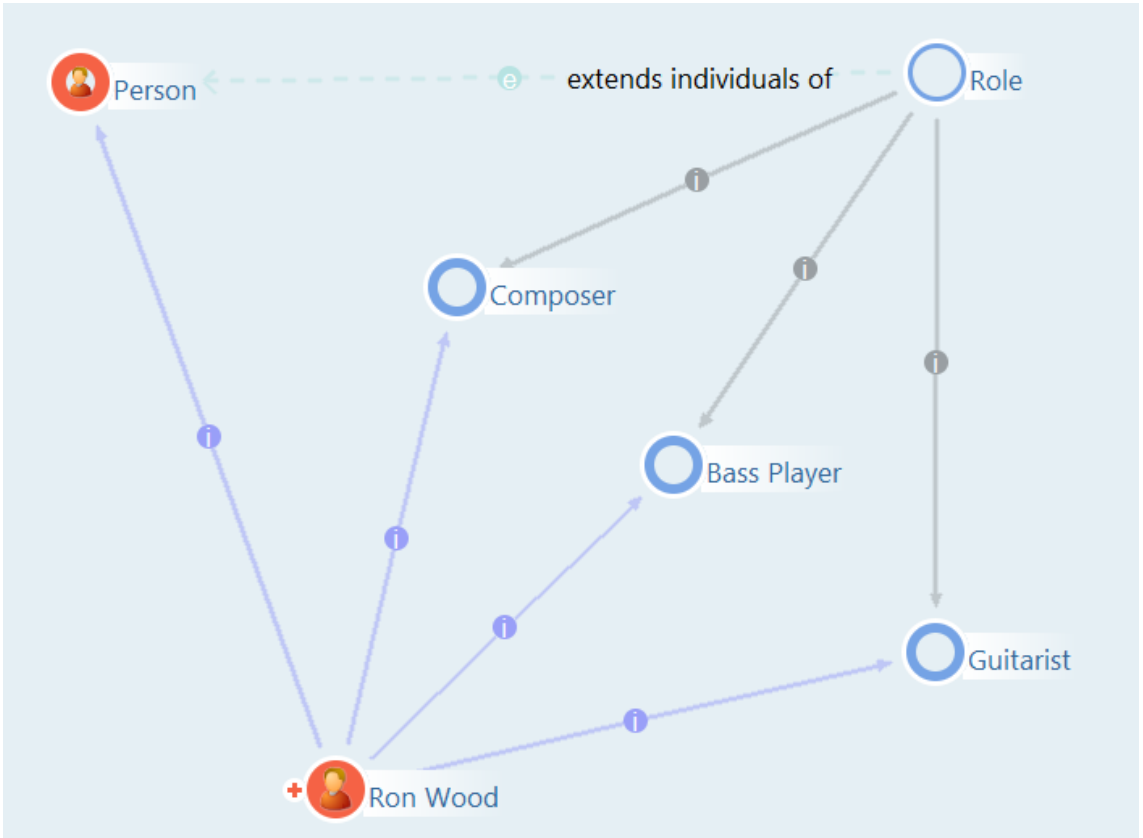


The fact that one person can play different roles in a knowledge network may be regulated via a special form of a object type. This may not contain any objects, but enhance objects from another object type (e.g. in this case "person"). For this purpose, the object type "role" is implemented into the 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":

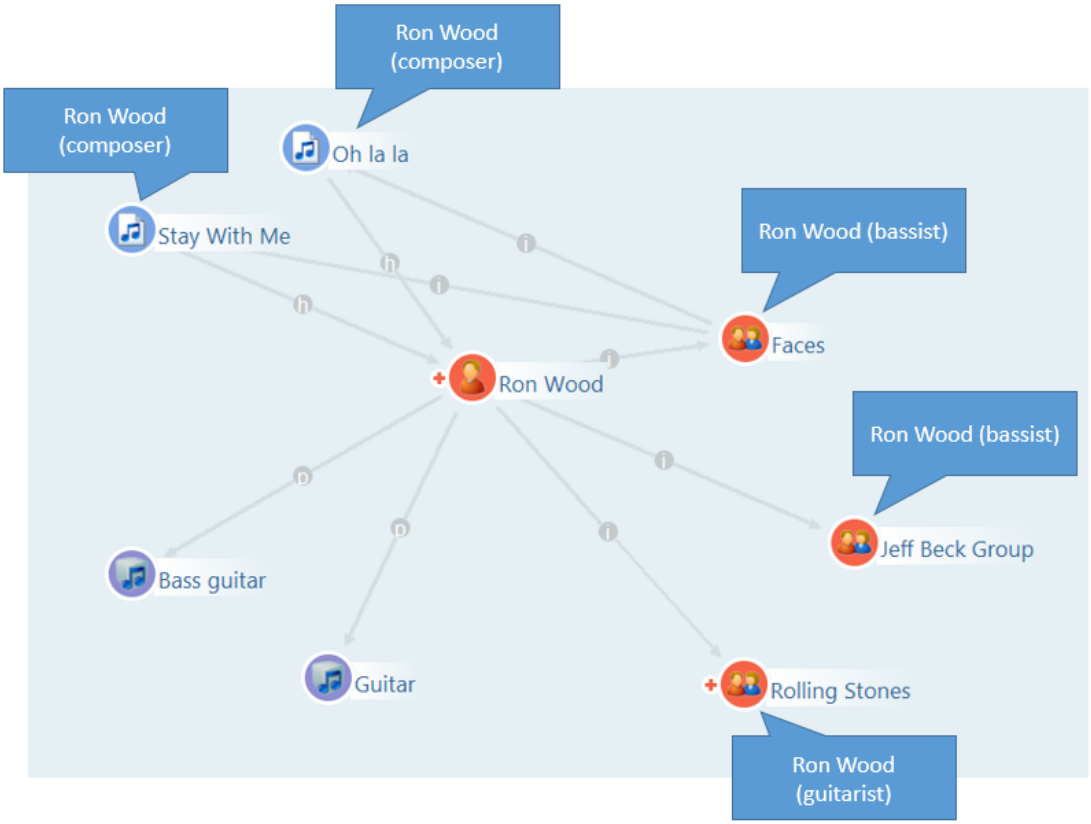


Enhancements are displayed in the graph editor as a blue dotted line:



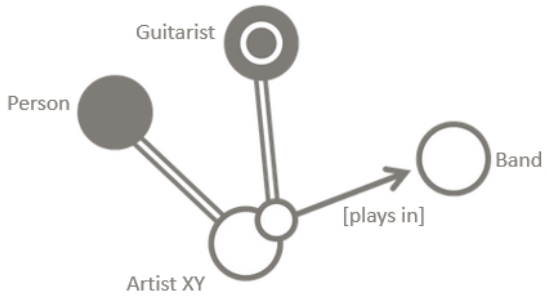
As a result of this enhancement we have achieved several things simultaneously:

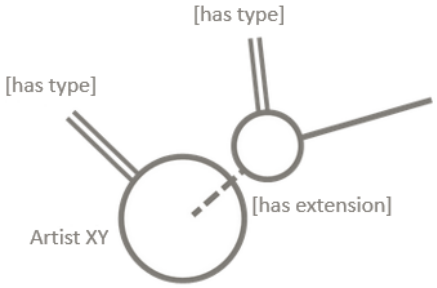
- We have formed sub objects for the persons (we can also imagine these as sections or - with persons - as roles). These sub objects may be viewed and queried individually. They are not independent, when the person is deleted the enhancement "guitarist" along with the relations to the bands or titles 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.

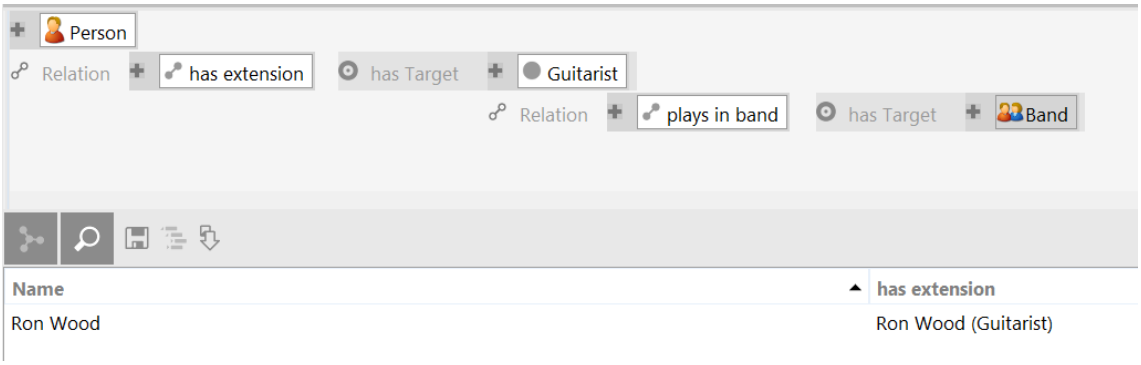
From a technical point of view, the enhancement is an independent object which is linked to the core individual by means of the system relation "has enhancement" or inverse "enhanced individual". Its type (system relation "has a type") forms the enhancement type.





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".



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

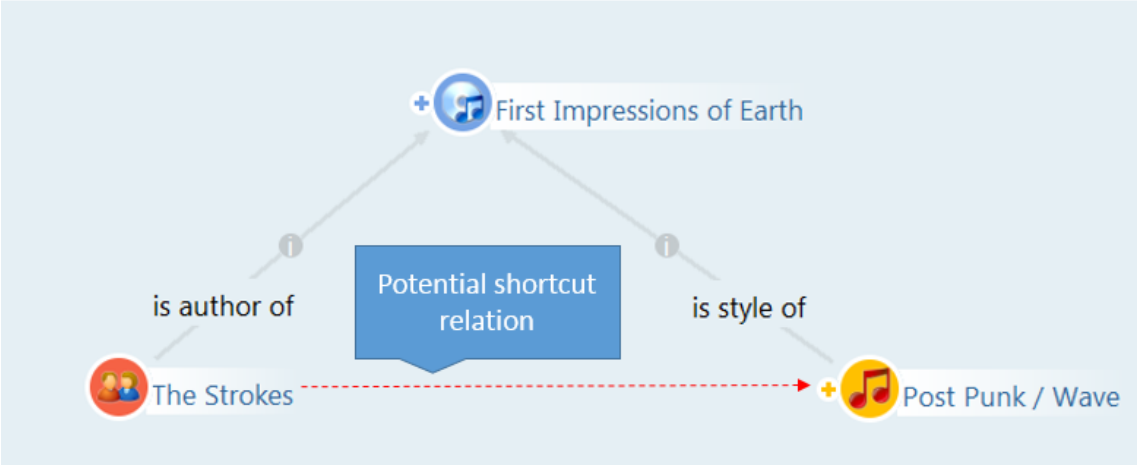
"outdoor event", for example, still the property "floodlight available". When we have placed these properties in mix-in they may be queried like direct properties in the events.

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".

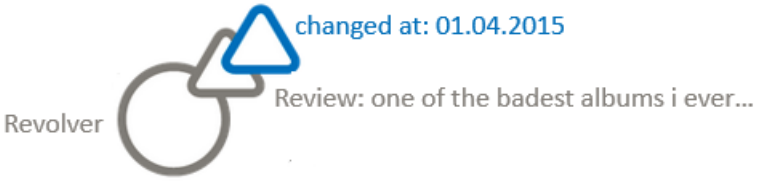
In the queries the shortcut relation can be used like any other relation as well.

In the current version of k-infinity 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

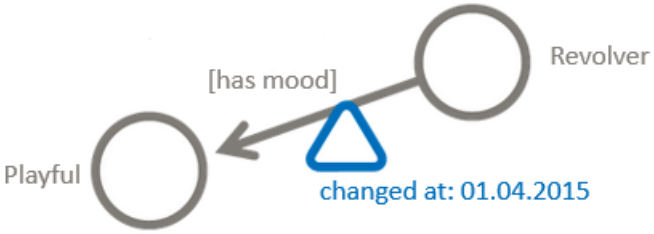
In Chapter 2.1 properties of a lesser complexity in object types for objects were defined. For example, users can add or edit contents to the music knowledge network which we are treating here as an example via a web application. It should, however, be noted which information was changed from whom and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

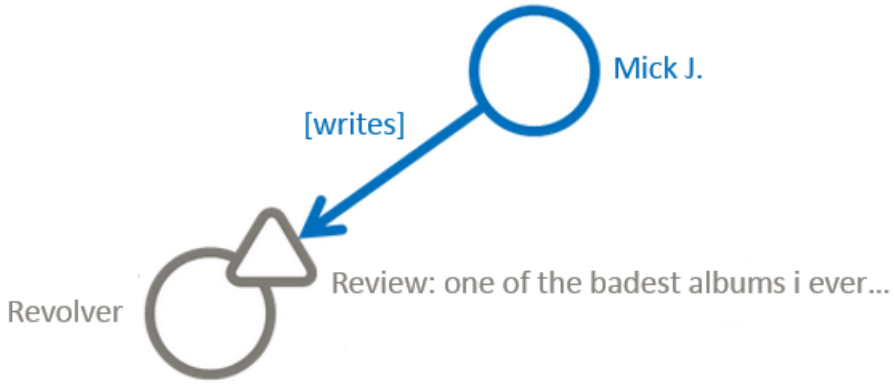


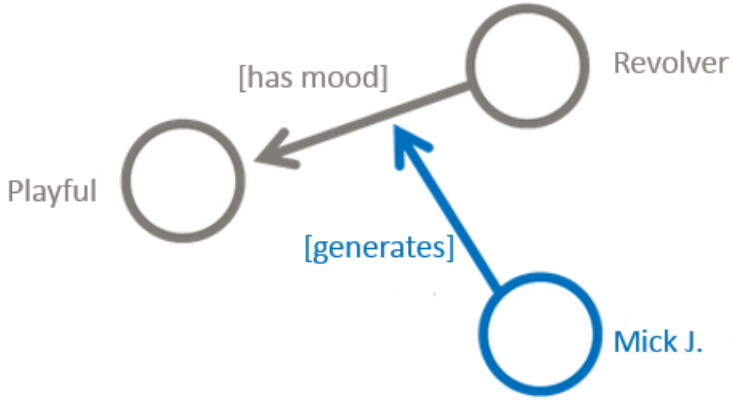
Attributes to relations:

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.

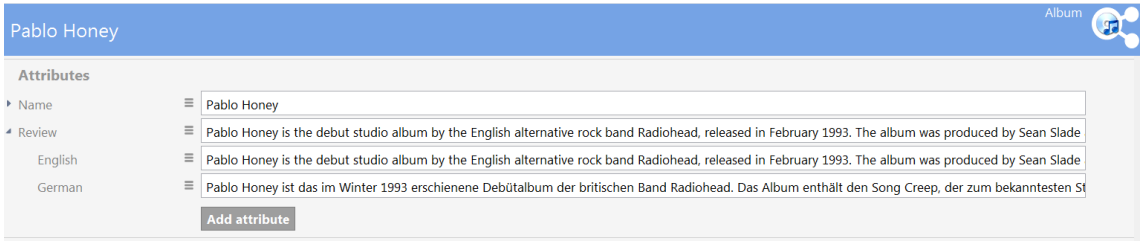
Modelling of meta properties may, of course, also be realised by implementing additional objects. In the last example the fact that the band performed at a festival enabled an object of the type "performance" to be modelled. A significant difference is that in the meta model the primary information can 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:

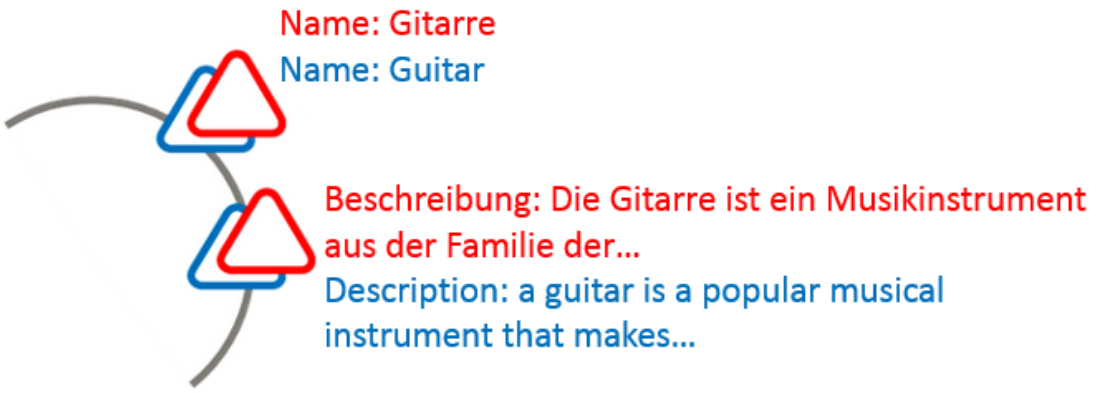


With data file attributes several images (e.g. with labels in other languages) may be uploaded analogically. In the case of selection attributes all selection options are deposited in the attribute definition; here it doesn't matter in which language the selection for the specific object is made.

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 k-infinity 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:



In k-infinity 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.

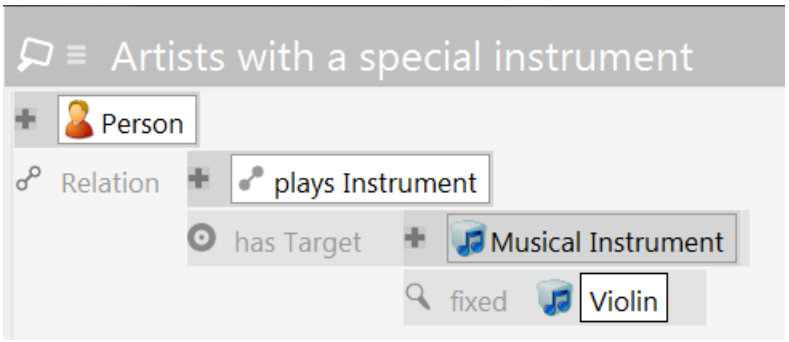
1.3 Search / queries

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

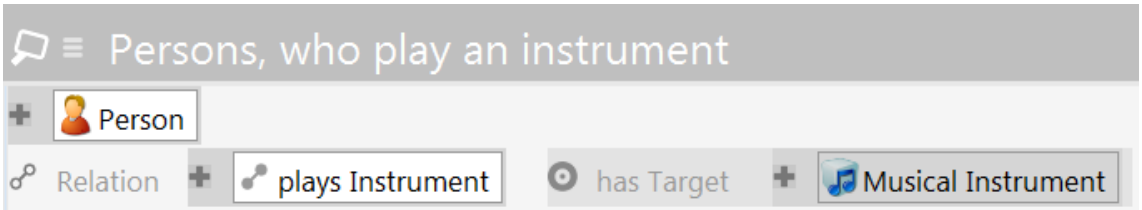
1.3.1 Structured queries

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.

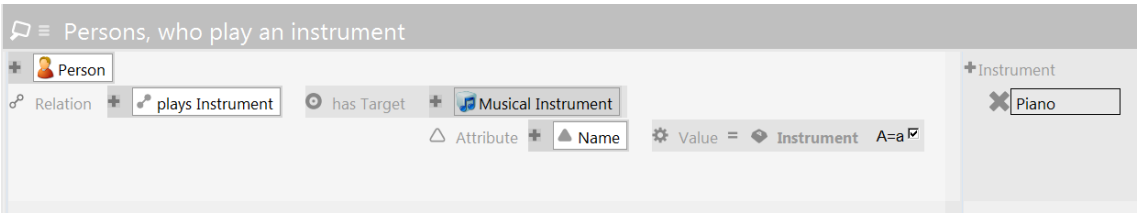


At first there is the type condition: objects of the type person are searched for. The second condition: the persons have to master an instrument. Third condition: this instrument has to be the violin.

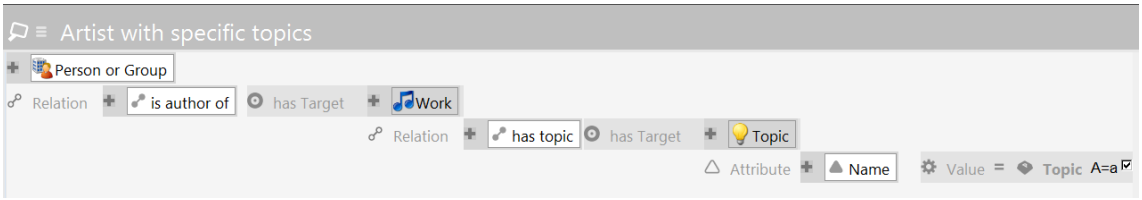
In the structured query the relation "plays an instrument", the type of the target of the relation and the value of the target "violin" form three different lines and thus also three search modes. The third condition that the instrument has to be a violin may also optionally be omitted. In the hit list 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:

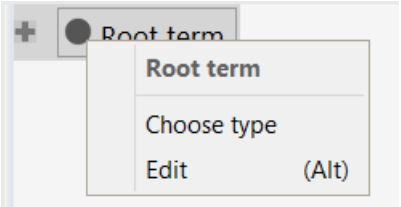


Slightly more complex example: persons or bands who deal with a certain issue in their songs (to be more exact in at least one). In this case you do not search for the name but the ID of the issue as the parameter - typical for searches, for example, which are queried via a REST service from the application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

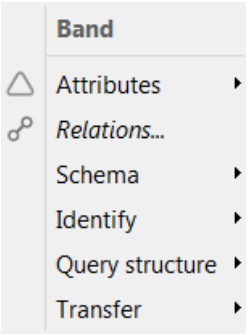
Interaction

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

Ein Hauptzweck der Strukturabfragen ist, in Anwendungen zu einem bestimmten Kontext Informationen zu liefern. Die Strukturabfrage aus dem letzten Abschnitt kann z.B. dem Endnutzer in einem Musikportal zu einem Thema wie Liebe, Drogen, Gewalt usw. eine Liste aller Künstler oder Bands generieren, die das Thema in ihren Liedern behandeln.

Dazu wird die Strukturabfrage in der Regel über ihren **Registrierungsschlüssel** in einen **REST-Service** eingebaut. Das Thema, für das sich der Nutzer gerade interessiert, geben wir der Abfrage mit seiner ID als Parameter mit.

Beispielszenario: Ein Nutzer sucht durch Eingabe eines Suchstrings nach einem Thema. Es liegt also keine ID vor, sondern nur einen String anhand dessen das Thema identifiziert werden soll. Dabei soll aber im Suchergebnis gleich angezeigt werden, welche Bands Songs zu dem Thema geschrieben haben. Zu diesem Zweck kann eine Strukturabfrage als eine Komponente in eine **Such-Pipeline** eingebaut werden - hinter die Abfrage, die den Suchstring verarbeitet.

Strukturabfragen sind unter anderem deshalb ein zentrales Werkzeug innerhalb von i-views, weil auch die Bedingungen für **Rechte und Trigger** mit Strukturabfragen formuliert werden: Angenommen es wird in einem Musikportal nur Künstlern und Bands erlaubt, Kommentare zu hinterlassen. Im Rechtesystem lässt sich entsprechend formulieren, dass nur Künstler und Bands, die zu einem bestimmten Thema mindestens einen Song geschrieben haben, zu diesem Thema Kommentare hinterlassen dürfen. Strukturabfragen können auch in Exporten benutzt werden, um zu bestimmen, welche Objekte exportiert werden soll.

Alle diese Verwendungen haben eines gemeinsam: wir sind nur an qualitativen, keinen gewichteten Aussagen interessiert. Das ist die Domäne der Strukturabfragen gegenüber den Such-Pipelines.

Nicht zuletzt sind Strukturabfragen auch Hilfsmittel für uns Knowledge-Engineers. Mit ihnen können wir uns einen Überblick über das Netz verschaffen und **Reports** sowie **To-do-Listen** zusammenstellen. Beispielfragen, die mithilfe von Strukturabfargen beantwortet werden können, sind:

- Zu welchen Themen gibt es wie viele Künstler/Bands?
- Müssen bestimmte Themen ausgebaut werden weil sich zu viele Relationen ansammelt haben, oder sollten umgekehrt spärlich besetzte Themen zusammenlegt oder

geschlossen werden?

Für diese Verwendung ist es sinnvoll, die Strukturabfragen in **Ordnern** organisieren zu können.

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 k-infinity 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.

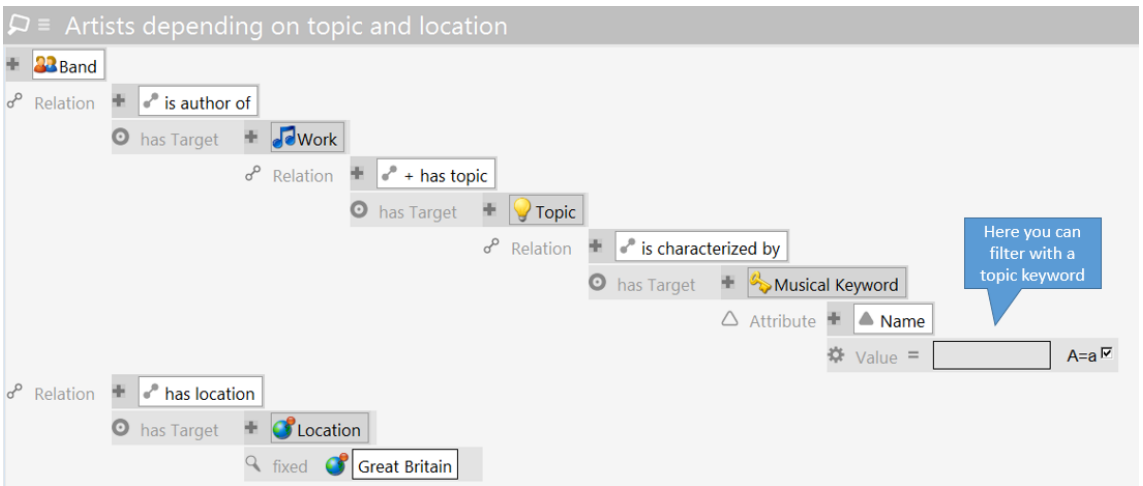
A structured query may be copied in order to create different versions, for example. Likewise there is the possibility of saving them in XML format, regardless of the network. The structured query may therefore be imported into another network. However, this is limited to versions of the same network,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

1.3.1.2 Setup 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 Several conditions

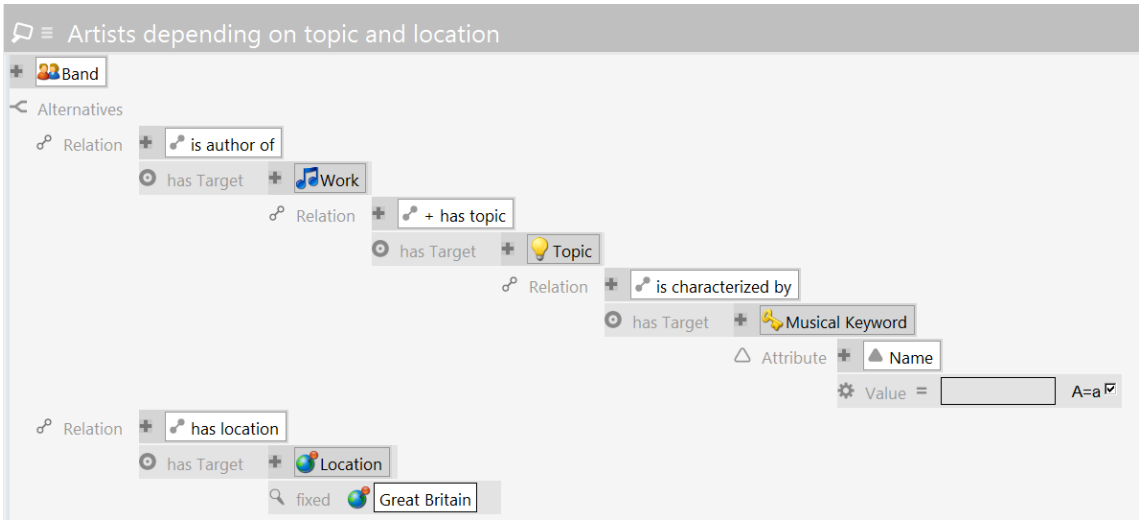
Condition chains may either be randomly deep or several parallel conditions may be expressed: additional conditions are added to any random condition element as a further branch:



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'. By clicking the symbol of the condition in the form of the relation "is the author of" you can select an alternative from the menu:



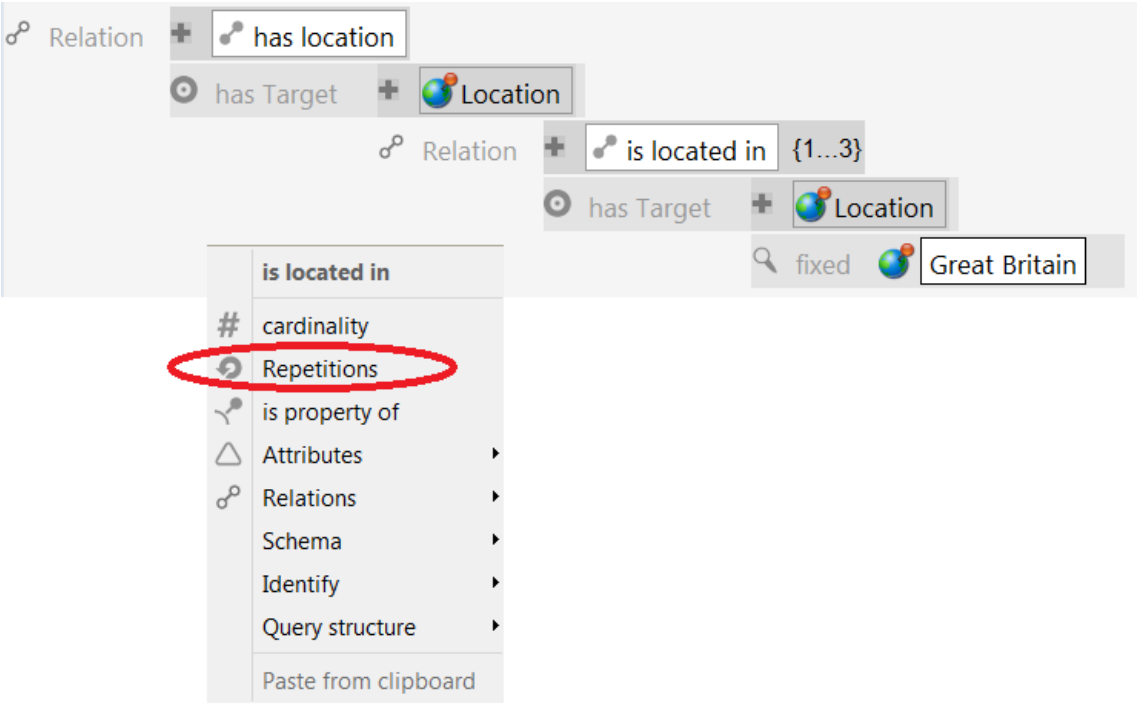
Alternative conditions - the band either has to be English or have songs on a certain subject

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 Transitive / repeating queries

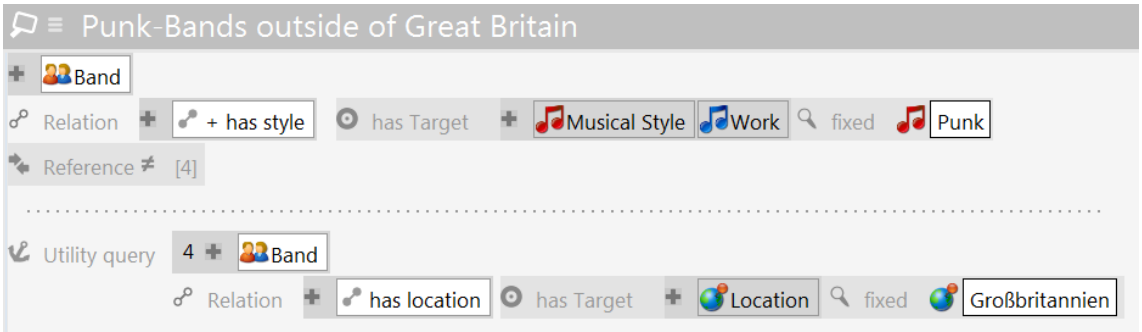
Let's assume the bands are assigned to either cities or countries within the network. Of these, in turn, it is known which cities are in which countries. In order to document these contents in the search it was possible to very simply expand the condition string: we were able, for example, to search 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 Negative 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 Equivalent conditions

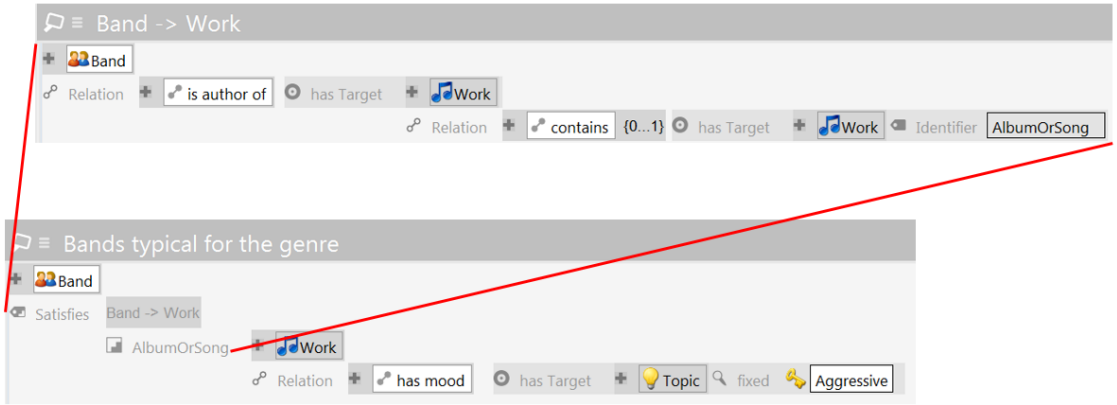
The reference allows references to be made to other conditions of the same query within a structured query:

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 Further options in setting up 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:



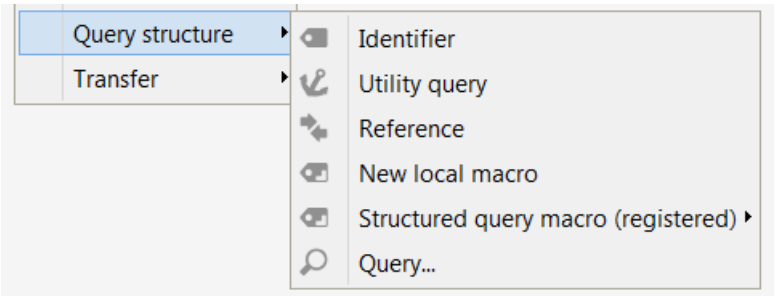
The objects which return those which are integrated into the structured query as macros have, of course, to match the condition with which they are linked from the point of view of their type. With the aid of the **identifier** function, the query (from the "invoking" query) can still be continued with 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":



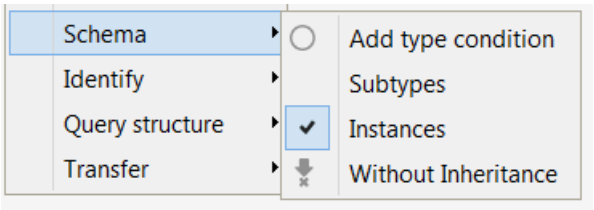
1.3.1.3 The conditions in detail

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.

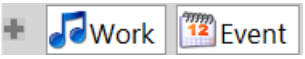
Alternatively, you can simply overwrite the text behind the type icon with the name of the object.

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:



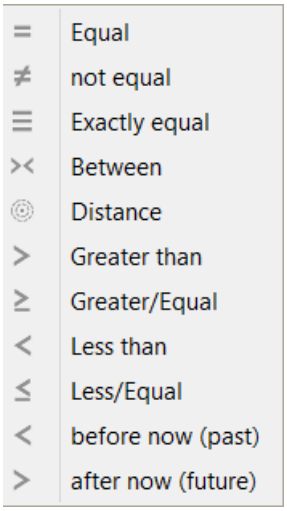
We can just search for types of objects instead of specific objects or both at the same time by checking the boxes "Subtypes" and "Instances" in the menu "Schema".

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 k-infinity offers us depends on the technical data type of the attribute:

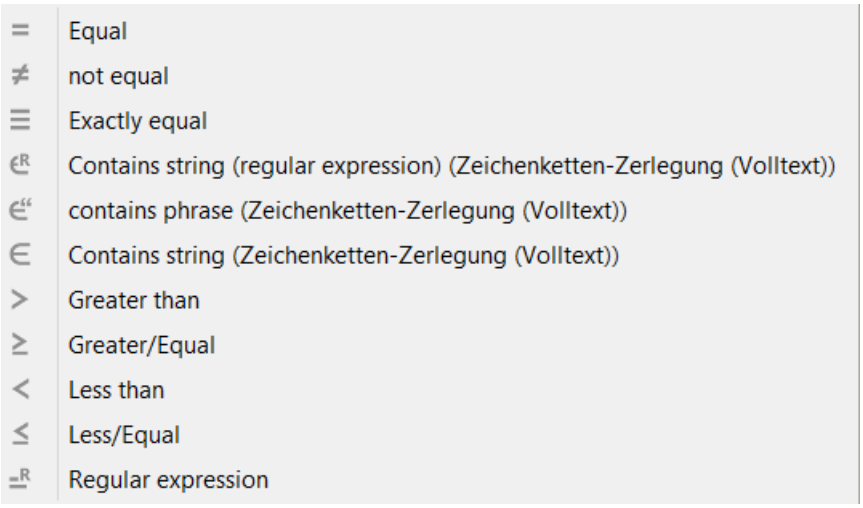


Comparison operators for dates and quantities

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.



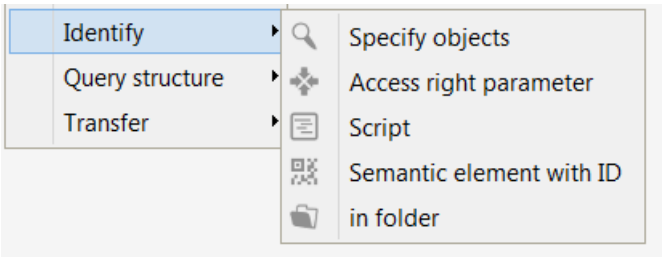
Comparison operators for character strings

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.

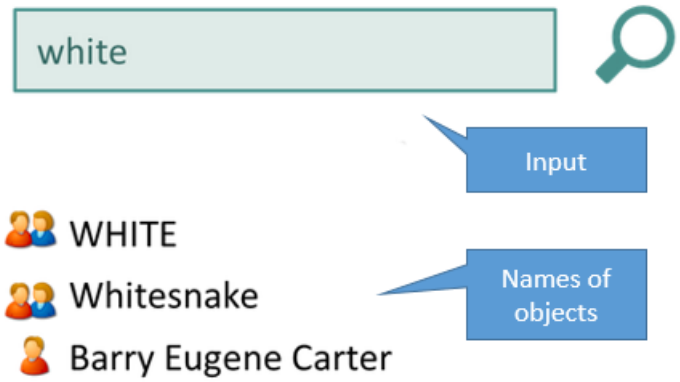
Script: the objects to be entered at this point are defined by the results of the script.

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).

1.3.2 Simple search / full-text search

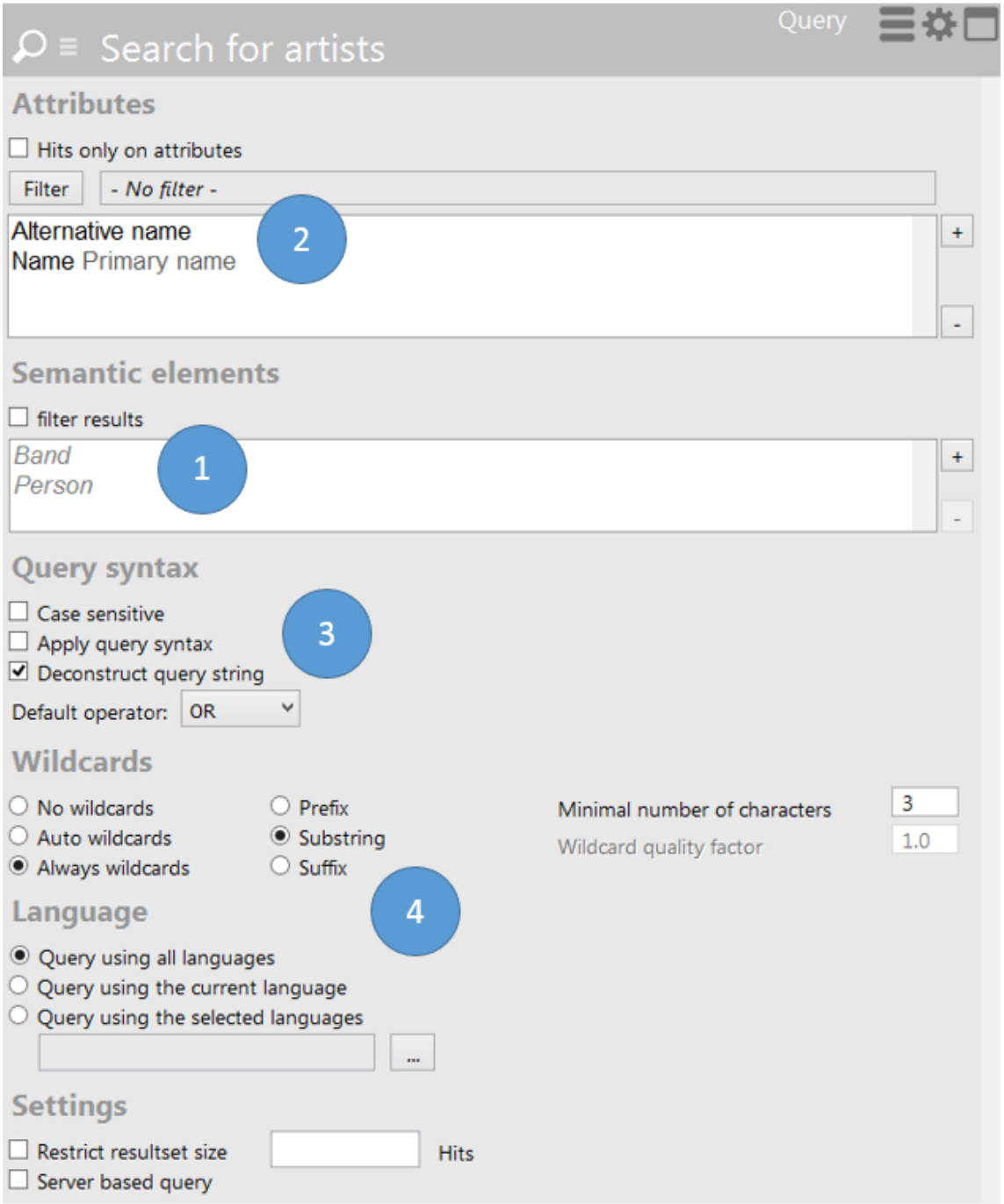
Processing the search queries of users may be carried out with or without interaction (e.g. with type-ahead suggestions). The starting point is, in any case, the character string entered. In configuring the simple search we can now define with which objects and in which attributes we search according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



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 - settings in detail

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:

- **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.

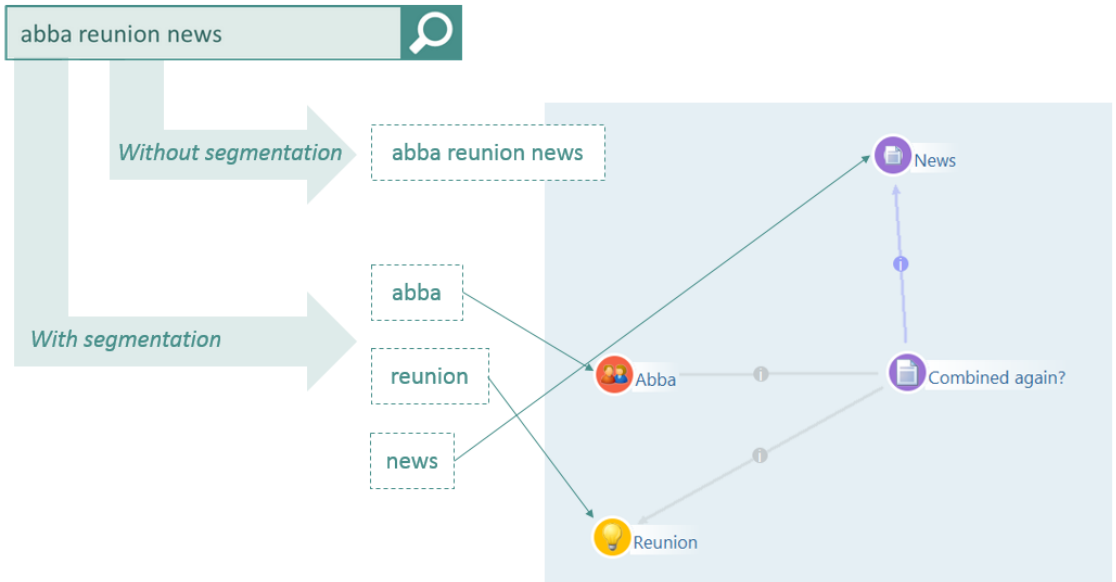
Translated attributes: in the case of translated attributes we can neither select a translation, nor have the language dynamically defined. Search for multilingual attributes, then in the active language or in all languages, depending on whether the option "in all languages" is checked.

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 k-Infinity server will be disburdened.

1.3.2.2 Multiple-word search input

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 Full-text search and indexing

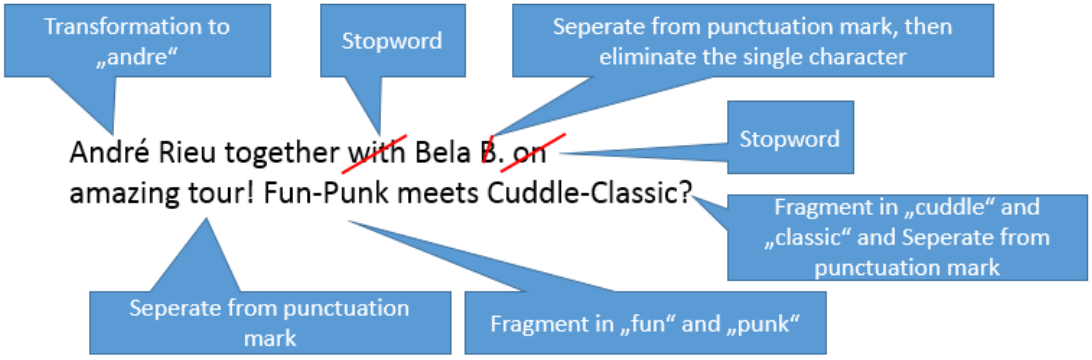
If we want to view or search through longer texts word by word, e.g. description attributes we recommended the use of full-text index. What does something like that look like?

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 k-infinity 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 k-infinity, 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:



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.:

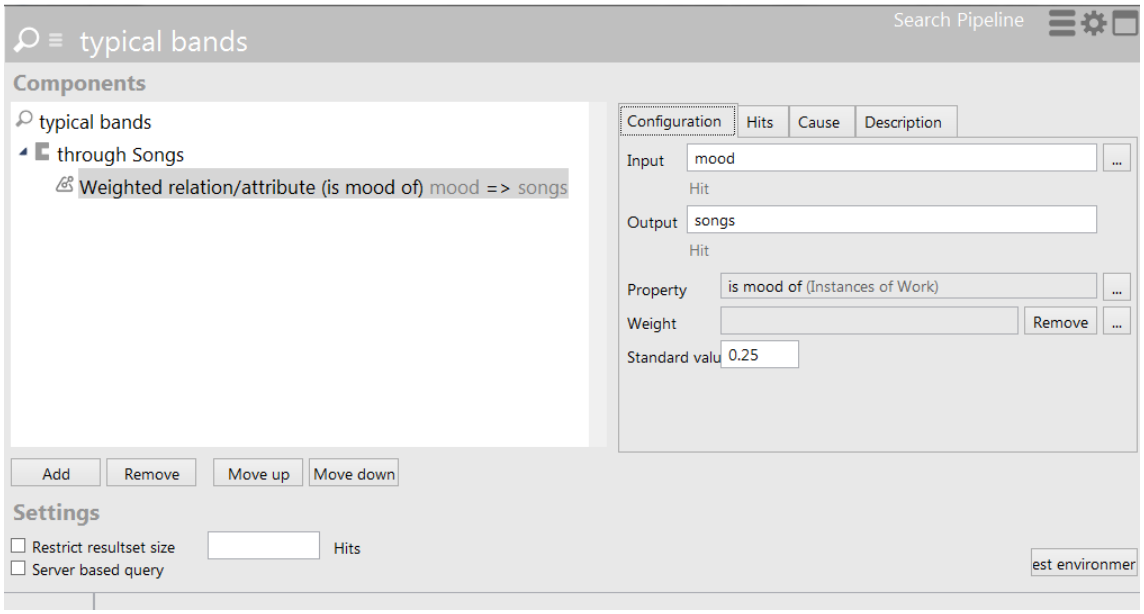
- traversing the network and thus determining the weighting
- performing structured queries and simple queries
- compiling hit lists

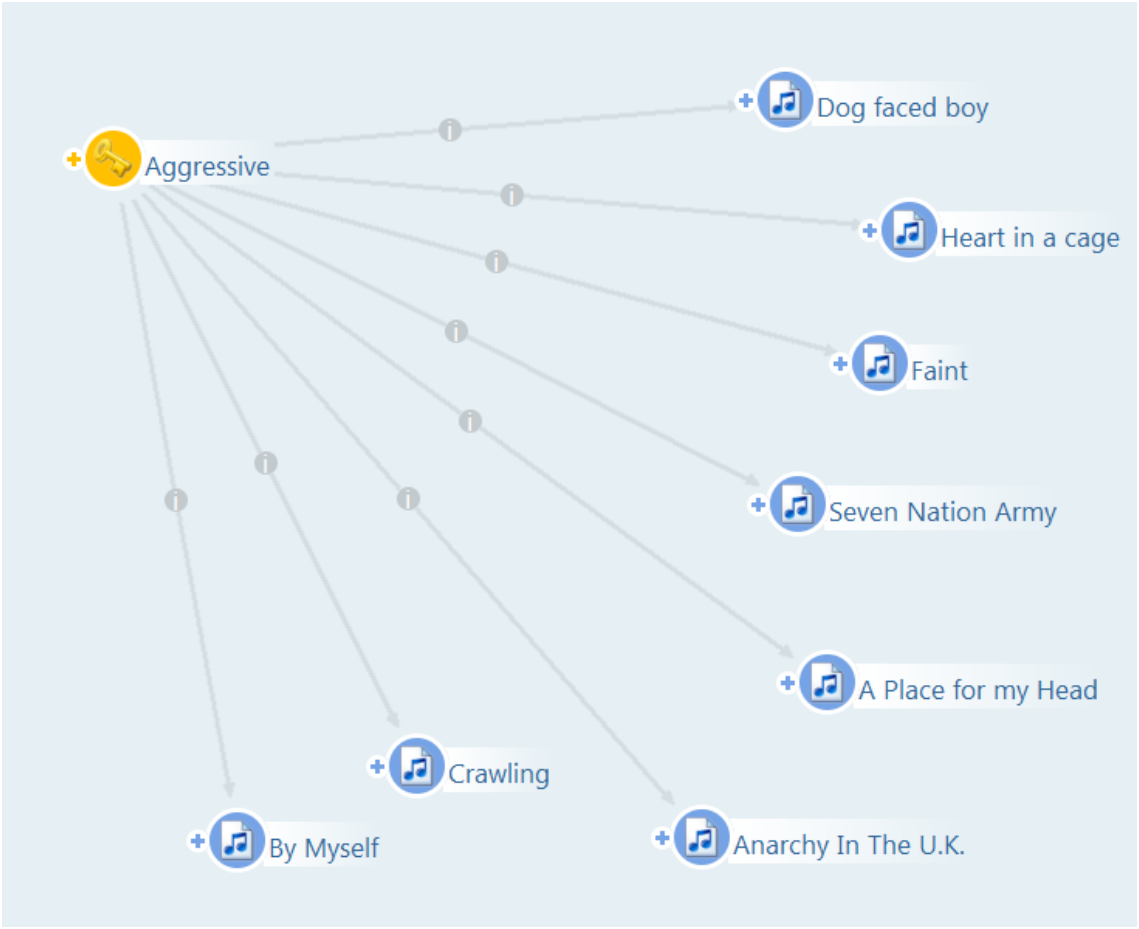
Every query step produces a query output (usually a number of objects). This query output may, in turn, be used as input for the following components in the pipeline.

Example

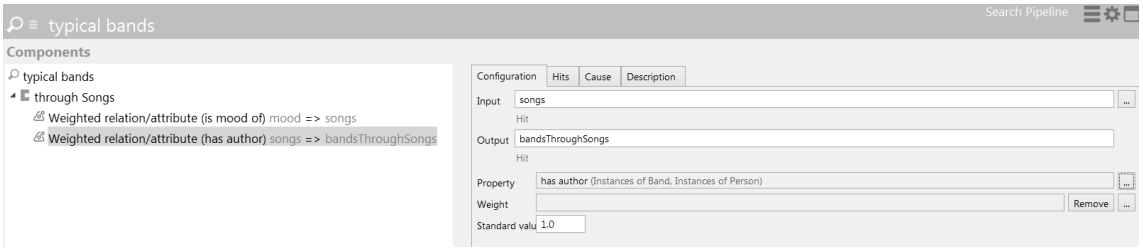
Let us assume that songs and artists from our musical network are characterised with tags named 'moods'. Based on a certain 'mood' we now want to find which bands best represent this mood.

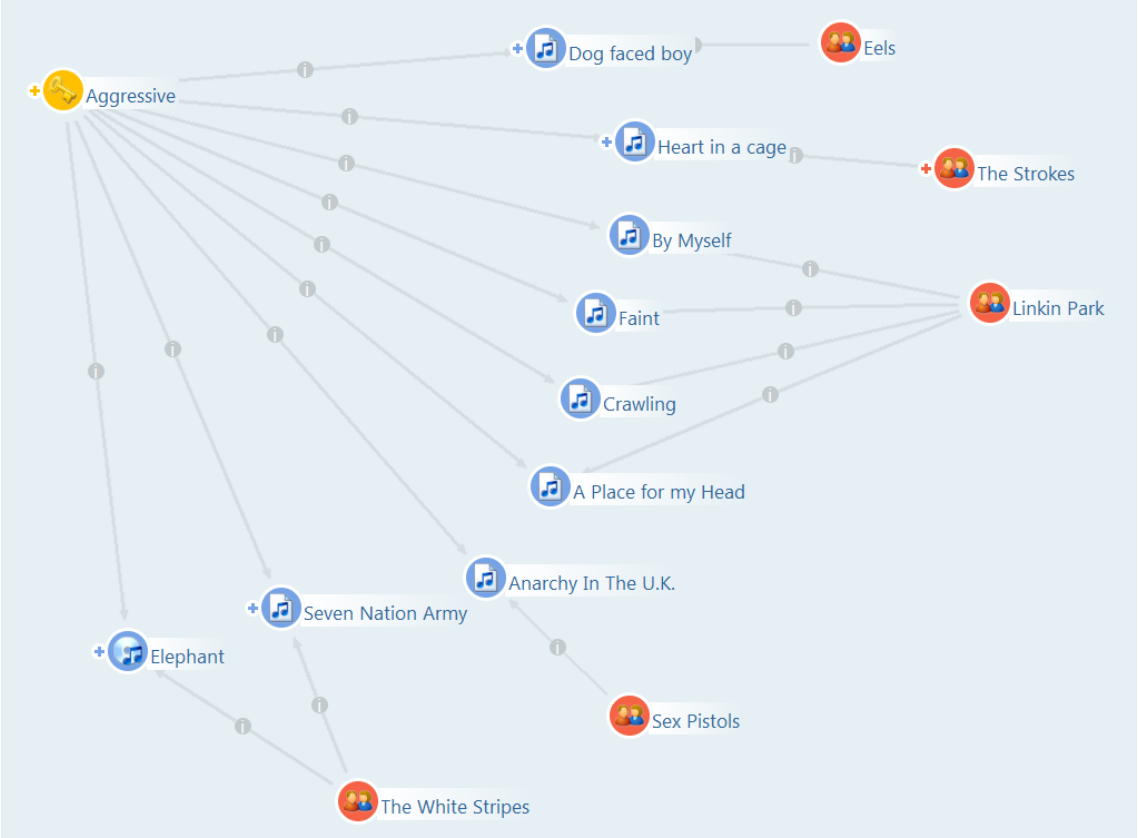
Step 1 of our search pipeline goes from a starting mood (in this case "aggressive") via the relation *is mood of* to the songs which are assigned to the mood 'aggressive':





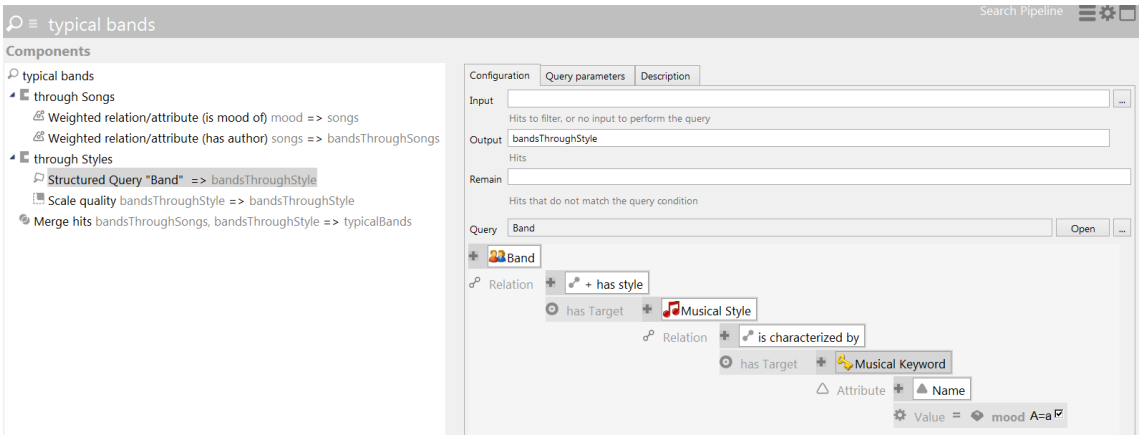
In the second step we go from the number of songs detected in the 'mood' searched for to the corresponding bands via the relation *has author*:



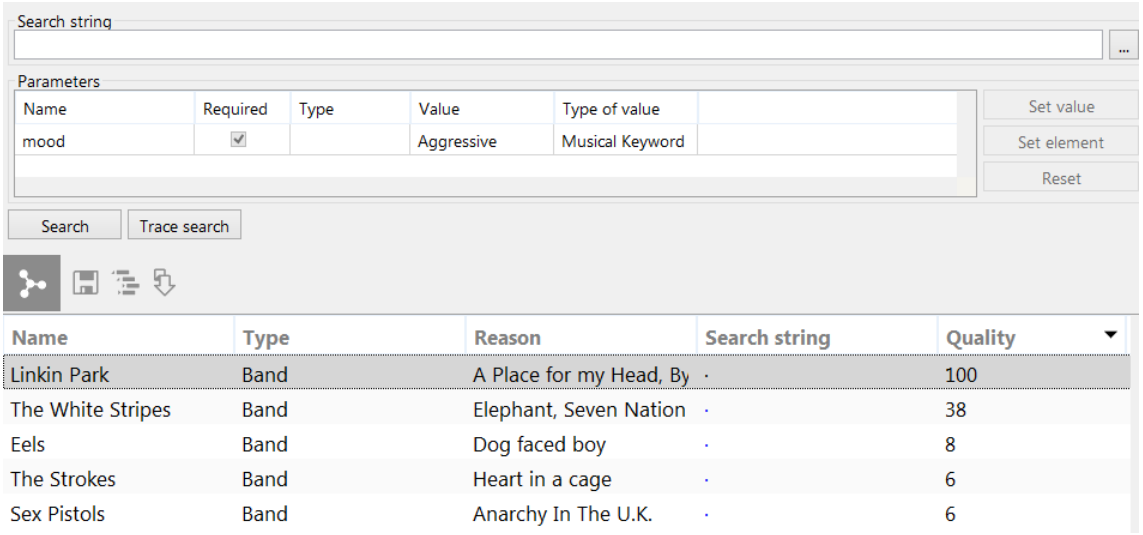


Now we would like to pursue a second path: from the starting point ’mood’ "aggressive" to the musical directions which are characterised by aggressiveness.

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:



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



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 Compilation 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.

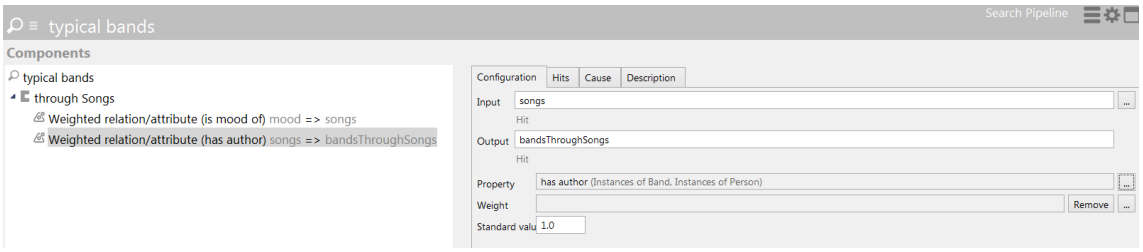
Grouping with blocks serves only to provide an overview, e.g. for the compilation of several components in a functional area of the search pipeline.

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.

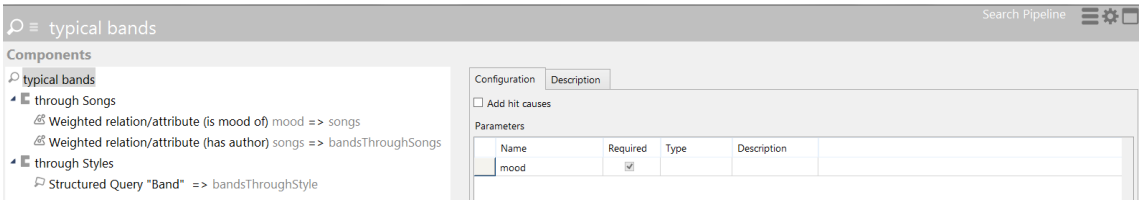
Configuration of a component

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":



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.



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.

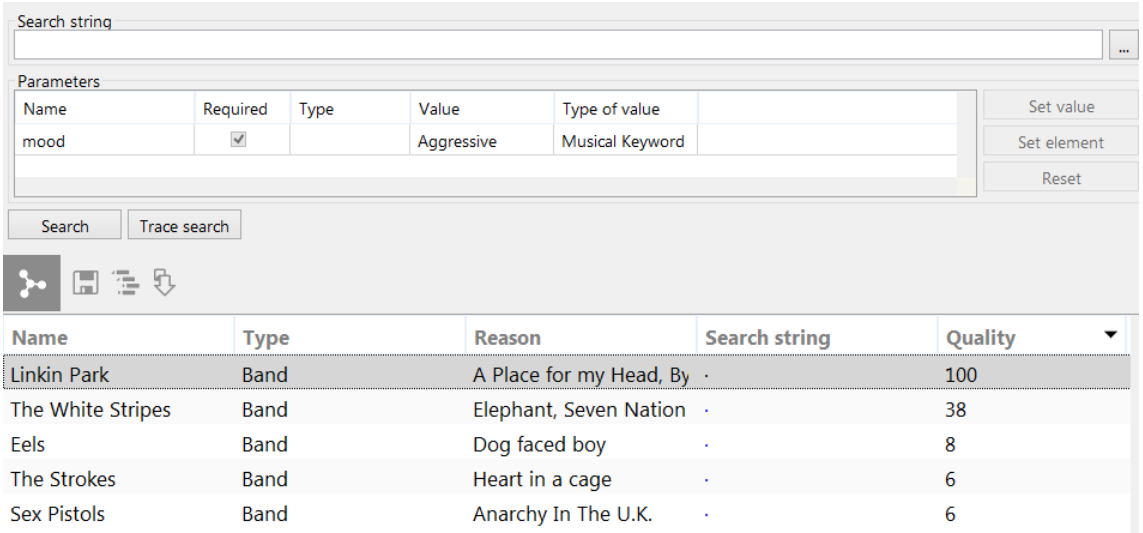
Output: we can stop the search at any stage and return the input. This component is also useful for testing the search pipeline.

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

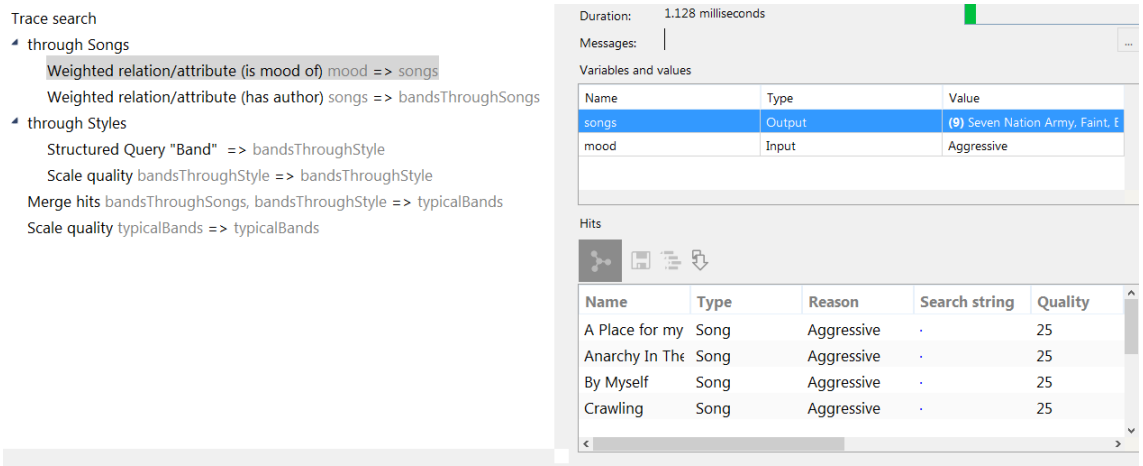
the 'legibility' of the search pipeline.

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.



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.



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 individual components

Gewichtete Relationen und Attribute

Ausgehend von semantischen Objekten können wir mit diesem Schritt den Graph traversieren und Relationsziele oder Attribute aufsammeln. Dazu müssen wir den Typ der Relation oder des Attributs angeben.

Achtung: Ausgabe sind nur noch die aufgesammelten Ziele, nicht mehr die Ausgangsmenge. Wenn diese angezeigt werden sollen, müssen wir anschließend die Eingangs- und die Ausgangstreffer zusammenfassen.

Bei der Traversierung einer Relation kann die Gewichtung der Treffer beeinflusst werden. Nehmen wir an, wir wollen den „Ausgangs-mood“ unserer Beispielsuche um „Unter-moods“ semantisch erweitern. Aber diese Indirektion soll sich in einem Ranking niederschlagen: Verbindungen zu Bands, die über die Untermoods laufen, sollen nicht so stark zählen wie Verbindungen über einen Ausgangsmood. Zu diesem Zweck können wir das Relation-Entlanggehen mit einem pauschalen Wert - z.B. 0,5 - belegen, und mit dem Eingangsgewicht verrechnen, beispielsweise multiplizieren. Dann zählen die in diesem Schritt hinzugefügten Untermoods nur halb so viel wie die direkten.

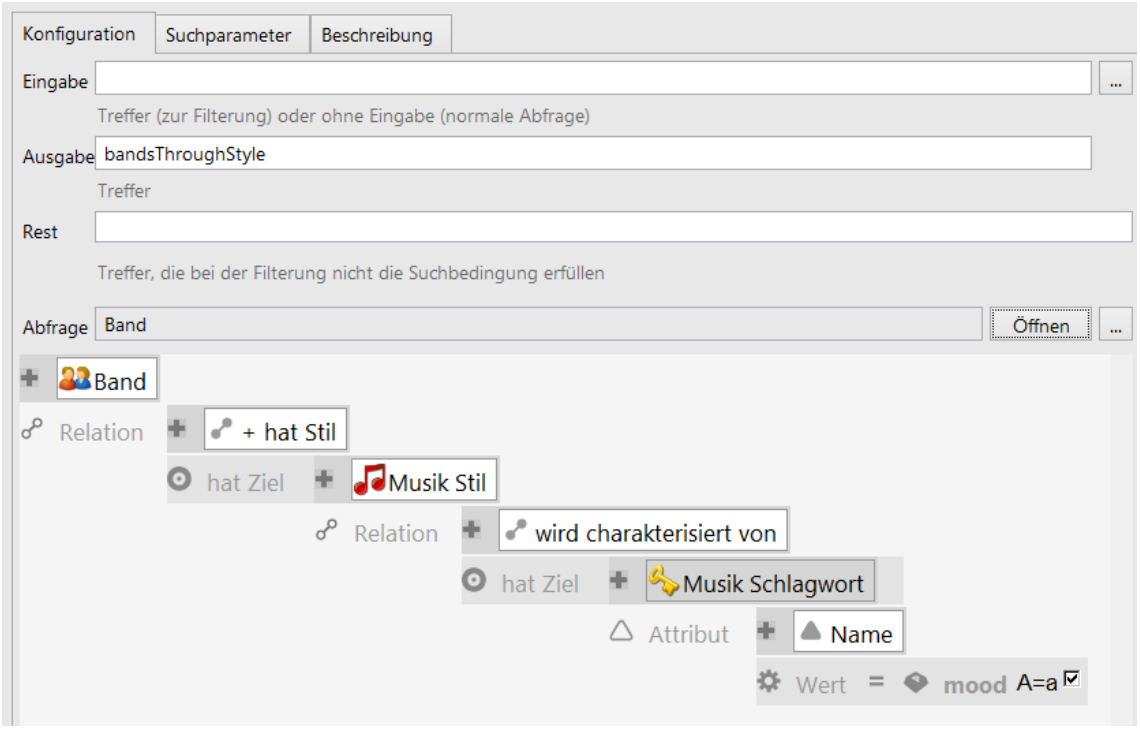
Statt eines pauschalen Gewichts für das Entlanggehen der Relation zu vergeben, können wir den Wert auch aus einer Metaeigenschaft des Basistyps Fließkommazahl der ausgewählten Relation auslesen. Falls dieses Attribut nicht vorhanden ist oder keines konfiguriert wurde, wird der Standardwert verwendet. Der Wert sollte zwischen 0 und 1 liegen. Die Treffererzeugung kann detaillierter konfiguriert werden: Bei Relationen kann optional auch für die Relationsquelle (statt für das Relationsziel) ein neuer Treffer erzeugt werden.

Wenn eine Relation als Eigenschaft ausgewählt wurde und Treffer für Relationsziele erzeugt werden, können wir optional auch die Relation transitiv verfolgen. Bei jedem Schritt verringert sich der Qualitätswert, bis der angegebene Schwellwert unterschritten wird. Falls ein Objekt mehr Relationen hat als bei maximaler Fanout angegeben, werden diese Relationen nicht verfolgt. Je höher der Dämpfungsfaktor ist, desto stärker wird der Qualitätswert verringert.

Strukturabfrage

Mit Strukturabfrage-Komponenten können wir entweder semantische Objekte suchen / von einer bestehende Menge zu anderen Objekten gehen (wie mit der gewichteten Relation) oder eine Menge filtern.

Wenn wir Objekte suchen, leiten wir unsere Ausgangsmenge von Treffern aus einem der vorhergehenden Schritte über den Parameternamen in die Suche ein. (Allgemein: Innerhalb der Expertensuche können Variablen der Such-Pipeline wie z.B. search-String über Parameter referenziert werden). Die Eingabe bleibt in diesem Fall leer.



Zur Filterung geben wir dagegen als Eingabe eine Menge von Objekten an. In der Ausgabe sind alle Objekte enthalten, auf die die Suchbedingung zutrifft. Objekte, die nicht zur Suchbedingung passen, können optional in einer weiteren Variable (Rest) gespeichert werden.

Wir können die Strukturabfrage entweder direkt in der Komponente ad hoc definieren oder eine bestehende Strukturabfrage verwenden.

Achtung: Wenn eine bestehende Suche ausgewählt wurde, wird keine Kopie angelegt, Änderungen, die wir an für Zwecke der Such-Pipeline an der Strukturabfrage vornehmen, ändern sie auch für alle anderen Verwendungen.

Abfrage

Mit der Komponente „Suche“ können einfache Suchen, Volltextsuchen und andere Such-Pipelines ausgeführt werden. Einfache Suchen und Volltextsuchen werden dabei mit einer Zeichenkette versehen, z.B. mit dem *searchString*: Das ist ein Parameter, der in allen Such-Pipelines zur Verfügung steht um die Nutzereingabe zu verarbeiten. Die Treffermenge der aufgerufenen Suche belegt die Ausgabe dieser Komponente.

Über das Einbinden von Such-Pipelines in anderen Such-Pipelines können wir häufiger auftretende Teilschritte ausfaktorisieren. Anderen Such-Pipelines können mehrere Parameter übergeben werden und ganze Treffermengen übergeben. Mit eingebundenen Such-Pipelines können wir auch mehrere Parameter austauschen, d.h. wir können in der eingebundenen Suche auf jede Teilschrittausgabe der umgebenden Suche zugreifen und umgekehrt. Wenn wir auf "ausgewählte Parameter gehen, können wir diese auch umbenennen, falls wir z.B. eine Treffermenge aus der eingebundenen Suche nutzen wollen, aber den Namen schon verwendet haben. Oder wir können, um solche Konflikte zu vermeiden, nur einen Teil der Parameter aus der eingebundenen Suche übernehmen.

Treffer zusammenfassen

Mit dieser Komponente können wir mehrere Treffermengen aus unterschiedlichen vorangegangenen Schritten zusammenfassen. Folgende Methoden zur Zusammenfassung stehen zur Verfügung:

Vereinigungsmenge: Alle Treffer, die in mindestens einer der Mengen vorkommen, werden als Ergebnis ausgegeben

Schnittmenge: Nur Treffer, die in allen Mengen vorkommen, werden als Ergebnis ausgegeben.

Bei Vereinigungsmengen und Schnittmengen kommt es vor, dass ein semantisches Objekt in mehreren Treffermengen vorkommt und zu einem Gesamttreffer mit neuer Trefferqualität verrechnet werden muss. Hier stehen wieder die erwähnten Verrechnungsmethoden zur Verfügung.

Differenz: Eine der Treffermengen muss als Ausgangsmenge definiert werden. Von dieser werden die anderen Mengen abgezogen.

Symmetrische Differenz: Die Ergebnismenge besteht aus den Objekten, die nur in genau einer Teilmenge enthalten sind (= alles außer dem Schnitt, bei zwei Mengen).

Es können drei unterschiedliche Arten von Gesamttreffern erzeugt werden. Die Auswahl ist insbesondere dann relevant, wenn die Teiltreffer zusätzliche Informationen tragen.

- Einheitliche Treffer erzeugen, ursprüngliche Treffer als Ursache merken: Es werden neue Treffer erzeugt, die den ursprünglichen Treffer als Ursache enthalten.
- Ursprüngliche Treffer erweitern: Der ursprüngliche Treffer wird kopiert und erhält einen neuen Qualitätswert. Falls mehrere Treffer für dasselbe semantische Objekt vorliegen, wird ein beliebiger Treffer gewählt.
- Einheitliche Treffer erzeugen: Es wird ein neuer Treffer erzeugt. Die Eigenschaften des ursprünglichen Treffers gehen verloren.

Teiltreffer zusammenfassen

Bei der Einzelverarbeitung ist es öfters notwendig, eine Gesamtmenge aus Teiltreffern zu erzeugen. Dieses ermöglicht die Komponente „Teiltreffer zusammenfassen“. Diese fasst alle Treffer einer oder mehrerer Teiltreffermengen zusammen. Der Unterschied zu Treffer zusammenfassen liegt darin, dass die Zusammenfassung erst am Ende erfolgt, nicht für jede einzelne Teiltreffermenge. Dies ist insbesondere bei der Berechnung der Qualität relevant, da Treffer zusammenfassen z.B. bei Median falsche Werte liefern würde.

Skript

Eine Such-Pipeline kann ein Skript (JavaScript oder KScript) enthalten. Dieses kann auf die Variablen der Such-Pipeline zugreifen. Außerdem kann ein Skript mehrere Parameter an die Such-Pipeline übergeben. Das Ergebnis des Skripts wird als Ergebnis der Komponente verwendet.

Die JavaScript-API sowie KScript sind in separaten Handbüchern beschrieben.

Qualität aus Attributwert übernehmen

Für Treffer können wir den Qualitätswert aus einem Attribut des semantischen Objekts übernehmen. Falls das Objekt nicht genau ein solches Attribut besitzt, wird der Standardwert verwendet. Der Wert sollte zwischen 0 und 1 liegen.

Gesamtqualität aus gewichteten Qualitäten berechnen

Um die Qualität eines Suchtreffers anzupassen, kann es hilfreich sein, aus einzelnen Teilqualitäten einen Gesamtwert zu berechnen. Die Qualitäten müssen dabei als Zahlenwerte vorliegen. Aus diesen Werten wird eine neue Gesamtqualität berechnet.

Gesamtqualität einer Treffermenge berechnen

Aus den einzelnen Qualitätswerten einer Treffermenge kann man eine Gesamtqualität berechnen.

Qualität beschränken

Treffermengen können wir auf Treffer beschränken, deren Qualitätswert innerhalb vorgegebener Schranken (Minimum und Maximum) liegen. Im Normalfall möchten wir Treffer, die unterhalb einer bestimmten Qualitätsschwelle liegen, ausfiltern.

Trefferanzahl beschränken

Falls die Gesamtzahl einer Treffermenge begrenzt werden soll, können wir die Komponente „Trefferanzahl beschränken“ hinzufügen. Mit der Option „Treffer gleicher Qualität nicht zerteilen“ verhindern wir, dass bei mehreren Treffern mit gleicher Qualität eine willkürliche Auswahl erfolgt, um die Gesamtzahl einzuhalten. Wir erhalten dann mehr Treffer als vorgegeben.

Für einige sehr spezielle Fälle können wir die Treffer auch zufällig auswählen lassen, z.B. wenn wir eine große Menge an Treffern gleicher Qualität haben und eine Vorschau generieren wollen.

Qualität skalieren

Die Qualitätswerte einer Treffermenge kann skaliert werden. Es wird eine neue Treffermenge mit skalierten Qualitätswerten berechnet. Die Berechnung erfolgt ein zwei Schritten:

1. Die Qualitätswerte der Treffer werden begrenzt. Die Grenzwerte können entweder festgelegt oder berechnet werden. Bei der Berechnung werden der minimale und der maximale Wert der Treffer ermittelt. Falls die Grenzen vorgegeben werden und ein Treffer einen Qualitätswert außerhalb der Grenzwerte hat, wird der Wert auf den Grenzwert

beschränkt. Falls man solche Treffer entfernen will, muss man die Komponente Qualität beschränken vorschalten. Beispiel: Abbilden von Prozentwerten auf Schulnoten. 30% ist Durchschnitt, über 90% ist Highscore. Die Werte können innerhalb von 30% bis 90% linear skaliert werden.

2. Anschließend werden die Qualitätswerte linear skaliert. Treffer mit dem minimalen/maximalen Eingangswert erhalten den minimalen/maximalen skalierten Wert.

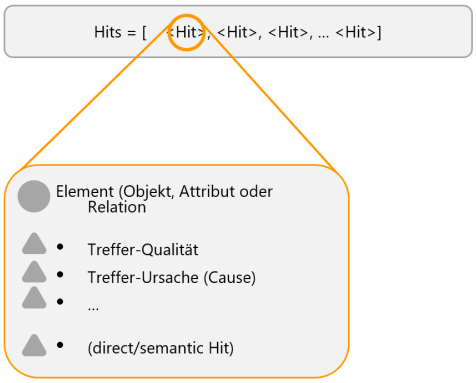
Trefferqualität berechnen

Mit Hilfe eines KPath-Ausdrucks wird für einen Treffer ein neuer Treffer mit berechneter Qualität erzeugt. Der KPath-Ausdruck wird ausgehend von der Eingabe berechnet.

1.3.4 Inhaltsmodell "Hit"

Damit für Suchabfragen sowohl Qualität als auch Ursachen mitverarbeitet und transportiert werden können, gibt es das Inhaltsmodell des Typs „Hit“. Ein „Hit“ kann als Container verstanden werden, der das Element inklusive mehrerer Eigenschaften zusammenfasst und temporär für den Kontext zur Verfügung stellt. Die enthaltenen Eigenschaften sind beispielsweise berechnete Treffer-Qualität, Treffer-Ursache, ChangeLog-Eintrag etc.

In Such-Pipelines stehen die Inhaltsmodelle „Hit“ und „Hits“ zur Verfügung. Der Typ „Hits“ ist dabei ein Array aus mehreren „Hit“-Elementen:



Metaattribute der Hits

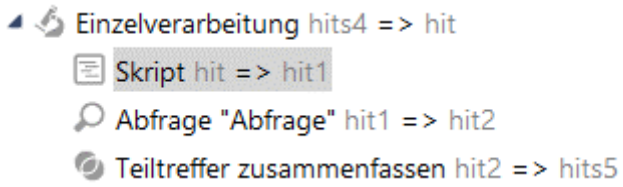
Außer dem semantischen Element werden in einem Hit folgende Metaattribute transportiert:

- **Treffer-Qualität (quality):** Kann in einer Such-Pipeline durch das Setzen einer Qualität einen Wert zwischen 0 und 1 annehmen; die Treffer einer Strukturabfrage enthalten per Default den Wert 1
- **Treffer-Ursache (cause):** Bezeichnet das Eingangs-Element, das zum Treffer geführt hat und um welchem Typ es sich handelt
- **Treffer-Ursache (snippet):** Bezeichnet den Inhalt bzw. den Suchbegriff, der zum Treffer geführt hat

Detaillierte Informationen zu den Metaattributen sind in der JavaScript-API dokumentiert.

Verwendung von Hits in Such-Pipelines

Wenn in einer Such-Pipeline eine Treffermenge mithilfe einer einfachen Abfrage verarbeitet werden soll, so ist aus Gründen der zum Array zusammengesetzten Treffermenge eine Einzelverarbeitung notwendig: Abfragen können einen einzelnen „Hit“ in Form eines Strings verarbeiten, jedoch keine „Hits“ (= Array). Die Umwandlung eines „Hit“ in einen String kann wiederum durch ein Skript erfolgen, das der einfachen Abfrage vorangestellt ist.



Beispiel-Skript für die Umwandlung eines Hit in einen String:

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

Verwendung von Hits in Tabellen

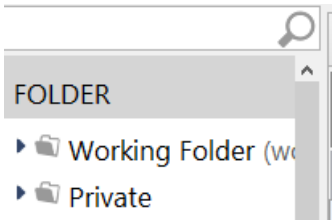
In der Spaltenelement-Konfiguration einer Tabelle steht die Option "Hits verwenden" zur Verfügung. Diese Option bestimmt, ob für die Anzeige von Suchergebnissen das gesamte Hit-Element (semantisches Element + Metaattribute) weitergereicht werden soll oder nur das semantische Element.

Verarbeitung von Hits in Tabellen per Skript

Wenn die Suchergebnisse per Skript weiterverarbeitet werden sollen, bestimmt die Option "Hits verwenden", ob das Suchergebnis wie ein Hit behandelt werden soll: Das Skript bekommt als JavaScript-Objekt entweder $k.SemanticElement oder $k.Hit zur Verarbeitung weitergereicht.

1.3.5 Searches 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 Full-text 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, k-infinity 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 be-

cause in the case of extensive fodlers (e.g. saved search results) the search for object names may take a very long time.

1.4 Folders and registration

Along with the objects and their properties we also create diverse other elements in a typical project: for example, we define queries and imports/exports or write scripts for special functions. We can organise everything which we create and configure in folders.

The folders are shared with all others who are working on the project. If we do not wish to do this we can deposit things in a private folder, e.g. 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 save objects manually by drag & drop for processing at a later date. To do this, we can simply push them into the folder using drag & drop and additionally there are operations for keeping output lists in folders. The collection of semantic objects merely keeps references on the objects: at that point in time when we delete one of these objects it will also be deleted in the collection.

Registration

Queries, scripts, etc. may be mutually invoked (a query may be setup in another query or in a script; conversely a script may be invoked from a search pipeline). For this purpose, there are registration keys with which can equip queries, import/export illustrations, scripts and even collections of semantic objects and structured folders to enable them to provide other configurations with functionality. The registration key has to be clear. Everything which a registration key contains will automatically be adopted in the folder "registered objects" or in a sub folder of this type.

Shift, copy, delete

Let us assume we have a folder named "playlist functions" in our project. This would perhaps contain an export, some scripts and a structured query "similar songs" which we want to use in a REST services. At that point in time when we give the structured query a registration key it will be adopted in the folder "registered objects" (paragraph "technology"). I.e. the structured query "similar songs" will appear in the folder "registered objects" under "query". It will also stay there when we remove it from our project sub-folder "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 and scripts may be contained in one or several folders simultaneously and must be in at least one folder. Not until we remove our query from the last folder will it actually be deleted. Only then does k-infinity also request a confirmation of the deletion action. The same applies to the removal of the registration key.

If we wish to delete the query in one step, regardless of in how many folders it is located, we

can only do this from the registry.

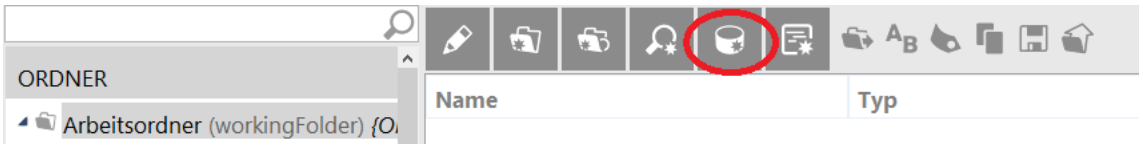
Folder settings

In the folder settings we can define quantitative limits for search outputs, folders and object lists (lists of the specific objects in the main window of the knowledge builder by selecting an object type on the left-hand side). Automatic query up to the number of objects specifies up to what number of objects the contents of the folders are shown without any further interaction by the user. If the limit set there is exceeded, the list will first remain empty and in the status bar the message "query not implemented" will appear. Carrying out a search without any input in the input bar shows all objects. At least until the second limit is reached: maximum number of query outputs, maximum number of outputs in lists of objects - in this instance high values - after these values no more output in fact, queries have to restrict, e.g. in lists of objects in which we also have the beginning of the name in the input box.

1.5 Import und Export

Mit den Abbildungen von Datenquellen können wir Daten aus strukturierten Quellen in i-views importieren und Objekte und ihre Eigenschaften in strukturierter Form exportieren. Die Quellen können Excel/CSV-Tabellen, Datenbanken oder XML-Strukturen sein.

Die Funktionen für den Import und Export decken sich größtenteils und sind daher alle in einem Editor verfügbar. Um auf die Funktionen für den Import und Export zugreifen zu können, muss zunächst ein Ordner (z.B. der Arbeitsordner) ausgewählt werden. Dort kann über die Schaltfläche "Neue Abbildung einer Datenquelle" eine Datenquelle ausgewählt werden, aus der importiert, bzw. in die exportiert werden soll.



Alternativ findet man die Schaltfläche auch im Reiter "TECHNIK" unter "Registrierte Objekte" -> "Abbildungen von Datenquellen".

Folgende Schnittstellen und Dateiformate stehen für den Import und Export zur Verfügung:

- CSV/Excel-Datei
- XML-Datei
- MySQL-Schnittstelle
- ODBC-Schnittstelle
- Oracle-Schnittstelle
- PostgreSQL-Schnittstelle
- Für den Austausch von Benutzer-IDs ist eine Standard-LDAP-Schnittstelle implementiert.

Im Folgenden wird anhand einer CSV-Datei beschrieben, wie man einen tabellenorientierten Import/Export anlegt. Da bis auf den XML-Import/Export alle Importe/Exporte tabellenorientiert sind und sich die einzelnen Datenquellen ansonsten nur in ihrer Konfiguration un-

terscheiden, kann das Beispiel der Abbildung der CSV-Datei auch auf die Abbildungen der anderen Datenbanken und Dateiformate übertragen werden.

1.5.1 Mappings of Data Sources

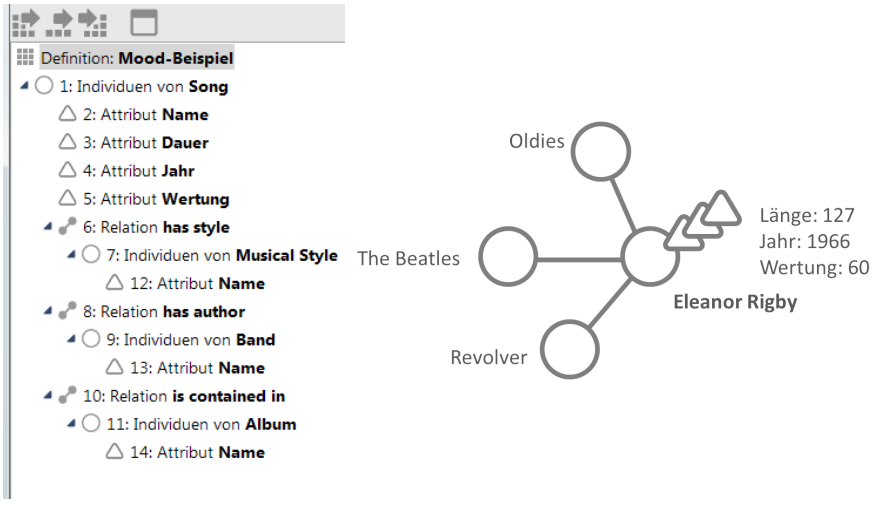
CSV-Dateien sind das Standard-Austauschformat von Tabellenkalkulationstools wie Excel. CSV-Dateien bestehen aus einzelnen Klartext-Zeilen, bei denen die Spalten durch ein fest vorgegebenes Zeichen wie z.B. ein Semikolon getrennt sind.

1.5.1.1 Functional principle

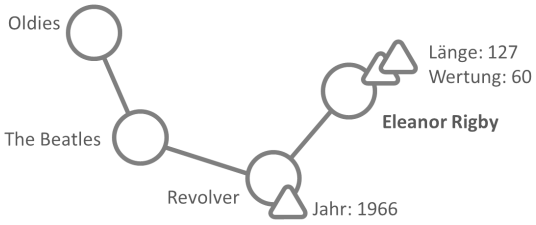
Nehmen wir als erstes Beispiel eine Tabelle mit Songs: Beim Import dieser Tabelle möchten wir für jede Zeile ein neues konkretes Objekt vom Typ Song anlegen. Aus den Inhalten der Spalten B bis G werden Attribute des Songs bzw. Relationen zu anderen Objekten:

	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	

Ausgehend vom Song, bauen wir die Struktur von Attributen, Relationen und Zielobjekten auf, die durch den Import angelegt werden soll (linke Seite). So wird für die Zeile 18 beispielsweise ein Objekt vom Typ Song mit folgenden Attributen und Beziehungen angelegt:



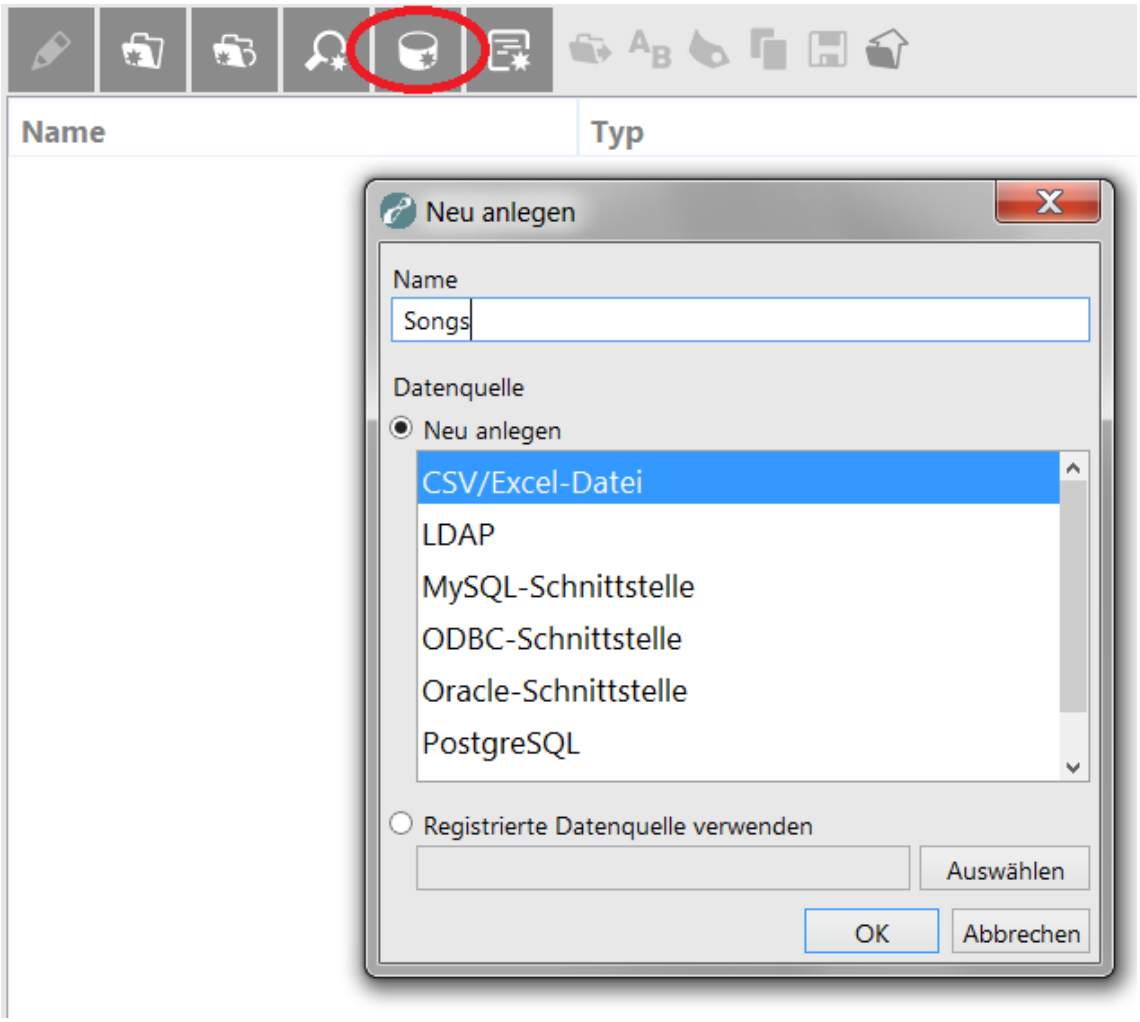
Wir können uns aber auch dafür entscheiden, die Angaben aus der Tabelle anders zu verteilen also z.B. Erscheinungsjahr und Interpret dem Album zuordnen und das Genre wiederum dem Interpreten. Eine Zeile bildet immer noch einen Kontext, muss deswegen aber nicht zu genau einem Objekt gehören:



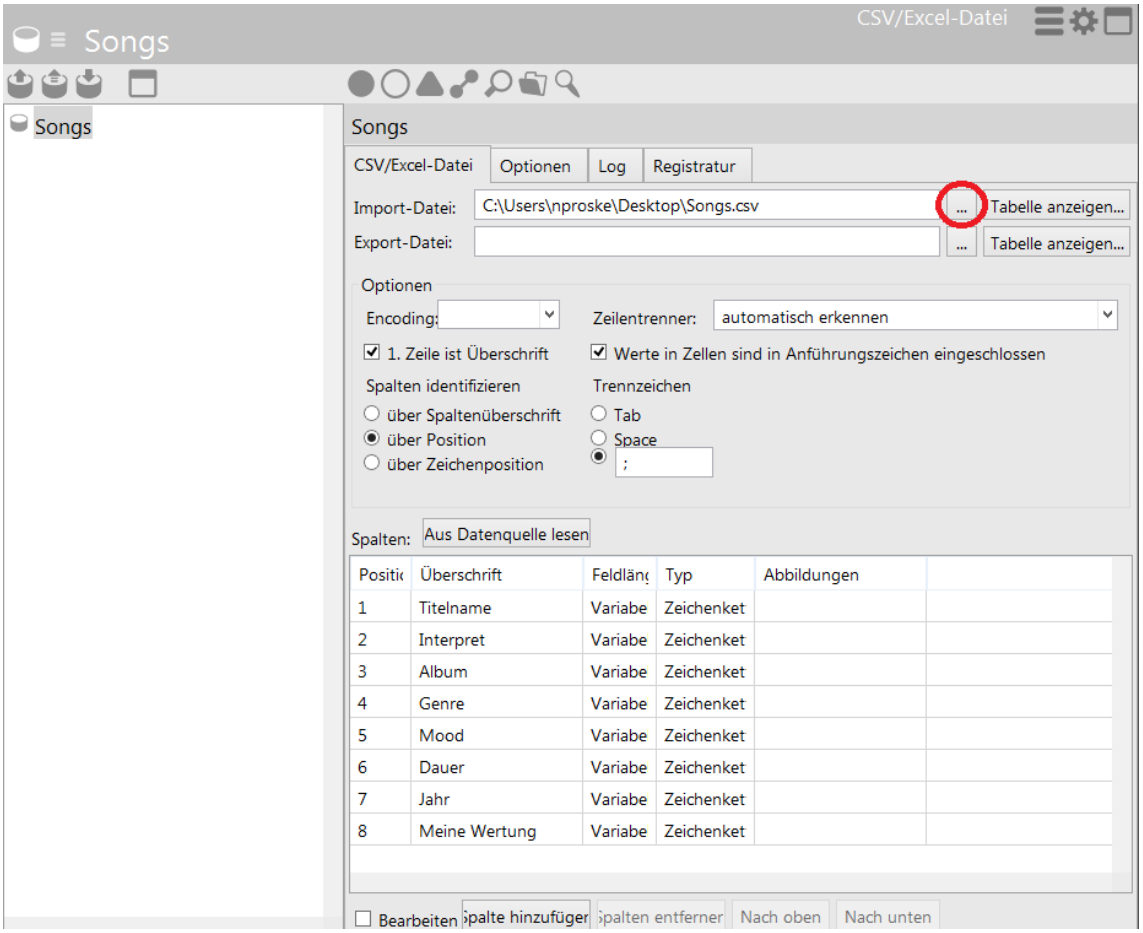
Überall, wo wir in unserem Beispiel neue konkrete Objekte als Relationsziele aufbauen, müssen wir immer mindestens ein Attribut zu diesem Objekt angeben, hier jeweils das Namensattribut, mit dem wir das entsprechende Objekt identifizieren können.

1.5.1.2 Datenquelle - Auswahl und Optionen

Nachdem wir, die Schaltfläche "*Neue Abbildung einer Datenquelle*" ausgewählt haben, öffnet sich ein Dialog, mit dem wir die Art der Datenquelle und den Namen der Abbildung angeben müssen. Haben wir die Datenquelle bereits in der semantischen Graph-Datenbank registriert, können wir diese im unteren Auswahlmenü finden.



Mit der Bestätigung auf "OK" öffnet sich der Editor für den Import und Export. Unter "Import-Datei" können wir den Pfad unserer zu importierenden Datei angeben. Alternativ können wir die Datei auch über den Button rechts daneben auswählen. Sobald die Datei ausgewählt wurde, werden die Spaltenüberschriften und ihre Positionen in der Tabelle ausgelesen und im Feld rechts unten angezeigt. Die Schaltfläche "*Aus Datenquelle lesen*" kann bei eventuellen Änderungen der Datenquelle die Spalten erneut auslesen. Die Spalte "Abbildungen" zeigt uns später jeweils auf welches Attribut die jeweilige Spalte der Tabelle abgebildet wird.



Die Struktur unserer Beispiel-Tabelle entspricht komplett den Standard-Einstellungen, sodass wir unter dem Menüpunkt *Optionen* nichts weiter berücksichtigen müssen. CSV-Dateien können jedoch sehr unterschiedliche Strukturen aufweisen, die mit folgenden Einstellungsmöglichkeiten berücksichtigt werden müssen:

Encoding: Hier wird die Zeichenkodierung der Import-Datei festgelegt. Zur Auswahl stehen ascii, ISO-8859-1, ISO-8859-15, UCS-2, UTF-16, UTF-8 und windows-1252. Ist nichts ausgewählt, wird die Standard-Einstellung übernommen, die der des laufenden Betriebssystems entspricht.

Zeilentrenner: In den meisten Fällen reicht die Einstellung "automatisch erkennen", die auch standardmäßig ausgewählt ist. Sollte man jedoch feststellen, dass Zeilenumbrüche nicht richtig erkannt werden, sollte man die entsprechende korrekte Einstellung manuell auswählen. Zur Auswahl stehen *CR* (*carriage return*, engl. für Wagenrücklauf), *LF* (*line feed*, engl. für Zeilenvorschub), *CR-LF* und *Keine*. Der Standard, der den Zeilenumbruch in einer Textdatei kodiert, ist bei Unix, Linux, Android, Mac OS X, AmigaOS, BSD und weiteren *LF*, bei Windows, DOS, OS/2, CP/M und TOS (Atari) *CR-LF* und bei Mac OS bis Version 9, Apple II und C64 *CR*.

1. Zeile ist Überschrift: Es kann vorkommen, dass die erste Zeile keine Überschrift enthält, was mit dem Entfernen des standardmäßig gesetzten Häkchens bei "*1. Zeile ist Überschrift*" dem System mitgeteilt werden muss.

Werte in Zellen sind in Anführungszeichen eingeschlossen wählt man aus, damit die Anführungszeichen nicht mit importiert werden, wenn man das nicht möchte.

Spalten identifizieren: Ob die Spalten über ihre Überschrift, die Position oder die Zeichenposition identifiziert werden, muss angegeben werden, da ansonsten die Tabelle nicht korrekt erfasst werden kann.

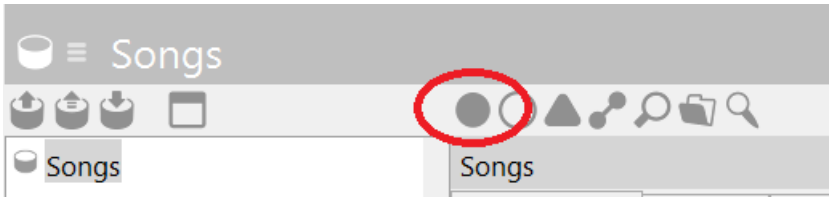
Trennzeichen: Falls ein anderes Trennzeichen als das standardmäßige Semikolon verwendet wird, muss dies ebenfalls angegeben werden, sofern die Spalte nicht über die Zeichenposition identifiziert wird.

Darüber hinaus gelten folgende Regeln: Falls ein Wert der Tabelle das Trennzeichen oder einen Zeilenumbruch enthält, muss der Wert in doppelte Anführungszeichen gestellt werden. Falls der Wert ein Anführungszeichen enthält, muss dieses verdoppelt (»**""**«) werden.

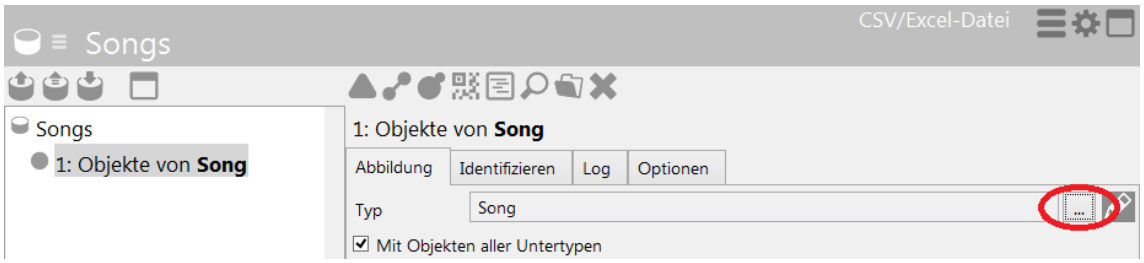
1.5.1.3 Definition von Zielstruktur und Abbildungen

1.5.1.3.1 Object mapping

Nun fangen wir an die Zielstruktur, die in der semantischen Graph-Datenbank entstehen soll, aufzubauen. In unserem Beispiel beginnen wir mit einer Objektabbildung der Songs. Um ein neues Objekt abzubilden müssen wir den Button "Neue Objektabbildung" bestätigen.

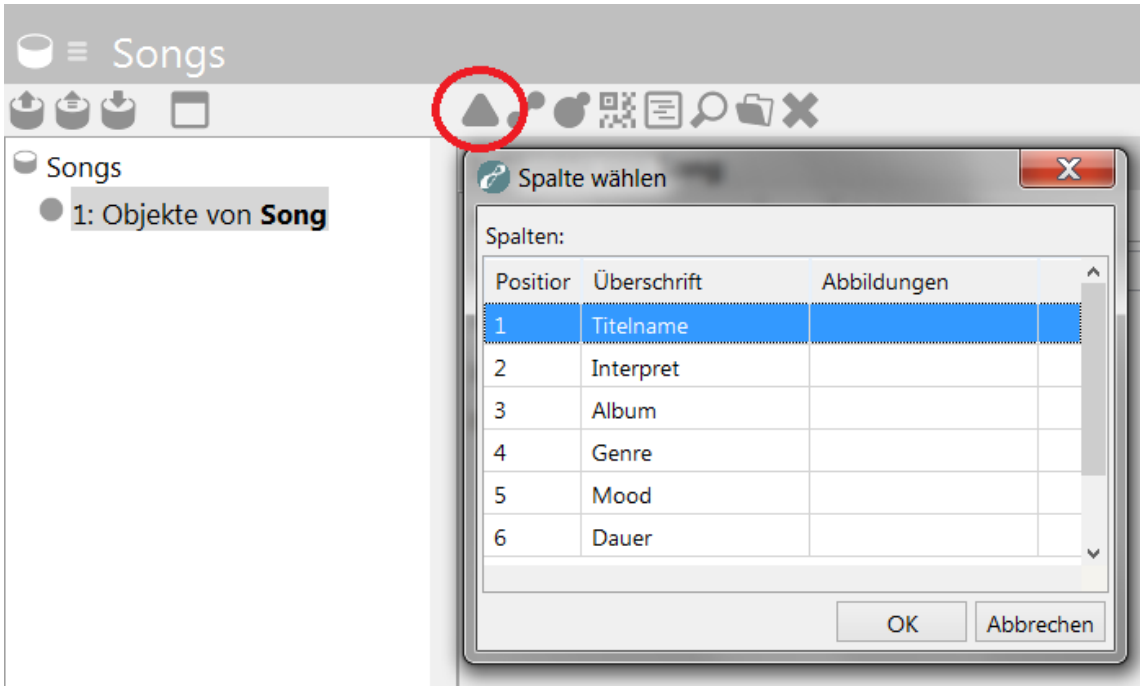


Als nächstes muss der Typ des zu importierenden Objektes angegeben werden. Ist der Haken bei "Mit Objekten aller Untertypen" gesetzt, werden beim Import auch Objekte aus allen Untertypen von "Song" berücksichtigt.

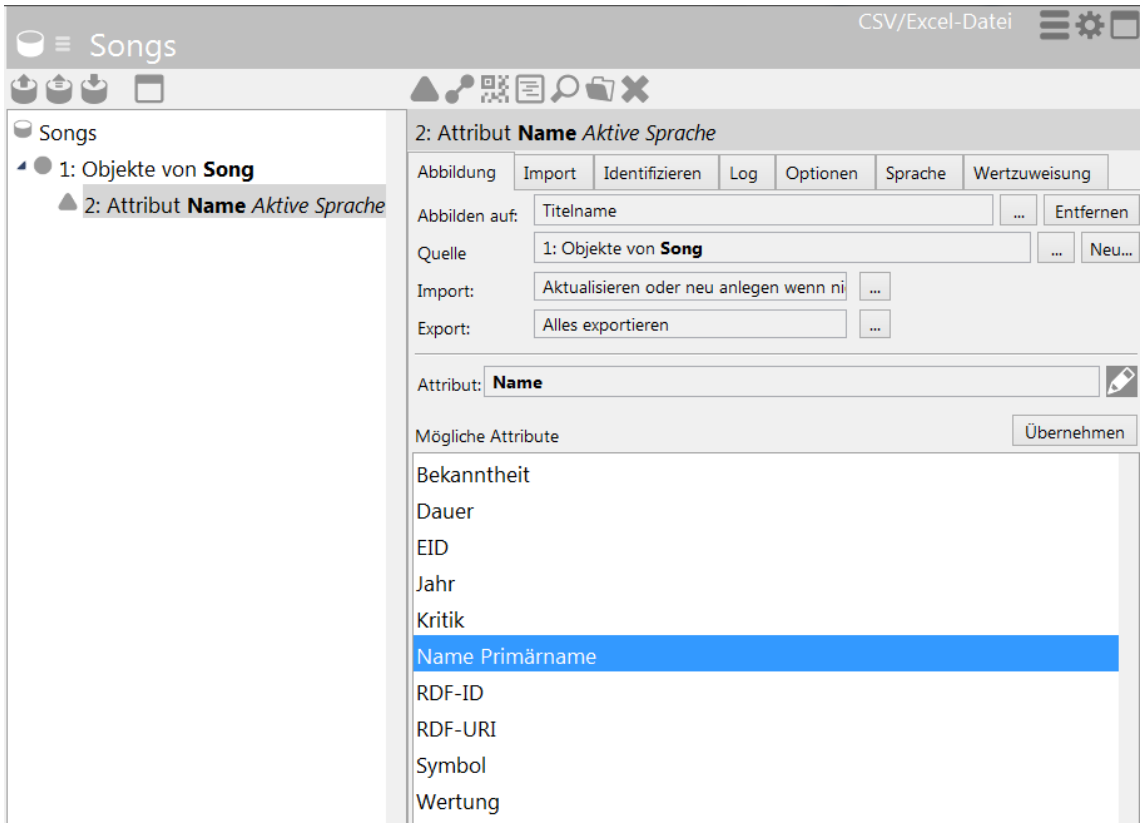


1.5.1.3.2 Attribute mapping / Identifing objects

Nun wollen wir die in der Tabelle enthaltenen Informationen mit der Objektabbildung der Songs verknüpfen. Es sind sowohl Attribute zu den einzelnen Songs, als auch Relationen vertreten. Um zunächst den Titelnamen eines Songs in der Abbildung anzulegen, fügen wir der Objektabbildung von Song ein Attribut hinzu. Ein Klick auf die Schaltfläche "*Neue Attributabbildung*" öffnet einen Dialog, mit dem die entsprechende Spalte aus der zu importierenden Tabelle ausgewählt werden muss.



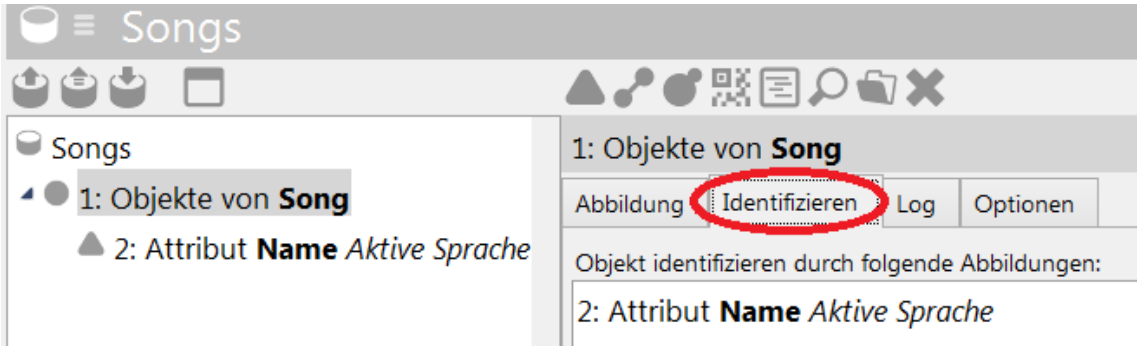
Da dieses Attribut, das erste ist, das wir zu der Objektabbildung von Songs angelegt haben, wird es daraufhin automatisch auf den Namen des Objekts abgebildet, da der Name in der Regel das meistgenutzte Attribut ist.



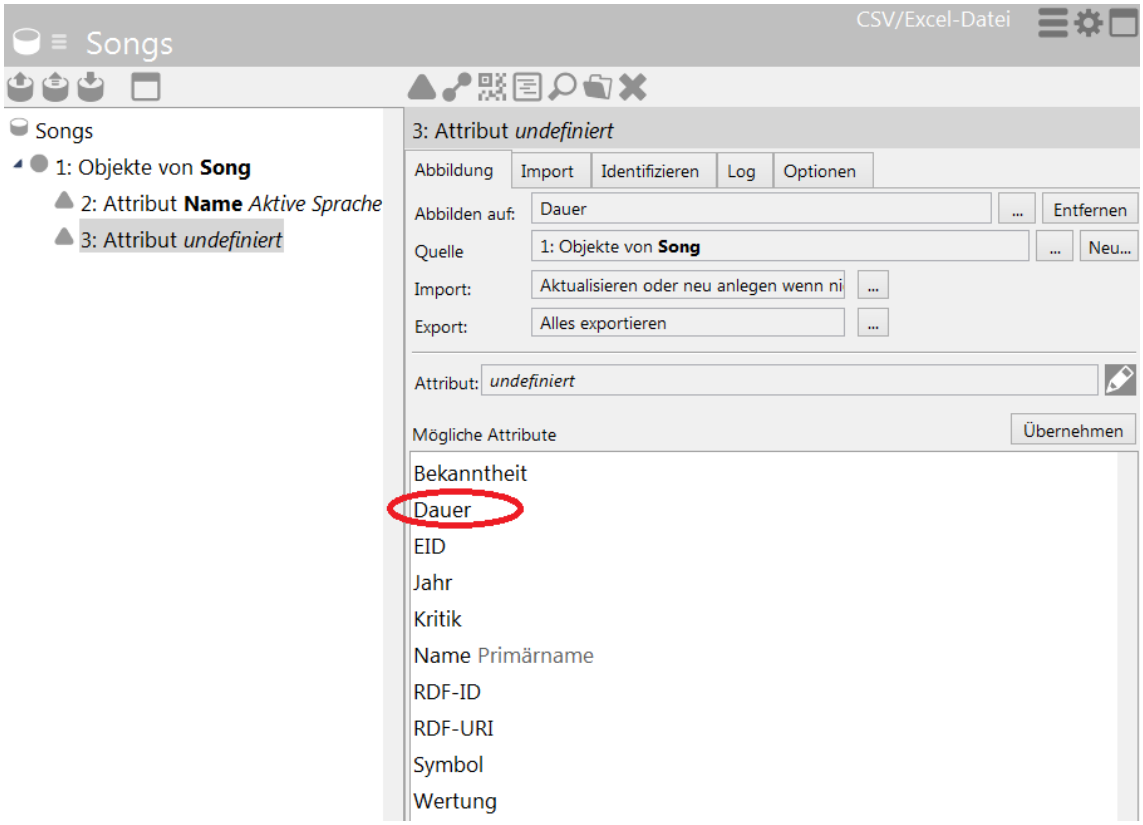
Das erste zu einem Objekt angelegte Attribut wird zudem automatisch zur **Identifizierung des Objektes** verwendet.

Ein Objekt muss über mindestens ein Attribut identifiziert werden - sei es über seinen Namen oder seine ID oder eine Kombination aus mehreren Attributen (wie bei Personen aus Vor- und Nachname und dem Geburtsdatum)-, damit es in der semantischen Graph-Datenbank eindeutig wiedergefunden werden kann, sollte es bereits vorhanden sein. So wird das ungewollte Anlegen von Dubletten beim Import vermieden.

Im Reiter "*Identifizieren*" kann das Attribut, das das Objekt identifiziert noch nachträglich geändert oder mehrere Attribute hinzugefügt werden. Zudem kann hier eingestellt werden, ob die Groß- und Kleinschreibung beim Abgleich der Werte beachtet werden soll und ob nach exakt gleichen Werten gesucht werden soll (ohne Indexfilter/Wildcards). Letzteres ist dann relevant, wenn im Index Filter oder Wildcards definiert sind, die z.B. festlegen, dass ein Bindestrich im Index entfallen soll. Der Begriff würde mit Bindestrich nicht gefunden werden, wenn nur über den Index gesucht wird, also müsste in diesem Fall hier der Haken gesetzt werden, sodass nach dem exakt gleichen Wert gesucht wird.

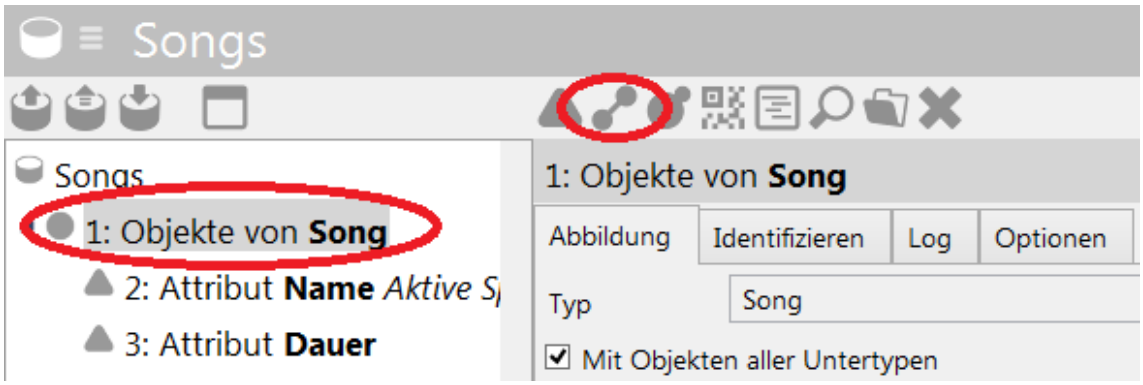


Jetzt können wir dem Objektmapping weitere Attribute hinzufügen, die nicht zur Identifizierung beitragen müssen, z.B. die Dauer eines Songs - dies geschieht wieder über die Schaltfläche "Neue Attributabbildung". (Achtung: Die Objektabbildung "Objekte von Song" muss zunächst wieder ausgewählt werden.) Jetzt wählen wir die Spalte "Dauer" aus der zu importierenden Tabelle aus. Dieses Mal müssen wir das Attribut, auf das die Spalte "Dauer" abgebildet werden soll, manuell auswählen. In dem Feld rechts unten befindet sich die Lister aller im Schema festgelegten möglichen Attribute, die uns für Objekte des Typs "Song" zur Auswahl stehen, darunter auch das Attribut "Dauer".

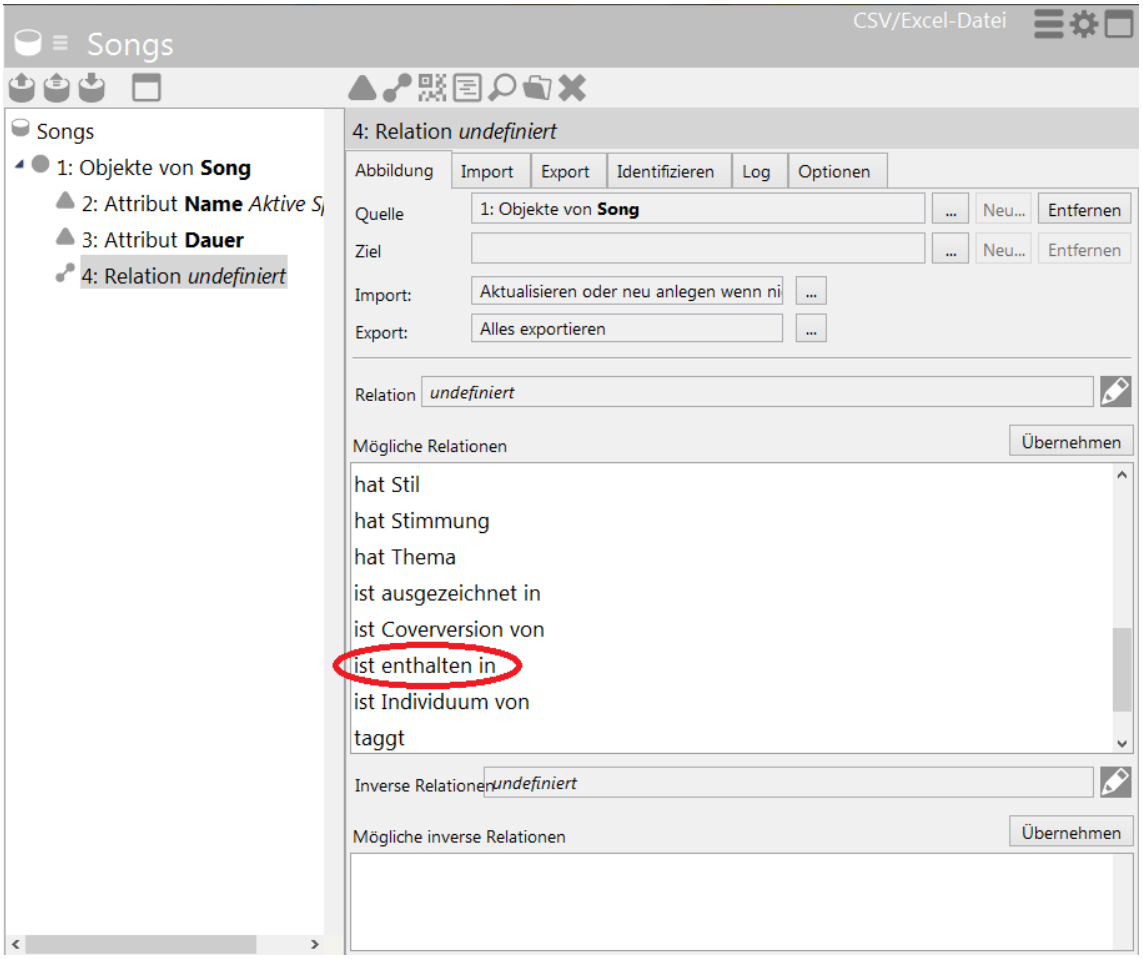


1.5.1.3.3 Relation mapping

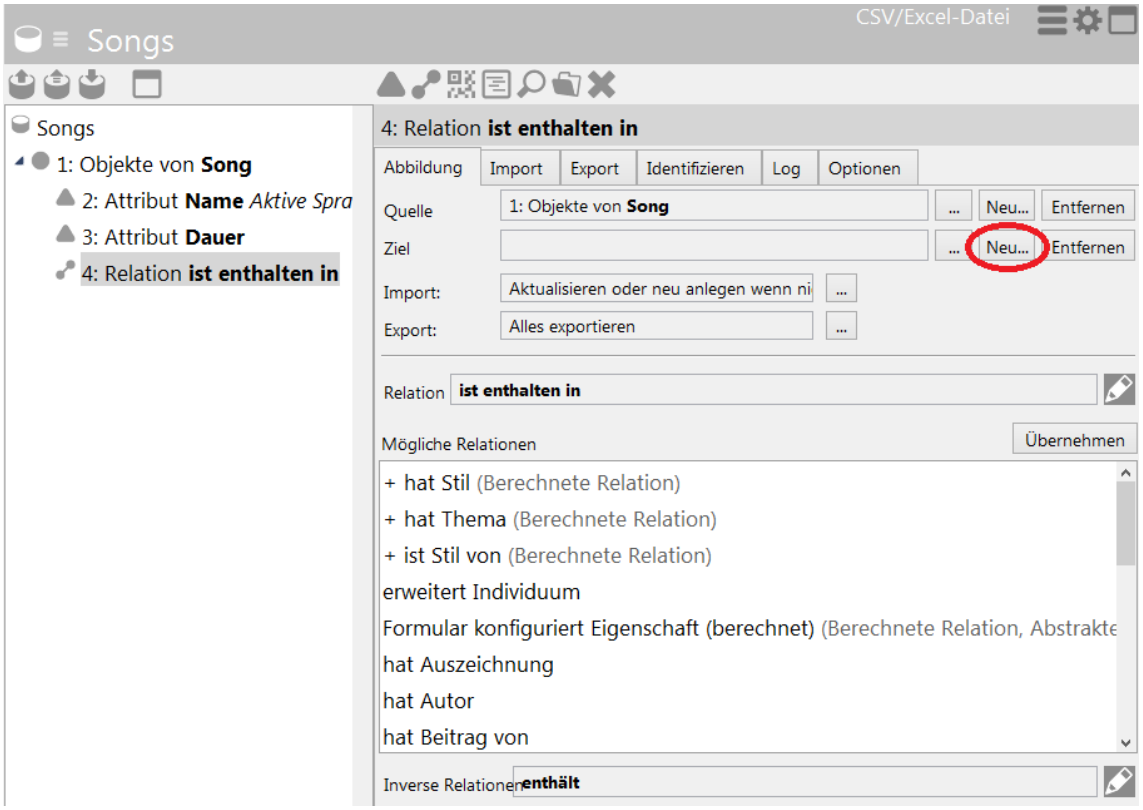
Als nächstes wollen wir das Album abbilden, auf dem der Song sich befindet. Da Alben eigene konkrete Objekte in der semantischen Graph-Datenbank sind, benötigen wir hierfür die Relation, die den Song und das Album verbindet. Um eine Relation abzubilden wählen wir zunächst das Objekt aus, für das die Relation definiert wird und klicken dann auf die Schaltfläche "*Neue Relationsabbildung*".



Daraufhin erhalten wir - wie bei den Attributen - eine Liste aller möglichen Relationen, auch die benötigte Relation "*ist enthalten in*" ist selbstverständlich dabei.



Nun müssen wir im nächsten Schritt festlegen, wo aus der Tabelle die Zielobjekte herkommen. Für das Ziel wird eine neue Objektabbildung gebraucht, die über die Schaltfläche "Neu" angelegt wird. Ist der Typ des Zielobjektes eindeutig im Schema definiert, wird dieser automatisch übernommen, ansonsten erscheint eine Liste der möglichen Objekttypen.

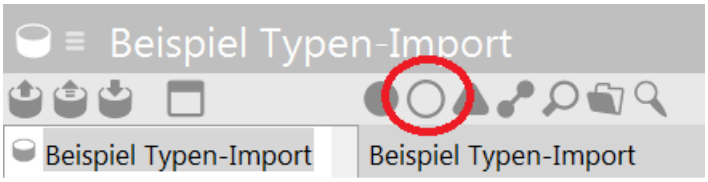


Bei der neuen Objektabbildung müssen wir anschließend wieder das Attribut auswählen, das das Zielobjekt identifiziert usw. So wird die Zielstruktur des Imports aufgebaut.

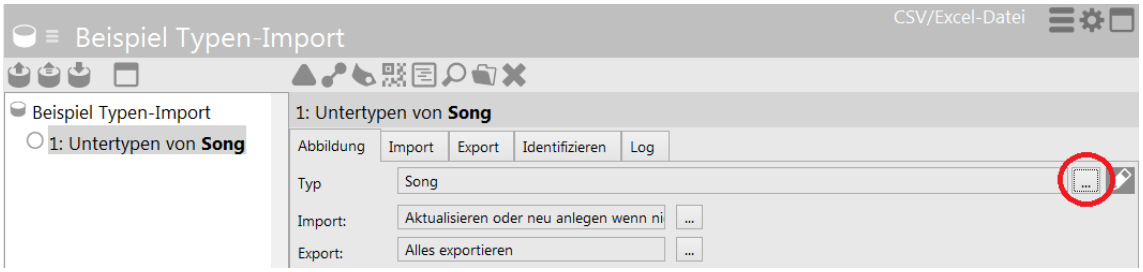
1.5.1.3.4 Type mapping

Auch Typen können importiert und exportiert werden. Nehmen wir beispielhaft an, wir wollten die Genres der Songs als Typen importieren.

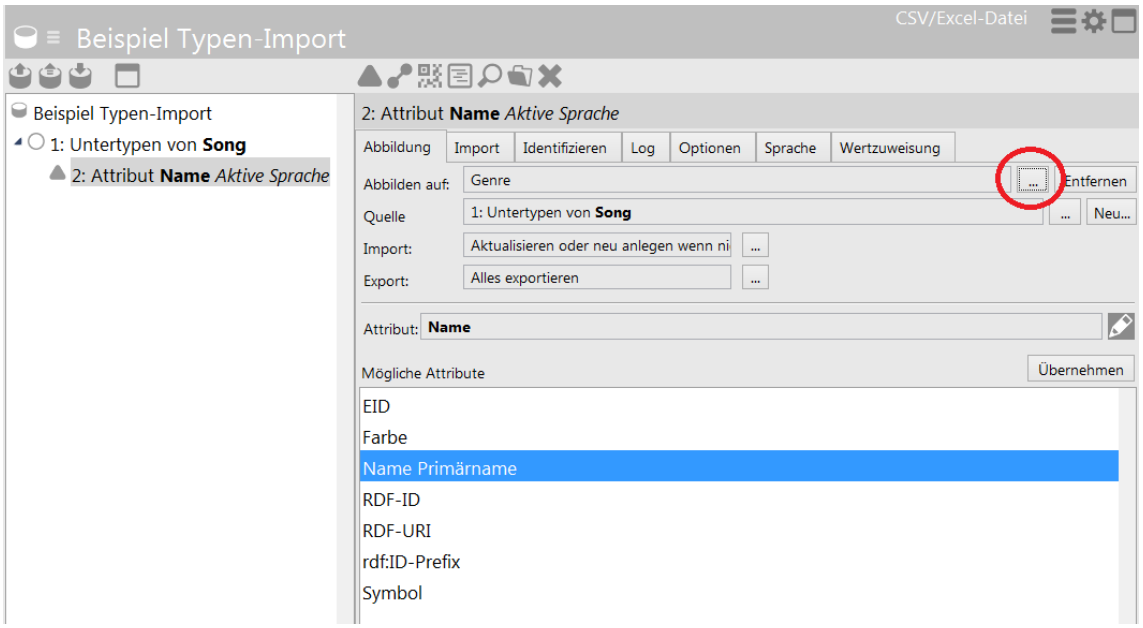
Um einen neuen Typen abzubilden, wählen wir den Button "Neue Typabbildung".



Daraufhin müssen wir den Obertyp der neu anzulegenden Typen angeben, in unserem Beispiel wäre der Obertyp "Song":

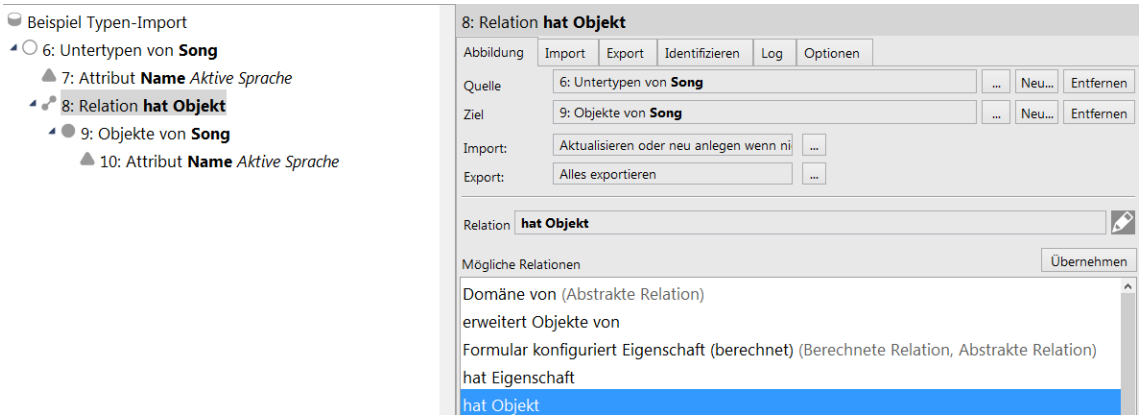


Anschließend müssen wir angeben, aus welcher Spalte der importierten Tabelle der Name unserer neuen Typen entnommen werden soll:



Schließlich müssen wir unter dem Reiter "Import" noch angeben, dass unsere neuen Typen nicht abstrakt sein sollen:

Wenn wir nun die entsprechenden Songs ihren neuen Typen zuordnen wollen, müssen wir die Systemrelation "hat Objekt" verwenden. In älteren Versionen von i-views heißt diese Relation "hat Individuum". Als Ziel wählen wir alle Objekte von Song (inkl. der Untertypen) aus, die sich über das Attribut Name entsprechend der Spalte Songtitel definieren.

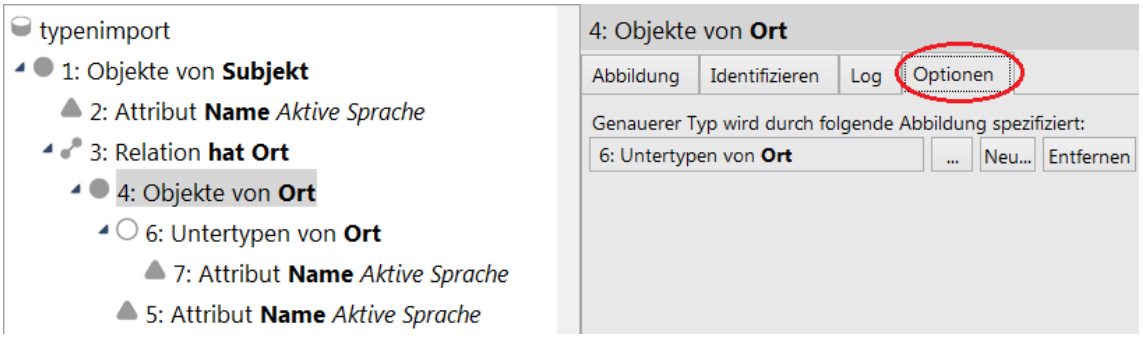


Importieren wir nun diese Abbildung erhalten wir das gewünschte Ergebnis. Die Songs, die bereits in der semantischen Graph-Datenbank vorhanden sind, werden durch die Import-Einstellung "Aktualisieren oder neu anlegen wenn noch nicht vorhanden" berücksichtigt und unter ihren neuen entsprechenden Typ geschoben, sodass kein Objekt doppelt angelegt wird (siehe Kapitel Einstellungen des Import-Verhaltens). Zur Erinnerung: Ein konkretes Objekt kann nicht mehreren Typen gleichzeitig angehören.

Es gibt noch einen Spezialfall. Angenommen, wir haben eine Tabelle, in der in einer Spalte verschiedene Typen vorkommen, dann können wir auch dies in unseren Importeinstellungen abbilden.

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

Dazu wählen wir die Abbildung der Objekte aus, denen wir die Untertypen zuordnen wollen (in diesem Fall "Objekte von Ort") und wählen dann unter dem Reiter "Optionen" den entsprechenden Obertyp aus.



Wichtig ist auch hier wieder nicht zu vergessen, unter dem Reiter "Import" festzulegen, dass der Typ nicht abstrakt sein soll, damit konkrete Objekte angelegt werden können.

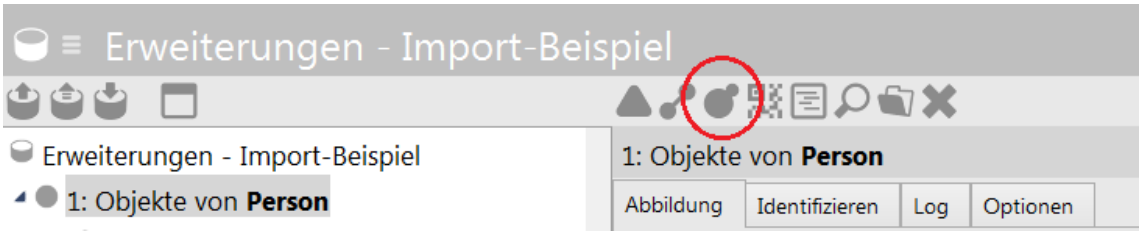
Vorsicht: Angenommen, Liverpool existiert bereits im Wissensnetz, ist jedoch dem Typ "Ort" zugeordnet, da dieser bis zu diesem Zeitpunkt noch keine Untertypen wie "Stadt" und "Land" besessen hat. In diesem Fall wird Liverpool **nicht** unter dem Typ Stadt neu angelegt. Begründung: die Objekte des Typs Ort werden lediglich über das Namensattribut identifiziert, nicht jedoch über den Untertyp.

1.5.1.3.5 Mapping of extensions

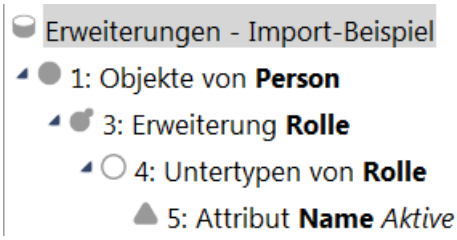
Auch Erweiterungen können importiert und exportiert werden. Angenommen, wir haben eine Tabelle, die die Rolle eines Bandmitglieds in einer Band zeigt:

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

Ron Wood ist Gitarrist bei den Faces und den Rolling Stones aber Bassist bei der Jeff Beck Group. Um dies abzubilden müssen wir das Objekt auswählen, zu dem im Schema eine Erweiterung definiert wurde und dann den Button "Neue Erweiterungsabbildung" betätigen.

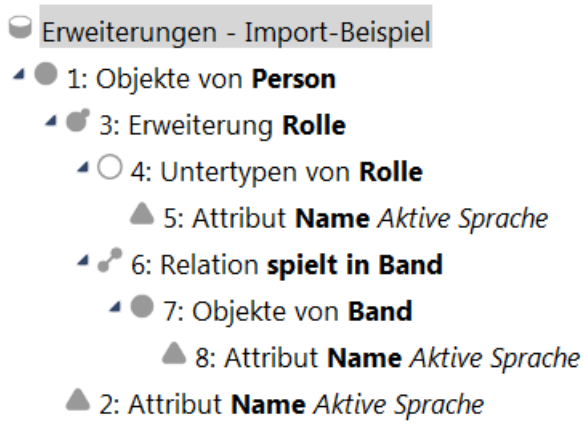


Die Abbildung einer Erweiterung fragt - wie eine Objektabbildung - einen zugehörigen Typen ab. Im Schema des Musik-Netzes ist der Typ "Rolle" ein abstrakter Typ. Deswegen muss in der Abbildung definiert werden, dass die Rolle auf Untertypen des Typs "Rolle" abgebildet weden sollen (siehe Kapitel Die Typabbildung).



Die Relation kann - wie auch bei Objekten und Typen - an der Erweiterung (bzw. an den

Untertypen einer Erweiterung) abgebildet werden.

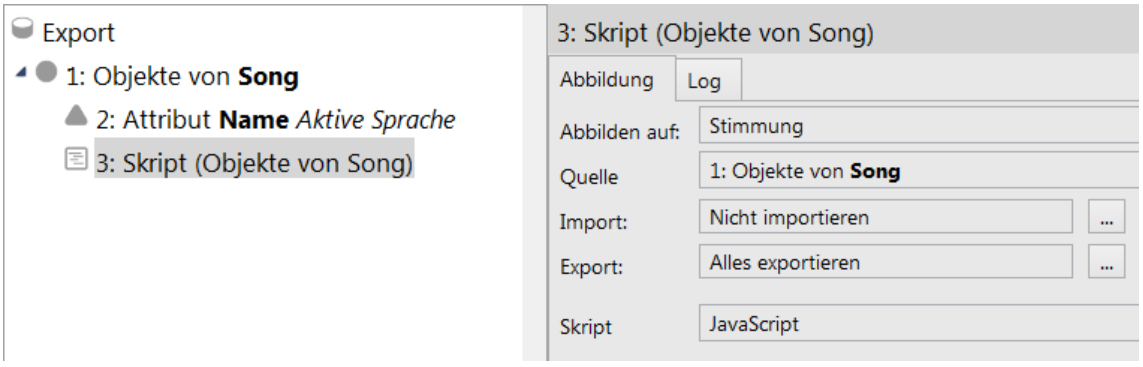


1.5.1.3.6 Die Skriptabbildung

Die Skriptabbildung kann ausschließlich beim Export verwendet werden. Das Skript kann entweder in JavaScript oder KScript geschrieben sein.

Die Skriptabbildung findet beispielsweise dann Verwendung, wenn wir drei Attribute aus der semantischen Graph-Datenbank zu einer ID zusammensetzen wollen. Allerdings kann es sein, dass der Export dann langsamer ist. (Bei einem Import könnte man dies einfacher über eine virtuelle Eigenschaft abbilden. Die Verwendung von virtuellen Eigenschaften wird im Kapitel Tabellenspalten erklärt.)

Der folgende Fall ist ein weiteres Beispiel für die Verwendung eines Skripts bei einem Export. Es zeigt wie mehrere Eigenschaften mit einem Trennzeichen in eine Zelle geschrieben werden können. In diesem Fall wollen wir eine Tabelle erzeugen, die in der ersten Spalte die Songnamen und in der zweiten Spalte alle Stimmungen der Songs mit Komma getrennt aufführt:

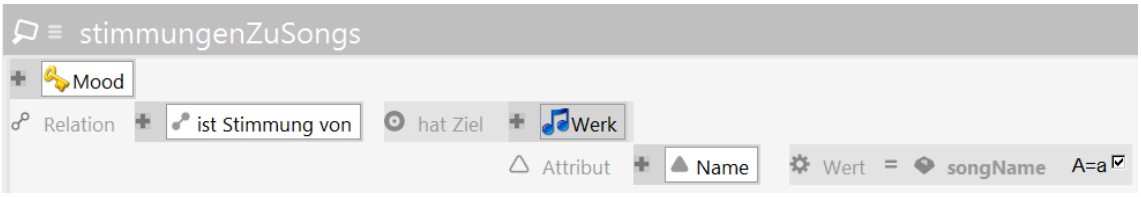


Um die zweite Spalte zu erzeugen benötigen wir folgendes Skript:

```
function exportValueOf(element)
{

    var stimmung = "";
    var relTargets = $k.Registry.query("stimmungenZuSongs").findElements({songName: element.attrib
    if(relTargets && relTargets.length > 0){
        for(var i = 0; i < (relTargets.length-1); i++){
            stimmung += relTargets[i].attributeValue("objektName") + ", ";
```

```
        }
        stimmung += relTargets[relTargets.length-1].attributeValue("objektName");
    }
    return stimmung;
}
```

Das Skript beinhaltet folgende Strukturabfrage (Registrierungsschlüssel: "stimmungenZuSongs"):



Über den Ausdruck "findElements" können wir auf einen Parameter (hier "songName") innerhalb der Abfrage zugreifen. "objektName" ist der interne Name des Namensattributs in diesem semantischen Modell.

Innerhalb der if-Anweisung sagen wir, dass wenn ein Element mehrere Relationsziele hat, diese durch ein Komma getrennt dargestellt werden sollen. Nach dem letzten Relationsziel, das die Schleife durchläuft, soll keine Komma mehr stehen. Auch wenn ein Element nur ein Relationsziel hat, wird dies demnach ohne Komma dargestellt.

Das Ergebnis ist eine Liste der Songs mit allen ihren Stimmungen, die durch Komma getrennt in der zweiten Spalte der Tabelle stehen:

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	

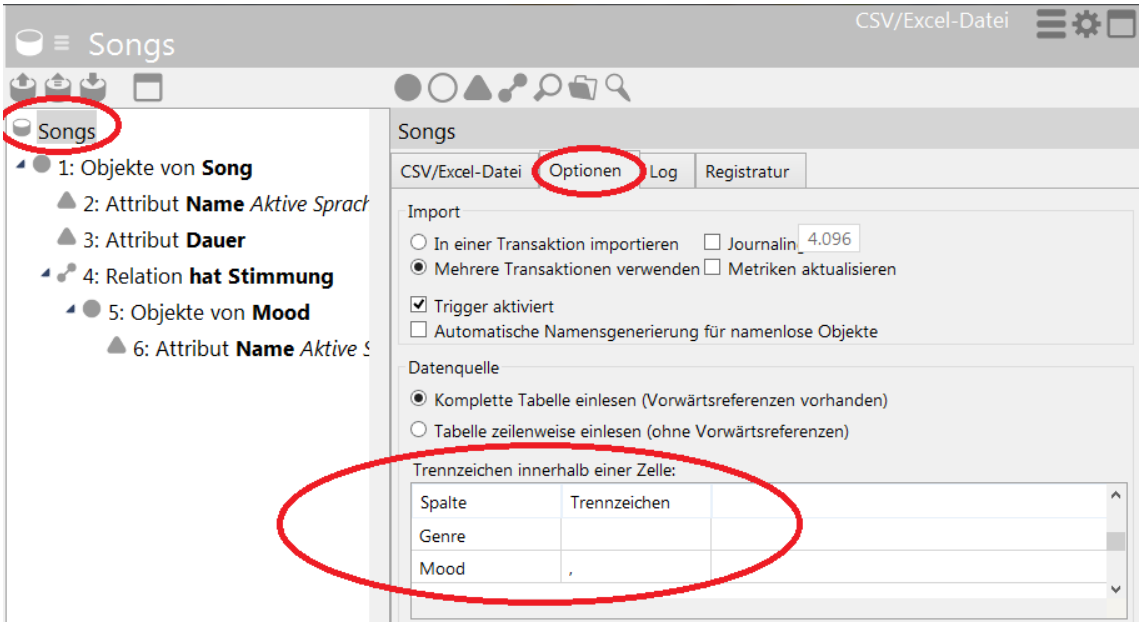
1.5.1.4 Abbildung von mehreren Werten für einen Objekttyp bei einem Objekt

Wenn für einen Objekttyp bei einem Objekt mehrere Werte angegeben sind (in unserem Beispiel etwa mehrere „Moods“ für jeden Song), dann gibt es drei Möglichkeiten, wie die Tabelle aussehen kann. Für zwei der drei Möglichkeiten muss der Import angepasst werden, was im Folgenden beschrieben ist.

Möglichkeit 1 - Trennzeichengetrennte Werte: Die einzelnen Werte befinden sich in einer Zelle und sind durch ein Trennzeichen (z.B. ein Komma) getrennt.

	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

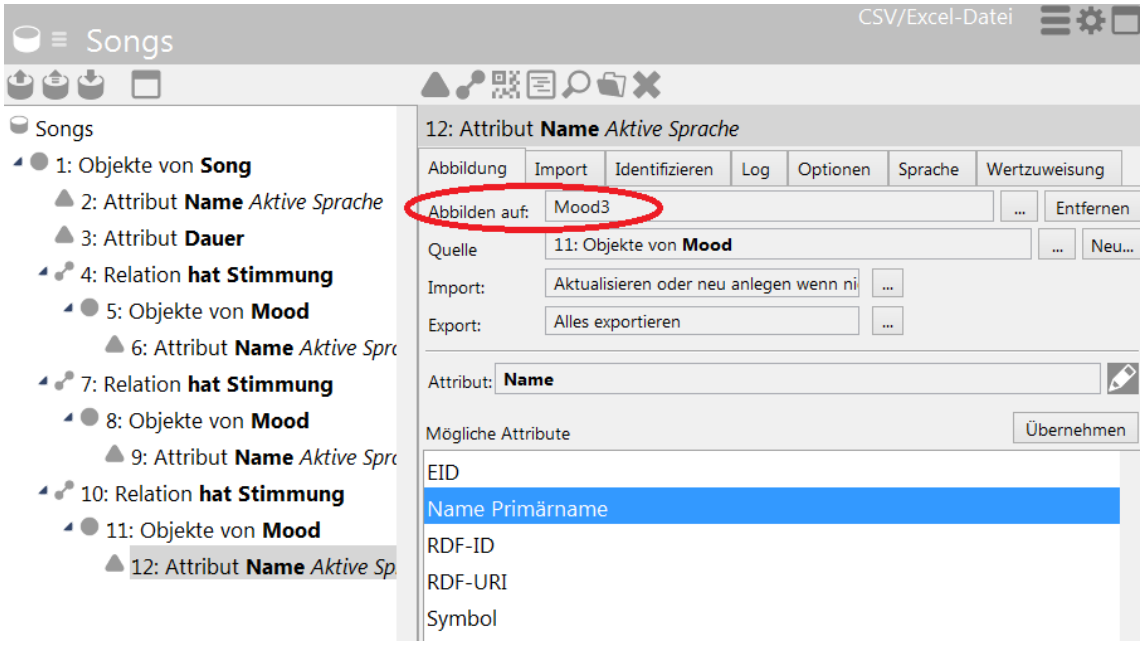
In diesem Fall gehen wir auf die Abbildung der Datenquelle, wo sich die allgemeinen Einstellungen befinden und dort auf den Reiter "*Optionen*". Hier finden wir im unteren Bereich die Einstellungsmöglichkeit, Trennzeichen innerhalb einer Zelle anzugeben. Nun müssen wir nur noch die entsprechende Spalte der zu importierenden Tabelle heraussuchen ("Mood") und in die Spalte "*Trennzeichen*" das verwendete Trennzeichen (",") eingeben.



Möglichkeit 2 - Mehrere Spalten: Die einzelnen Werte befinden sich jeweils in einer eigenen Spalte, wobei nicht jedes Feld ausgefüllt sein muss. Es werden so viele Spalten benötigt, wie maximal Moods pro Song vorhanden sind.

	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 diesem Fall muss die entsprechende Relation so oft angelegt werden, wie Spalten vorhanden sind. In diesem Beispiel müssen demnach die erste Relation auf "Mood1", die zweite Relation auf "Mood2" und die dritte Relation auf "Mood3" abgebildet werden.



Möglichkeit 3 - Mehrere Zeilen: Die einzelnen Werte befinden sich jeweils in einer eigene Zeile. Achtung: Hierbei ist es zwingend nötig, dass die Attribute, die für die Identifizierung des Objektes benötigt werden (in diesem Fall der Titelname), in jeder Zeile auftreten, ansonsten würden die Zeilen als jeweils eigenes Objekt ohne Name gedeutet werden und ein korrekter Import wäre nicht möglich.

	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 diesem Fall sind keine besonderen Import-Einstellungen nötig, da das System über das identifizierende Attribut das Objekt erkennt und die Relationen korrekt zieht.

1.5.1.5 Einstellungen des Import-Verhaltens

Beim Import-Vorgang wird immer geprüft, ob ein Attribut bereits vorhanden ist. Das „Identifizieren“ schließt von Attributen auf die konkreten Objekte. Wenn wir nun im Folgenden von „bereits vorhandenen Attributen“ sprechen, dann sind das Attribute, die im Wert genau mit dem Wert aus der Spalte, auf die sie abgebildet sind, übereinstimmen. Wenn wir von bereits vorhandenen Objekten sprechen, dann sind das konkrete Objekte, die durch ein bereits vorhandenes Attribut identifiziert werden.

Beispiel: Wenn in unserem Netz bereits ein Song mit dem Namen „Eleanor Rigby“ existiert, dann ist das Namensattribut (abgebildet auf die Spalte „Titelname“ unserer Importtabelle) ein existierendes Attribut und folglich der Song ein existierender Song, solange der Song nur

über das Namensattribut identifiziert wird.

Mit den Einstellungen für das Importverhalten können wir steuern, wie der Import auf bereits vorhandene und neue Wissensnetzelemente reagieren soll. Folgende Tabelle zeigt eine Kurzbeschreibung der einzelnen Einstellungen, während die Unterkapitel dieses Kapitels ausführliche und anschauliche Beschreibungen beinhalten.

Einstellung	Kurzbeschreibung
Aktualisieren	Vorhandene Elemente werden überschrieben (aktualisiert), keine neuen Elemente werden angelegt.
Aktualisieren oder neu anlegen wenn nicht vorhanden	Vorhandene Elemente werden überschrieben, sollten keine vorhanden sein, werden sie neu angelegt.
Alle mit selbem Wert löschen (nur bei Eigenschaften verfügbar)	Alle Attributwerte, die mit dem importierten Wert übereinstimmen, werden für die jeweils entsprechenden Objekte gelöscht.
Alle vom selben Typ löschen	Alle Attributwerte des ausgewählten Typs werden für die entsprechenden Objekte gelöscht, unabhängig davon, ob die Werte übereinstimmen oder nicht.
Löschen	Wird verwendet, um genau das eine Element zu löschen.
Neu anlegen	Legt eine neue Eigenschaft/Objekt an, ohne zu beachten, ob der Attributwert oder das Objekt bereits vorhanden ist.
Neu anlegen wenn noch nicht vorhanden (nur bei Attributen verfügbar)	Nur, wenn noch kein Attribut des gewünschten Typs vorhanden ist, wird eines angelegt.
Nicht importieren	Kein Import.
Synchronisieren	Um die zu importierenden Inhalte mit den Inhalten der Datenbank zu synchronisieren, werden bei dieser Aktion alle Elemente, die noch nicht vorhanden sind, neu angelegt, alle die sich geändert haben, aktualisiert und alle, die nicht mehr vorhanden sind, gelöscht.

Bei einem Import müssen wir uns für jedes abgebildete Objekt, jede abgebildete Relation und jedes abgebildete Attribut einzeln entscheiden, welche Import-Einstellung wir jeweils verwenden wollen. Anders als in anderen Editoren des Knowledge-Buildes "vererbt" sich eine Einstellung nicht an die darunterliegenden Abbildungselemente. Auch die Import-Einstellung für ein Objekt "vererbt" sich nicht an seine Attribute.

1.5.1.5.1 Aktualisieren

Wird diese Einstellung bei einem **Attribut** angewendet, sorgt sie dafür, dass der Wert aus der Tabelle den Attributwert genau eines bereits existierenden Attributs überschreibt. Mit dieser Einstellung werden keine neuen Attribute angelegt. Falls das Objekt mehr als einen Attributwert des ausgewählten Typs hat, wird kein Wert importiert.

Verwendet man die Einstellung "Aktualisieren" bei einem identifizierenden Attribut, während man bei dem dazugehörigen Objekt die Einstellung "Aktualisieren oder neu anlegen, wenn nicht vorhanden" verwendet, erscheint die Fehlermeldung "Attribut nicht gefunden", wenn das identifizierende Attribut nicht in i-views vorhanden ist.

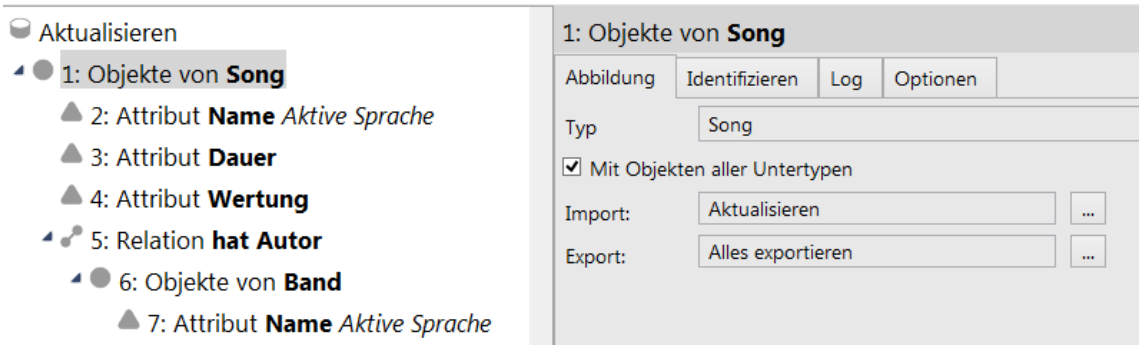
Wird "Aktualisieren" bei einem **Objekt** angewendet, sorgt die Einstellung dafür, dass alle Eigenschaften des Objekts durch den Import hinzugefügt, bzw. geändert werden können. Neue Objekte werden nicht angelegt.

Beispiel: Angenommen wir führen eine Datenbank mit unseren Lieblings-Songs. Nun haben wir eine Liste mit Songs, die neue Informationen, enthalten. Wir wollen diese Informationen in unsere Datenbank bringen, gleichzeitig aber nicht, dass Songs importiert werden, die nicht zu unseren Lieblings-Songs gehören. Hierfür verwenden wir die Einstellung "Aktualisieren".

Der Song "About A Girl" ist bereits im Knowledge-Builder vorhanden.

Song	Dauer	Wertung	Autor
About A Girl	168	5	Nirvana

Die Import-Tabelle enthält Angaben zu Dauer, Wertung und Autor des Songs.



Wir legen für Objekte von Song fest, dass sie aktualisiert werden sollen. Alle Attribute, Relationen und Relationsziele erhalten die Import-Einstellung "Aktualisieren oder neu anlegen wenn noch nicht vorhanden".

Das Ergebnis: Der Song wurde aktualisiert und hat nun neue Attribute und Relationen erhalten. Bereits vorhandene Eigenschaften wurden aktualisiert (Wertung).

1.5.1.5.2 Aktualisieren oder neu anlegen wenn nicht vorhanden

Diese Import-Einstellung wird in den meisten Fällen benötigt und ist darum als Standard-Einstellung gesetzt. Wenn Elemente bereits vorhanden sind, werden sie aktualisiert. Wenn Elemente noch nicht vorhanden sind, werden sie in der Datenbank neu angelegt.

1.5.1.5.3 Alle mit selben Wert löschen

Diese Import-Einstellung ist nur bei Eigenschaften (Relationen und Attributen) verfügbar und wird nur verwendet, wenn über die Import-Einstellung "Löschen" nicht gelöscht werden kann. Mit "Löschen" kann dann nicht gelöscht werden, wenn eine Relation oder ein Attribut bei einem Objekt mehrmals mit denselben Werten vorkommt. Versucht man es dennoch, erscheint eine Fehlermeldung. Zum Beispiel kann es sein, dass der Song "About A Girl" versehentlich zweimal mit der Band "Nirvana" über die Relation "hat Autor" verknüpft wurde.

In solchen Fällen greift die Import-Einstellung "Löschen" nicht, da sie bei Mehrfachvorkommen nicht weiß, welche der Relationen sie löschen soll. Hier muss also "Alle mit selben Wert löschen" verwendet werden.

1.5.1.5.4 Alle vom selben Typ löschen

Diese Import-Einstellung wird verwendet, wenn alle Attribute, Objekte oder Relationen eines Typs gelöscht werden sollen, unabhängig von den vorhandenen Werten. Im Gegensatz dazu berücksichtigen die Einstellungen "Löschen" und "Alle mit selben Wert löschen" die vorhandenen Werte. Es werden nur die Elemente der Objekte gelöscht, die in der Import-Tabelle vorkommen.

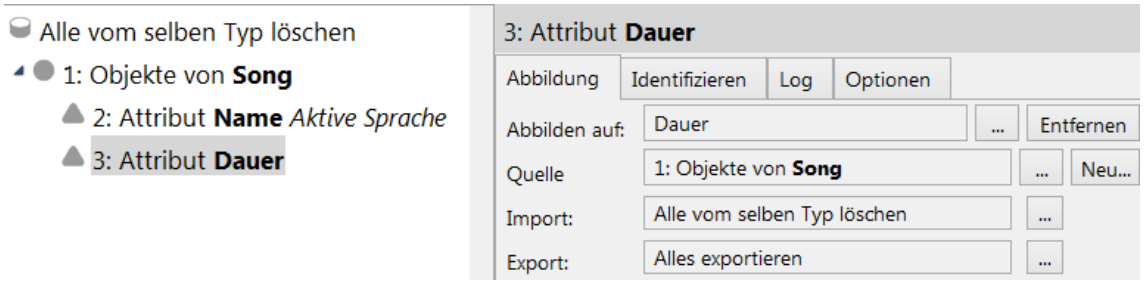
Beispiel: Wir haben eine Import-Tabelle mit Songs und der Dauer der Songs. Wir sehen, dass sich die Dauer in vielen Fällen unterscheidet und beschließen, die Dauer für diese Songs zu löschen, damit wir keinesfalls falsche Angaben haben.

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

Die Dauer in der Import-Tabelle unterscheidet sich bei den meisten Songs...

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

... von der Dauer der Songs in der Datenbank.



Beim Attribut "Dauer" verwenden wir die Import-Einstellung "Alle vom selben Typ löschen".

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

Nach dem Import, sind alle Attributwerte des Attributtyps Dauer für diese 4 Songs gelöscht.

1.5.1.5.5 Löschen

Die Import-Einstellung "Löschen" wird verwendet, um genau das eine Objekt / genau die eine Relation / genau den einen Attributwert zu löschen. Falls keine oder mehrere Objekte / Relationen / Attributwerte mit den zu importierenden Elementen übereinstimmen, erscheint diesbezüglich eine Fehlermeldung und die betroffenen Elemente werden nicht gelöscht.

1.5.1.5.6 Neu anlegen

Diese Import-Einstellung legt eine neue Eigenschaft / ein neues Objekt an, ohne zu beachten, ob der Attributwert oder das Objekt bereits vorhanden ist. Einzige Ausnahme: Sollte eine Eigenschaft nur einmal vorkommen (man beachte die Einstellung "kann mehrfach vorkommen" bei der Attributdefinition), so wird das neue Attribut nicht angelegt und eine Fehlermeldung erscheint, die dies mitteilt.

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 Neu anlegen wenn noch nicht vorhanden

Diese Import-Einstellung ist nur bei Attributen verfügbar. Ein neuer Attributwert wird nur angelegt, wenn das entsprechende Attribut noch keinen Wert hat. Die Werte müssen nicht gleich sein, es geht nur um das Vorhandensein, bzw. Nichtvorhandensein irgendeines Wertes des entsprechenden Attributtyps. Der Import mehrerer Attributwerte gleichzeitig auf einen Attributtyp ist nicht möglich, da hier nicht entschieden werden kann welcher der Attributwerte verwendet werden soll.

Beispiel: Angenommen, wir haben eine Import-Tabelle, die Musiker mit ihren alias-Namen beinhaltet. Einige Musiker haben auch mehrere alias-Namen. Hier können wir die Einstellung "Neu anlegen wenn noch nicht vorhanden" nicht verwenden, da dann alle Musiker mit

mehreren alias-Namen keinen erhalten würden.

1.5.1.5.8 Nicht importieren

Mit der Import-Einstellung "Nicht importieren" können wir sagen, dass ein Objekt oder eine Eigenschaft nicht importiert werden soll. Dies ist dann nützlich, wenn wir bereits eine Abbildung definiert haben und diese wiederverwenden wollen, jedoch bestimmte Objekte und Eigenschaften nicht noch einmal importieren wollen.

1.5.1.5.9 Synchronisieren

Die Import-Einstellung "Synchronisieren" ist mit Vorsicht zu genießen, denn sie betrifft als einzige Import-Einstellung nicht nur die Objekte und Eigenschaften in i-views, die in ihren Werten mit denen der Import-Tabelle übereinstimmen, sondern darüber hinaus alle Elemente des gleichen Typs in i-views. Wenn man eine Import-Tabelle mit i-views synchronisiert, bedeutet das prinzipiell, dass das Ergebnis in i-views nach dem Import exakt so aussehen soll wie in der Tabelle.

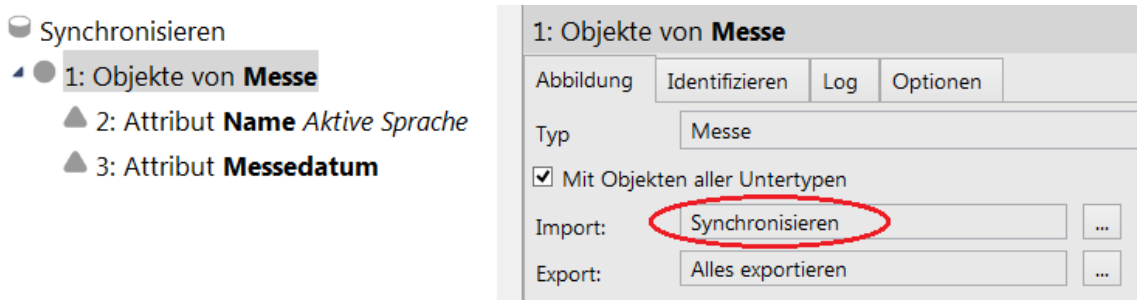
Wenn Objekte eines Typs synchronisiert werden, werden alle Objekte dieses Typs gelöscht, die nicht in der Import-Tabelle vorkommen. Die Objekte, die vorkommen, werden aktualisiert und die Objekte, die nicht in i-views vorkommen, werden neu angelegt.

Beispiel: Wir wollen die Musikmessen in i-views (links) mit einer Tabelle mit den Messen und ihrem Datum (rechts) synchronisieren:

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

Für die Objekte des Typs Messe wählen wir die Import-Einstellung "Synchronisieren", für die einzelnen Attribute *Name* und *Messedatum* wird die Import-Einstellung "Aktualisieren oder neu anlegen, wenn nicht vorhanden" verwendet:



Das Attribut Name ist das identifizierende Attribut von Messe. Für das Objekt Musikmesse 2015 fehlt der Name in der Import-Tabelle. Importieren wir die Tabelle so, erhalten wir dazu eine Fehlermeldung:

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

Nach dem Import sehen wir nun, dass durch den Import zwei Objekte entfallen sind, die keine Entsprechung in der Import-Tabelle hatten. Das Datum bei Musikmesse 2016 wurde

aktualisiert:

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

Wenn **Attribute** synchronisiert werden gilt folgendes: Wenn ein bestehendes Attribut durch einen Import keinen Wert erhält, wird es für das entsprechende Objekt der Import-Tabelle gelöscht. Wenn das bestehende Attribut einen anderen Wert hat als in der Import-Tabelle, wird es aktualisiert, auch dann, wenn es mehrmals vorkommen darf. Wenn das Attribut noch nicht vorhanden ist, wird es neu angelegt.

Wenn **Relationen** synchronisiert werden, und sie erhalten keinen Wert werden sie für das entsprechende Objekt gelöscht. Wenn die bestehende Relation einen anderen Wert hat, als in der Import-Tabelle, wird sie aktualisiert. Sollte es das Zielobjekt noch nicht in der Datenbank geben, wird es neu angelegt, vorausgesetzt, das Zielobjekt hat eine entsprechende Import-Einstellung zugewiesen bekommen. Kann das Zielobjekt nicht neu angelegt werden, da hier beispielsweise die Import-Einstellung "Aktualisieren" zugewiesen wurde erscheint eine Fehlermeldung, die uns mitteilt, dass das Zielobjekt nicht gefunden wurde und es wird nicht neu angelegt.

1.5.1.6 Tabellenspalten

Bei Abbildungen von Datenbank-Queries sind die Spalten, die zum Import zur Verfügung stehen, durch die Datenbanktabellen bzw. durch das Select-Statement vorgegeben. Beim Abbilden von Dateien können die Spalten mit der Schaltfläche "Aus Datenquelle lesen" aus der Datei übernommen werden. Man kann sie aber auch von Hand angeben. Dann hat man die Wahl, ob man eine Standard-Spalte oder eine virtuelle Eigenschaft anlegen möchte.

Will man aus der semantischen Graph-Datenbank exportieren, muss man die Spalten von Hand eingeben. Es können nur Standard-Spalten, nicht jedoch virtuelle Spalten exportiert werden.

Virtuelle Tabellenspalte / Virtuelle Eigenschaft
Virtuelle Spalten sind zusätzliche Spalten, die es erlauben die Inhalte, die wir in einer Spalte der zu importierenden Tabelle vorfinden, mit regulären Ausdrücken zu transformieren. Beispiel: Nehmen wir an in unserer Import-Tabelle steht bei den Jahreszahlen immer ein a.d. dahinter. Das können wir bereinigen, indem wir eine virtuelle Spalte anlegen, die aus der Spalte Jahr nur die ersten 4 Zeichen übernimmt.

Auch beim Export können wir virtuelle Eigenschaften definieren.

Den reguläre Ausdruck schreiben wir dabei einfach in die Spaltenüberschrift (in den Namen der Spalte). Dabei werden Teilzeichenketten, die in spitze Klammern <...> eingeschlossen sind, nach den folgenden Regeln ersetzt, wobei *n, n1, n2, ...* für die Inhalte anderer Tabellenspalten mit der Spaltennummer n stehen.

Ausdruck	Beschreibung	Beispiel	Eingabe	Ausgabe
<n**p**>	Druckausgabe des Inhalts von Spalte n	Treffer: <1p>	1 (integer) 'keine' (String)	Treffer: 1 Treffer: 'keine'

<n**s**>	Ausgabe der Zeichenkette in Spalte n	Hallo <1s>!	'Peter'	Hallo Peter!
<n**u**>	Ausgabe der Zeichenkette in Spalte n in Großbuchstaben	Hallo <1u>!	'Peter'	Hallo PETER!
<n**l**>	Ausgabe der Zeichenkette in Spalte n in Kleinbuchstaben	Hallo <1l>!	'Peter'	Hallo peter!
<n**c**start-stop>	Teilzeichenkette von Position start bis stop aus Spalte n	<1c3-6> <1c3> <1c3->	'Spalten'	alte ten alten
<n**m**regex>	Test, ob der Inhalt von Spalte n den regulären Ausdruck regex matcht. Die folgenden Ausdrücke werden nur ausgewertet, wenn der reguläre Ausdruck zutrifft.	<1m0[0-9]>hi <1m$>test	01 123 (leer) 123	hi (leer) test (leer)
<n**x**regex>	Test, ob der Inhalt von Spalte n den regulären Ausdruck regex matcht. Die folgenden Ausdrücke werden nur ausgewertet, wenn der reguläre Ausdruck **nicht** zutrifft.	<1x0[0-9]>hallo	01 123	(leer) hallo
<n**e**regex>	Selektiert alle Treffer von regex aus dem Inhalt von Spalte n. Einzeltreffer sind im Ergebnis durch Komma voneinander getrennt.	<1eL+> <1e\d\d\d\d>	HELLO WORLD 02.10.2001	LL,L 2001
<n**r**regex>	Entfernt alle Treffer von regex aus dem Inhalt von Spalte n	<1rL>	HELLO WORLD	HEO WORD
<n**g**regex>	Überträgt den Inhalt aller Gruppen des regulären Ausdrucks	<1g\+(\d+)\->	+42-13	42

<n**f**format>	Formatiert Zahlen, Datums- und Zeitangaben aus Spalten gemäß der Formatangabe 'format'	<1f#,0.00>	3,1412 1234,5	3,14 1.234,50
		<1fd/m/y>	1. Mai 1935	1/5/1935
		<1fdd/mmm>	1. Mai 1935	01/Mai

Tabellespalten können auch unabhängig von ihrer Spaltennummer referenziert werden, indem eigens definierte Bezeichner verwendet werden. Der Vorteil hierin ist, dass bei einer Änderung der Spalten-Reihenfolge der Importtabelle die Zuordnung nicht verloren geht.

Der Bezeichner für die jeweilige Spalte der Importtabelle wird in die Spalte mit der Überschrift *Bezeichner* der Spaltendefinitionstabelle eingetragen. Referenziert werden diese Spalten durch Anlegen einer virtuellen Tabellenspalte, die den Bezeichner als Tabellenspalten-Überschrift enthalten (siehe Beispiel 2).

Ausdru	**Beschreibung**	**Be**	**Ein**	**Ausgabe**
<**$**name**$**regex>	Referenz auf eine Spalte mittels eindeutigem Spalten-Bezeichner *name* und anschließender Transformierung durch regulären Ausdruck *regex*. Die **$**-Zeichen sind funktioneller Bestandteil der Bezeichner-Syntax.	<$NCompany$SOMPANY #1>		

Beispiel 1: Verwendung von regulären Ausdrücken (Referenz über Spaltennummer)

Angenommen wir haben eine Import-Tabelle, in der konkrete Objekte ohne Namen vorkommen. In unserem Datenmodell sollen diese Objekte jedoch als eigene Objekte modelliert werden. Ein Beispiel: zu einem Lastpunkt steht in Spalte 88 sein Hauptwert, der Drehmoment. Als Definition unserer virtuellen Spalte, die für den Namen dieses Lastpunktes stehen soll, geben wir also den Ausdruck *Lastpunkt* <*88s*> ein. Der daraus entstehende Name für einen Lastpunkt mit dem Drehmoment von 850 wäre demnach "Lastpunkt 850".

Wir können die virtuelle Eigenschaft auch nutzen, um einen Usernamen herzustellen, der aus den ersten 4 Buchstaben des Vornamens und des Nachnamens zusammengesetzt ist. Heißt die Person Maximilian Mustermann und wir definieren die virtuelle Spalte mit dem entsprechenden Ausdruck <1c1-4><2c1-4>, erhalten wir das Ergebnis "MaxiMust".

Die virtuelle Eigenschaft kann auch dazu genutzt werden, einem User beim Import ein initiales Passwort anzulegen. Der Ausdruck könnte *Pass4*<*2s*> lauten. Das daraus resultierende Passwort für Maximilian Mustermann wäre "Pass4Mustermann".

Ein etwas umfangreicheres Beispiel zeigt, wie die virtuelle Eigenschaft dazu genutzt werden kann, Objekten die korrekte direkte Obergruppe zuzuordnen:

Tabelle

#	Gruppe	Nomenklatur mdı	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

Die drei rechten Spalten sind virtuelle Spalten.

<1mUG>: In die erste der virtuellen Spalten wird die Nummer der Obergruppe des Objekts nur geschrieben, wenn der Begriff "UG" (für Untergruppe) in der ersten Spalte für das Objekt vorkommt.

<2c1-3>000: Die Nummer die in die Spalte geschrieben werden soll, setzt sich aus den ersten drei Zeichen der zweiten Spalte zusammen und drei Nullen.

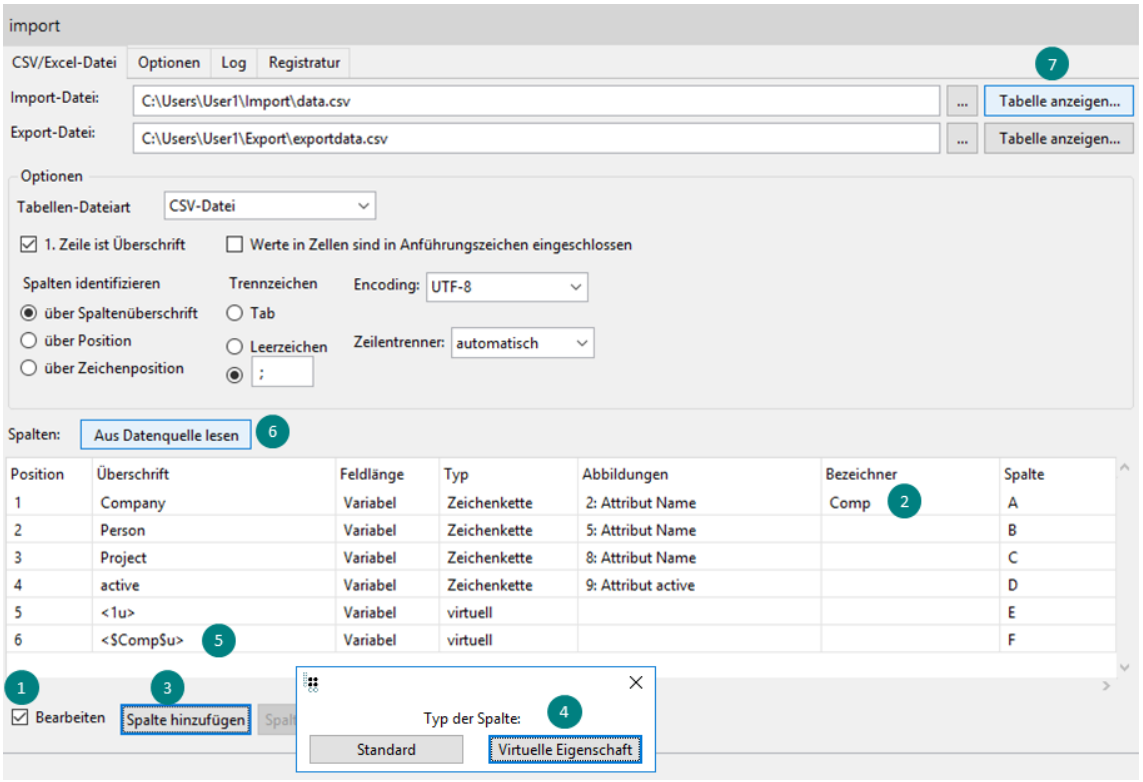
<1m>: Nur wenn die erste Spalte für das Objekt leer ist, also keinen Wert enthält, wird die Nummer der Obergruppe des Objekts in die Spalte geschrieben.

<2c1-4>00: Die Nummer, die in die Spalte geschrieben werden soll, setzt sich aus den ersten vier Zeichen der zweiten Spalte zusammen.

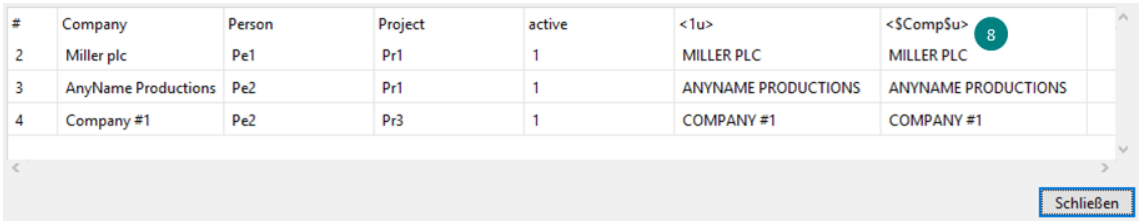
Heimtextil 2016: Dieser Ausdruck wird für alle Objekte in die Spalte geschrieben.

Beispiel 2: Verwendung individueller Bezeichner (in Kombination mit regulären Ausdrücken)

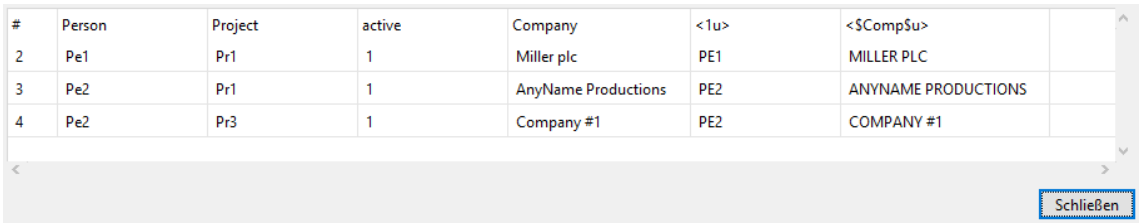
Im folgenden Beispiel werden die Inhalte der Spalte **Company** mittels virtueller Spalten in Großbuchstaben transformiert: Spalte 5 verwendet eine Referenz per Spalten-Nummer, Spalte 6 verwendet eine Referenz per Spalten-Bezeichner.



Ein Klick auf die Vorschau zeigt die transformierten Spalteneinträge:



Folgende Abbildung zeigt die Auswirkung von vertauschten Spalten einer Importtabelle: Während bei der alleinigen Verwendung von regulären Ausdrücken (<1u>) die falsche Spalte transformiert wird, bleibt bei Verwendung eines Bezeichners mit nachgelagertem regulären Ausdruck (<$Comp$u>) der Inhalt gleich.



1.5.1.7 Configuration of further table-oriented data sources

Datenbanken

In einem Mapping für eine PostgreSQL-, Oracle- oder ODBC-Schnittstelle müssen die Datenbank, der Benutzer und das Passwort angegeben werden.

Angabe der Datenbank

Die Angabe für die Datenbank setzt sich aus Name des Host, dem Port und dem Namen der Datenbank zusammen. Die Syntax lautet.

Datenbank-System	Angabe der Datenbank
PostgreSQL	hostname:port_datenbank
Oracle	//hostname:[port][/datenbankService]
ODBC	Name der konfigurierten Datenquelle
MySQL	Getrennte Konfiguration von Datenbank und Hostname

Benutzername und Passwort konfigurieren

Benutzername und Passwort werden so angegeben, wie sie in der Datenbank abgelegt sind. Unter dem Punkt Tabelle kann die Tabelle angegeben werden, die importiert werden soll. Für den Import besteht aber auch die Möglichkeit, dass unter dem Punkt "Query" eine Query formuliert wird, die angibt, welche Daten importiert werden sollen.

Encoding

Handelt es sich um ein PostgreSQL-Mapping, dann kann auf dem Reiter "Encoding" das Encoding angegeben werden.

Spezielle Anforderungen der Oracle-Schnittstelle

Die Funktion zum direkten Import aus einer Oracle Datenbank setzt voraus, dass auf dem importierenden Rechner bestimmte Laufzeit-Bibliotheken installiert sind.

Direkt benötigt wird das "Oracle Call Interface" (OCI) und zwar in einer Version, die laut Oracle zu dem Datenbankserver passt, der angesprochen werden soll. D.h., um eine Oracle 11i Datenbank anzusprechen, sollte auf dem importierenden Rechner das OCI in Version 11 installiert sein. Das OCI lässt sich am einfachsten installieren, wenn man den "Oracle Database Instant Client" installiert. Die Package Version "Basic" ist ausreichend. Der Client ist entweder vom Serverbetreiber zu bekommen oder von Oracle nach Registrierung unter http://www.oracle.com/technology/tech/oci/index.html ladbar.

Nach der Installation ist sicher zu stellen, dass die Bibliothek für den importierenden Client auffindbar ist, entweder indem sie im gleichen Verzeichnis liegt oder für das entsprechende Betriebssystem passende Umgebungsvariablen definiert werden (ist beim OCI dokumentiert).

Je nach Betriebssystem auf dem der Import stattfinden soll, sind weitere Bibliotheken notwendig, die nicht immer installiert sind.

- MS Windows: neben der benötigten "oci.dll" sind noch zwei weitere Bibliotheken notwendig: advapi32.dll (Erweitertes Windows 32 Base-API) und mscvr71.dll (Microsoft C Runtime Library)

Bis auf den XML-Import/Export sind alle Importe/Exporte tabellenorientiert und unterscheiden sich nur in der Konfiguration der Quelle. Für die Beschreibung einer tabellenorientierten Abbildung kann das Beispiel der CSV-Datei herangezogen werden.

1.5.1.8 Mapping of a XML file

Das Prinzip von XML-Dateien ist, die unterschiedlichen Angaben zu einem Datensatz über Tags (<>) explizit zu machen (nicht über Tabellenspalten). Dementsprechend sind Tags auch die Grundlage der Abbildung beim Import von XML-Strukturen in i-views.

Ein Beispiel: Nehmen wir an, unsere Liste von Songs liegt als XML-Datei vor:

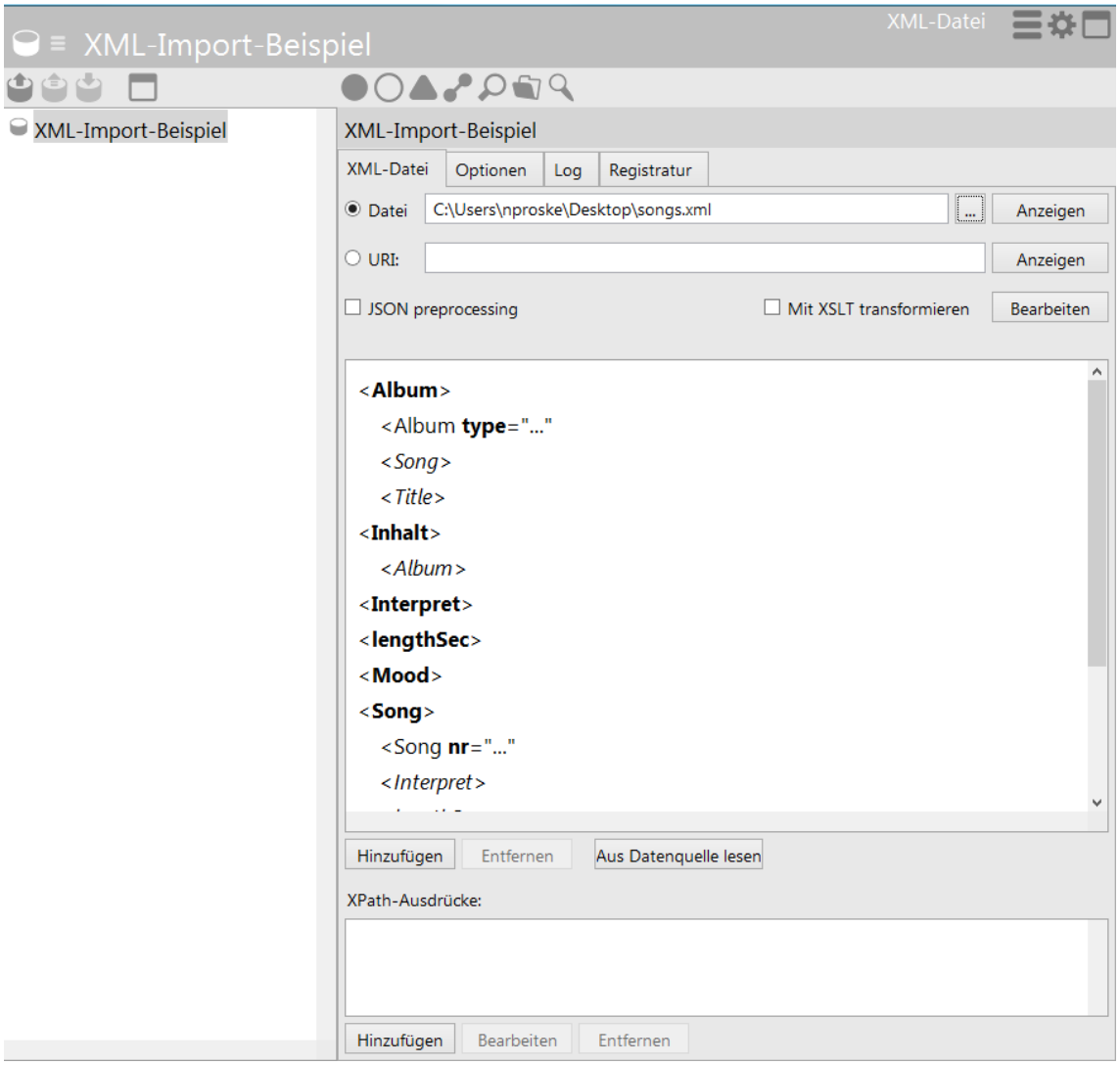
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Inhalt>
    <Album type="Oldie">
        <Title>Revolver</Title>
        <Song nr="1">
            <Title>Eleanor Rigby</Title>
            <lengthSec>127</lengthSec>
            <Interpret>The Beatles</Interpret>
            <Thema>Mental illness</Thema>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

Wenn wir nun diese XML-Datei importieren wollen, wählen wir bei der Auswahl des Typs der Datenquelle "XML-Datei" aus, wodurch sich der Editor für den Im- und Export von XML-Dateien öffnet. Bereits in der Angabe des Dateistandortes gibt es Unterschiede zum Editor für CSV-Dateien. Wir können nun zwischen einem lokalen Dateipfad und der Angabe einer URI wählen.

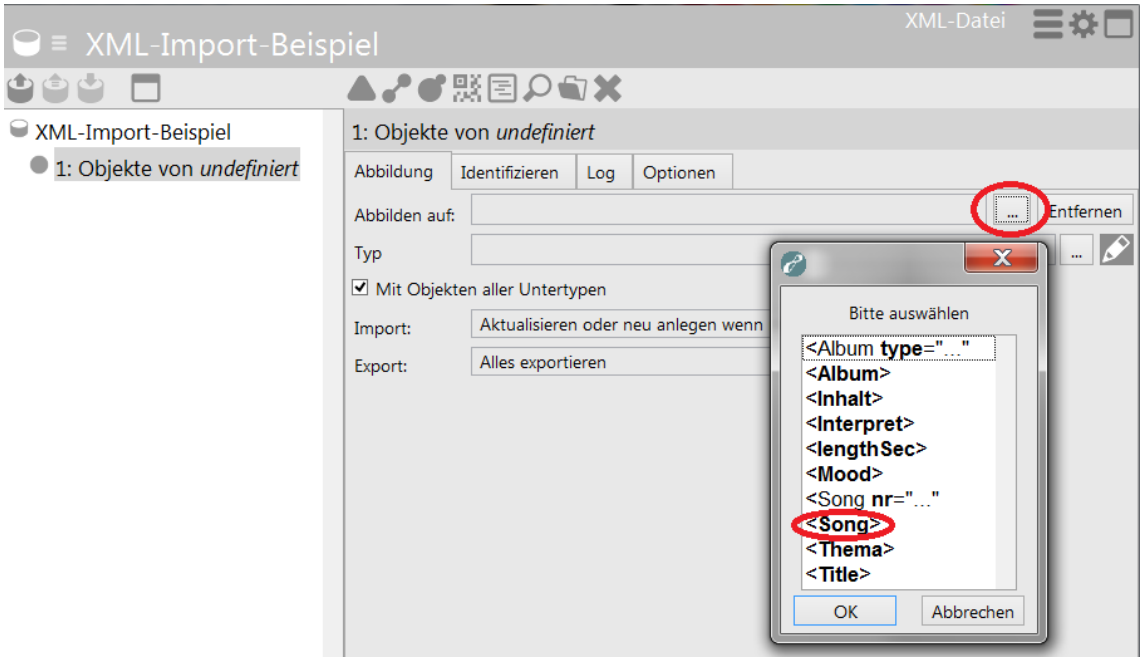
JSON preprocessing ermöglicht das Umwandeln einer JSON-Datei in XML vor dem eigentlichen Import.

Mit XSTL transformieren kann man auswählen, wenn man die XML-Daten aus der ausgewählten XML-Datei noch vor dem Import in andere XML-Daten umwandeln möchte, um beispielsweise die Struktur zu ändern oder einzelne Werte weiter aufzutrennen. Über die Schaltfläche "Bearbeiten" öffnet sich die XML-Datei, in der man die Änderungen mittels XSLT definieren kann.

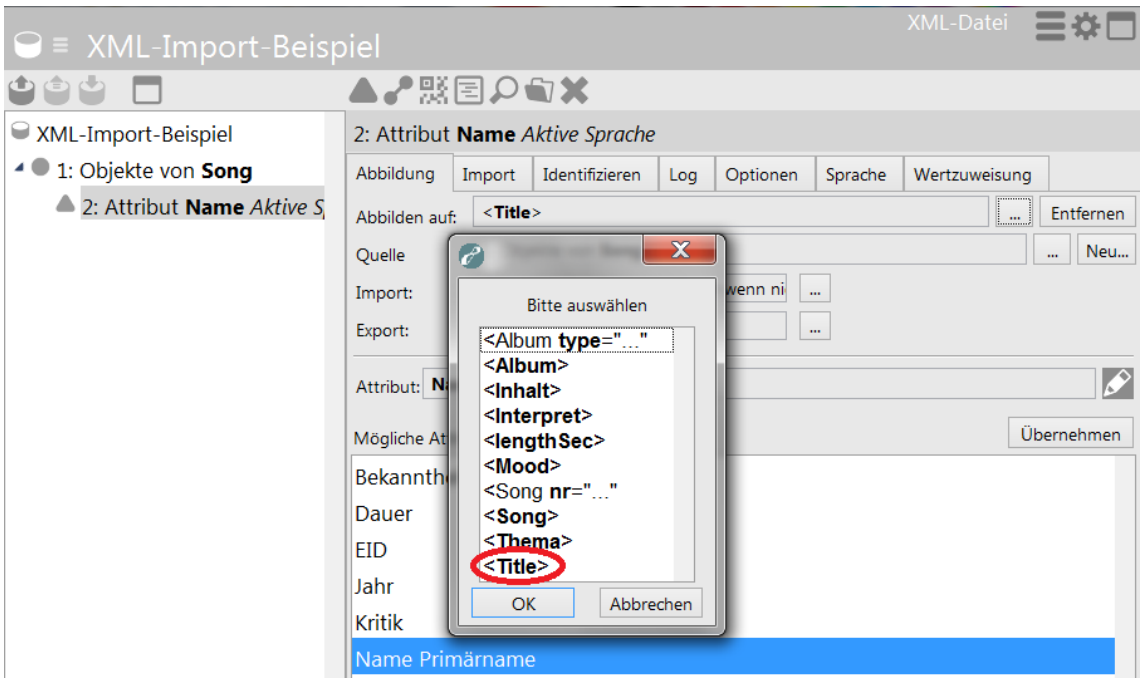
Ist die Datei ausgewählt, können wir mit dem Button "Aus Datenquelle lesen" die XML-Struktur auslesen lassen, die uns daraufhin in rechten Fenster angezeigt wird.



Wir wollen die einzelnen Songs unserer Liste importieren. Darum legen wir eine neue Objektabbildung an und wählen über den Button bei "Abbilden auf" den Tag <*Song*> aus. Im Gegensatz zum CSV-Import, bei dem nur Attributwerte eine Entsprechung in der CSV-Tabelle finden und eine einzelne Zeile für ein Objekt steht, sodass auch nur die Attributwerte abgebildet werden müssen, erfolgt das Mapping der semantischen Objekte hier über die XML-Struktur. Darum müssen auch für alle abzubildenden Objekte jeweils ein entsprechendes Tag der XML-Datei angegeben werden.



Wie in unserem Beispiel, sind die Tags ohne Kontext nicht immer eindeutig: <Title> wird sowohl für Titel von Alben als auch für Songtitel verwendet. Erst in Kombination mit dem umgebenden Tag wird der Objekttyp klar. Oft laufen der Kontext der XML-Struktur und der Kontext der Abbildungshierarchie synchron: Da wir nun bereits festgelegt haben, dass die Objekte auf den Tag <Song> abgebildet werden sollen, ist durch die XML-Struktur klar, welcher <Title>-Tag nun gemeint ist, wenn wir <Title> mit dem Namensattribut von Songs mappen. Dort, wo Abbildungshierarchie und Tagstruktur nicht parallel laufen, können wir im XML-Import zusätzlich zu den in der XML-Datei vorkommenden Tags Ketten bilden - mit XPath.

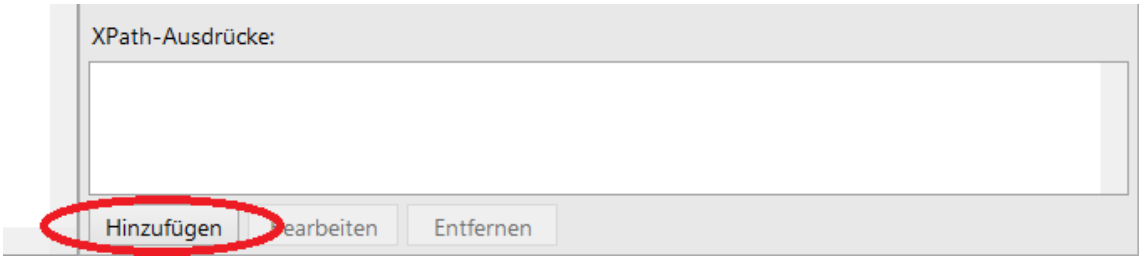


Wie auch beim CSV-Import muss über den Reiter "Identifizieren" bei der Objektabbildung festgelegt werden, durch welche Attributwerte das Objekt in der semantischen Graph-Datenbank

identifiziert werden soll. Das erste angelegte Attribut für ein Objekt wird auch hier wieder automatisch als identifizierendes Attribut verwendet.

Möglichkeiten mit XPath-Ausdrücken

Angenommen wir würden nur Songs aus Alben des Musik-Stils "Oldie" importieren wollen. In unserem XML-Dokument ist die Information über den Musik-Stil direkt im Album-Tag angegeben unter *type="..."*. Im Editor müssen wir also einen XPath-Ausdruck definieren, der den Pfad im XML-Dokument beschreibt, der nur diejenigen Songs enthält, die aus Oldie-Alben stammen. Im rechten unteren Bereich des Editors finden wir ein Feld zum hinzufügen von XPath-Ausdrücken.



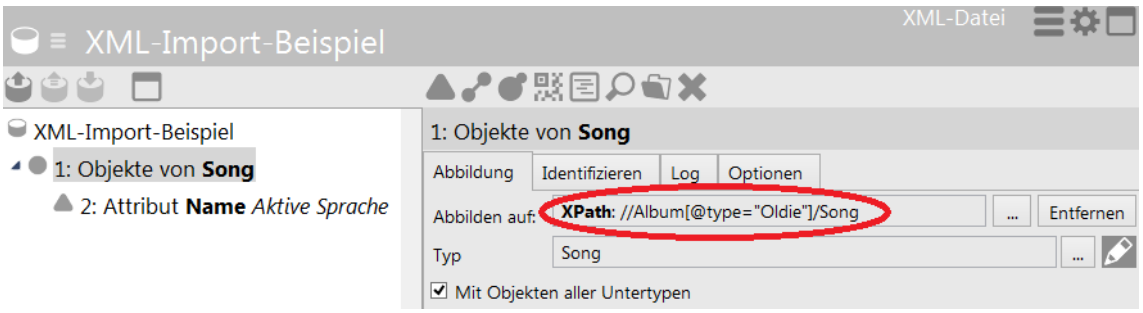
Der passende XPath-Ausdruck lautet:

```
//Album[@type="Oldie"]/Song
```

Erklärung im Einzelnen:

//Album	Selektiert alle Alben, wobei es keine Rolle spielt, wo sie sich im Dokument befinden.
Album[@type="Oldie"]	Selektiert alle Alben vom Typ "Oldie"
Album/Song	Selektiert alle Songs, die Subelemente von Alben sind.

Diesen Ausdruck können wir nun verwenden, um eine Entsprechung für die Objektabbildung der Songs zu definieren.

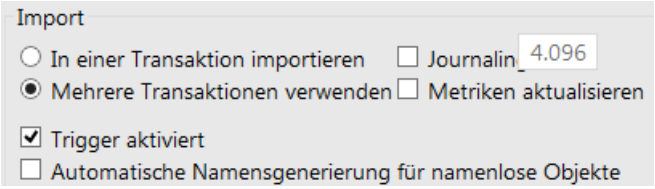


Mit XPath stehen uns außerdem viele weitere nützliche Selektions-Funktionen zur Verfügung. So können wir beispielsweise Elemente über ihre Position im Dokument selektieren, Vergleichsoperatoren einsetzen, sowie alternative Pfade angeben.

1.5.1.9 Weitere Optionen, Log und Registratur

1.5.1.9.1 Weitere Optionen beim Import

Im Reiter "Optionen" stehen uns folgende Funktionen zur Auswahl, die unabhängig von der Datenquelle sind:



In einer Transaktion importieren: Ist langsamer als ein Import mit mehreren Transaktionen und sollte darum nur dann verwendet werden, wenn es beim Import mit mehreren Transaktionen ansonsten zum Konfliktfall kommen kann, wenn viele Personen zur selben Zeit im Knowledge-Builder arbeiten oder wenn man Daten importieren will, bei denen es eine Rolle spielt, dass sie nicht getrennt voneinander betrachtet werden. Beispiel 1: Es erfolgt stündlich ein Import mit dem Status der Auslastung von Maschinen. Die addierten Werte der Auslastung dürfen einen gewissen Wert nicht übersteigen, da es ansonsten eventuell zum Stromausfall kommen kann. Damit diese Regel (z.B. mithilfe eines Skripts) berücksichtigt werden kann, müssen alle Werte gemeinsam betrachtet und dann importiert werden. Beispiel 2: Es erfolgt ein Import mit Personen, von denen höchstens eine den Hauptschlüssel haben kann, weil nur ein Hauptschlüssel existiert. Auch hier muss der Import in einer Transaktion erfolgen, da bei mehreren Transaktionen der Fehler übersehen werden könnte, dass bei zwei Personen das Attribut für den Hauptschlüsselbesitz gesetzt wurde.

Mehrere Transaktionen verwenden: Standardeinstellung für einen schnellen Import.

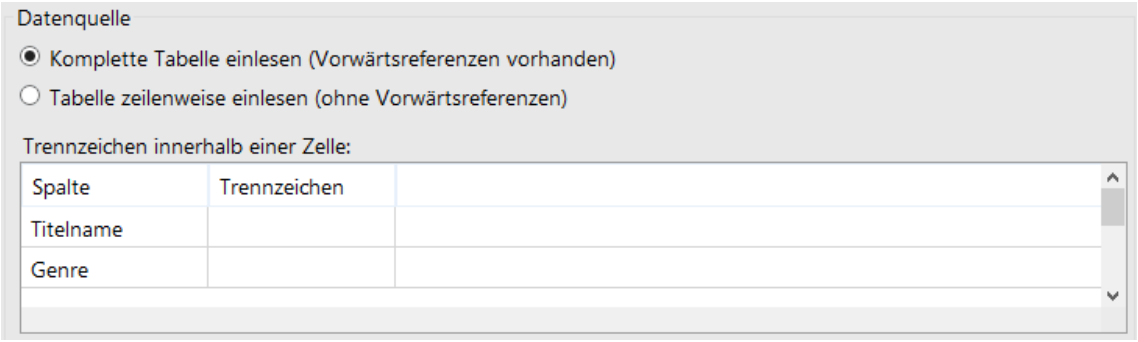
Journaling: Das Journaling sollte verwendet werden, wenn extrem viele Daten mit einem Import gelöscht oder geändert werden. Erst nach jeweils 4.096 Einträgen (die Zahl ist variabel), sollen die Änderungen, bzw. Löschungen für diese Einträge auch am Index vorgenommen werden. Dadurch wird der Import beschleunigt, da nicht für jede einzelne Änderung/Löschung der Index herangezogen werden muss, sondern spätestens nach 4.096 Veränderungen diese gemeinsam in den Index übernommen werden.

Metriken aktualisieren: Die Metriken sollten aktualisiert werden, wenn der Import eine große Auswirkung auf die Menge von Objekttypen oder Eigenschaftstypen hat, wenn also sehr viele Objekte oder Eigenschaften eines Typs der semantischen Graph-Datenbank hinzugefügt werden. Würden die Metriken nicht aktualisiert, könnte dies negative Auswirkungen auf die Performance von Suchen haben, in denen die entsprechenden Typen eine Rolle spielen.

Trigger aktiviert: Ob Trigger beim Import aktiviert sein sollen oder nicht kann hier über das Häkchen bestimmt werden. Falls es gewünscht ist, dass ein Trigger greift und ein anderer nicht, müssen zwei verschiedene Abbildungen mit den entsprechenden semantischen Elementen definiert werden. Informationen zu Triggern stehen im Kapitel Trigger zur Verfügung.

Automatische Namensgenerierung für namenlose Objekte: Ermöglicht die automatische Namensgenerierung für namenlose Objekte.

Liegt eine tabellenorientierte Quelle vor, können wir folgende Einstellungen vornehmen:



Komplette Tabelle einlesen: Obwohl es länger dauern kann, die komplette Tabelle auf einmal einzulesen, macht es Sinn diese Option auszuwählen, wenn Vorwärtsreferenzen vorhanden sind, d.h., wenn Relationen zwischen den zu importierenden Objekten gezogen werden sollen. In diesem Fall müssen nämlich beide Objekte bereits vorhanden sein, was beim zeilenweisen Einlesen der Tabelle nicht der Fall ist. Zudem ist die Fortschrittsanzeige genauer als beim zeilenweisen Einlesen.

Tabelle zeilenweise einlesen: Das zeilenweise Einlesen der Tabelle sollte immer dann verwendet werden, wenn keine Querreferenzen in der Tabelle vorhanden sind, da der Import so schneller geht.

Trennzeichen innerhalb einer Zelle: siehe Kapitel Abbildungen von mehreren Werten für einen Objekttyp bei einem Objekt.

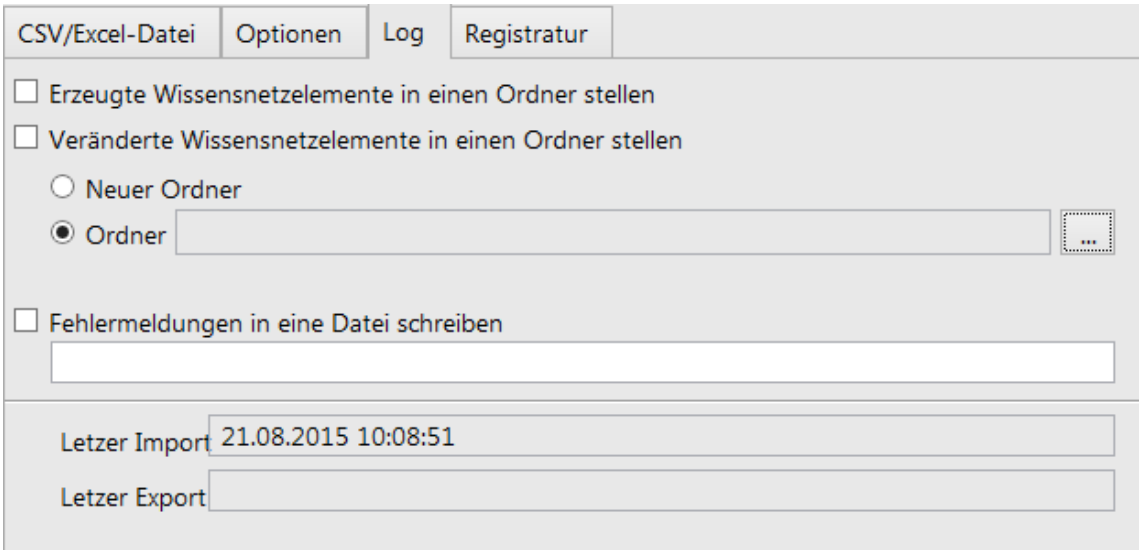
Liegt eine XML-basierte Datenquelle vor, stehen uns folgende Funktionen zur Verfügung:

Inkrementeller XML-Import: Der XML-Import erfolgt schrittweise**.** Die Schritte werden durch das partitionierende Element festgelegt.

DTD einlesen: Liest die Dokumenttypdefinition (DTD) ein**.**

1.5.1.9.2 Log

Die Funktionen im Reiter "Log" ermöglichen Änderungen, die beim Import vorgenommen werden, verfolgen zu können.

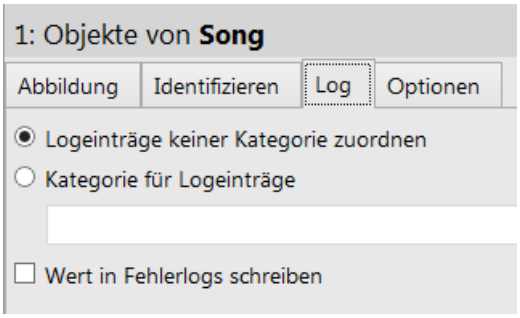


Erzeugte Wissensnetzelemente in einen Ordner stellen: Werden neue Objekte, Typen oder Eigenschaften durch den Import erzeugt, können diese in einen Ordner in der semantischen Graph-Datenbank gestellt werden.

Veränderte Wissensnetzelemente in einen Ordner stellen: Alle Eigenschaften oder Objekte, deren Eigenschaften sich durch den Import geändert haben, können in einen Ordner gestellt werden.

Fehlermeldungen in eine Datei schreiben: Beim Import können Fehler auftreten (z.B. kann es sein, dass ein identifizierendes Attribut für mehrere Objekte vorkam und daher das Objekt nicht eindeutig identifiziert werden konnte). Diese Fehler werden standardmäßig nach einem Import in einem Fenster angezeigt und man hat dann die Möglichkeit den Fehlerbericht zu speichern. Wenn dies automatisch passieren soll, kann hier der Haken gesetzt und eine Datei angegeben werden.

Letzter Import / Letzter Export: Hier werden das Datum und die Uhrzeit des zuletzt vorgenommen Imports und des zuletzt vorgenommen Export angezeigt.



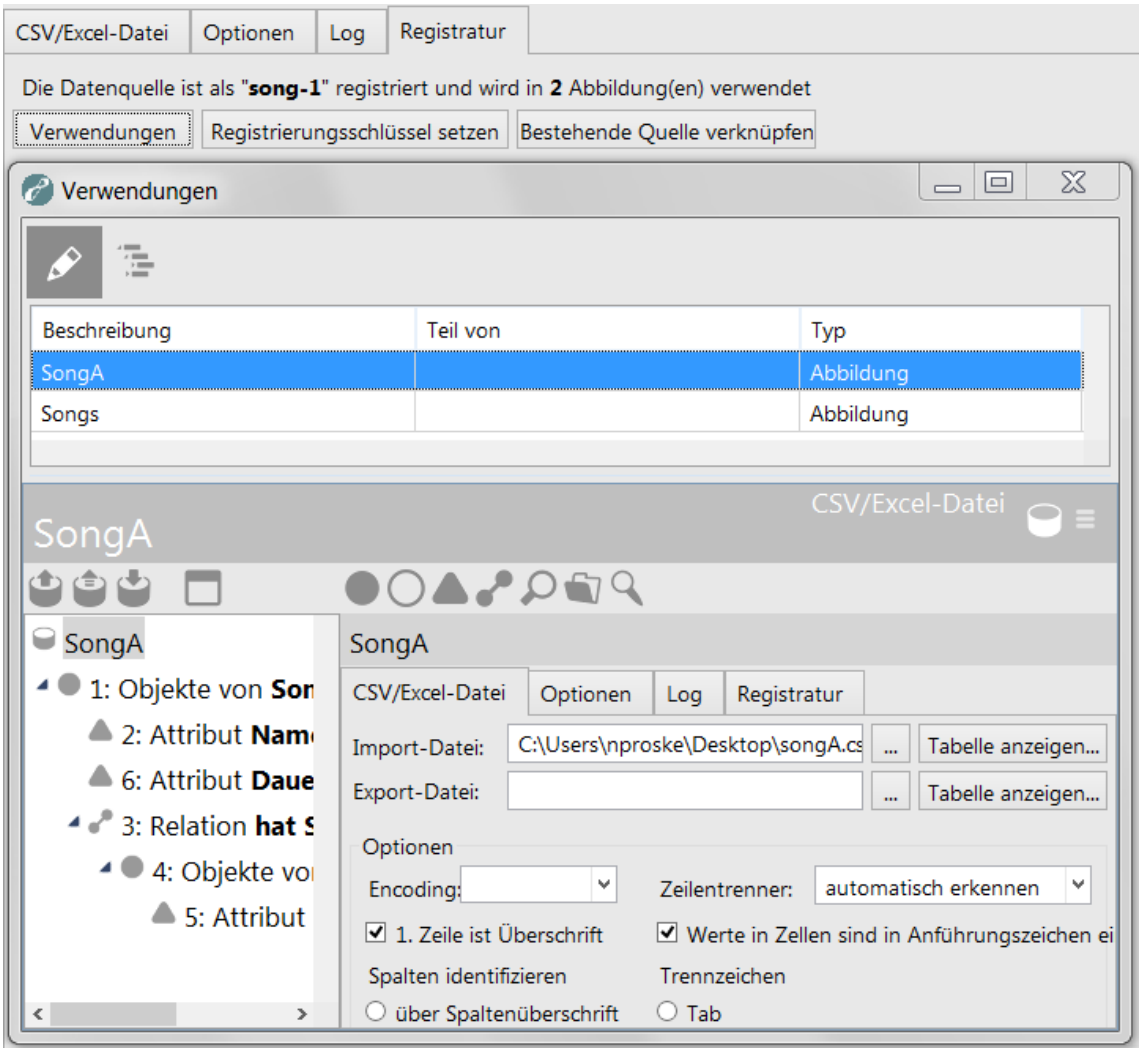
Auch bei den einzelnen Abbildungs-Objekten ist der Reiter "Log" verfügbar. Hier kann bei Bedarf eine Kategorie für Logeinträge eingetragen werden. Zudem kann festgelegt werden, dass der Wert des entsprechenden Objekts / der entsprechenden Eigenschaft in den Fehlerlog geschrieben werden soll. Dies ist standardmäßig nicht aktiviert, um das Offenlegen sensibler Daten (z.B. Passwörter) zu vermeiden.

1.5.1.9.3 Registratur

Unter dem Reiter "Registratur" findet man die Funktion "Registrierungsschlüssel setzen" mit der man die Datenquelle für andere Importe und Exporte registrieren kann.

Die Funktion "Bestehende Quelle verknüpfen" ermöglicht die Wiederverwendung einer registrierten Quelle.

Unter "Verwendungen" kann man einsehen, wo eine Datenquelle noch verwendet wird:



1.5.2 Attribute types and formats

Eine häufig auftretende Aufgabe einer Attributabbildung ist der Import bestimmter Daten von konkreten Objekten, beispielsweise von Personen: Telefonnummer, Geburtsdatum etc.

Beim Import von Attributen, für die i-views ein bestimmtes Format verwendet (z.B. Datum), müssen die Einträge der zu importierenden Spalte in einer Form vorliegen, die von i-views unterstützt wird. Beispielsweise kann in ein Attributfeld vom Typ Datum keine Zeichenkette der Form abcde... importiert werden; in einem solchen Fall wird für das entsprechende Objekt kein Wert importiert.

Die folgende Tabelle listet die von i-views unterstützten Formate beim Import von Attributen auf. Ein Tabellenwert ja oder 1 wird also beispielsweise korrekt als boolescher Attributwert (bei einem entsprechend definierten Attribut) importiert, ein Wert wie ein oder ähnliches hingegen nicht.

Attribut	Unterstützte Werte-Formate
Auswahl	Die Abbildung der Import- auf die Attributwerte kann über den Reiter "Wertzuweisung" konfiguriert werden.
Boolesch	Die Abbildung der Import- auf die Attributwerte kann über den Reiter "Wertzuweisung" konfiguriert werden.
Datei	Das Importieren von Dateien (z.B. Bildern) ist möglich. Dazu muss entweder der absolute Pfad zur Datei angegeben werden oder die zu importierenden Dateien müssen im gleichen (oder einem anzugebenden Unterverzeichnis) liegen wie die Import-Datei.
Datum	• <day> <monthName> <year>, z. B. 5 April 1982, 5-APR-1982 • <monthName> <day> <year>, z. B. April 5, 1982 • <monthNumber> <day> <year>, z. B. 4/5/1982 Das Trennzeichen zwischen <day>, <monthName> und <year> kann z.B. ein Leerzeichen, ein Komma oder ein Bindestrich sein (es sind aber noch weitere Zeichen möglich). Gültige Monatsnamen sind: • 'Januar', 'Februar', 'März', 'April', 'Mai', 'Juni', 'Juli', 'August', 'September', 'Oktober', 'November', 'Dezember' • 'Jan', 'Feb', 'Mrz', 'Mär', 'Apr', 'Mai', 'Jun', 'Jul', 'Aug', 'Sep', 'Okt', 'Nov', 'Dez'. Achtung: Zweistellige Jahreszahlen xy werden zu 20xy expandiert (aus 4/5/82 wird also 4/5/2082). Wenn das Mapping auf "Frei definierbares Format" eingestellt ist, können folgende Tokens verwendet werden: YYYY und YY (Jahr), MM und M (Monatsnummer), MMMM (Monatsname), MMM (abgekürzter Monatsname), DD und D (Tag)
Datum und Uhrzeit	Für Datum und Uhrzeit siehe die jeweiligen Attribute. Das Datum muss vor der Uhrzeit stehen. Wenn die Uhrzeit weggelassen wird, wird 0:00 verwendet.
Farbe	Import nicht möglich.
Festkommazahl	Import möglich.
Ganzzahl	• Ganzzahlen beliebiger Größe • Fließkommazahlen (mit Punkt getrennt), z.B. 1.82. Die Zahlen werden beim Import gerundet.
Internet-Verknüpfung	Jede beliebige URL möglich.

Uhrzeit	<hour>: <minute>: <second> <am/pm>, z.B. 8:23 pm (wird zu 20:23:00) <minute>, <second> und <am/pm> können weggelassen werden. Wenn das Mapping auf "Frei definiertes Format" eingestellt ist, könne folgende Tokens verwendet werden: hh und h (Stunde), mm und m (Minute), ss und s (Sekunde), mmm (Millisekunde)
Zeichenkette	Jede beliebige Zeichenkette. Es wird keine Dekodierung vorgenommen.

Boolesche Attribute und Auswahlattribute

Auswahl- oder boolesche Attribute können nur Werte aus einer vorgegebenen Menge annehmen; bei Auswahlattributen ist dies eine vorgegebene Liste, bei booleschen Attributen das Wertepaar ja/nein in Form eines Klickfelds. Beim Import dieser Attribute kann angegeben werden, wie die Werte aus der Import-Tabelle in Attributwerte der semantischen Graph-Datenbank übersetzt werden. Zum einen können die Werte so, wie sie in der Tabelle stehen, übernommen werden; entsprechen sie keiner der in der semantischen Graph-Datenbank definierten möglichen Werte des Attributs, werden sie nicht importiert. Zum anderen können Wertzuweisungen zwischen Tabellenwerten und Attributwerten, die dann importiert werden, festgelegt werden.

1.5.3 Export Configration

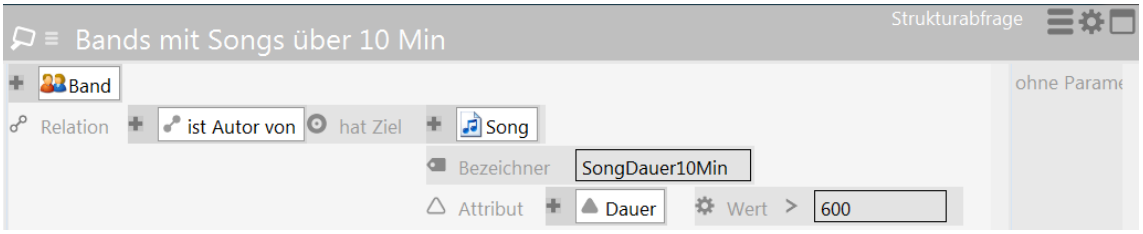
Der Export von Daten aus einer semantischen Graph-Datenbank in eine Tabelle wird in demselben Editor wie der Import und ganz analog vorbereitet:

1. Im einem Tabellen-Mapping-Ordner im Hauptfenster wird ein neues Mapping angelegt.
2. Im Tabellen-Mapping-Editor wird die zu erzeugende Datei angegeben.

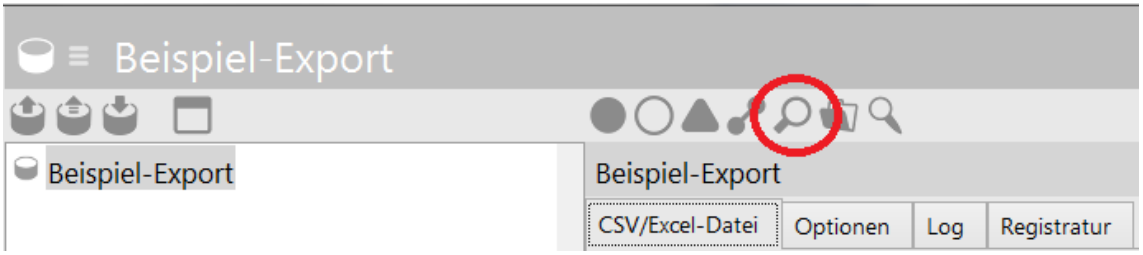
Der Unterschied zum Import liegt darin, dass die Spalten jetzt nicht aus der Tabelle eingelesen werden, sondern im Tabellen-Mapping-Editor angelegt werden müssen. Da der Import- und der Export-Editor derselbe sind, muss man beim Anlegen einer neuen Spalte zunächst auswählen, ob es sich um eine *Standard*-Spalte oder eine *Virtuelle Eigenschaft* handelt. Virtuelle Eigenschaften sind bei einem Export jedoch nicht verwendbar.

Export von Strukturabfragen

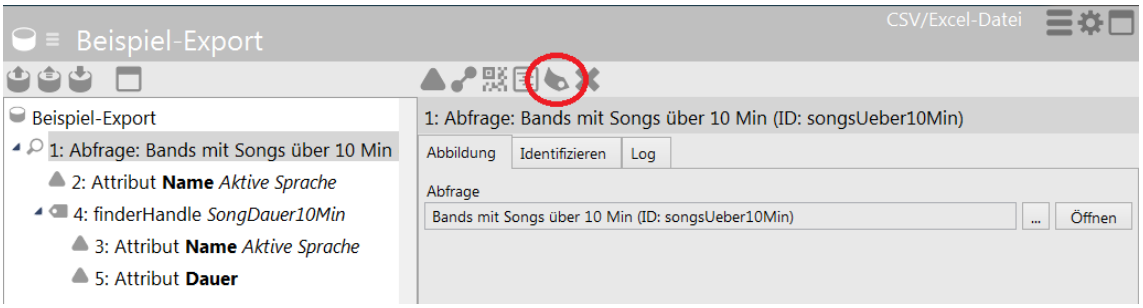
Es besteht die Möglichkeit das Ergebnis einer Strukturabfrage zu exportieren. Diese Vorgehensweise bietet sich an, wenn nur bestimmte Objekte, die durch eine Suche eingeschränkt werden, exportiert werden sollen. Nehmen wir als Beispiel an, wir wollten alle Bands, die Songs geschrieben haben, die länger als 10 min. dauern, exportieren. Dafür müssen wir zunächst eine Strukturabfrage definieren, die die gewünschten Objekte zusammenstellt.



Auf diese Strukturabfrage greifen wir dann von der Konfiguration des Exports aus zu. Dazu wählen wir im Kopf der Mapping-Konfiguration anstelle einer Objekt-Abbildung die Abbildung einer Abfrage. Die Strukturabfrage benötigt einen Registrierungsschlüssel, um auf sie zugreifen zu können.

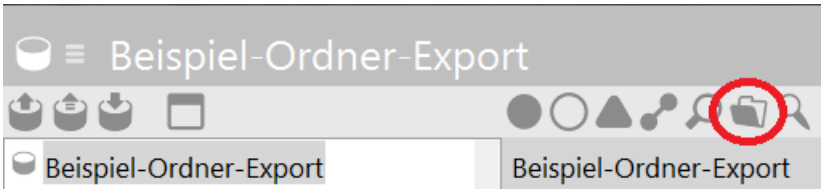


Damit werden nur noch die Ergebnisse der Strukturabfrage exportiert. Für diese Objekte können wir jetzt wieder Eigenschaften angeben, die in den Export mit aufgenommen werden sollen: z.B. Gründungsjahr der Band, Mitglieder und Songs. Jetzt kann es aber vorkommen, das wir von den Bands, die wir so zusammengestellt haben, nicht alle Songs exportieren wollen, sondern gerade nur die, die auch dem Suchkriterium entsprechen, in unserem Beispiel die Songs über 10 min. Dazu können wir die einzelnen Suchbedingungen in der Strukturabfrage mit Bezeichnern belegen. Diese Bezeichner können dann wiederum in der Export-Definition angesprochen werden.



Export von Sammlungen semantischer Objekte

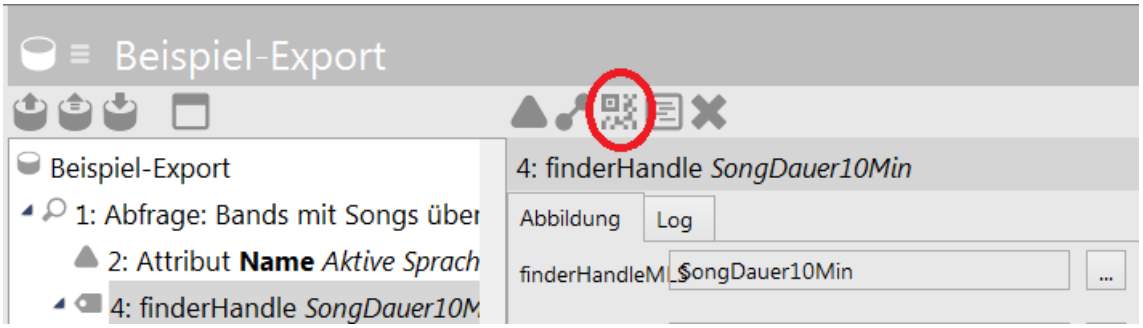
Auch Sammlungen semantischer Objekte können exportiert werden. Diese brauchen ebenfalls einen Registrierungsschlüssel, den man unter TECHNIK -> Strukturordner setzen kann.



Export der Frame-ID

Die Abbildung der Frame-ID ermöglicht es uns, die in der semantischen Graph-Datenbank für ein Wissensnetzelement vergebene ID, zu exportieren. Hierzu wählen wir einfach das

Objekt, den Typ oder die Eigenschaft aus, für die wir die ID brauchen und wählen dann den Button "Neue Abbildung der Frame-ID":



Wir können außerdem entscheiden, ob wir die ID im Format eines Strings wollen (ID123_456) oder, ob wir sie als 64 Bit Integer ausgegeben haben wollen.

Export mithilfe von Skripten

Schließlich steht uns beim Export noch ein weiteres mächtiges Werkzeug zur Verfügung: die Skriptabbildung. Informationen hierzu sind im Kapitel Die Skriptabbildung verfügbar.

Export-Aktionen bei Datenbankexporten

Die Abbildung der Eigenschaften eines Objekts wird für einen Export in eine Datenbank genauso vorgenommen wie für einen Import und wie für alle anderen Mappings. Einzig ist für den Export die Export-Aktion zu bestimmen. Diese gibt an, welche Art von Query in der Datenbank ausgeführt werden soll. Es stehen drei Export-Aktionen zur Verfügung:

Folgende Aktionen stehen in dem sich öffnenden Auswahldialog zur Verfügung:

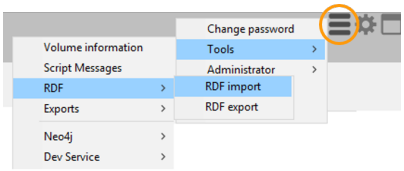
- **Datensätze in Tabelle neu anlegen**: Es werden neue Datensätze in der Datenbanktabelle hinzugefügt. Diese Aktion entspricht einem INSERT.
- **Existierende Datensätze aktualisieren**: Die Datensätze werden über eine ID in der Tabelle identifiziert. Sie werden nur überschrieben, wenn der Wert sich geändert hat. Gibt es keinen passenden Datensatz, dann wird ein neuer hinzugefügt. Diese Aktion entspricht einem UPDATE.
- **Tabelleninhalt beim Export überschreiben**: Alle Datensätze werden erst gelöscht und dann neu geschrieben. Diese Aktion entspricht einem DELETE auf der ganzen Tabelle mit folgendem INSERT.

1.5.4 RDF import and export

RDF is a standard format for semantic data models. RDF import and export allow us to share the data from i-views with other applications, but also to move data from one i-views knowledge network to another.

In the RDF export, the entire knowledge network is exported to an RDF file. The RDF import, on the other hand, is interactive and selective. That we can specify what should and should not be imported at the schema level as well as at the individual objects and properties.

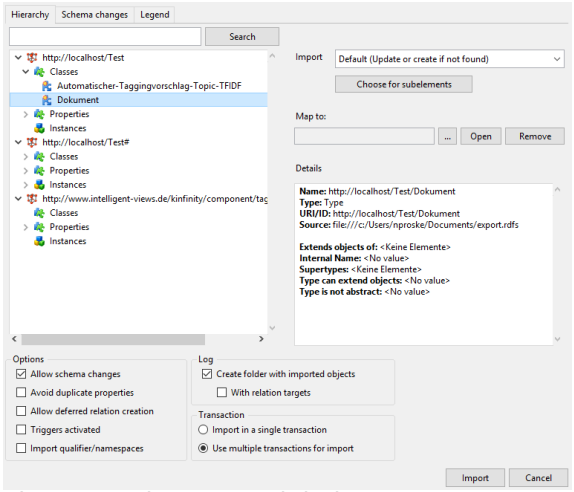
You can find the RDF import and RDF export in the actions menu of the knowledge builder:



Matching the objects from RDF with the existing objects in the semantic network

If the RDF data comes from the same schema as the network into which it is imported - e.g. from a backup - then the RDF import automatically maps objects and object types by their ID. In the import settings we can now determine the details, e.g. existing objects are to be updated by the import, new ones are created etc.

If the data comes from another source, the default setting is the import into its own sub-net. But we can also integrate this external information - by manual assignments with the mapping-to-function in the mapping interface.

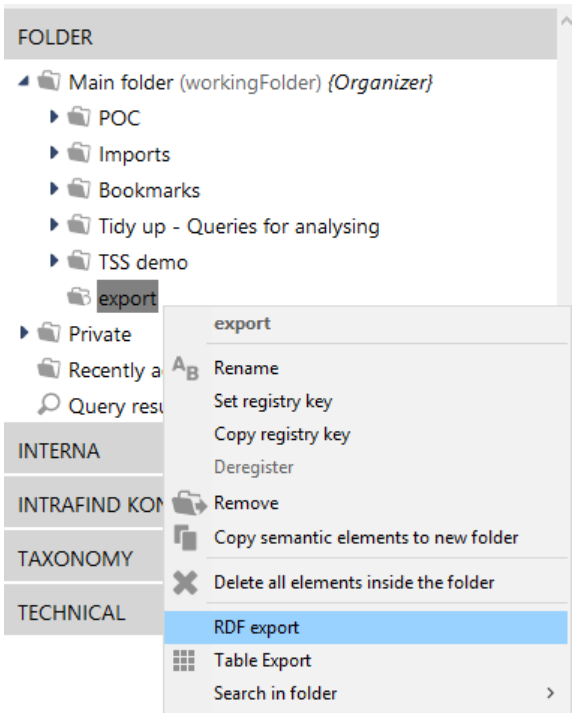


There are also some global settings: Do we even want to allow changes to the schema? Do we allow properties to be created multiple times? Finally, all schema changes are displayed on a separate tab.

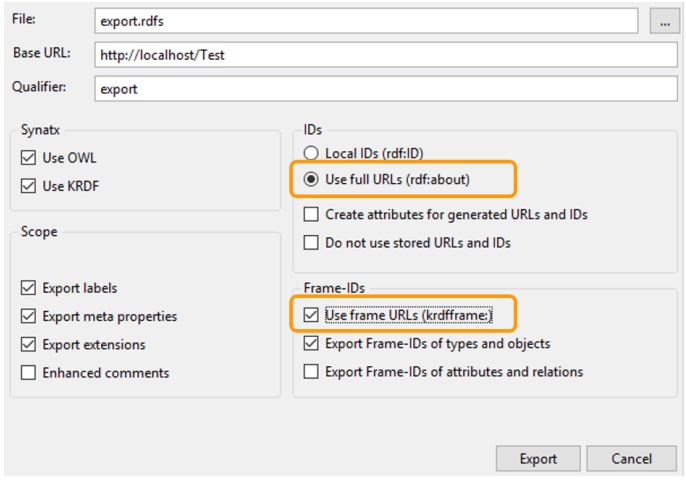
1.5.5 Restore deleted individuals from a backup

RDF export and import is useful for recovering deleted individuals from a backup network. Proceed as follows:

1. Open the backup network in the Knowledge Builder
2. Create a new folder and put the individuals to be restored there. To do this, right-click in the list view of the individuals to be acquired to open the context menu and select "Copy content to new folder", choosing a new folder as the storage destination
3. Open the RDF export via the context menu on the newly created folder:

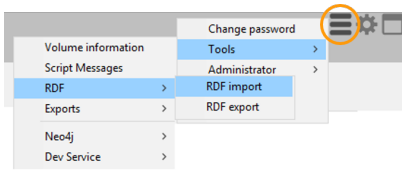


4. Specify a file name in the export dialog box, select the options "Use full URLs (rdf: about)" and "Use frame URLs (krdfframe :)" and execute the export:

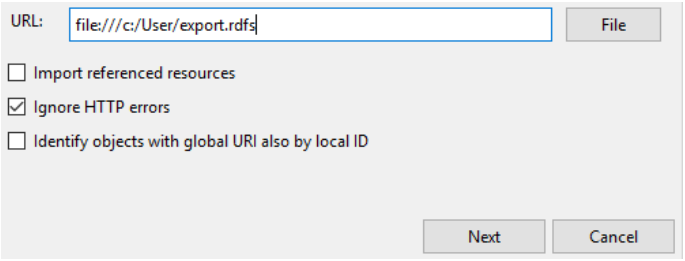


Note: The option "Use KRDF" causes i-views specific content to be included. This could not be completely mapped using RDF syntax.

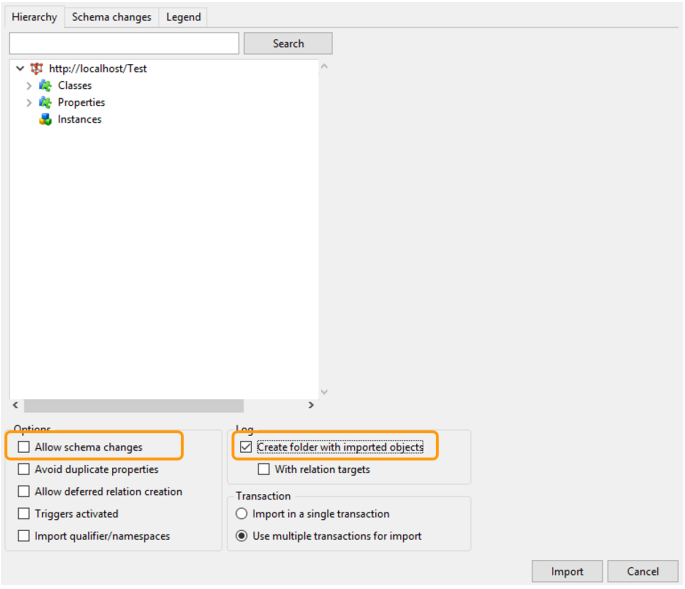
5. Close the Knowledge Builder and open the target network in the Knowledge Builder
6. In the main menu under Tools> RDF> RDF Import open the RDF import dialog:



7. Choose file and click „Next“:



8. In the selection dialog, deactivate the option "Allow changes to the schema", activate "Create folder with imported objects":



9. Execute the import

Review the recovered individuals

1.6 Access rights and triggers

This section is about checking access rights and triggers:

- **Access rights**regulate which operations are allowed to be performed by certain user groups on the semantic model. In i-views they are defined in the rights system.The rights system is located in the area of *Technology* > *rights*.
- **Triggers**are automatic operations that are triggered on a given incident and perform the associated actions. The Triggerarea can be found under *Technology* > *Trigger*.

The rights system and triggers are initially not activated in a newly created semantic graph database. These areas must first be activated before they can be used.

When creating rights and triggers, the basic procedure is identical:
Filters are needed to check if certain conditions are met or not. If these conditions are met, an access right or ban is granted to the rights system and a log is entered for triggers or a

script is executed. In the rights system, the arrangement of the filters is called the Rights Tree and for triggers the TriggerTree.

1.6.1 Testing Access Rights

With Rights we regulate the access of users to the data in the semantic network. The two fundamental aims that can be achieved with the so-called rights system are:

Protection of sensitive data: Users or user groups are only shown the data that they are allowed to read. This ensures secrecy and confidentiality restrictions.

Work-specific overview: Certain users often need only a portion of the model's data to work with the system. With the help of the rights system it is possible to show them only the elements which they need for the accomplishment of their tasks.

The rights system of i-views is characterized by a high degree of flexibility. It can be confidently tailored to different requirements of a project. Defining rules in the rights tree, consisting of individual filters and decision makers, creates a network-specific configuration of the rights system. There are many ways to compose these rules for the rights system, creating highly differentiated rights. It is not possible to list all potential combinations of confgurations; here an advice for the individual case must be found.

How does the rights system work?

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

- *Read*: an element should be displayed.
- *Modify*: an element should be changed.
- *Create*: a new element should be created.
- *Delete*: an element should be deleted.

If the access right is to be checked in a specific access situation, the **Rights Tree**is processeduntil a decision can be made for or against the access in this situation. The Rights tree consists of conditions against which the access situation is checked. To check the conditions, **Filters**are used which filter elements of the knowledge network and operations. At the end of a subtree of filters in the Rights Tree are the **Decision Makers**. Of these, access is either allowed or denied.

With regard to the access situation, aspects are selected which are used as a condition for the permission or the rejection of the access. In access situations, the following aspects are often used for the decision:

- the operation (create, read, delete or modify)
- the item to access.
- the current user

It may be that only one aspect of the access situation is selected as a condition, but a combination of the listed aspects may also be queried, e.g.: "Paul [user] may not delete [operation] any descriptions [item]".

1.6.1.1 Activation of the Access Right System

In a newly created knowledge network, the Rights System is disabled by default. For it to be usable, it must be activated in the settings of the Knowledge Builder.

Instructions for activating the Rights System

1. In the Knowledge Builder, go to the *Settings* menu and select the *System* tab. Select the field *Rights*.
2. Set a check mark 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 object type "Person". (Type can not be abstract.)
4. If you have connected the Knowledge Portal of i-views, enter a user in the*Default Web User*field (object of the previously defined person object type).

Before activating the Rights System, the folder is named Rights (*disabled*). If the Rights System has been activated, the folder is named *Rights*. By deactivating the Rights System, access right checks are no longer performed. The defined rules in the Rights Tree are retained and will be used again when the Rights System is reactivated.

Note: If you access the item from the web frontend without special login, the person specified under *Default Web User*will be used.Usually you put here a dummy person named "anonymous" or "guest".

In order for the rights system to work in the Knowledge Builder as well, the user account 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 that was specified in the User Type field when activating the rights system.

The link is generally necessary for the use of the operation parameter *User* for search filters or for the use of the access parameter *User* for structure queries, if the rights system or the search is not executed in an application but in the Knowledge Builder itself.

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

1. In the Knowledge Builder, go to the *Settings*menu and select the *System*tab. There, select the *User*field.
2. Select the user to be linked. *Linking*allows the user to associate with a person object that is not yet linked to a Knowledge Builder account. The *Unlink* operation causes the link of the Knowledge Builder account to the person object to be dismissed.

Note: The currently logged in user can not be linked.

Users with administrator rights are generally allowed to perform all operations, regardless of which rights have been defined in the Rights System. The definition as administrator is also carried out in the *Settings* menuon the *System*tab in the *User*field.

1.6.1.2 Access Rights Tree

Traversing the Rights Tree

The Rights Tree consists of rules that are defined in a tree. The branches of the tree, also called subtrees, consist of the conditions that should be checked. The conditions are defined in the system as filters, which are nested. During evaluation, the system runs the tree from top to bottom. If a condition matches the access situation, then the check goes to the next

filter of the subtree. This filter is checked again. This will be done until the end of the subtree, there is an access right or ban. If a condition does not match the access situation, the system moves to the next subtree. If the system is processing, the rights tree encounters an access right or ban and the rights' check is terminated with this result. The branches (subtrees) of the tree are thus processed one after the other, the tree is "traversed" until a decision can be made.

Filters and decision makers are nested inside each other in the form of folders, resulting in a tree construct consisting of different subtrees. A folder can have multiple subfolders (multiple following filters on one level), creating branches in the Rights Tree. Folders that have been defined on one level will be processed sequentially (from top to bottom).

Creating the Rights Tree

When creating the Rights Tree, it is important to group the rules in a useful way, because if a decision for an access permit or an access ban has been made, no further rules are checked. Therefore, exceptions should be defined before the global rules.

The two basic cases that must be distinguished are:

- **negative configuration:**in the lowest subtree everything is allowed, above bans are formulated.
- **positive configuration:**at the bottom everything is forbidden, except of what is allowed above.

The sequence of the subtrees is therefore crucial in the creation of the Rights Tree. On the other hand, the order of the conditions in a subtree (whether we first check the operation and then the property or vice versa) is arbitrary.

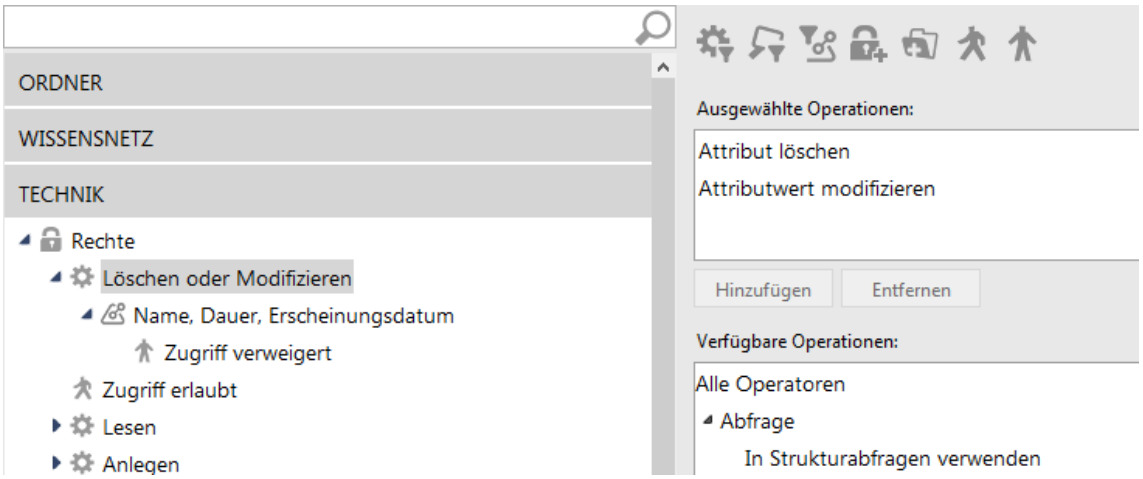
For the definition of a subtree of the Rights Tree, it is not absolutely necessary to use all filter types. A sub-tree consists of at least one filter and one decision maker. An exception is the last subtree, which normally consists of only one decision maker, who allows all remaining operations (which were not prohibited previously in the rights tree) or prohibits all remaining operations (which were not allowed previously in the rights tree).

Example: Rights Tree

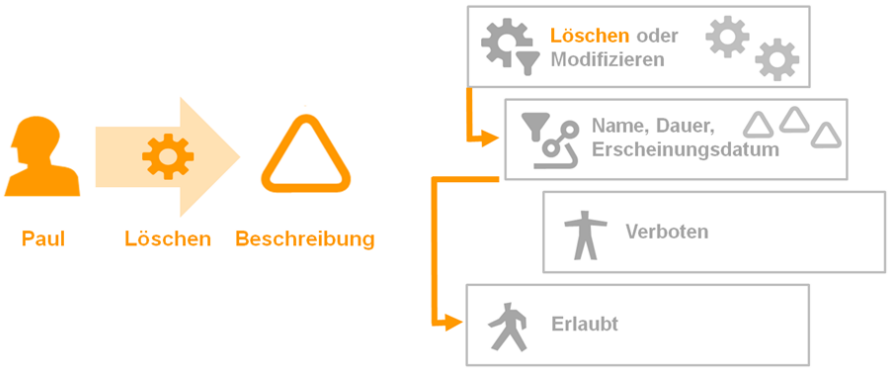
This simple example shows a rights tree consisting of a rights tree and a default decision maker that allows everything:

In the rights branch the deletion or modification of the attributes Name, Duration and Release Date is forbidden. For this purpose, an operation filter is used that has the "Delete or Modify" operation (Löschen oder Modifizieren) as a condition. Only these operations are allowed through by this operations filter. The next filter is a Property filter that filters for specific properties. In this case, the attributes "Name", "Duration", and "Release Date" are filtered regardless of which object

or property they are stored on. The lasst node of the rights branch is the decision maker "Forbidden" (Verboten), which prohibits any access operation that fits on the two featured filters. If one of the two conditions does not affect the access situation, the default decision maker "Allowed"(Erlaubt) is executed.



Checking an operation using the rights tree example:



The left side shows the operation to be tested: The user Paul wants to delete (Löschen) the attribute Description (Beschreibung). On the right side the rights tree is shown. Checking the condition of the first filter turns out positive since Paul wants to perform the delete operation. In the rights tree the next filter of the rights tree will be excecuted. This is the property filter of the attributes "Name", "Duration" and "Release Date". The check of the filter turns out negative as the description is not one of the filtered properties. The processing of the subtree has started. It switches to the next subtree of the rights tree. This is already the default decision maker "Allowed", which allows anything that is not explicitly forbidden in the rights tree.

1.6.1.3 Decider

Entscheider stehen immer an der letzten Stelle eines Rechteteilbaumes. Durch die Kombination mit Filtern werden Zugriffsituationen bestimmt in denen der Zugriff explizit erlaubt bzw. verboten ist. Wenn bei der Traversierung des Rechtebaumes ein Entscheider erreicht wird, dann wird mit dieser Entscheidung die Rechteprüfung beantwortet. Die zu prüfende Operation wird dann entweder erlaubt oder abgewiesen. Der Rechtebaum wird dann nicht weiter geprüft.

Symbol	Zugriffsrecht	Beschreibung
	Zugriff gewähren	Der Zugriff wird in der zu prüfenden Zugriffssituation erlaubt.
	Zugriff verweigern	Der Zugriff wird in der zu prüfenden Zugriffssituation nicht erlaubt.

Es gibt grundsätzlich zwei verschiedene Entscheider einen positiven - Zugriff erlaubt und einen negativen - Zugriff verboten.

Anleitung zum Anlegen eines Entscheiders

1. Wählen Sie im Rechtebaum die Stelle aus, an der sie einen Entscheider anlegen wollen.
2. Über die Buttons und werden neue Entscheider als Unterordner des aktuell ausgewählten Ordners angelegt.
3. Geben Sie dem Ordner einen Namen.

1.6.1.4 Assemble Access Rights

Für die Definition von Rechten werden Filter und Entscheider im Rechtebaum miteinander kombiniert. Im Kapitel Filter werden die verschiedenen Filterarten und deren Einsatzmöglichkeiten dargestellt. Die Entscheider *Zugriff gewähren* oder *Zugriff verweigern* stellen jeweils den letzten Knoten eines Teilbaumes des Entscheidungsbaumes dar. Wird ein Entscheider erreicht, so wird mit dieser Entscheidung die Traversierung des Rechtebaumes beendet.

Um Regeln im Rechtesystem zu definieren stehen die folgenden Funktionen zur Verfügung:

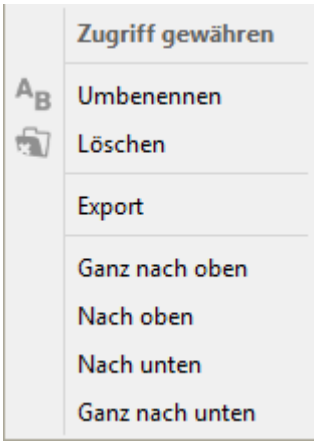
Symbol	Funktion	Beschreibung
	Neuer Operationsfilter	Ein neuer Operationsfilter wird erstellt.
	Neuer Suchfilter	Ein neuer Suchfilter wird erstellt.
	Neuer Eigenschaftsfilter	Ein neuer Eigenschaftsfilter wird erstellt.
	Neuer Strukturordner	Ein neuer Strukturordner wird erstellt.
	Zugriff gewähren	Ein positiver Entscheider, der den Zugriff erlaubt, wird erstellt.
	Zugriff verweigern	Ein negativer Entscheider, der den Zugriff verbietet, wird erstellt.

Um Rechte sinnvoll zu strukturieren, können Strukturordner verwendet werden. Sie haben keinen Einfluss auf die Traversierung des Rechtebaumes. Sie dienen lediglich dazu bei einer Vielzahl von Rechten, inhaltlich zusammengehörige Teilbäume des Rechtebaumes zu grup-

pieren.

Anordnung von Ordner im Rechtebaum ändern

Um die Filter und Entscheider im Rechtebaum in die richtige Reihenfolge zu bringen, kann über ein Klick mit der rechten Maustaste ein Kontextmenü aufgerufen werden:



In diesem Kontextmenü kann der Filter oder Entscheider umbenannt, gelöscht und exportiert sowie die Position im Rechtebaum verändert werden. Liegen zwei Ordner (Filter oder Entscheider) auf der gleichen Ebene, kann mithilfe der Funktion *Nach oben*, *Nach unten* der Ordner im Rechtebaum weiter nach vorne oder hinten verschoben werden. *Ganz nach oben* und *Ganz nach unten* verschiebt den Ordner entsprechend an die erste bzw. letzte Stelle der Ebene im Rechtebaum.

Sollen Ordner ineinander geschachtelt werden, also die Ebene im Entscheidungsbaum verändert werden, kann dies mit Drag & Drop durchgeführt werden.

Zusammensetzen von Rechten

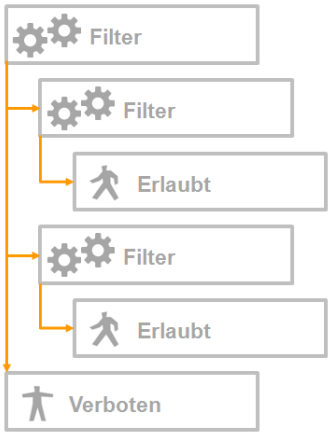
Durch das Zusammensetzen von Filtern und Entscheidern im Rechtebaum gibt es eine Vielzahl von Kombinationsmöglichkeiten um Rechte zu definieren. Es gibt grundsätzlich 3 verschiedene Vorgehensweisen um Rechte zu definieren:

- Definition von Rechten für jede mögliche Zugriffsituation
- Positiv-Konfiguration
- Negativ-Konfiguration

Da die Definition von Zugriffsrechten für jede mögliche Zugriffsituation eine sehr aufwendige Vorgehensweise ist, wird i.d.R. eine der beiden anderen Konfigurationswiesen angewendet. Diese werden in den beiden folgenden Abschnitten erläutert.

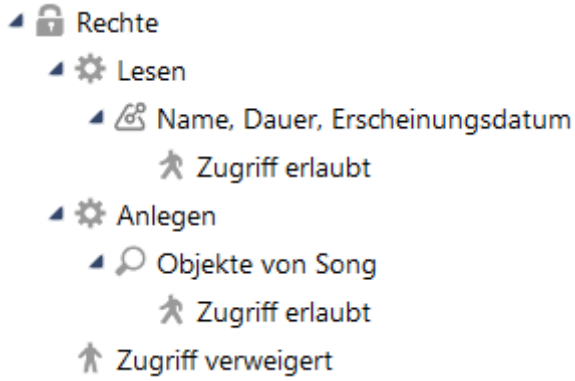
1.6.1.4.1 Positive Configuration

Wenn im Rechtebaum nur Rechte definiert werden, die bestimmte Zugriffe erlauben und alle anderen Zugriffe, über die nichts ausgesagt wird, verboten sind, spricht man von einer Positiv-Konfiguration des Rechtebaumes. In jedem Teilbaum des Rechtebaumes werden Regeln definiert, die bestimmte Operationen erlauben. Alle zu prüfenden Operationen durchlaufen den Rechtebaum: Passt die zu prüfende Operation nicht auf die Bedingungen der Teilbäume, wird sie am Ende des Rechtebaumes abgelehnt.



Beispiel: Positiv-Konfiguration

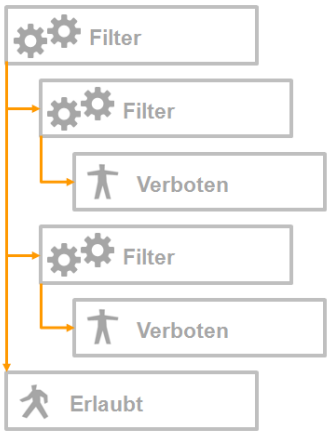
Dieses Beispiel zeigt, wie ein positiv formulierter Rechtebaum im Knowledge-Builder aussehen kann:



Der erste Teilrechtebaum definiert den lesenden Zugriff auf die Attribute Name, Dauer und Erscheinungsdatum. Die Operation Lesen wird für diese Attribute erlaubt. Der zweite Teilrechtebaum erlaubt das Anlegen von neuen Objekten des Typs Song. Alle anderen Operationen werden am Ende des Rechtebaumes generell verboten.

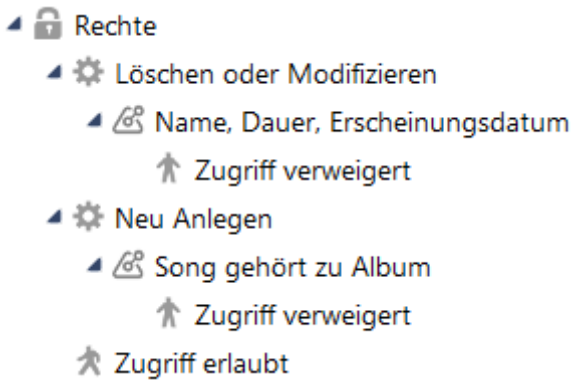
1.6.1.4.2 Negative Configuration

Werden im Rechtebaum Regeln definiert, die bestimmte Operationen ablehnen und alle nicht darauf passenden zu prüfenden Operationen erlaubt werden, spricht man von einer Negativ-Konfiguration. In den Teilbäumen des Rechtebaumes werden bestimmte Operationen verboten. Passt eine zu prüfende Operation nicht auf die Bedingungen der Teilbäume, dann wird die Operation am Ende des Rechtebaumes erlaubt.



Beispiel: Negativ-Konfiguration

Dieses Beispiel zeigt, wie ein negativ formulierter Rechtebaum im Knowledge-Builder aussehen kann:

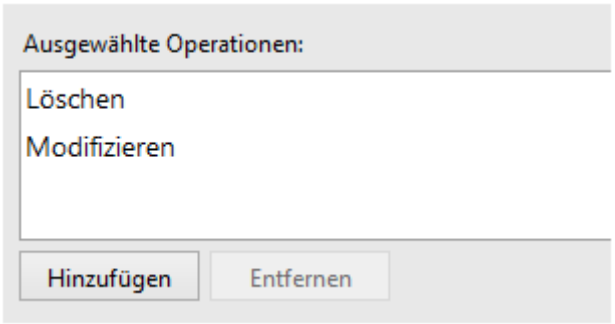


Der erste Teilrechtebaum verweigert im Gegensatz zum Beispiel Positiv-Konfiguration die Zugriffsrechte für das Löschen und Modifizieren der Attribute Name, Dauer und Erscheinungsdatum. Der Zweite Teilrechtebaum verbietet das Löschen der Relation die Songs mit dem Album verbindet, in dem sie enthalten sind. Alle anderen Operationen dürfen durchgeführt werden.

1.6.1.4.3 Example

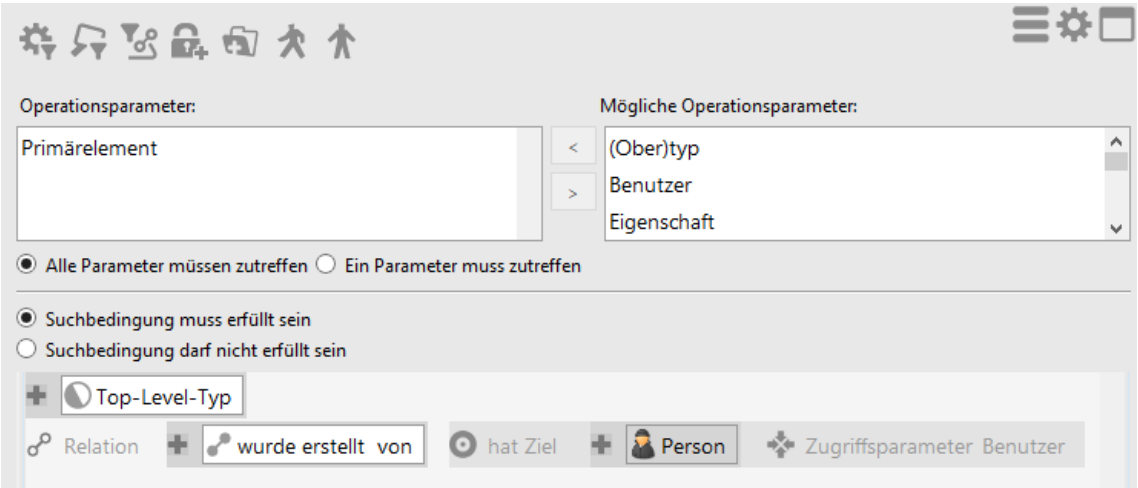
Was wird gebraucht um dieses Recht in i-views zu definieren? Zum einen wird ein Operationsfilter benötigt, da es um das Ändern und Löschen von Elementen geht. Zum anderen muss der Zusammenhang zwischen dem Benutzer und dem Element, an dem er eine Operation ausführen möchte, formuliert werden - das geht nur mithilfe von Suchfiltern.

Operationsfilter



Im Operationsfilter wurden die Operationen Löschen und Modifizieren ausgewählt.

Suchfilter



Im Suchfilter wird die Relation wurde erstellt von mit dem Relationsziel Person ausgewählt. An dem Relationsziel Person wurde der Zugriffsparameter Benutzer angegeben. Die Einstellung Alle Parameter müssen zutreffen und Suchbedingung muss erfüllt sein sind ausgewählt. In diesem Fall wurde der Operationsparameter Primärelement ausgewählt.

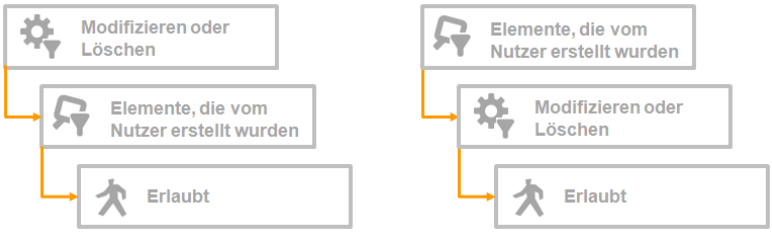
Ein Frage, die das Schema betrifft, ist: An welchen Elementen ist die Relation *wurde erstellt von* definiert? Es gibt verschiedene Möglichkeiten diese Relation in einem semantischen Netz umzusetzen:

1. Fall Definition an Objekten und Typen: Nur an Objekten und Typen wird die Relation verwendet.
2. Fall Definition an allen Elementen: An allen Objekten, Typen, Erweiterungen, Attributen und Relationen wird die Relation verwendet.

Im ersten Fall macht es Sinn den Operationsparameter Primärelement oder übergeordnetes Element zu verwenden. Definiert man das Recht mit dem übergeordneten Element, so gilt es für nicht nur für das Objekt an sich sondern auch für alle Eigenschaften, die an Objekten gespeichert sind, welche vom Nutzer erstellt wurden. Verwendet man stattdessen den Operationsparameter Primärelement so gilt das Recht ebenfalls für alle Metaeigenschaften des Objektes.
Im zweiten Fall wird der Operationsparameter Zugriffselement verwendet, da nur die Elemente geändert werden dürfen, an denen die Relation *wurde erstellt von* mit dem entsprechenden Relationsziel, dem Benutzer, vorkommt.

Das Recht im Rechtebaum zusammensetzen

Es gibt zwei verschiedene Varianten die Filter zu kombinieren. Gibt es in dem Rechteteilbaum keine Verzweigungen so ist die Reihenfolge der Teilbäume nicht relevant.

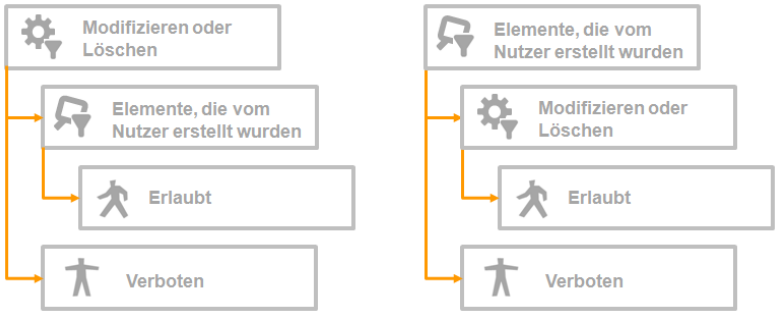


Die Graphik zeigt, die zwei möglichen Kombinationsweisen: Version 1 (links) erst Operationsfilter dann Suchfilter, Version 2 (rechts) erst Suchfilter dann Operationsfilter, als letztes folgt jeweils der Entscheider Erlaubt.

Empfehlung: Es ist sinnvoll den Operationsfilter an erster Stelle zu haben, so ist es möglich unter ihm alle anderen Rechte, welche auf die selbe Operation filtern, anzulegen. Dies schafft eine einfacher nachvollziebare Struktur in den Rechtebaum.

Erweitertes Recht: Elemente die nicht vom Nutzer erstellt wurden, dürfen nicht geändert oder gelöscht werden

Das Recht impliziert das Verbot für alle Elemente, die nicht vom Nutzer erstellt wurden - jedoch haben wir das in der Rechtedefinition noch nicht ausgedrückt. Dafür mussen wir bei der Rechteerstellung den Entscheider Zugriff verboten berücksichtigen. Betrachtet man beide Rechteversionen und kombiniert diese mit dem negativen Entscheider, kommen folgende Varianten heraus. Jedoch haben die beiden Varianten unterschiedliche Auswirkungen im Rechtesystem.



Fügt man an die beiden eben dargestellten Kombinationsweisen jeweils den Entscheider Verboten hinzu, so entstehen die beiden Versionen: Version 1 (links) erst Operationsfilter, dann Suchfilter und Entscheider Erlaubt. Auf den Operationsfilter folgt außerdem in einem zweiten Teilbaum der Entscheider Verboten. Version 2 (rechts) erst Suchfilter, dann Operationsfilter und Entscheider Erlaubt. In dieser Version folgt auf den Suchfilter ein zweiter Teilbaum mit dem Entscheider Verboten.

Auswirkungen der verschiedenen Versionen auf das Rechtesystem

Version 1 (links)

- Erlaubt wird das Modifizieren und Löschen selbst erstellter Elemente.
- Verboten wird das Modifizieren und Löschen aller anderen Elemente.
- Es wird keine Aussage über alle anderen Operationen gemacht.

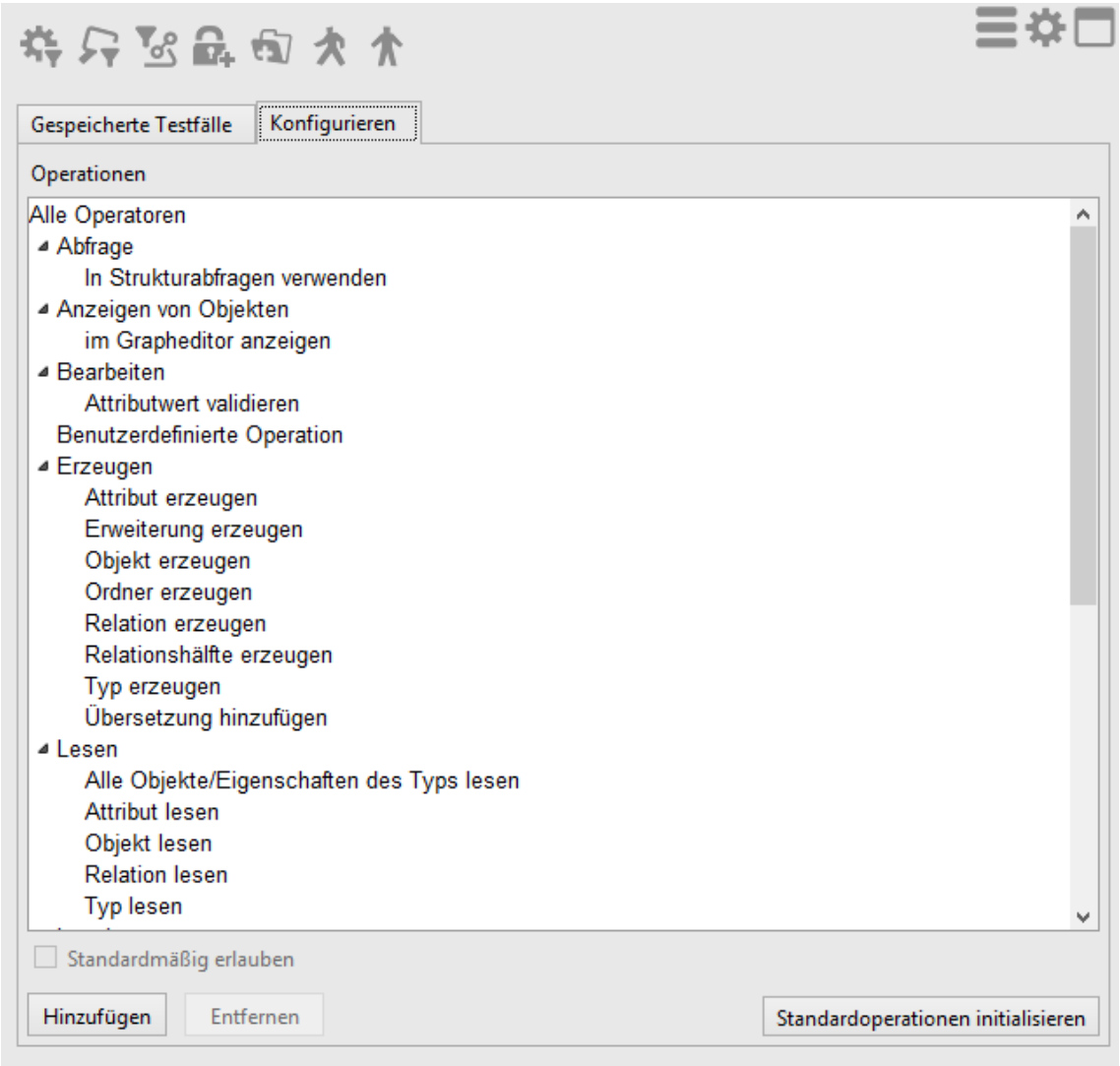
Version 2 (rechts)

- Erlaubt wird das Modifizieren und Löschen selbst erstellter Elemente.
- Verboten werden alle anderen Operationen auf selbst erstellte Elemente (wie z.B. das Lesen)
- Es wird keine Aussage über alle anderen Elemente gemacht.

Die Punkte zeigen, dass Version 2 **nicht** das geforderte Zugriffsrecht ausdrückt. Nur Version 1 formuliert das gewünschte Zugriffsrecht - Jeder Benutzer darf selbst erstellte Elemente ändern oder löschen sowie Elemente, die nicht vom Nutzer erstellt wurden, dürfen nicht geändert oder gelöscht werden.

1.6.1.5 Configuration og User Generated Operations

Wird im Bereich *System* der Ordner *Rechte* ausgewählt, werden im Hauptfenster die Reiter *Gespeicherte Testfälle* und *Konfigurieren* angeboten. Auf dem Reiter *Konfigurieren* können eigene Operationen konfiguriert werden.



Die Konfiguration von eigenen Operationen findet i.d.R. nur dann Anwendung, wenn der Knowledge-Builder zusammen mit anderen Anwendungen verwendet wird. Eigene Operationen sind anwendungsspezifische Operationen, die gemeinsam geprüft werden sollen.

Dabei geht es darum, dass eine Kette von Operationen geprüft werden soll und nicht nur eine Operation.

Anleitung zur Konfiguration von eigenen Operationen

1. Wählen Sie im Knowledge-Builder den Bereich *System* den Ordner *Rechte* aus.
2. Wählen Sie im Hauptfenster den Reiter *Konfigurieren* aus.
3. Klicken Sie auf *Hinzufügen*, damit eine neue Operation erstellt wird.
4. Geben Sie in nachfolgenden Fenstern für die neue Operation einen internen Namen und eine Beschreibung an.
5. Die neue Operation wird als *Benutzerdefinierte Operation* hinzugefügt.
6. Über *Entfernen* können benutzerdefinierte Operationen wieder gelöscht werden.

1.6.2 trigger

Trigger sind automatische Operationen, die in i-views ausgeführt werden, wenn ein bestimmtes Ereignis eintritt. Sie helfen dabei Arbeitsabläufe zu unterstützen, in dem immer gleich bleibende Arbeitsschritte automatisiert werden.

Beispiele für den Einsatz von Trigger sind:

- Versenden von E-Mails aufgrund einer bestimmten Änderung
- die Bearbeitung von Dokumenten in einer bestimmten Reihenfolge durch bestimmte Personen
- die Kennzeichnung von Aufgaben als offen oder erledigt aufgrund einer bestimmten Bedingung
- die Erstellung von Objekten und Relationen, wenn eine bestimmte Änderung durchgeführt wird
- die Berechnung von Werten in einer vorher definierten Art und Weise
- automatische Generierung des Namensattribut von Objekten (z.B. Zusammensetzung aus Eigenschaften des Objektes)

Wie funktionieren Trigger?

Trigger sind eng verwandt mit dem Rechtesystem. Sie nutzen den selben Filtermechanismus, um festzulegen, wann ein Trigger ausgelöst wird. Die Filter werden in einem Baum angeordnet, dem Triggerbaum, der wie der Rechtebaum aufgebaut ist. Er besteht aus Filtern, mit denen Bedingungen definiert werden, wann eine Trigger-Aktion ausgeführt werden soll. Tritt durch die Durchführung einer Operation eine Zugriffsituation ein, welche auf die definierten Bedingungen passt, wird die zugehörige Trigger-Aktion ausgeführt.

Trigger-Aktionen sind in den meisten Fällen Skripte, die abhängig von den Elementen der Zugriffsituation, mit diesen Operationen durchführen. Somit ist es möglich gleichbleibende Arbeitsschritte zu automatisieren oder intelligente Auswertungen auf Grundlage von bestimmten Konstellationen im sem. Netz durchzuführen. In Skripten können jegliche Operationen auf Elemente, die in Abhängigkeit von komplexen Auswertungen stehen, ausgeführt werden und damit situations- und anwendungsspezifische Anforderungen an das sem. Netz gewährleisten. Die meisten Trigger sind aus diesem Grund i.d.R. projekt- und netzspezifisch; Für den Einzelfall sollte eine Beratung durchgeführt werden.

1.6.2.1 Activate Trigger

Um mit Triggern arbeiten zu können, muss die Trigger-Funktionalität zunächst im Knowledge-Builder aktiviert werden.

Anleitung zur Aktivierung von Triggern

1. Rufen Sie die *Einstellungen* des Knowledge-Builders auf.
2. Wählen Sie dort den Reiter *System* und das Feld *Trigger* aus.
3. Setzten Sie im Feld *Trigger aktiviert* einen Haken.

Hier kann ein *Limit für rekursive Trigger* angeben werden. Die Standardeinstellung ist "Keine". Als rekursive Trigger werden Trigger bezeichnet, die sich selber aufrufen. Dies passiert, wenn im Trigger-Skript selbst Operationen im sem. Netz durchgeführt werden, die wiederum selbst auf die Filterdefinition des Triggers passen.

Vor der Aktivierung der Trigger-Funktionalität, ist der Ordner im Technikbereich als *deaktiviert* gekennzeichnet. Durch die Aktivierung wird der *Ordner* dann in Trigger umbenannt.

Anmerkung: Wenn in Triggern der aktuelle Nutzer verwendet wird (z.B. in Suchfiltern oder über die entsprechende Skriptfunktion) und der Nutzer nicht in einer Anwendung Operationen ausführt sondern im Knowledge-Builder selbst, ist die Verknüpfung des Knowldge-Builder-Benutzer-Accounts mit einem Personenobjekt notwendig. Wie eine solche Verknüpfung erstellt wird, wird im Kapitel Aktivierung des Rechtesystems erklärt.

1.6.2.2 Trigger Tree

Der Triggerbaum ist wie der Rechtebaum aufgebaut. Er besteht aus Ästen (Teilbäumen), die aus Filtern und Triggern bestehen. Die Filter sind die Bedingungen, die geprüft werden müssen, damit der Trigger am Ende des Teilbaumes ausgeführt werden kann, wenn alle vorher zu prüfenden Bedingungen erfüllt sind.

Der Triggerbaum wird bei jeder Operation auf die Daten abgefragt - der Baum wird "traversiert". Passt ein Teilbaum auf die Zugriffsituation, so wird der Trigger ausgeführt. Passt die Bedingung eines Filters nicht auf die Zugriffssituation, so wird zum nächsten Teilbaum gewechselt. Nach der Ausführung einer Trigger-Aktion wird der Triggerbaum weiter durchlaufen, im Gegensatz zum Rechtesystem, dessen Abarbeitung mit dem Erreichen eines Entscheiders beendet ist. Um im Triggerbaum zu definieren, dass nach der Ausführung einer Aktion keine weiteren Filter geprüft werden sollen, dient die Schaltfläche *Keine weiteren Trigger auslösen*:

Symbol	Funktion	Beschreibung
	Keine weiteren Trigger auslösen	Die Traversierung des Triggerbaumes wird beendet.

Am Ende eines Teilbaumes steht im Gegensatz zum Rechtesystem kein Entscheider sondern Aktionen zur Verfügung.

Sym-bol	Funktion	Beschreibung
	Trigger definieren	Es wird eine neue Trigger-Aktion erstellt.

Die verfügbaren Trigger-Aktionen sind:

- *Log eintragen*: Ein Logeintrag wird geschrieben.
- *Script ausführen* > *JavaScript*: Eine Script-Datei in JavaScript wird ausgeführt.
- *Script ausführen* > *KScript*: Eine Script-Datei in KScript wird ausgeführt.

Gestaltung des Triggerbaumes

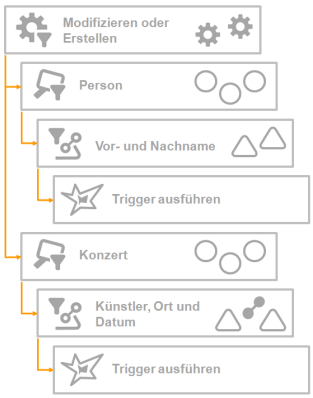
Bei der Gestaltung des Triggerbaumes hat die Reihenfolge, in der man die Trigger definiert i.d.R. keinen Einfluss auf die Performance von i-views. Beim Rechtebaum gibt es Empfehlung zur Gestaltung, die aber nicht auf den Triggerbaum übertragbar sind, da nach Ausführung einer Trigger-Aktion der Triggerbaum weiter traversiert wird.

Für die übersichtlichere Gestaltung der Trigger können diese in Strukturordnern gesammelt werden. Die Strukturordner selbst haben keinen Einfluss auf die Traversierung des Triggerbaumes.

Sym-bol	Funktion	Beschreibung
	Strukturordner	Strukturordner für die Gruppierung von Teilbäumen

Beispiel: Triggerbaum

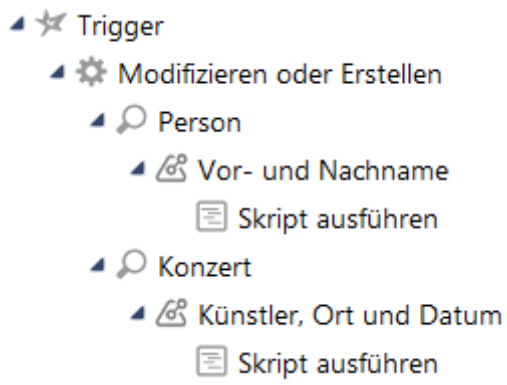
Dieses Beispiel zeigt einen Triggerbaum, der die Namen von Personen und Konzerten automatisch aus Eigenschaften der Objekte zusammensetzt:



Dieser einfache Triggerbaum beginnt mit einem Operationsfilter und teilt sich nach dem Operationsfilter in zwei getrennte Teilbäume. Wird einer der beiden Operationen Modifizieren oder Erzeugen ausgeführt, wird diese vom Operationsfilter durchgelassen. Der Teilbaum Person filtert Operationen, die an Attributen, Relationen von Objekten des Typs Person durchgeführt werden. Ist von der Operation entweder das Attribut Vorname oder das Attribut Nachname betroffen,

wird diese vom Eigenschaftsfilter durchgelassen. Das dazugehörige Skript, welches das Namensattribut einer Person aus Vor- und Nachname zusammensetzt wird ausgeführt. Der zweite Teilbaum bezieht sich ebenfalls auf den Operationsfilter Modifizieren oder Erstellen. Er filtert jedoch Attribute und Relationen, die an Objekten des Typs Konzert gespeichert sind. Der Eigenschaftsfilter lässt nur Operationen durch, welche an den Attributen oder Relationen zum Datum, dem Veranstaltungsort oder dem Künstler durchgeführt werden. Treffen diese Bedingungen zu, wird das zugehörige Skript ausgeführt, welches den Namen des Konzertes zusammensetzt.

So würde dieser Trigger Baum in i-views aussehen:



1.6.2.3 Create Trigger

Wie im Abschnitt Triggerbaum beschrieben, bestehen Trigger aus Filtern und Trigger-Aktionen. Diese werden miteinander kombiniert, so dass eine bestimmte Trigger-Aktion nur dann ausgeführt wird, wenn sie benötigt wird.

Die folgenden Funktionen stehen im Bereich Trigger zur Verfügung:

Symbol	Funktion	Beschreibung
	Neuer Operationsfilter	Ein neuer Operationsfilter wird erstellt.
	Neuer Suchfilter	Ein neuer Suchfilter wird erstellt.
	Neuer Eigenschaftsfilter	Ein neuer Eigenschaftsfilter wird erstellt.
	Neuer Löschfilter	Ein neuer Löschfilter wird erstellt.
	Neuer Strukturordner	Ein neuer Strukturordner wird erstellt.
	Neuer Trigger	Eine neue Trigger-Aktion wird erstellt.
	Keine weiteren Trigger auslösen	Ein neuer "Stopp"-Ordner wird erstellt. Dieser beendet die Traversierung des Triggerbaumes.

Bei der Erstellung von Triggern sollten zwei grundsätzliche Eigenschaften des Trigger Mechanismus beachtet werden:

- Die Ausführung eines Trigger-Skriptes kann dazu führen, dass weitere Trigger ausgelöst werden. Dies passiert, wenn im Trigger-Skript selbst Operationen in der semantischen Graph-Datenbank ausgeführt werden.
- Nach der Ausführung einer Trigger-Aktion wird der Triggerbaum weiter durchlaufen. Alle Trigger-Aktionen der Teilbaume, die auf die Zugriffssituation zutreffen, werden ausgeführt.

1.6.2.4 Trigger Action

Trigger-Aktionen dienen dazu, intelligente Operationen in der semantischen Graph-Datenbank durchzuführen, welche beispielsweise Arbeitsabläufe automatisieren oder unterstützen. Sie werden jedoch nur ausgeführt, wenn die Zugriffsituation und die Verknüpfungen im semantischen Netz einen bestimmten Zustand annehmen, der durch den Filter definiert wird.

Anleitung zum Anlegen von Trigger-Aktionen

1. Wählen Sie im Triggerbaum die Stelle, an der die Trigger-Aktion angelegt werden soll.
2. Fügen Sie über den Button einen neuen Trigger ein.
3. Wählen Sie den Aktionstyp aus der Liste aus: Log eintragen oder Skript ausführen (Wenn Sie ein Skript ausführen wollen, wählen Sie die Skriptsprache aus.)
4. Der Trigger wird als Unterordner des aktuell ausgewählten Ordners erstellt.

1.6.2.4.1 Script Trigger

Für die Ausführung des Skriptes muss ein Operationsparameter angegeben werden. Im Gegensatz zu Suchfiltern, kann nur ein Operationsparameter angegeben werden. Auf dem im Operationsparameter enthaltenem Element startet die Ausführung des Skriptes.

Zeitpunkt/Art der Ausführung

- Vor der Änderung: Der Trigger wird ausgeführt bevor die Operation durchgeführt wird.
- Nach der Änderung: Der Trigger wird direkt nach der Durchführung der Operation ausgeführt.
- Ende der Transaktion: Der Trigger wird erst am Ende der gesamten Transaktion ausgeführt.
- Job-Client: Der Jobclient bestimmt den Zeitpunkt der Ausführung.

Beachte: Trigger, die bei Löschoperationen ausgelöst werden, sollten vorzugsweise als Zeitpunkt *Vor der Änderung* verwenden, da ansonsten das zu löschende Element nicht mehr zur Verfügung steht. Für andere Operationen bietet sich als Zeitpunkt eher *Nach der Änderung* oder *Ende der Transaktion* an, da dann beispielsweise eine Eigenschaft zu dem neu erstellten Element hinzugefügt werden kann oder automatisch der Name aus verschiedenen Eigenschaften eines Objektes generiert werden kann, wenn eine oder mehrere Eigenschaften geändert wurden.
Werden z.B. mehrere Datensätze in einem Import in i-views importiert, die eine Trigger-Aktion auslösen, die auf Basis von importierten Relationen, Aktionen im sem. Netz durchführen, kann es sinnvoll sein den Import in einer Transaktion durchzuführen und entsprechend *Ende der Transaktion* als Zeitpunkt der Ausführung auszuwählen, da sonst noch nicht alle Relationen die das Script benötigt importiert wurden.

Je Operationsparameter nur ein mal ausführen

Ist diese Einstellung ausgewählt, dann wird das in Operationsparameter ausgewählte Element maximal ein mal pro Transaktion ausgeführt. Wenn diese Einstellung gesetzt ist, sollte der Ausführungszeitpunkt auf *Ende der Transaktion* gesetzt werden, damit im Skript der endgültige Zustand des Elements verwendet wird.

Beispiel: Bei Personen soll der Name des Objekts aus Vorname und Nachname zusammengesetzt werden. Mit dieser Einstellung wird bei gleichzeitiger Änderung von Vor- und Nachname der Trigger nur ein mal ausgeführt.

Ausführung löst keine Trigger aus

Mit dieser Einstellung wird festgelegt, dass durch die Operationen, die innerhalb eines Triggers ausgeführt werden, keine weiteren Trigger ausgelöst werden können. Mit dieser Einstellung lassen sich Endlosschleifen vermeiden.

Bei Skriptfehlern Skript weiter ausführen

Ist diese Einstellung aktiv, so wird versucht nach Ausführungsfehlern wieder aufzusetzen und die Ausführung des Skriptes fortzuführen. Diese Einstellung eignet sich vorwiegend für Skripte, die voneinander unabhängige Anweisungen ausführen sollen, nicht für solche, die auf vorherige Schritte des Skriptes aufbauen.

Transaktion abbrechen, wenn Trigger fehlschlägt

Diese Einstellung legt das Abbruchverhalten bei Skriptfehlern fest. Tritt bei der Ausführung des Skriptes ein Fehler auf und diese Einstellung ist aktiv, werden alle Aktionen der Transaktion rückgängig gemacht. Ist diese Einstellung nicht aktiv, werden alle Aktionen durchgeführt außer diese, die von der Fehlerstelle betroffen sind. Die ursprüngliche Aktion, die zum Aufruf des Triggers geführt hat, wird trotzdem durchgeschrieben.

Ausführen während eines Daten-Refactorings

Unter Daten-Refactoring werden Operationen zur Umstrukturierung des semantischen Netzes verstanden wie z.B. *Typ wechseln* oder *Relationsziel neu wählen*. Daten-Refactoring-Operationen können unter Umständen ungewollte Trigger-Aktionen auslösen und in bestimmten Fällen auch Fehler bei der Durchführung des Skriptes erzeugen. Aus diesem Grund kann pro Trigger eingestellt werden, ob er bei Daten-Refactorings ausgeführt werden soll.

Der Funktionsrumpf für Skript-Trigger wird automatisch angelegt.

Das Skript hat drei Parameter:

parameter	$k.SemanticElement / $k.Folder	Der ausgewählte Parameter
access	object	Objekt mit Daten der Änderung (neuer Attribuwert usw.)
user	$k.User	Der Benutzer der die Änderung ausgelöst hat

Folgendes Beispiel setzt die Attribute mit den internen Namen "geaendertAm" / "geaendertVon". Als Parameter sollte hier "Primäres Kernobjekt" ausgewählt werden.

```
/**
 * 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 });
}
```

Das Parameter "access" kann (je Operation variierend) folgende Eigenschaften enthalten:

Eigenschaft	Beschreibung
accessedObject	Zugriffselement
core	Kernobjekt
detail	Detail
inversePrimaryCoreTopic	Primäres Relationsziel
inverseRelation	Inverse Relation
inverseTopic	Relationsziel
operationSymbol	"read", "deleteRelation", etc.
primaryCoreTopic	Primäres Kernobjekt
primaryProperty	Primäreigenschaft
primaryTopic	Primärelement
property	Eigenschaft
topic	Übergeordnetes Element
user	Benutzer (identisch zu "user"-Parameter der Funktion)

1.6.2.4.2 Log Trigger

Möchte man die Trigger-Funktionalität überwachen bzw. dokumentieren, wann welcher Trigger ausgelöst wurde und welche Operationen im sem. Netz ausgeführt wurden, eignen sich Log Trigger. Der Log wird in das jeweilige Log File (bridge.log, batchtool.log etc.) geschrieben in dessen Anwendungsumgebung die Operation, welche den Trigger ausgelöst hat, durchgeführt wird.

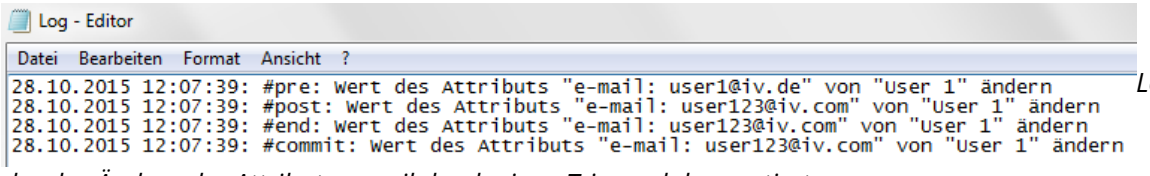
Zeilen des Logeintrages	Zustand des sem. Netzes zum Zeitpunkt
#pre	vor Auslösung
#post	nach Auslösung
#end	am Ende der Transaktion
#commit	bei erfolgreicher Beendigung der Transaktion

Logeinträge dienen dazu nachzuvollziehen, ob in einer bestimmten Zugriffsituation, die tatsächlich geschehen ist, ein Trigger ausgeführt wurde und was er gemacht hat. Im Gegensatz dazu kann in der Testumgebung getestet werden, ob in einer bestimmten Zugriffsituation ein Trigger ausgelöst werden würde oder nicht, ohne dass die konkrete Zugriffsituation durchgeführt wird.

Anleitung zum Anlegen von Log Triggern

1. Wählen Sie im Triggerbaum das Trigger-Skript aus, welches geloggt werden soll.
2. Erstellen Sie über den Button ein Trigger vom Typ *Log eintragen* im Triggerbaum direkt vor dem Skript-Trigger.

Beispiel:

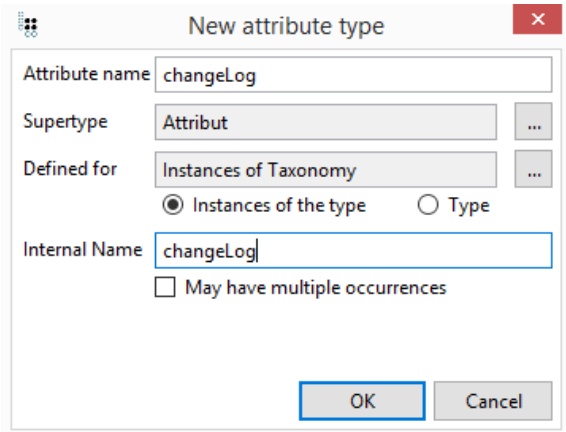


Logeintrag der das Ändern des Attributs e-mail durch einen Trigger dokumentiert.

1.6.2.4.3 ChangeLog Trigger

If you want to monitor user activity on objects, you should set up a changeLog trigger, also known as a change history.

For this purpose, a string attribute must first be defined, which receives the internal name "changeLog". This changeLog attribute must be defined for all elements where it should document user activities.



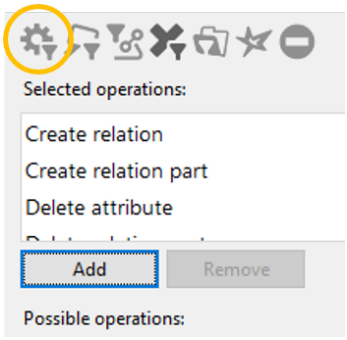
The triggermust contain the operation filters, that should log the change history, and the elements, where the attribute should be visible. See in example for learning how to build the trigger.

The triggerscript looks like this:

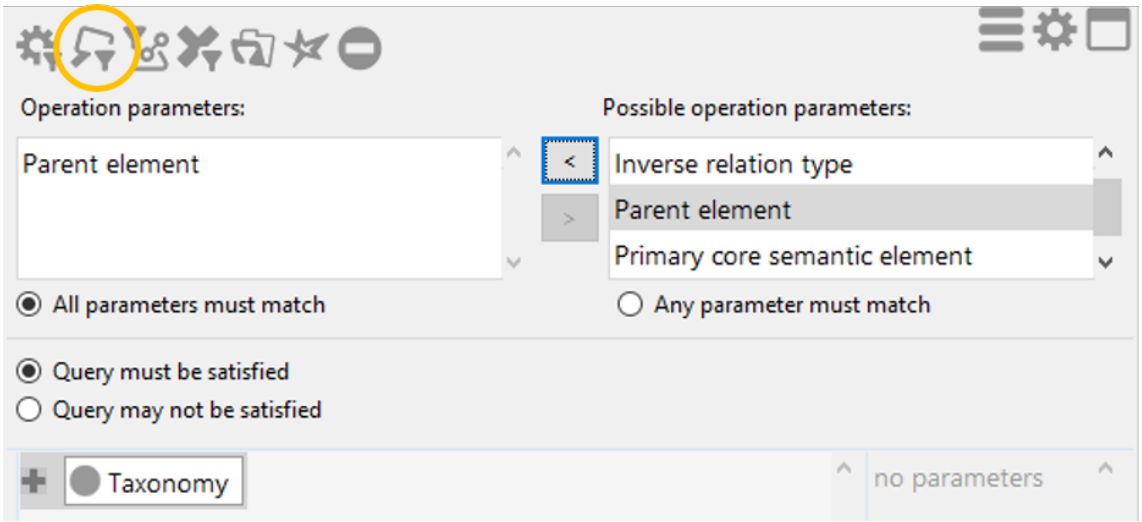
```
/**
 * Perform the trigger
 * @param parameter The chosen parameter, usually a semantic element
 * @param {object} access Object that contains all parameters of the access
 * @param {$k.User} user User that triggered the access
**/

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

Example

In a network, a changeLog is to be stored on all objects oft he taxonomy from the knowledge network. The objects should be used to log property modifications, creations and deletions. First, an operation filter was created that responds to the operations "Delete attribute", "Modify attribute value", "Create relation", "Create relation part" and "Delete relation part".

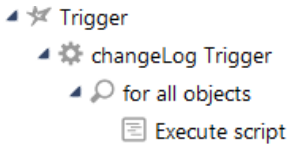


The next step is to define a search filter that determines which parent elements the operations are performed on.



In the triggerscript, the operation parameter "Parent element" was set because it corresponds to the search filter.

The triggerrules (operation filter, search filter and triggerscript) must follow the test sequence and therefore are in the hierarchy tree like this:



As soon as the triggeris finished and you modify an object a changeLog attribute will be created.

By clicking on "open", the table opens in which you can see who made when which change on which knowledge network element with which value.

1.6.3 Filter

Mithilfe von Filtern werden die Bedingungen im Rechtebaum bzw. im Triggerbaum definiert, um Zugriffssituationen einschränken zu können, wann ein Entscheider bzw. Trigger ausgeführt werden soll. Neue Filter werden im Baum unterhalb des aktuell ausgewählten Knotens angelegt. Auf diese Weise werden sie untereinander geschachtelt.

Im Rechtesystem stehen die drei Filterarten Operationsfilter, Suchfilter und Eigenschaftsfilter zur Verfügung. Zusätzlich zu den drei grundsätzlichen Filterarten bietet der Bereich Trigger einen spezifischen Filter - den Löschfilter.

Es gibt verschiedene Arten von Filtern - Wann benutzen wir welchen Filter?

Symbol	Filter	Beschreibung
	Operationsfilter	Filtert die Operationen; Auswahl aus Liste
	Suchfilter	Filtert Elemente durch Strukturabfrage

	Eigenschaftsfilter	Filtert Relationen und Attribute; Auswahl aus Liste
	Löschfilter	Filtert das Löschen von Elementen

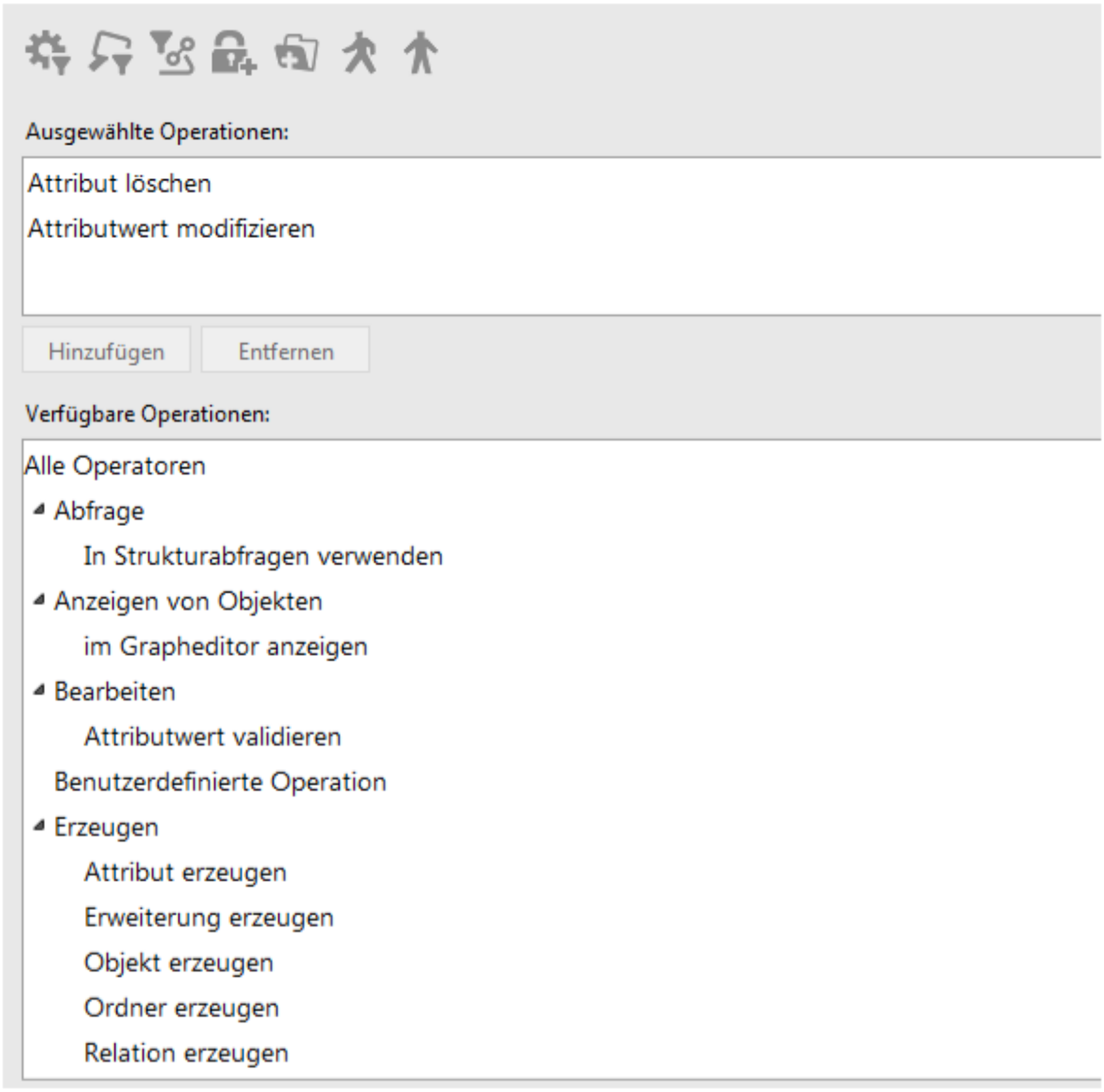
Operationen können nur mit einem Operationsfilter bestimmt werden. Benutzer können nur durch Suchfilter bestimmt werden. Eigenschaften können entweder mit Such- oder Eigenschaftsfiltern bestimmt werden. Die Verwendung von Eigenschaftsfiltern ist dann sinnvoll, wenn unabhängig von weiteren Eigenschaften im semantischen Modell wie Relationen zum Nutzer, Eigenschaften gefiltert werden sollen. Vor allem wenn große Mengen von Eigenschaften gefiltert werden sollen, ist es einfacher und übersichtlicher, das in einer Liste zu tun, anstatt in einer Strukturabfrage. Sollen Relationen zum Zugriffsobjekt oder zum Nutzer einbezogen werden, muss allerdings ein Suchfilter verwendet werden.

Anleitung zum Anlegen eines Filters

1. Wählen Sie im Rechte- bzw. Triggerbaum die Stelle aus, an der Sie einen neuen Filter anlegen wollen.
2. Erstellen Sie über die Buttons , , oder einen neuen Filter.
3. Der Filter wird als Unterordner des aktuell ausgewählten Ordners im Baum angelegt.
4. Geben Sie dem Ordner einen Namen.

1.6.3.1 Gear Filter

Für welche Operationen ein Zugriffsrecht gelten soll oder ein Trigger ausgeführt werden soll, kann nur mithilfe von Operationsfiltern angegeben werden. Durch die Auswahl der gewünschten Operation kann diese dem Filter hinzugefügt oder wieder entfernt werden.



Die Operationen sind in Gruppen gegliedert. Wählt man den übergeordneten Knoten einer Gruppe aus, werden auch alle darunterliegenden Operationen mit gefiltert. Wenn beispielsweise die *Erzeugen* Operation ausgewählt wird, dann werden die Operationen *Attribut erzeugen*, *Erweiterung erzeugen*, *Ordner erzeugen*, *Relation erzeugen*, *Relationshälfte erzeugen*, *Typ erzeugen* und *Übersetzung erzeugen* vom Filter berücksichtigt.

Im Kapitel Operationen werden alle verfügbaren Operationen aufgelistet und zusätzlich wird angegeben, welche Operationsparameter in Kombination verwendet werden können. Die verschiedenen Operationsparameter werden entsprechend im Kapitel Operationsparameter erklärt.

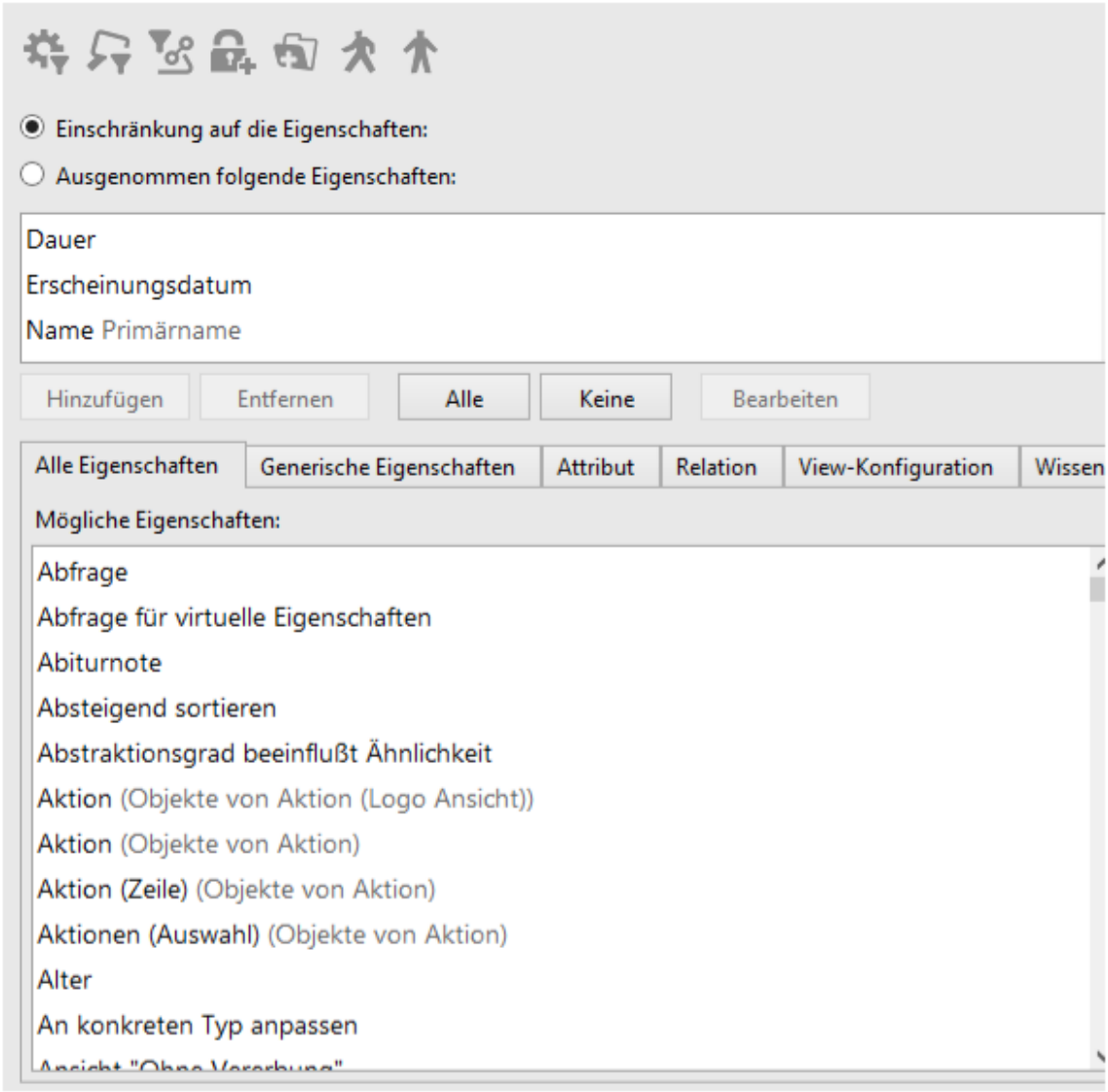
1.6.3.2 Property Filter

Mit Eigenschaftsfiltern können Attribute und Relationen gefiltert werden. Es gibt zwei verschiedene Vorgehensweisen einen Eigenschaftsfilter zu verwenden:

- *Einschränkung auf Eigenschaften*: Angabe der Eigenschaften für die die Bedingung gelten soll. Nachfolgende Filter oder Entscheider des Teilbaumes werden nur ausgeführt,

wenn die Zugriffseigenschaft mit den ausgewählten Eigenschaft übereinstimmt.

- *Ausgenommen folgende Eigenschaften*: Angabe der Eigenschaften für die die Bedingung nicht gelten soll. Stimmt die Zugriffseigenschaft mit einer der ausgewählten Eigenschaften überein, werden nachfolgende Filter, Entscheider oder Trigger nicht ausgeführt.



Über *Hinzufügen* und *Entfernen* können die unten aufgeführten Eigenschaften selektiert werden. Alle unten stehenden Eigenschaften können mithilfe von *Alle* ausgewählt werden. *Keine* entfernt alle ausgewählten Eigenschaften. Über das *Bearbeiten* Feld wird der Detaileditor des Attributs oder der Relation aufgerufen, das oder die im oberen Auswahlfeld markiert ist. Die Reiter *Alle Eigenschaften*, *Generische Eigenschaften*, *Attribut, Relation*, *View-Konfiguration* und *Wissensnetz* sollen dem Anwender helfen, die zu filternden Eigenschaften schneller zu finden. Im Reiter *Wissensnetz* werden alle selbst angelegten Relationen und Attribute angezeigt.

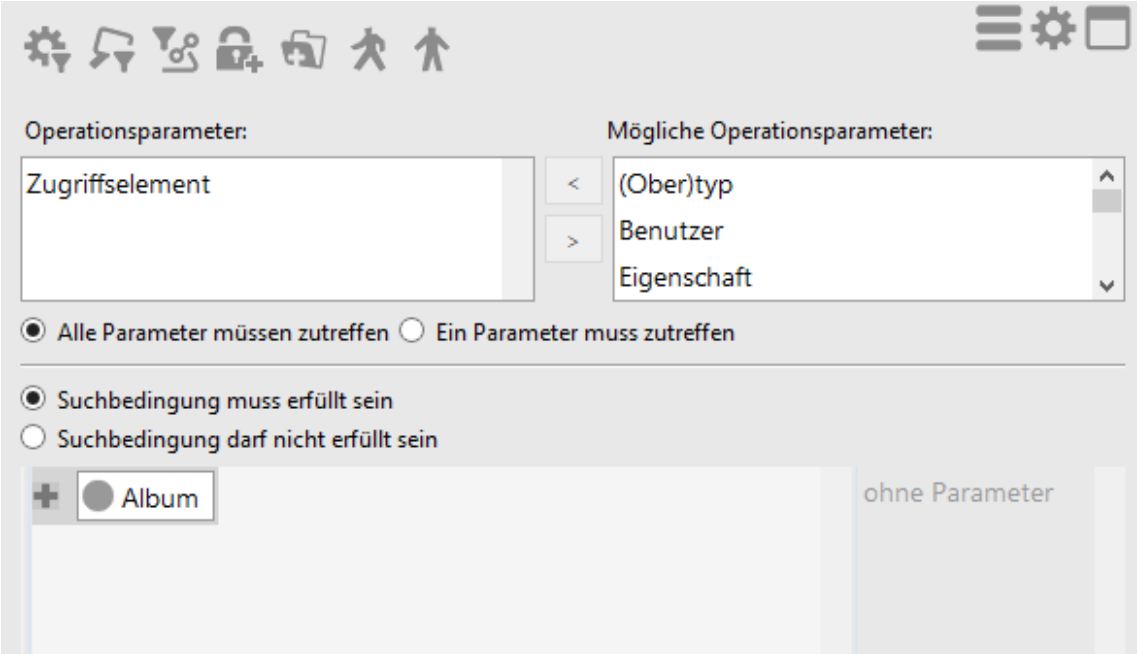
1.6.3.3 Expert Query Filter

Suchfilter ermöglichen es Elemente im Umfeld des Elementes, auf das zugegriffen werden soll, einzubeziehen. So können nicht nur einzelne Eigenschaften sondern auch Zusammen-

hänge zwischen Objekten, Eigenschaften und Attributen in die Rechte- bzw. Triggerdefinition einbezogen werden. Bei der Verwendung von Suchfiltern muss ein Operationsparameter angegeben werden, mit dem das Ergebnis der Strukturabfrage verglichen wird. Alle verfügbaren Operationsparameter werden im Kapitel Operationsparameter erklärt.

Es gibt zwei verschiedene Vorgehensweise Suchfilter zu definieren:

- *Suchbedingung muss erfüllt sein*: Diese Einstellung ist initial ausgewählt. Stimmt das Suchergebnis der Strukturabfrage mit dem Operationsparameter überein, ist die Bedingung des Filters erfüllt und nachfolgende Filter, Entscheider oder Trigger werden ausgeführt.
- *Suchbedingung darf nicht erfüllt sein*: Liefert die Strukturabfrage als Ergebnis das selbe Element wie der Zugriffsparameter, ist die Bedingung nicht erfüllt und die Prüfung des Rechte- bzw. Triggerbaumes wechselt zum nächsten Teilbaum. Ist das Ergebnis der Strukturabfrage ein anderes als der Zugriffsparameter liefert, ist die Bedingung erfüllt und der nachfolgende Filter, Entscheider oder Trigger wird ausgeführt.



Die Objekte des Typs links oben, die auf die Suchbedingung passen, sind das Ergebnis der Strukturabfrage. Diese werden mit dem Element, das vom Operationsparameter übergeben wird, verglichen. In der Strukturabfrage können Zugriffsparameter verwendet werden, mit diesen können beispielsweise der Benutzer, das Zugriffsobjekt usw. in die Suche einbezogen werden.

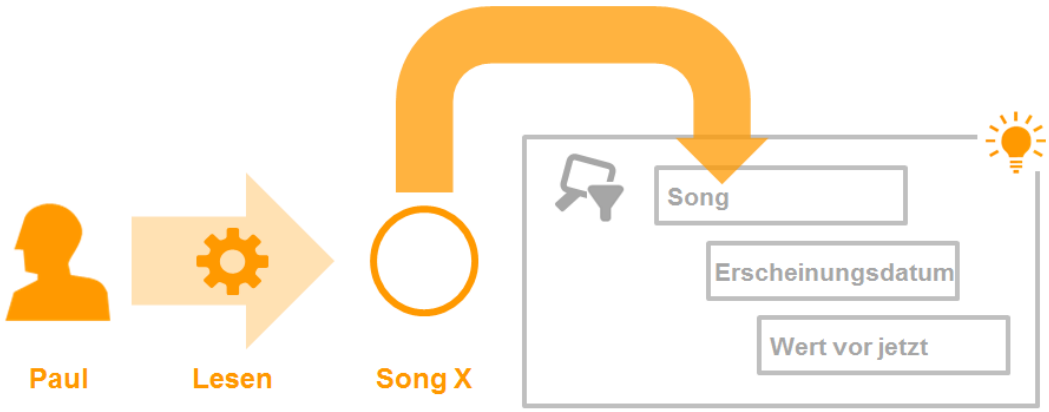
Bei der Auswahl der Operationsparameter kann konfiguriert werden, ob

- alle ausgewählten Parameter zutreffen müssen (*Alle Parameter müssen zutreffen*)
- oder nur ein Parameter zutreffen muss (*Ein Parameter muss zutreffen*).

Beachte: Initial ist die Einstellung *Alle Parameter müssen zutreffen* ausgewählt. Werden beispielsweise die Operationsparameter *Zugriffselement* und *Primärelement* ausgewählt, ist die Bedingung nur dann erfüllt, wenn das Ergebnis der Strukturabfrage sowohl Zugriffselement als auch Primärelement der zu prüfenden Operation ist.

Beispiel 1: Suchfilter im Rechtesystem

Es soll ein Recht definiert werden, das besagt, dass bereits veröffentlichte Songs von allen gesehen werden dürfen unveröffentlichte Songs hingegen nicht.

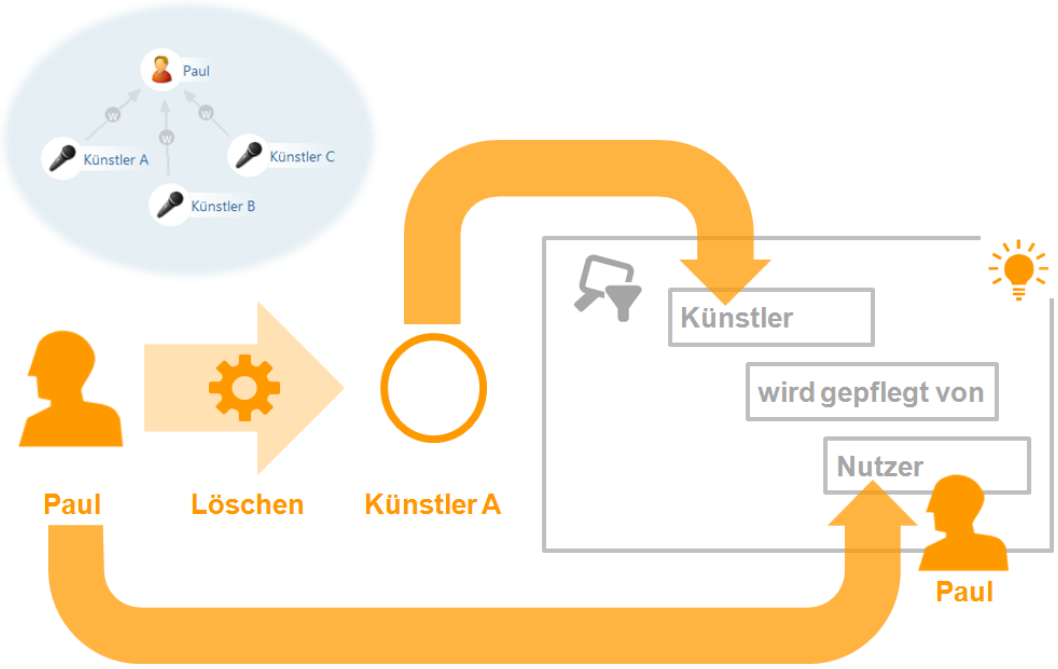


In diesem Beispiel möchte die Benutzer Paul den Song X lesen. Diese Operation wird nun vom Rechtesystem geprüft. Dort ist ein Suchfilter definiert, der prüft, ob der Song bereits veröffentlicht ist. In der Strukturabfrage des Suchfilters werden Objekte vom Typ Song gesucht, mit der Einschränkung, dass das Attribut Erscheinungsdatum in der Vergangenheit liegt. Die Strukturabfrage liefert alle Songs, die diese Bedingung erfüllen. Ist der Song X einer davon, fällt die Prüfung des Filters positiv aus und der auf den Suchfilter nachfolgende Ordner (mit einem Filter oder Entscheider) wird ausgeführt.

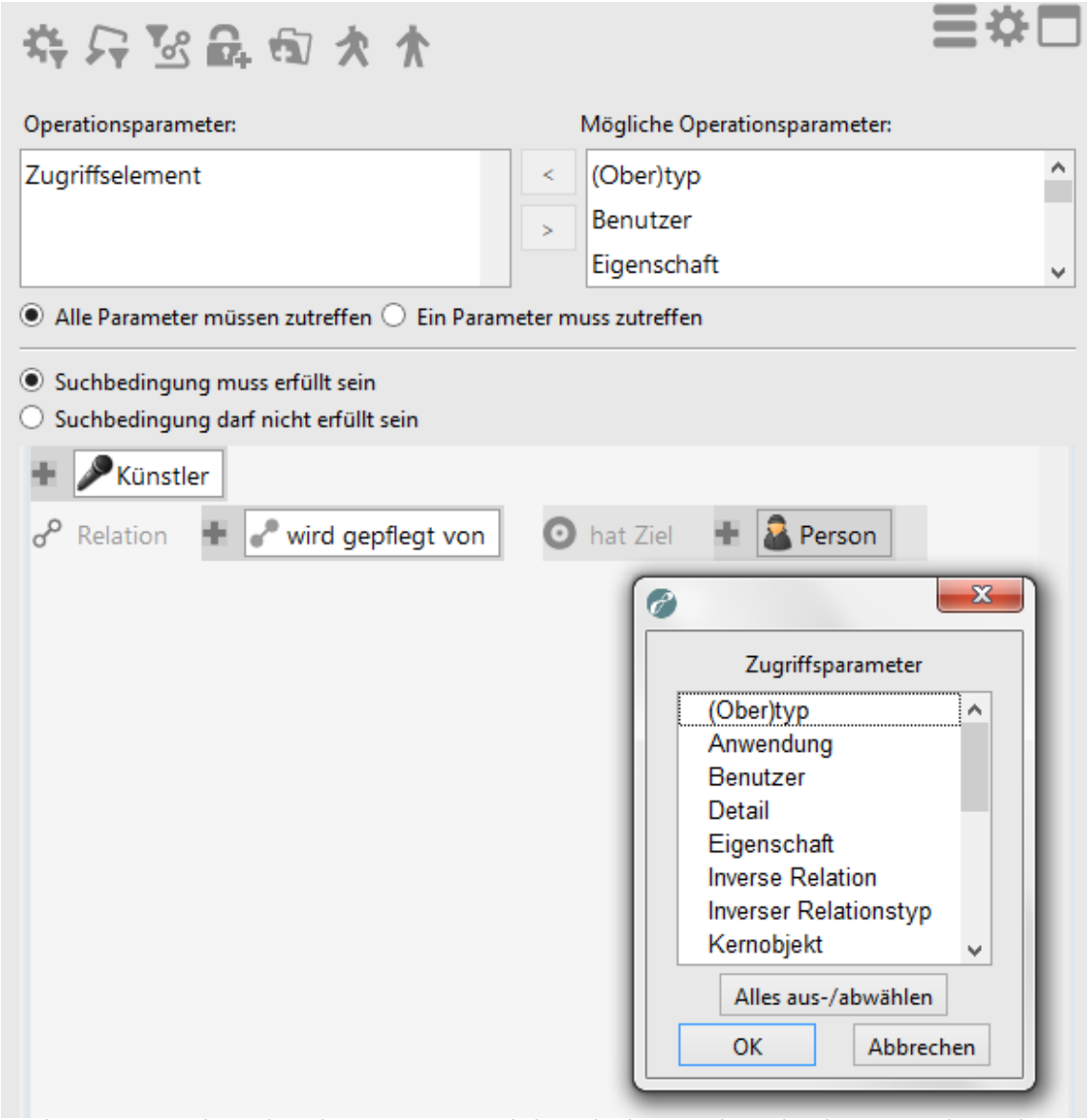
Bei dem Suchfilter wurden die Einstellungen Suchbedingung muss erfüllt sein und Alle Parameter müssen zutreffen ausgewählt.

Beispiel 2: Suchfilter im Rechtesystem

In den meisten Fällen gibt es eine Verbindung zwischen dem Benutzer, der zugreifen will und den Objekten oder Eigenschaften, auf die er zugreifen will. Ein Beispiel dafür wäre: "Mitarbeiter einer Abteilung, die eine Branche betreuen, dürfen alle Kunden aus dieser Branche bearbeiten." Eine andere Version dieses Beispiels, das unten dargestellt wird, wäre: "Nutzer, die einen Künstler pflegen, dürfen diesen bearbeiten und löschen."



Auf der linken Seite ist ein Ausschnitt des Wissensnetzes abgebildet: Das Objekt Paul ist mit den Objekten Künstler A, Künstler B und Künstler C über die Relation pflegt verknüpft. Die inverse Relation von pflegt ist wird gepflegt von, die zwischen den Objekten Künstler A, Künstler B, Künstler C und dem Objekt Paul besteht und im Suchfilter abgefragt wird. Diese Relation im semantischen Netz steht dafür, dass eine Person für die Datenpflege rund um einen Künstler verantwortlich ist.

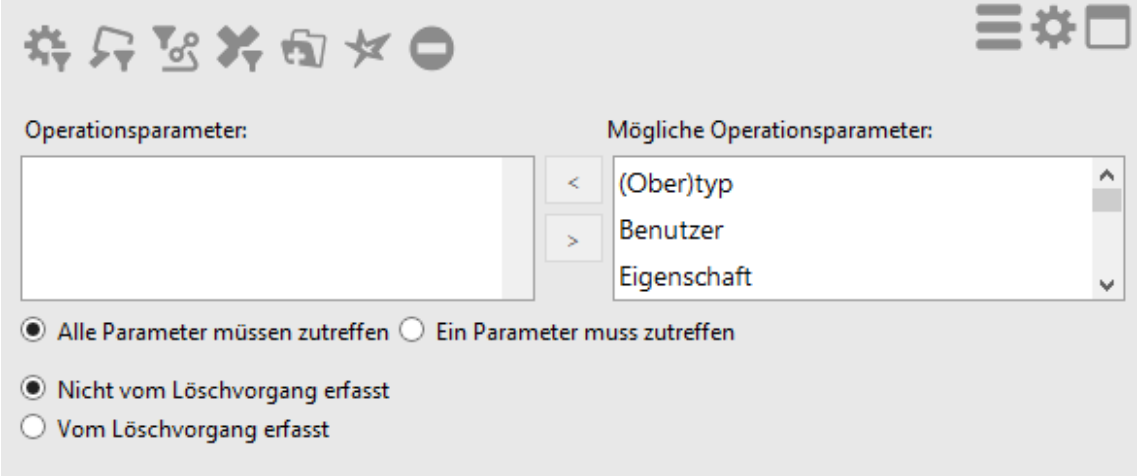


In diesem Beispiel möchte der Benutzer Paul das Objekt Künstler A löschen. Der dazugehörige Suchfilter liefert als Suchergebnis alle Künstler die von einem bestimmten Benutzer gepflegt werden. Der aktuelle Benutzer wird als Zugriffsparameter in die Strukturabfrage übergeben. Zugriffsparameter in Strukturabfragen werden im Kapitel Strukturabfragen erklärt. Somit liefert die Suche in dieser Zugriffsituation alle Künstler, die von Paul gepflegt werden. Da Künstler A einer davon ist, fällt die Prüfung des Suchfilters positiv aus.

Von der Zugriffssituation werden in diesem Beispiel zwei Aspekte in den Suchfilter eingebracht. Das ist der Künstler der gelöscht werden soll und der Benutzer. Der Suchfilter kann entsprechend auf zwei verschiedene Arten definiert werden. Entweder wird der Künstler als Zugriffselement an den Suchfilter übergeben und der Benutzer als Zugriffsparameter in der Strukturabfrage verwendet. Oder der Benutzer wird als Operationsparameter Benutzer an den Suchfilter übergeben und die Firma als Zugriffsparameter Zugriffselement in der Strukturabfrage verwendet.

1.6.3.4 *

Löschfilter stehen nur bei der Definition von Triggern zur Verfügung. Sie werden dazu eingesetzt, in einer Löschsituation zu testen, ob das übergeordnete Element auch von dem Löschvorgang betroffen ist. Will man beispielsweise, dass ein Trigger nicht ausgeführt wird, wenn ein Objekt samt all dessen Eigenschaften gelöscht wird, aber dann wenn eine bestimmte Eigenschaft des Objektes gelöscht wird, muss ein Löschfilter verwendet werden.



Bei der Definition eines Löschfilters, muss mindestens ein Operationsparameter angegeben werden, der bestimmt, die Löschung welches Objektes getestet werden soll.

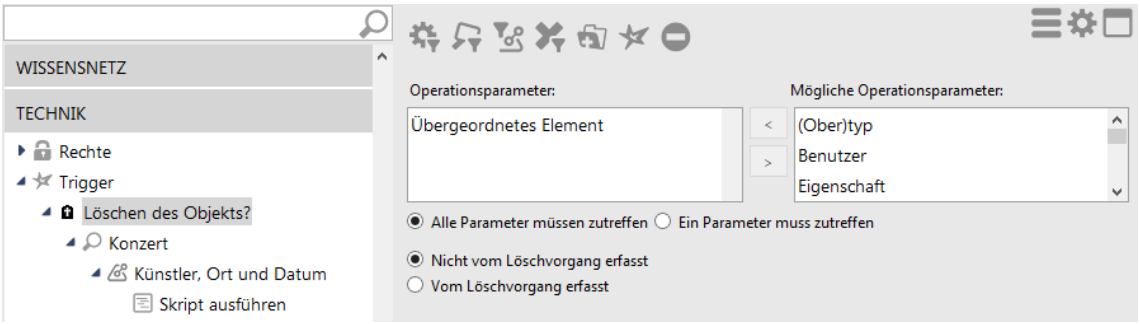
- *Alle Parameter müssen zutreffen*: Alle angegebenen Operationsparameter müssen zutreffen. Werden beispielsweise zwei Operationsparameter angegeben (Zugriffsobjekt und Primärobjekt), dann wird geprüft, ob der Löschvorgang sowohl für Zugriffsobjekt als auch für Primärobjekt gilt, das kann nur der Fall sein, wenn das Primärobjekt auch das Zugriffsobjekt ist.
- *Ein Parameter muss zutreffen*: Nur einer der angegebenen Operationsparameter muss zutreffen.

Anmerkung: In den meisten Fällen bietet sich der Operationsparameter übergeordnetes Element oder Primärobjekt an, da überprüft werden soll, ob entweder nur die Eigenschaft gelöscht wird, oder ob die Eigenschaft gelöscht wird, weil das gesamte Objekt gelöscht wurde.

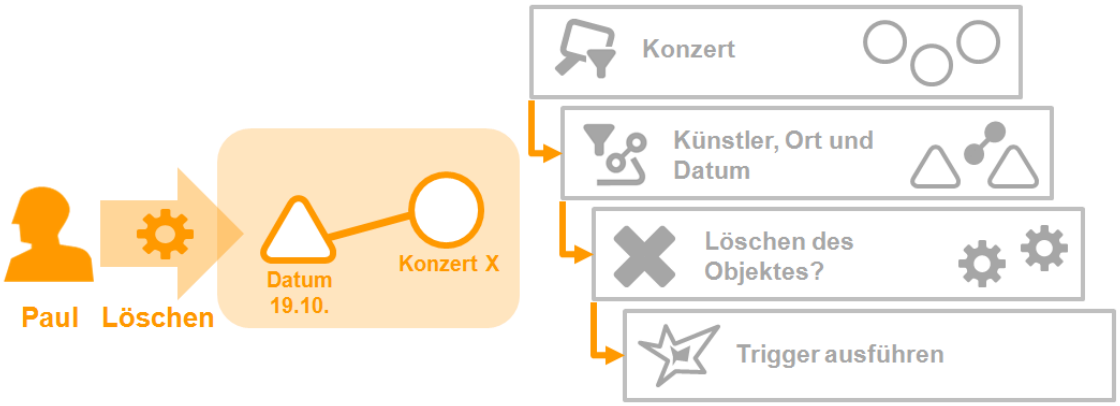
- *Nicht vom Löschvorgang erfasst*: Die Bedingung des Filters ist positiv, wenn das in Operationsparameter übergebene Element in dieser Transaktion nicht gelöscht wird.
- *Vom Löschvorgang erfasst*: Die Bedingung des Filters ist entsprechend positiv, wenn das in Operationsparameter übergebene Element in dieser Transaktion gelöscht wird.

Beispiel: Löschfilter bei Triggern

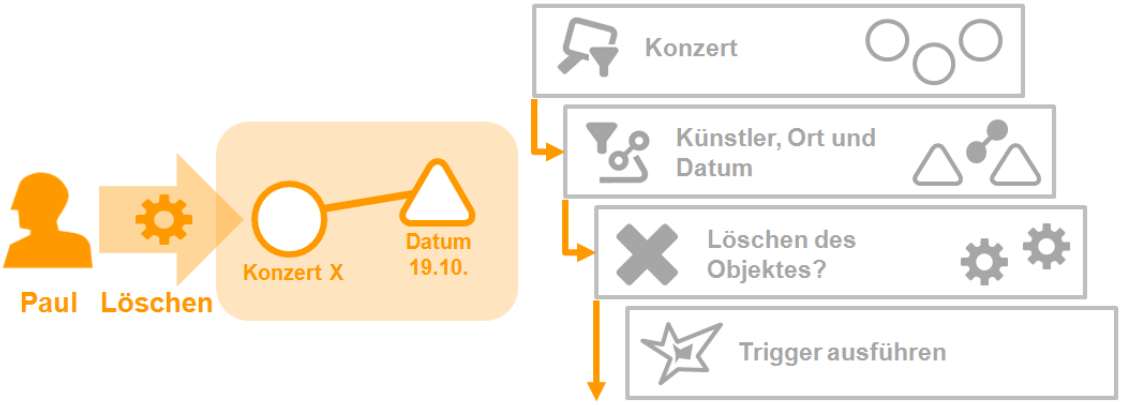
In diesem Beispiel soll ein Trigger nur dann ausgeführt werden, wenn der Künstler, der Ort oder das Datum einer Veranstaltung geändert oder gelöscht wird, aber nicht wenn das Objekt gelöscht wird, an denen die Eigenschaften gespeichert sind. Dafür wird die Einstellung *Nicht vom Löschvorgang erfasst* verwendet. Ist das übergeordnete Zugriffselement vom Löschvorgang erfasst, das in diesem Fall das Konzert-Objekt selbst ist, dann wird die Prüfung des Teilbaumes, aufgrund des negativen Ergebnisses des Filters, abgebrochen.



Verwendet wird der Operationsparameter Übergeordnetes Element und die Einstellung Nicht vom Löschvorgang erfasst.



In dieser beispielhaften Zugriffsituation wird das Attribut Datum mit dem Wert "19.10." am Objekt "Konzert X" gelöscht. Das Objekt selbst wird nicht gelöscht. Der Suchfilter "Konzert", der mit dem Operationsparameter übergeordnetes Zugriffselement definiert ist, und der Eigenschaftsfilter "Künstler, Ort und Datum" werden positiv beantwortet. Der darauffolgende Löschfilter liefert ebenfalls eine positive Antwort, da das Objekt an dem die Eigenschaft gespeichert ist (übergeordnetes Zugriffselement) nicht vom Löschvorgang betroffen ist - entsprechend der Einstellung Nicht vom Löschvorgang erfasst des Löschfilters.



In dieser Zugriffsituation wird das Objekt "Konzert X" vom Nutzer Paul gelöscht. Durch das Löschen des Objektes werden automatisch alle Eigenschaften des Objektes mit gelöscht - also auch alle Attribute des Objektes. Die Prüfung des Triggerbaums wird sowohl für die Löschung des Objektes

als auch des Attributes durchgeführt. Der Suchfilter "Konzert" und der Eigenschaftsfilter "Künstler, Ort und Datum" sind in der Prüfung des Triggerbaumes für den Löschvorgang des Attributs erfüllt. Der Löschfilter selbst ist in dieser Situation nicht erfüllt, da das Objekt "Konzert X" an dem die Eigenschaft "Datum 19.10." gespeichert ist, gelöscht wird.

Die Verwendung von Löschfiltern ist z.B. dann sinnvoll, wenn das Trigger-Skript den Namen des Objektes aus dessen Eigenschaften zusammensetzt. So wird der Name Objektes nicht erst mehrmals geändert, wenn die Eigenschaften des Objekts gelöscht werden, sondern das Objekt und alle damit verbunden Eigenschaften werden gelöscht ohne, dass das Skript ausgeführt wird, welches den Namen zusammensetzt. Dies erspart i.d.R. unnötige Berechnungszeit und kann in bestimmten Anwendungsszenarien, z.B. wenn der Trigger eine E-Mail Benachrichtigung schickt, dass ein Objekt umbenannt wird, durchaus sinnvoll sein (, da so das Verschicken von zahlreichen überflüssigen E-Mails zur Namensänderung vermieden wird).

1.6.4 operation parameters

Operationsparameter steuern bei Suchfiltern, mit welchem Element das Ergebnis der Strukturabfrage für die Prüfung der Bedingung verglichen werden soll. Im einfachsten Fall wird das Ergebnis mit dem Element verglichen, mit dem die zu prüfende Operation durchgeführt werden soll. Mithilfe von Operationsparametern kann das übergebene Element verändert werden. Es kann der aktuelle Benutzer oder Elemente aus dem Umfeld des Elements ausgewählt werden, die als Vergleichselement für den Suchfilter verwendet werden sollen.

Sie werden unter anderem auch bei Löschfiltern und Skript-Triggern verwendet. Dort geben sie an, ausgehend vom Element auf dem der Zugriff durchgeführt wird, auf welchem Element das Skript ausgeführt werden soll bzw. das Löschen welchen Elements gefiltert werden soll.

Wann ist dies sinnvoll? Statt des betroffenen Objekts ein Element aus dessen Umgebung zum Vergleich herziehen zu können, ist in manchen Fällen unverzichtbar: z.B. wenn es darum geht Zugriffsrechte für das Anlegen neue Objekte oder Typen zu prüfen. Es ist nicht möglich eine Strukturabfrage zu definieren, die das noch nicht angelegte Objekt zurückliefert. In diesem Fall muss der Suchfilter gegen etwas anderes verglichen werden, nämlich gegen den Typ des anzulegenden Objekts und bei Objekttypen gegen den Obertyp des anzulegenden Typs.

Operationsparameter	Beschreibung
(Ober)typ	Der (Ober)typ ist bei Typen der Obertyp des Typs. Bei Objekten ist der (Ober)typ der Typ des Objektes. Bei Attributen oder Relationen ist der (Ober)typ der Typ der Eigenschaft.
Benutzer	Der *Benutzer* ist das Objekt des Benutzers, der die Operation ausführt.
Eigenschaft	Die *Eigenschaft* ist die von der Operation betroffene Eigenschaft (Attribut oder Relation). Wird die Operation an einem Objekt, Typ oder Erweiterung durchgeführt, ist der Operationsparameter *Eigenschaft* leer.
Inverse Relation	Falls die von der Operation betroffene Eigenschaft eine Relation ist, enthält der Parameter die inverse Relationshälfte.

Inverser Relationstyp	Der *Inverse Relationstyp* ist der Typ der inversen Relation. Dieser kann bei Erzeugung von Relationen verwendet werden.
Kernobjekt	Wenn das übergeordnete Element eine Erweiterung ist, dann ist das *Kernobjekt* das Objekt an dem die Erweiterung gespeichert ist. Ansonsten ist das *Kernobjekt* identisch mit Zugriffselement.
Ordner	Der Operationsparameter *Ordner* ist der von der Operation betroffene Ordner.
Primäreigenschaft	Bei Metaeigenschaften ist die *Primäreigenschaft* die dem Objekt, Typ oder Erweiterung nächste Eigenschaft. Ansonsten ist *Primäreigenschaft* identisch mit Eigenschaft.
Primäres Kernobjekt	Wenn das Primärelement eine Erweiterung ist, dann ist das *Primäre Kernobjekt* das Kernobjekt der Erweiterung. Ansonsten ist das *Primäre Kernobjekt* identisch mit Kernobjekt.
Primäres Relationsziel	Das *Primäre Relationsziel* ist das Primärelement des Relationsziels.
Primärelement	Falls das übergeordnete Zugriffselement eine Eigenschaft ist, ist das *Primärelement* das Objekt, der Typ oder die Erweiterung an dem die Eigenschaft gespeichert ist (transitiv). Ansonsten ist das *Primärelement* identisch mit dem übergeordneten Element.
Relationsziel	Falls die von der Operation betroffene Eigenschaft eine Relation ist, enthält der Parameter *Relationsziel* das Relationsziel der Relationshälfte. (Die Relationsquelle wäre in diesem Fall das übergeordnete Element.)
Übergeordnetes Element	Das *Übergeordnete Element* ist das von der Operation betroffene Objekt, der Typ oder die Erweiterung. Bei Eigenschaften ist das *Übergeordnete Element* das Objekt, der Typ oder die Erweiterung an der die Eigenschaft gespeichert ist.
Zugriffselement	Das *Zugriffselement* ist das von der Operation betroffene Element.

1.6.4.1 Access Object

Das Zugriffselement ist das Element aus dem semantischen Netz auf das gerade zugegriffen wird. Bei Suchfiltern im Rechtesystem ist das Zugriffselement beispielsweise das Element auf das durch eine Operation zugegriffen werden soll. Beim Prüfen einer Zugriffsituation wird dann das Element an den Suchfilter übergeben, an dem die Operation durchgeführt werden soll. Der Suchfilter vergleicht dann das Zugriffselement mit dem Ergebnis der Strukturabfrage.

1.6.4.2 User

Der Parameter Benutzer ist unabhängig vom Zugriffselement immer das Benutzerobjekt des aktuell angemeldeten Nutzers. Hierfür muss der Knowledge-Builder-Account mit einem Wissensnetzobjekt verknüpft werden. Wie die Verknüpfung vorgenommen wird, wird im Kapitel Aktivierung des Rechtesystems vorgestellt.

Zugriffselement	Benutzer
Objekt, Typ, Erweiterung oder Eigenschaft	Objekt des aktuell angemeldeten Nutzers

1.6.4.3 Superior Type

Der Parameter (Ober)typ wird beispielsweise dann verwendet, wenn im Rechtesystem Operationen geprüft werden sollen, die neue Elemente anlegen. Beim Anlegen von Elementen kann der Suchfilter nicht so definiert werden, dass er das noch nicht angelegte Element findet. Der Suchfilter muss auf dem Obertyp oder Typ des Elements arbeiten, welches angelegt werden soll. Bei der Erstellung von Objekten, Attributen und Relationen wird der Typ des Objektes, Attributes oder der Relation verwendet. Bei Typen wird der Obertyp des anzulegenden Typs verwendet.

Zugriffselement	(Ober)typ
Objekt oder Erweiterung	Der Typ des Objektes oder der Erweiterung
Typ	Der Obertyp
Eigenschaft	Der Typ der Eigenschaft

1.6.4.4 Semantic Element

Das übergeordnete Element wird dann verwendet, wenn direkte Eigenschaften eines Elementes abgefragt werden sollen.

Zugriffselement	Übergeordnetes Element
Objekt, Typ oder Erweiterung	Das Zugriffselement selbst
Eigenschaft	Objekt, Typ oder Erweiterung an dem oder der die Eigenschaft gespeichert ist
Metaeigenschaft	Eigenschaft, an der die Metaeigenschaft gespeichert ist

1.6.4.5 Property

Als Eigenschaften werden Attribute und Relationen verstanden. Der Operationsparameter enthält das Attribute oder die Relation auf der die Operation durchgeführt wird. Wird die Operation auf einem Objekt oder Typ durchgeführt, ist der Operationsparameter Eigenschaft leer.

Zugriffselement	Eigenschaft
Attribute oder Relation	Das Zugriffselement selbst
Objekt, Typ oder Erweiterung	Leer

1.6.4.6 Inverse Relation

Die inverse Relation ist die "Gegenrichtung" einer Relationshälfte. Betrachtet man eine Relationshälfte als gerichteten Graphen, so besteht eine Relation aus zwei entgegengesetzten Graphen (der "Hinrichtung" und der "Rückrichtung" der Relation), die zwischen zwei Elementen aufgehängt ist. Die inverse Relation ist also die entgegengesetzte Relationshälfte. Die inverse Relationshälfte hat als Relationsziel die Relationsquelle der Relationshälfte und umgekehrt.

Zugriffselement	Inverse Relation
Relationshälfte	Die inverse Relationshälfte
Objekt, Typ, Erweiterung oder Attribut	Leer

1.6.4.7 Inverse Relation Type

Der inverse Relationstyp ist der Typ der inversen Relation.

Zugriffselement	Inverser Relationstyp
Relationshälfte	Typ der inversen Relationshälfte
Objekt, Typ, Erweiterung oder Attribut	Leer

1.6.4.8 Relation Target

Das Relationsziel ist nicht die Quelle sondern das "Ziel" einer Relationshälfte. Es kann auch als Relationsquelle der inversen Relationshälfte betrachtet werden.

Zugriffselement	Relationsziel
Relationshälfte	Das Relationsziel ist die Relationsquelle der inversen Relation
Objekt, Typ, Erweiterung oder Attribut	Leer

1.6.4.9 Primary Object

Das Primärelement liefert immer ein Objekt, Typ oder Erweiterung. Wird das Primärelement auf Metaeigenschaften ausgeführt, werden die Eigenschaften transitiv abgearbeitet, bis das Objekt, der Typ oder die Erweiterung gefunden wurde, an dem die Eigenschaften aufgehängt sind.

Zugriffselement	Primärelement
Objekt, Typ oder Erweiterung	Das Zugriffselement selbst
Eigenschaft	Objekt, Typ oder Erweiterung an dem oder der die Eigenschaft gespeichert ist
Metaeigenschaft	Objekt, Typ oder Erweiterung an dem oder der die Eigenschaft gespeichert ist, an der wiederum die Metaeigenschaft gespeichert ist (transitiv)

1.6.4.10 Primary Relation Target

Das primäre Relationsziel ist im Gegensatz zum Primärelement einer Relationshälfte nicht das Objekt, der Typ oder die Erweiterung an der die Relationshälfte angebracht ist sondern das Objekt, der Typ oder die Erweiterung an der die inverse Relationshälfte aufgehängt ist.

Zugriffselement	Primäres Relationsziel
Relationshälfte	Das Primärelement des Relationsziels (Objekt, Typ oder Erweiterung an dem oder der die inverse Relationshälfte gespeichert ist)
Relationshälfte deren Relationsziel eine Eigenschaft oder Metaeigenschaft ist	Das Primärelement des Relationsziels (Objekt, Typ oder Erweiterung der Metaeigenschaft oder Eigenschaft an der die inverse Relationshälfte gespeichert ist)
Objekt, Typ, Erweiterung oder Attribut	Leer

1.6.4.11 Core Object

Das Kernobjekt wird verwendet, wenn mit Erweiterungen gearbeitet wird. Das Kernobjekt liefert anstatt der Erweiterung das Objekt, an dem die Erweiterung gespeichert ist.

Zugriffselement	Kernobjekt
Objekt, Typ oder Eigenschaft	Das Zugriffselement selbst
Erweiterung	Das Objekt an dem die Erweiterung gespeichert ist

1.6.4.12 Primary Core Object

Wenn bei einem Element das zugehörige Objekt oder der zugehörige Typ verarbeitet werden soll, muss das primäre Kernobjekt verwendet werden. Im Gegensatz zum Primärelement werden keine Erweiterungen zugelassen. Bei diesen wird das Kernobjekt ausgegeben.

Zugriffselement	Primäres Kernobjekt
Erweiterung	Das Objekt an dem die Erweiterung gespeichert ist
Objekt oder Typ	Das Zugriffselement selbst
Eigenschaft oder Metaeigenschaft einer Erweiterung	Das Objekt an dem die Erweiterung gespeichert ist
Eigenschaft oder Metaeigenschaft eines Objektes oder Typs	Primärelement - Objekt oder der Typ an dem die Eigenschaft gespeichert ist (transitiv)

1.6.4.13 Primary Property

Die Primäreigenschaft ist immer eine Eigenschaft. Sie ähnelt dem Primärelement in der Hinsicht, dass sie transitiv Metaeigenschaften abarbeitet. Sie liefert aber im Gegensatz die letzte Eigenschaft die vor dem Primärelement kommt - also die Eigenschaft, die direkt am Primärelement gespeichert ist.

Zugriffselement	Primäreigenschaft
Eigenschaft	Das Zugriffselement selbst
Metaeigenschaft (oder Metaeigenschaft einer Metaeigenschaft)	Die Eigenschaft, die dem Objekt, Typ oder der Erweiterung am nächsten ist

Objekt, Typ oder Erweiterung	Leer

1.6.4.14 Folder

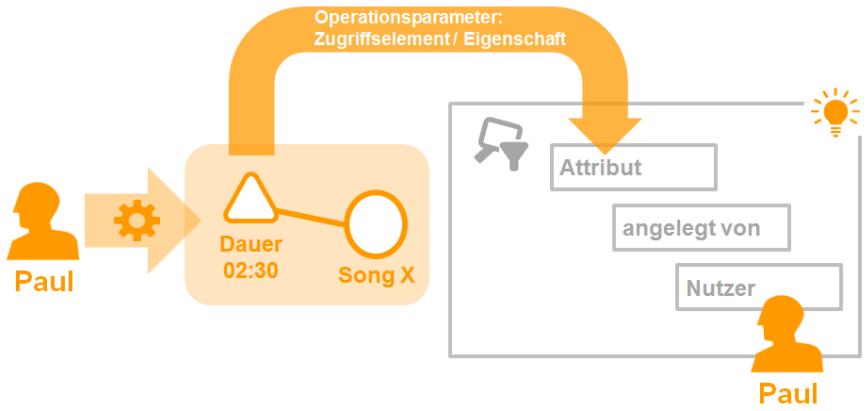
Soll ein Ordner aus dem Bereich *Ordner* des Wissensnetzes als Parameter an die Suche übergeben werden, dann muss der Operationsparameter Ordner verwendet werden.

Zugriffselement	Ordner
Ordner	Das Zugriffselement selbst
Objekt, Typ, Erweiterung oder Eigenschaft	Leer

1.6.4.15 Examples

Beispiel 1: Zugriffselement und Eigenschaft im Rechtesystem

Das unten aufgeführte Beispiel zeigt auf der linken Seite die Zugriffsituation und auf der rechten Seite den dazugehörigen Suchfilter.



Zugriffsituation: Der Nutzer Paul möchte das Attribut Dauer von Song X ändern.

Suchfilter: Es werden alle Attribute gefiltert die, die von einem bestimmten Benutzer angelegt wurden. In der Strukturabfrage wird der Zugriffsparameter Benutzer verwendet, der die Objekte von Nutzer auf die Person einschränkt, welche die Operation ausführen möchte. Entsprechend sind das alle Attribute, die von Paul angelegt wurden.

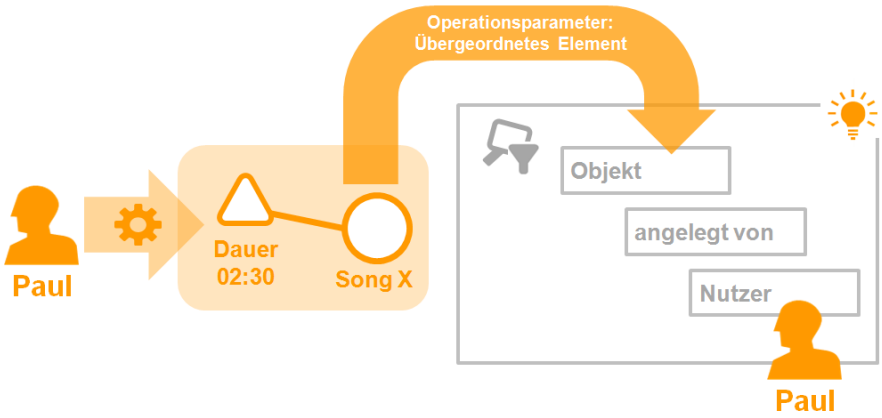
Prüfung der Zugriffsrechte: Für die Prüfung der Zugriffsrechte wird das Attribut (das Zugriffselement/die Eigenschaft), an dem die Operation durchgeführt werden soll, an den Suchfilter übergeben. Ist dieses Attribut in der Menge der Suchergebnisse enthalten, dann ist die Prüfung des Suchfilters positiv.

Operationsparameter: Das Attribut Dauer selbst wird an den Suchfilter übergeben. In

diesem Fall könnte sowohl der Operationsparameter Zugriffselement als auch Eigenschaft verwendet werden, da das Attribut Dauer selbst eine Eigenschaft ist und das Zugriffselement der Operation darstellt.

Beispiel 2: Übergeordnetes Element und Primärelement im Rechtesystem

Dieses Beispiel zeigt auf der linken Seite die Zugriffsituation und auf der rechten Seite den dazugehörigen Suchfilter.



Zugriffssituation: Der Nutzer Paul nimmt eine Änderung des Attributes Dauer, das aktuell den Wert 02:30 annimmt und zum Objekt Song X gehört, vor.

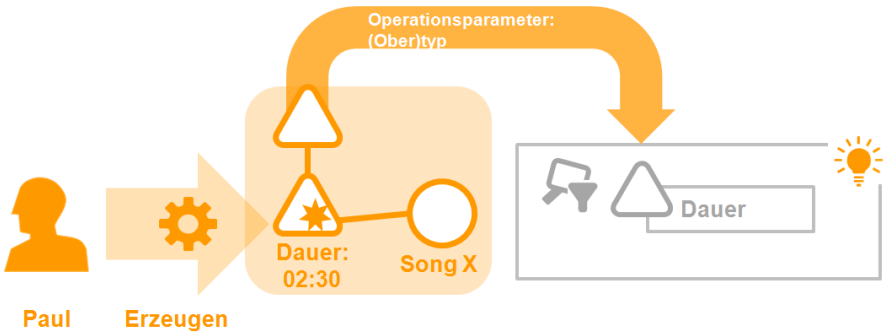
Suchfilter: Der Suchfilter ist so definiert, dass er alle Objekte sucht, die von einem bestimmten Benutzer angelegt wurden, das ist als Zugriffselement der aktuell angemeldete Nutzer. Entsprechend findet der Suchfilter alle Objekte, die von Paul angelegt wurden.

Prüfung der Zugriffsrechte: Ist in der Ergebnismenge des Suchfilters der Song X enthalten, wird der nachfolgende Ordner (Filter oder Entscheider) ausgeführt.

Operationsparameter: Die Verwendung des Operationsparameter übergeordnetes Element führt dazu, dass nicht das Attribut Dauer an dem die Änderung stattfinden soll, an den Suchfilter übergeben wird, sondern das Objekt an dem es definiert wurde. Das ist in diesem Fall der Song X. Neben dem übergeordneten Element könnte in diesem Fall auch der Operationsparameter Primärelement verwendet werden. Der Operationsparameter übergeordnetes Element führt dazu, dass alle Eigenschaften und das Objekt selbst positiv von Filter bewertet würde. Zusätzlich würde der Operationsparameter Primärelement auch Metaeigenschaften des Objektes zulassen, egal wie viele andere Eigenschaften zwischen Objekt und Metaeigenschaft hängen.

Beispiel 3: (Ober)typ im Rechtesystem

Das Beispiel stellt auf der linken Seite die Zugriffsituation dar und auf der rechten Seite wird der Suchfilter abgebildet, der in dieser Situation zum Einsatz kommt.



Zugriffsituation: Der Nutzer Paul möchte das Attribut Dauer am Objekt Song X erstellen. Es soll den Wert 02:30 haben.

Suchfilter: Der Suchfilter liefert den Attributtyp Dauer.

Prüfung der Zugriffsrechte: Ist das zu erstellende Attribut vom Typ Dauer, dann fällt die Prüfung des Suchfilters positiv aus.

Operationsparameter: Bei der Erstellung von Elementen, kann kein Suchfilter definiert werden, der das zu erstellende Element zurückliefert und damit die Zugriffsrechte prüfen kann. Bei der Erstellung von Elementen muss also ein anderer Operationsparameter als Zugriffselement ausgewählt werden. Der Operationsparameter (Ober)typ ist in diesen Situationen geeignet. In diesem Beispiel wird der Typ des Attributes verwendet, das ist der Attributtyp Dauer.

1.6.5 Operations

In Operationsfiltern können Operationen angegeben werden, die dann im Filterprozess von Operationsfilter zugelassen werden. Wird in der Zugriffssituation eine andere Operation ausgeführt, als im Operationsfilter angegeben, wird bei der Traversierung des Rechte bzw. Triggerbaumes zum nächsten Teilbaum gewechselt.

Die allgemeinen Operationen *Erzeugen*, *Lesen*, *Modifizieren* und *Löschen* bestehen aus mehreren einzelnen Operationen. Wird eine der Operationsgruppen verboten, werden somit auch alle darin enthaltenen Operationen nicht erlaubt und umgekehrt wird eine Operationsgruppe erlaubt, so werden alle enthaltenen Operationen automatisch mit erlaubt.

Die Tabelle zeigt eine Übersicht zu allen verfügbaren Operationen, die in Operationsfiltern ausgewählt werden können. Je nach Operation können nur bestimmte Operationsparameter in Suchfiltern verwendet werden. Diese werden in der Spalte Operationsparameter angegeben.

Anmerkung: Abgeleitete Operationsparameter wie z.B. Primärelement oder primäres Kernobjekt können immer dann eingesetzt werden, wenn der Parameter von dem sie abgeleitet sind, verwendet werden kann.

Besonderheiten bei Triggern
Bei Triggern können keine lesenden Operationen verwendet werden. Außerdem stehen bei Triggern die Operationsgruppen Abfrage (Operation: In Strukturabfrage verwenden), Anzeige von Objekten (Operation: Im Grapheditor anzeigen) und Bearbeiten (Operation: Attributwert validieren) nicht zur Verfügung.

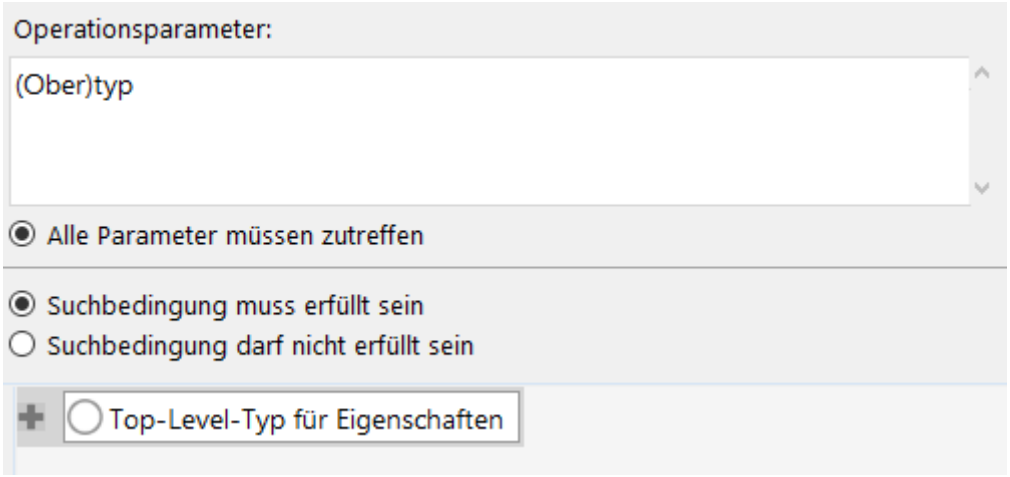
Außerdem steht bei den Erzeugen Operationen bei Triggern der Operationsparameter Zugriffselement zur Verfügung, wenn Zeitpunkt/Art der Ausführung auf *Nach der Änderung* oder *Ende der Transaktion* gesetzt ist.

Operationsgruppe	Operation	Operationsparameter
Abfrage	In Strukturabfrage verwenden	Zugriffselement
Anzeigen von Objekten	im Grapheditor anzeigen	Zugriffselement
Bearbeiten	Attributwert validieren	Zugriffselement, Eigenschaft, übergeordnetes Element, (zu prüfender Parameter: Attributwert)
Benutzerdefinierte Operation		
Erzeugen	Attribut erzeugen	(Ober)typ, übergeordnetes Element
	Erweiterung erzeugen	(Ober)typ, übergeordnetes Element, Kernobjekt
	Objekt erzeugen	(Ober)typ
	Ordner erzeugen	Ordner
	Relation erzeugen	(Ober)typ, übergeordnetes Element, Relationsziel, inverser Relationstyp
	Relationshälfte erzeugen	(Ober)typ, übergeordnetes Element, Relationsziel
	Typ erzeugen	(Ober)typ
	Übersetzung hinzufügen	Zugriffselement, Eigenschaft, übergeordnetes Element
Lesen	Alle Objekte/Eigenschaften des Typs lesen	(Ober)typ
	Attribut lesen	Zugriffselement, Eigenschaft, übergeordnetes Element
	Objekt lesen	Zugriffselement, übergeordnetes Element
	Relation lesen	Zugriffselement, übergeordnetes Element, Eigenschaft, inverse Relation, Relationsziel, inverses Relationsziel
	Typ lesen	Zugriffselement, übergeordnetes Element
Löschen	Attribut löschen	Zugriffselement, übergeordnetes Element

	Erweiterung löschen	Zugriffselement, Eigenschaft, übergeordnetes Element
	Objekt löschen	Zugriffselement, übergeordnetes Element
	Ordner löschen	Ordner
	Relationshälfte löschen	Zugriffselement, inverse Relation, Eigenschaft, übergeordnetes Element, Relationsziel, inverses Relationsziel
	Typ löschen	Zugriffselement, übergeordnetes Element
	Übersetzung entfernen	Zugriffselement, Eigenschaft, übergeordnetes Element
Modifizieren	Arttributwert modifizieren	Zugriffselement, Eigenschaft, übergeordnetes Element
	Ordner modifizieren	Ordner
	Schema modifizieren	Zugriffselement, übergeordnetes Element
	Typ wechseln	Zugriffselement, übergeordnetes Element
Werkzeuge verwenden	Export	
	Import	
	Script bearbeiten/ausführen	

Objekt lesen
Die Operation *Objekt lesen* deckt das Anzeigen von Objekten auf dem Reiter Objekte bei dem enstprechenden Objekttyp ab. Die Operation verbietet aber nicht das Anzeigen des Objektes, wenn es über ein verknüpftes Objekt aufgerufen wird. In diesem Fall gelten dann die Operationen für Eigenschaften *Attribut lesen* und *Relation lesen*.

Alle Objekte/Eigenschaften des Typs lesen
Diese Operation steuert speziell die Leserechtprüfung bei der Abarbeitung einer Struktursuche. Standardmäßig prüft eine Struktursuche alle Zwischenergebnisse. Eine Suche nach allen Mitarbeitern mit Gehalt größer 10.000€ würde also keine Treffer liefern, wenn das Gehalt nicht lesbar ist, auch wenn die entsprechenden Mitarbeiter-Objekte lesbar wären. Dieses Verhalten ist oft erwünscht, aber selten performant. Speziell bei einem umfangreich konfigurierten Rechtesystem, dessen Abarbeitung signifikant viel Rechenleistung erfordert, empfiehlt sich die Steuerung, welche Zwischenergebnisse einer Struktursuche nicht geprüft werden müssen, weil eine Prüfung der Endergebnisse ausreichend ist. In den meisten Wissensnetzen kann für alle Eigenschaftstypen ("Top-Level-Typ für Eigenschaften") eine Erlaubnis erteilt werden.

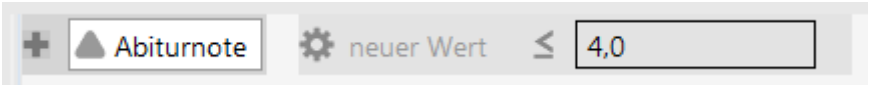


Zur Überprüfung, welche Zwischenergebnisse geprüft werden, kann man diese Information in einer Struktursuche einblenden lassen. Dies geschieht über "Einstellungen->Persönlich->Strukturabfrage->Leserechtprüfungen anzeigen".

In Strukturabfrage verwenden (veraltet)
Ist ein negatives Zugriffsrecht für ein Element definiert, das auf die Operation *In Strukturabfrage verwenden* gefiltert wird, dann darf das Element nicht in einer Strukturabfrage verwendet werden. Es wird auch dann nicht in Strukturabfragen berücksichtigt, wenn der (abstrakte) Obertyp angegeben wird.

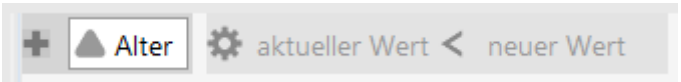
Attributwert validieren
Die Operation *Attributwert validieren* wird dann verwendet, wenn der zu setzende Attributwert bestimmte Bedingungen erfüllen muss. Die Definition der Bedingung an den Attributwert wird in einer Strukturabfrage gemacht. Dort stehen für die Validierung des Attributwertes zwei Definitionsmöglichkeiten zur Verfügung:

- *Bedingung für den zu setzenden Attributwert*:
  Der neue Wert des Attributes kann durch Vergleich mit einem angegebenen Wert in der Strukturabfrage validiert werden.

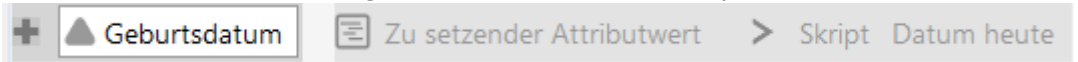
  Beispiel: Der Attributwert darf nur kleiner gleich 4,0 sein.

- *Vergleiche mit dem zu setzenden Attributwert*:
  Hierbei wird der aktuelle Wert mit dem neuen Wert verglichen.

  Beispiel: Der neue Wert des Attributs Alter darf in diesem Fall nur größer werden. Kleinere Werte werden nicht zugelassen.

- Vergleiche den zu setzenden Wert mit dem Ergebnis eines Skripts:
  Hierbei wird zunächst ein Vergleichswert mittels eines Skriptes ermittelt.

  Das Skript wird mit einem Parameter-Objekt aufgerufen, welches folgende Eigenschaften enthält:

Für die Validierung stehen verschiedene Vergleichsoperatoren zur Verfügung, mit denen der

zu setzende Attributwert gegen einen anderen Wert geprüft werden kann.
Entspricht der neue Wert nicht der definierten Bedingung, so ergibt die Prüfung des Filters ein negatives Ergebnis, sofern die initiale Einstellung *Suchbedingung muss erfüllt sein* ausgewählt ist.

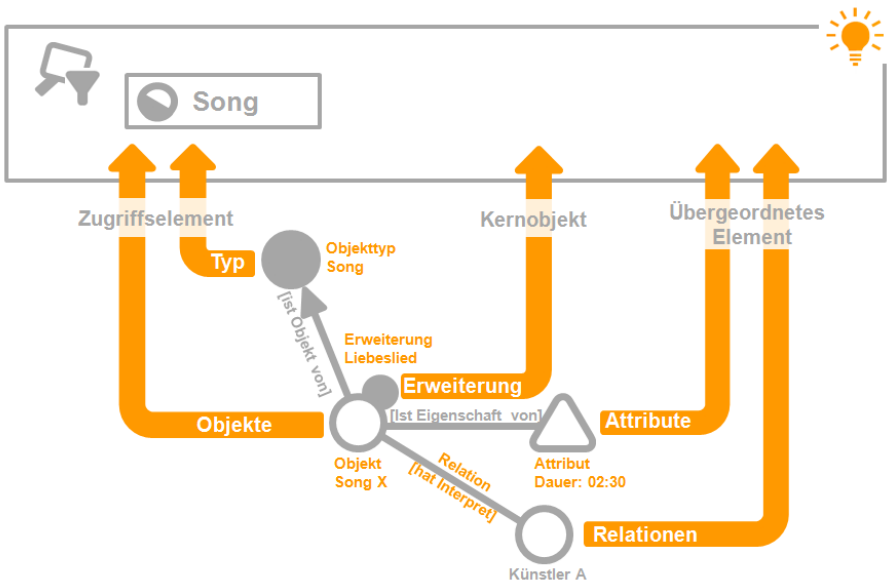
Schema modifizieren
Die Operation Schema modifizieren, betrifft Änderungen am Definitionsbereich von Relationen und Änderungen an der Typenhierarchie (*ist Untertyp von* und *ist Obertyp von* Relationen).

1.6.5.1 *

In diesem Beispiel wird gezeigt wie Operationsgruppen (Lesen, Erzeugen, Modifizieren, Löschen) bei der Rechtedefinition sinnvoll eingesetzt werden können. Es sollen alle Operationen für den Typ Song und dessen Objekte verboten werden. Dies umfasst die folgenden Aktionen:

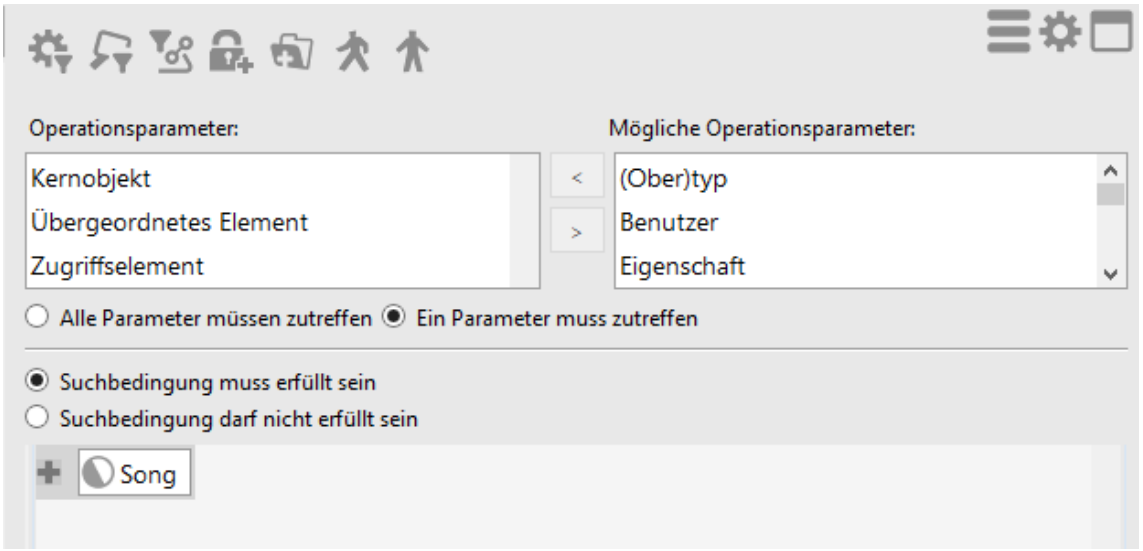
- Das Löschen des Objekttyps Song
- Das Löschen von bestimmten Songs (Objekte von Songs)
- Das Löschen von Attributen, welches an einem Song vorkommt
- Das Löschen von Relationen, die an einem Song vorkommt (Relationsziel und -quelle)
- Das Löschen von Erweiterungen, die Objekte von Song erweitern
- Das Löschen von Attribut- und Relationstypen die Objekte oder Untertypen von Song als Definitionsbereich haben

Sollen beispielsweise alle Löschen Operationen bei einem Objekt und dem dazugehörigen Typen verboten werden, muss man bei der Auswahl der Operationsparameter im Suchfilter des Rechtes darauf achten alle Löschen Operationen durch die entsprechenden Parameter abzudecken:



Der verwendete Suchfilter hat als einzige Bedingung den Objekttyp Song, bei dem die Einstellung Objekte und Untertypen ausgewählt ist. Der Operationsparameter Zugriffselement deckt den Objekttyp Song und alle Objekte, die zu diesem Typ gehören, ab. Der Parameter Kernobjekt deckt die Erweiterungsobjekte ab, die zu Songs gehören. Attribute und Relationen werden durch den Operationsparameter übergeordnetes Element abgedeckt.

Im Rechtebaum kommt der Operationsfilter der Operation Löschen an erster Stelle. Darauf folgt der unten abgebildete Suchfilter und als letztes der Entscheider Zugriff verweigert.

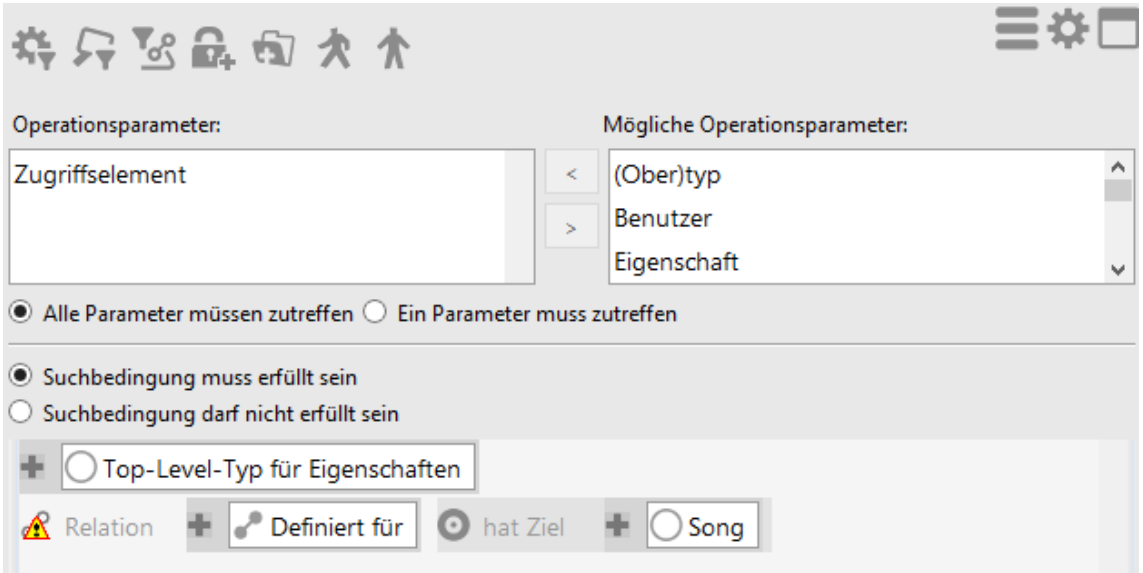


Im Beispiel verwendeter Suchfilter: Kernobjekt, übergeordnetes Element und Zugriffselement wurden als Operationsparameter ausgewählt. Die Einstellungen Ein Parameter muss zutreffen und Suchbedingung muss erfüllt sein werden verwendet.

Erweiterung des Rechtes um Attribut- und Relationstypen

Ein so definiertes Recht deckt die alle bis auf einen der oben formulierten Anforderungspunkte des Rechtes ab. Lediglich das Löschen von Attribut- und Relationstypen, die für Objekte und Untertypen von Songs definiert sind, wird in dieser Rechtedefinition nicht berücksichtigt.

Eine Erweiterung der Rechtedefinition wird durch den folgenden Filter realisiert:

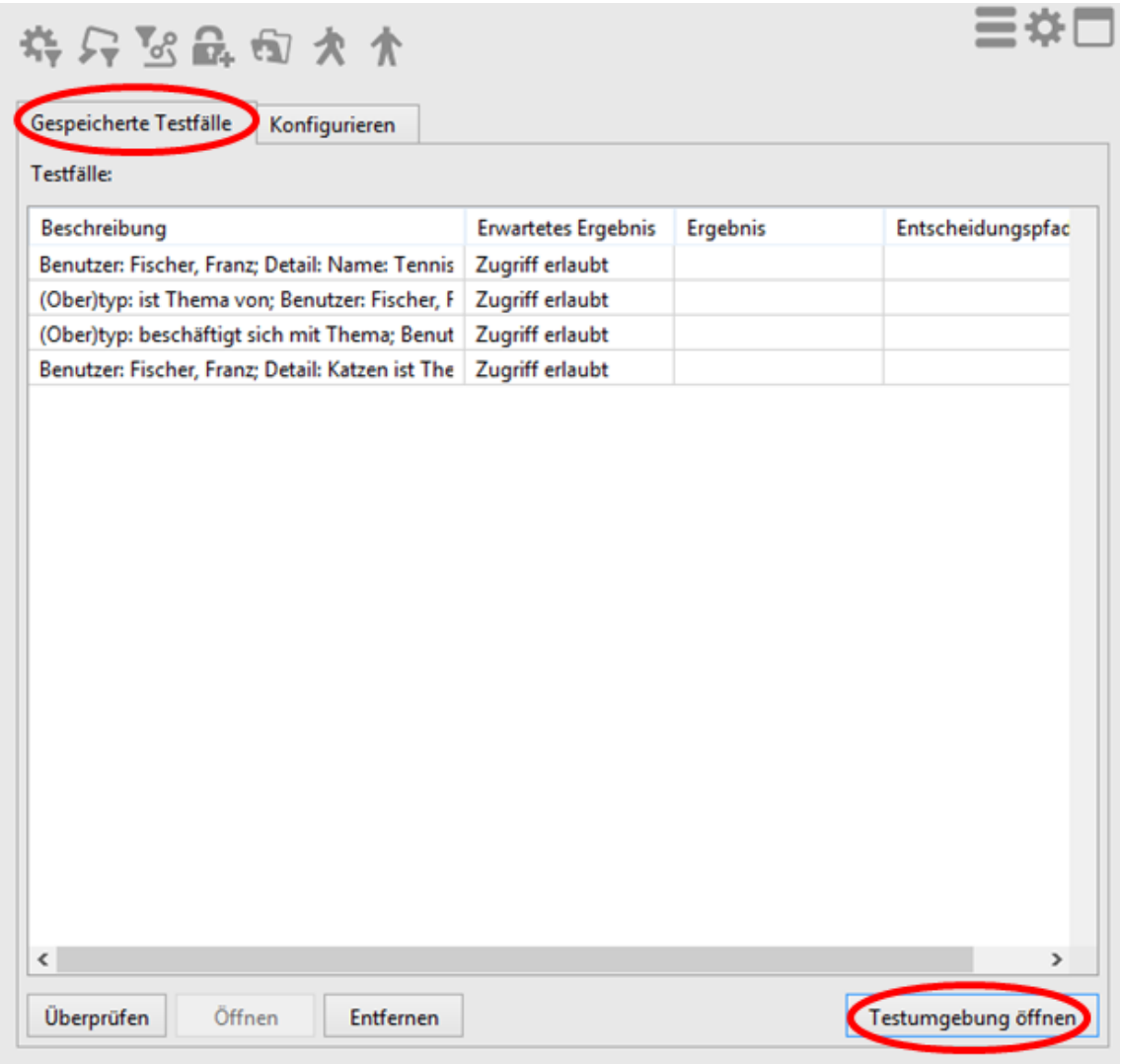


Der Suchfilter erfasst alle Eigenschaftstypen (Attribut- und Relationstypen) die für Objekte bzw. Untertypen von Songs definiert sind. In der Suchfilterdefinition wird der Parameter Zugriffselement und die Einstellung Suchbedingung muss erfüllt sein verwendet.

1.6.6 Test Environment

Wird im Bereich *System* der Ordner *Rechte* ausgewählt, werden im Hauptfenster die Reiter *Gespeicherte Testfälle* und *Konfigurieren* angeboten. Der Bereich des Testsystems befindet sich im Reiter *Gespeicherte Testfälle*. Das Testsystem für Trigger wird über den Bereich *System* im Ordner *Trigger* aufgerufen.

Hier können die gespeicherten Testfälle erneut getestet werden. Die Testoberfläche in der die Testfälle definiert werden können, kann über die Schaltfläche *Testumgebung öffnen* aufgerufen werden.



Zusätzlich zu den Funktionalitäten, die in den folgenden Kapiteln Eine Zugriffssituation testen und Testfälle definieren beschrieben werden, gibt es die Möglichkeit direkt an einem Objekt oder Typ Zugriffsrechte zu testen. Über das Kontextmenü (rechte Maustaste) die Funktion Zugriffsrechte auswählen. Dort stehen die folgenden Menüpunkte zur Auswahl:

- **Objekt:** Es werden alle Operationen (Modifizieren, Löschen, Lesen und im Graph-Editor anzeigen) am Objekt geprüft und deren Ergebnis ausgegeben.
- **Alles:** Es werden alle Operationen (Modifizieren, Löschen, Lesen und im Graph-editor anzeigen) am Objekt und all dessen Eigenschaften (Attribute und Relationen) geprüft.

- **Testumgebung Berechtigungssystem:** Die Testumgebung für die Rechteprüfung wird geöffnet.

1.6.6.1 Test Environment

Zum Testen des Rechtesystems und der Trigger-Funktionalität sind zwei Bereiche relevant:

- Die Testumgebung selbst: Die Testumgebung bietet die Möglichkeit für einen bestimmten Testfall die Zugriffsrechte bzw. wann ein Trigger ausgeführt wird zu testen.
- Der Reiter *Gespeicherte Testfälle*: Hier werden die Testfälle aufgelistet und für spätere Überprüfungen zur Verfügung gestellt.

Anleitung zum Öffnen der Testumgebung

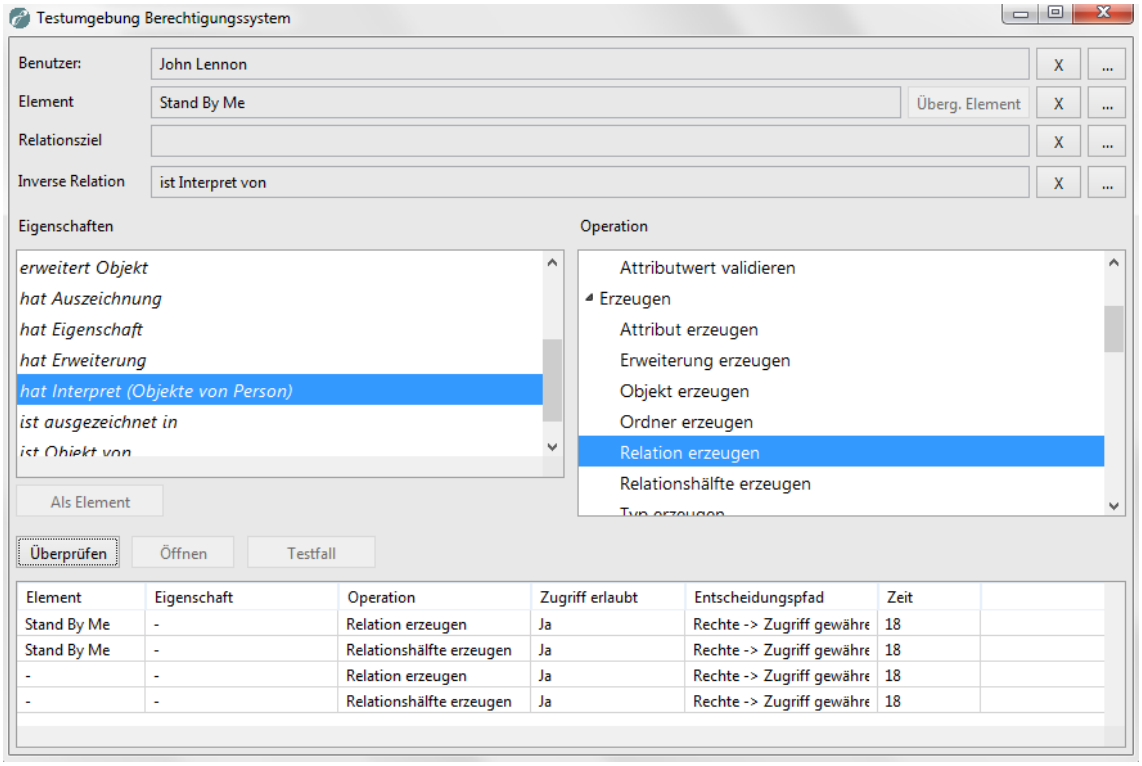
1. Wählen Sie im Knowledge-Builder im Bereich *Technik* den Ordner *Rechte* bzw. *Trigger* aus.
2. Wenn Sie im Rechtesystem arbeiten, wählen Sie im Hauptfenster den Reiter *Gespeicherte Testfälle* aus.
3. Klicken Sie *Testumgebung öffnen* (rechts unten) an, damit sich die Testumgebung in einem neuen Fenster öffnet.

Die Testumgebung besteht aus mehreren Bereichen: Im oberen Bereich wird der Benutzer und das Element definiert, an dem die Eigenschaft angebracht ist, die geprüft werden soll. Das Element kann ein Objektes, ein Typs oder eine Eigenschaft (wenn diese als Element übergeben wird) sein.

Der Bereich *Eigenschaften* listet alle Eigenschaften des ausgewählten Elements aus. Nicht kursive Eigenschaften, sind konkrete Eigenschaften, die bereits am Objekt oder der Eigenschaft vorliegen. Kursive Eigenschaften hingegen sind Eigenschaften, die vom Schema her angelegt werden können, aber noch nicht wurden. Soll die Erstellung einer neuen Eigenschaft getestet werden, muss die Eigenschaft in kursiv-Form ausgewählt werden.

Im Fenster *Operation* kann die Operation ausgewählt werden, die getestet werden soll. Je nach ausgewählten Parametern, ist eine Rechteprüfung möglich oder nicht.

Beachte: Soll eine Eigenschaft einer Eigenschaft also eine Metaeigenschaft getestet werden, dann muss die Eigenschaft im Eigenschaftsfenster markiert werden und die Schaltfläche *Als Element* ausgewählt werden. Dann wird beispielsweise bei Relationen die konkrete Relation zwischen zwei Objekten oder Eigenschaften als Objekt ausgewählt. Jetzt stehen im Eigenschaftsfenster alle Eigenschaften der konkreten Relation zur Verfügung. (Dies geht auch mit Attributen.) Über die Schaltfläche *Überg. Element* kann dieser Schritt wieder rückgängig gemacht werden.



Das Ergebnis der Prüfung wird im unteren Fenster angezeigt. Hierfür muss die Schaltfläche *Überprüfen* ausgewählt werden. Das Ergebnisfenster zeigt alle getesteten Fälle an.

- *Element*: das Objekt, der Typ oder die Eigenschaft an dem oder der die Eigenschaft definiert ist
- *Eigenschaft*: die konkrete Eigenschaft die getestet werden soll (ist leer wenn kursive Eigenschaften getestet werden)
- *Operation*: die Operation, die überprüft werden soll
- *Zugriff erlaubt*: das Ergebnis der Prüfung des Testfalls
- *Entscheidungspfad*: die entsprechenden Ordner, die zu dem Testergebnis führen
- *Zeit*: die Zeit, die für die Rechteprüfung benötigt wurde

Beachte: Bei der Prüfung von Relationen werden i.d.R. die Relation, die inverse Relation und beide Relationshälften einzeln getestet.

1.6.6.2 Define Testcases

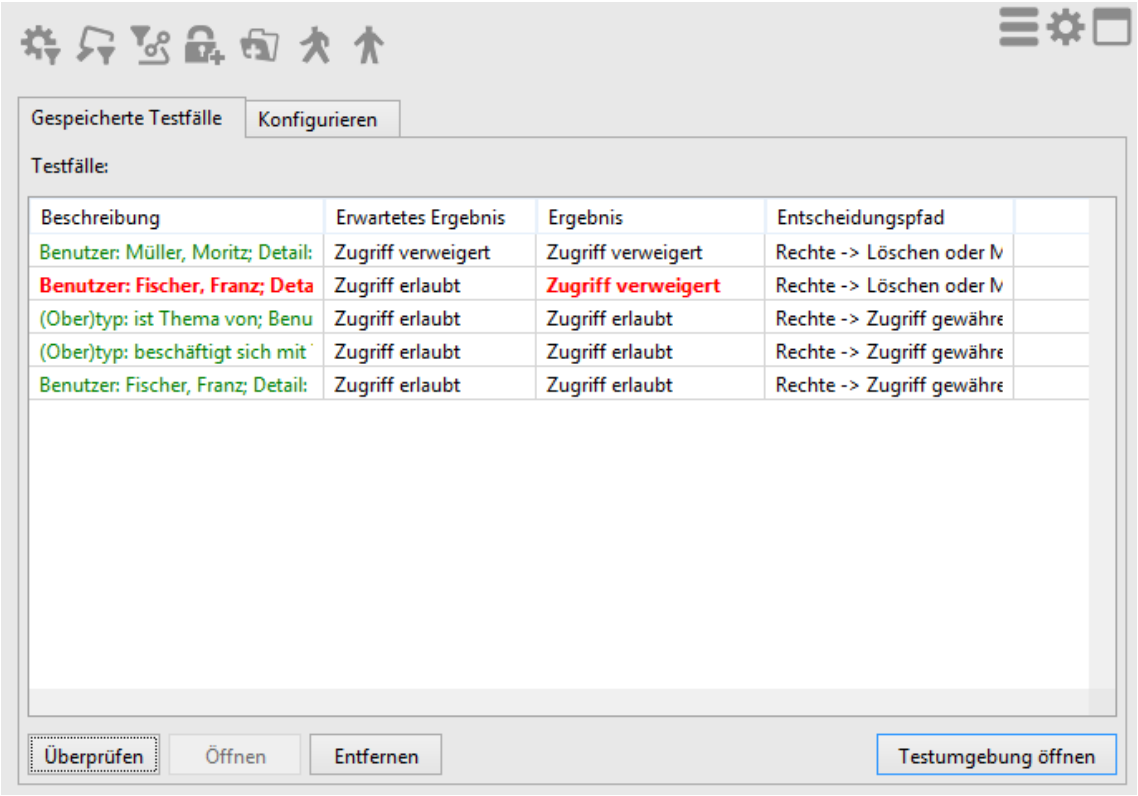
Um die Funktionalität des Rechtesystems zu überwachen, können Testfälle gespeichert werden. Dies ist gerade dann wichtig, wenn Änderungen am Rechtesystem vorgenommen werden und hinterher geprüft werden soll, ob das neue Ergebnis noch dem erwarteten Ergebnis entspricht. Alle gespeicherten Testfälle werden auf dem Reiter *Gespeicherte Testfälle* angezeigt. Dort können alle Testfälle gleichzeitig geprüft werden.

Anleitung zur Definition eines Testfalls

1. Wählen Sie in der Testumgebung das Element und die zu prüfende Eigenschaft aus.
2. Wählen Sie die Operation aus, die getestet werden soll.

3. Betätigen Sie die Schaltfläche *Überprüfen*. Jetzt werden die Zugriffsrechte für die abgegebenen Parameter getestet.
4. Wählen Sie in der Ergebnisausgabe den Testfall aus, der gespeichert werden soll. (Es kann immer nur eine Operation als Testfall gespeichert werden.)
5. Betätigen Sie die Schaltfläche Testfall. Der ausgewählte Testfall wird gespeichert und steht für spätere Prüfungen zur Verfügung.

Mehrere Testfälle gleichzeitig testen



Screenshot mit gespeicherten Testfällen, der zweite Testfall wird in Rot angezeigt.

In Grün werden alle Testfälle angezeigt, deren Testergebnis mit dem erwarteten Testergebnis übereinstimmen. Wird ein Testfall Rot angezeigt, dann ist das Ergebnis der Prüfung ein anderes als das erwartete Testergebnis. Das erwartete Testergebnis wird dadurch bestimmt, dass bei der Definition des Testfalls die Prüfung des Testfalls erstmalig durchgeführt wurde. Das Ergebnis dieser ersten Prüfung wird bei späteren Prüfungen des Testfalls als erwartetes Ergebnis angezeigt. Im Testsystem ist das erwartete Ergebnis entweder *Zugriff erlaubt* oder *Zugriff verweigert*; Bei Triggern ist das erwartete Ergebnis entweder *Skript ausführen* oder "nichts passiert" in Form eines Bindestriches.

Gespeicherte Testfälle können über *Entfernen* gelöscht werden. Soll ein Testfall bearbeitet werden, kann dies über die Schaltfläche *Testumgebung öffnen* gemacht werden. Der Testumgebung werden dann alle Parameter des Testfalls übergeben.

1.7 View configuration

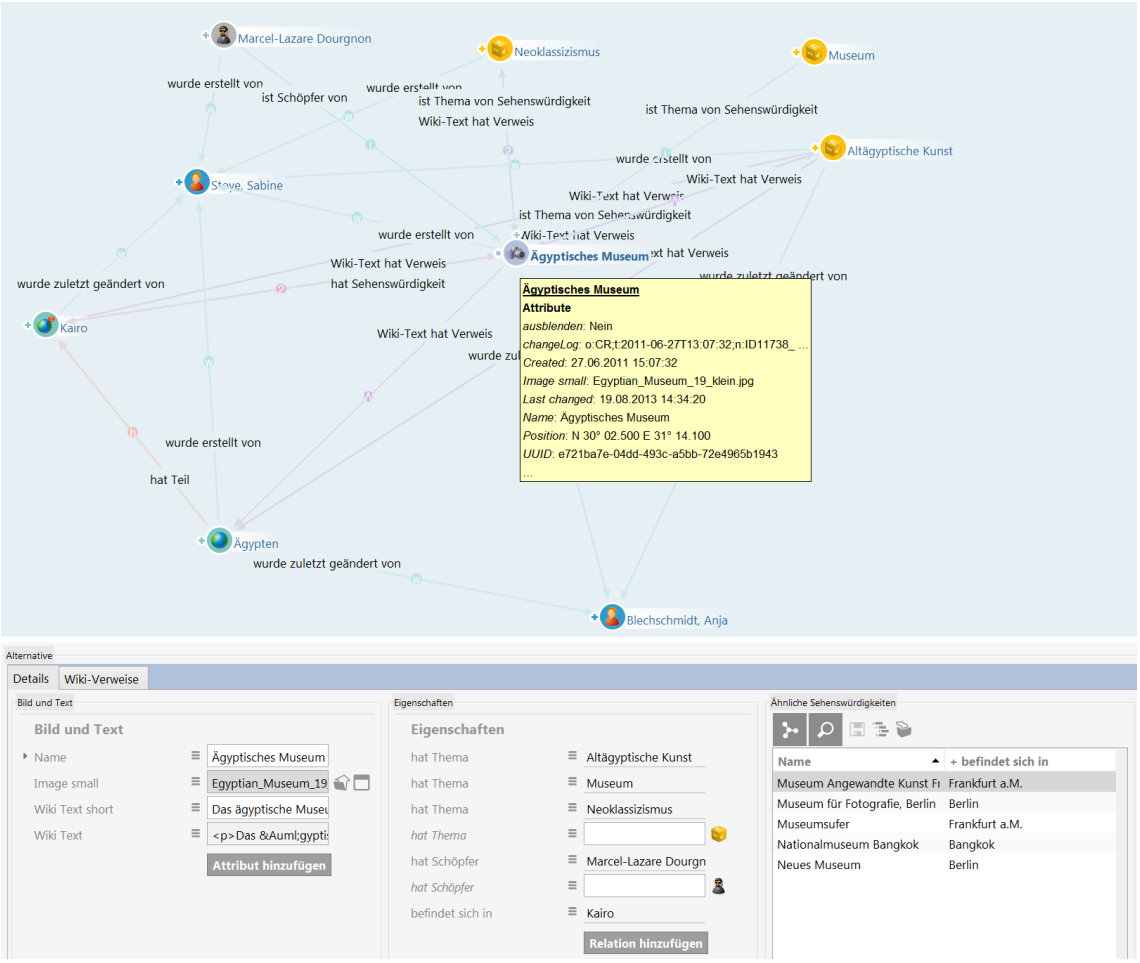
Die View-Konfiguration ermöglicht es, verschiedene Sichten auf die Daten von i-views zu konfigurieren. Die konfigurierten Sichten kommen in Anwendungen zum Einsatz. Es können beispielsweise Teilausschnitte des semantischen Modells gezeigt oder bestimmte Zusam-

menstellungen der Daten (z.B. in Formularen, Tabellen, Ergebnislisten u.v.m.) erstellt werden.

So können wir u. a. folgende Fragen entscheiden und die entsprechend gewünschten Ansichten mit View-Konfigurationen erstellen:

- Wie sollen die Eigenschaften von bestimmten Objekten dargestellt werden?
- In welcher Reihenfolge sollen die Eigenschaften dargestellt werden?
- Wenn wir ein neues Objekt anlegen, welche Attribute und Relationen sollen dann so dargestellt werden, damit sie auf keinen Fall übersehen und nicht ausgefüllt werden?
- Wie soll die Liste von Objekten zu einem Typ aussehen?
- Soll es überhaupt eine einfache Liste sein oder sollen die Objekte in Tabellen dargestellt werden?
- Welche Elemente sollen dann in den einzelnen Spalten zu sehen sein?
- Sollen Relationsziele direkt dargestellt werden? Oder nur bestimmte Attribute?
- Sollen wir verschiedene Reiter definieren, die zusammengehörige Eigenschaften und Attribute zusammenfassen? ...

Ein Beispiel: Konkrete Personen haben die Eigenschaften Name, Alter, Geschlecht, Adresse, Festnetznummer, E-Mail, Mobilnummer, Fax, *kennt*, *ist befreundet mit* und *ist Kollege von*. Nun könnten wir mithilfe der View-Konfiguration mehr Struktur in die Ansicht der Daten bringen, indem wir einen Reiter mit der Überschrift „Allgemeines“ definieren, der Name, Alter und Geschlecht zusammenfasst, einen mit der Überschrift „Kontaktdaten“, der Adresse, Festnetznummer, E-Mail, Mobilnummer und Fax beinhaltet und einen Reiter mit der Überschrift „Kontakte“, der die Eigenschaften *kennt*, *ist befreundet mit* und *ist Kollege von* enthält.



Beispiel einer View-Konfiguration. Oberer Screenshot: Unkonfigurierter Ausschnitt eines Objektes in der Graph-Ansicht mit allen seinen Eigenschaften. Unterer Screenshot: Konfigurierte Ansicht desselben Objektes, in der zusammengehörige Eigenschaften gruppiert, unwichtige Relationen weggelassen und Ähnlichkeitsbeziehungen direkt dargestellt sind.

Ein Spezialfall der View-Konfiguration ist die Konfiguration der Ansicht der Daten im Knowledge-Builder, denn auch der Knowledge-Builder ist eine Anwendung, in der verschiedene Sichten auf die Daten möglich sind. Hilfreich ist dies dann, wenn wir den Knowledge-Builder als Preview benutzen wollen, um bestimmte Konfigurationen auszuprobieren. Die View-Konfiguration im Knowledge-Builder kann so konfiguriert werden, dass wichtige zu ergänzende Eigenschaften gut sichtbar abgefragt werden, wie beispielsweise die Detailseiten von Objekten. Dies ist besonders hilfreich, wenn Daten systematisch erfasst werden sollen.

1.7.1 Concept

Das Konzept von i-views besteht darin, dass Wissensnetzelemente zur Konfiguration verwendet werden. Die Ansichten im Knowledge-Builder werden mithilfe einer voreingestellten View-Konfiguration generiert.

1.7.1.1 View-Konfiguration

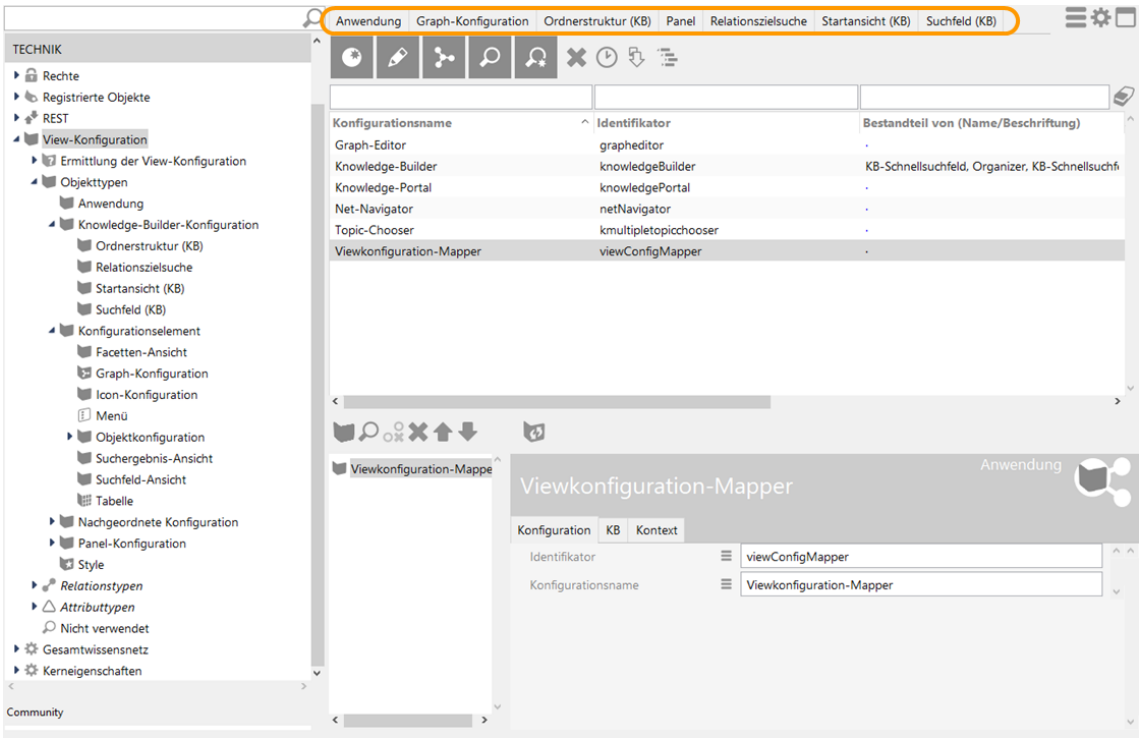
Die View-Konfiguration ist dazu vorgesehen, die Daten des Wissensnetzes für die Anwendungen so aufzubereiten, dass sie entweder im Knowledge-Builder oder mithilfe der Bridge

in Form einer Anwendung im Web-Frontend dargestellt werden können.

Im Wissensnetz lassen sich daher spezielle "View-Konfigurationen" sowohl für die Verwendung im Knowledge-Builder als auch für Anwendungen wie dem Viewconfiguration-Mapper erstellen.

Die View-Konfiguration im Knowledge-Builder enthält folgende Kategorien:

- Anwendungen
- Graph-Konfiguration
- Konfiguration der KB-Ordner-Struktur
- Panel
- Relationszielsuche
- Startansicht (KB)
- Suchfeld (KB)



Näheres hierzu ist im Kapitel "Kontext / Verwendung von View-Konfigurationen" beschrieben.

1.7.1.2 Viewkonfiguration-Mapper

Der Viewkonfiguration-Mapper dient dazu, die vorkonfigurierten Ansichten der View-Konfiguration auf das Web-Frontend des Browsers abzubilden, also zu "mappen".

Die Struktur des Viewkonfiguration-Mappers ist grundsätzlich hierarchisch aufgebaut und enthält die Panels zum Aufbau des Layouts (= Inhaltsanordnung) des Web-Frontends. Zum Anzeigen der Inhalte benötigt ein Panel eine Sub-Konfiguration, die sogenannte "View" (= aufbereiteter Inhalt).

Konkret enthält der Viewkonfiguration-Mapper ein Hauptfensterpanel und beliebig viele Dialog-Panel. Das Hauptfensterpanel spiegelt den gesamten Darstellungsbereich der Web-

seite im Web-Frontend wider und enthält beispielsweise folgende Panels:

- Fenstertitelpanel
- Panel mit festgelegter Ansicht
- Panel mit flexibler Ansicht
- Panel mit linearem Layout
- Panel mit wechselndem Layout

Zu beachten ist, dass der Viewkonfiguration-Mapper eine Single-Page-Applikation ist, d. h. es wird nicht die Sichtbarkeit von Panels über mehrere Seiten hinweg gesteuert, sondern die Sichtbarkeit der in fest vorhandenen Panels enthaltenen Elemente.

1.7.1.3 Erstellung und Aktualisierung von View-Konfigurationen

Erstellung

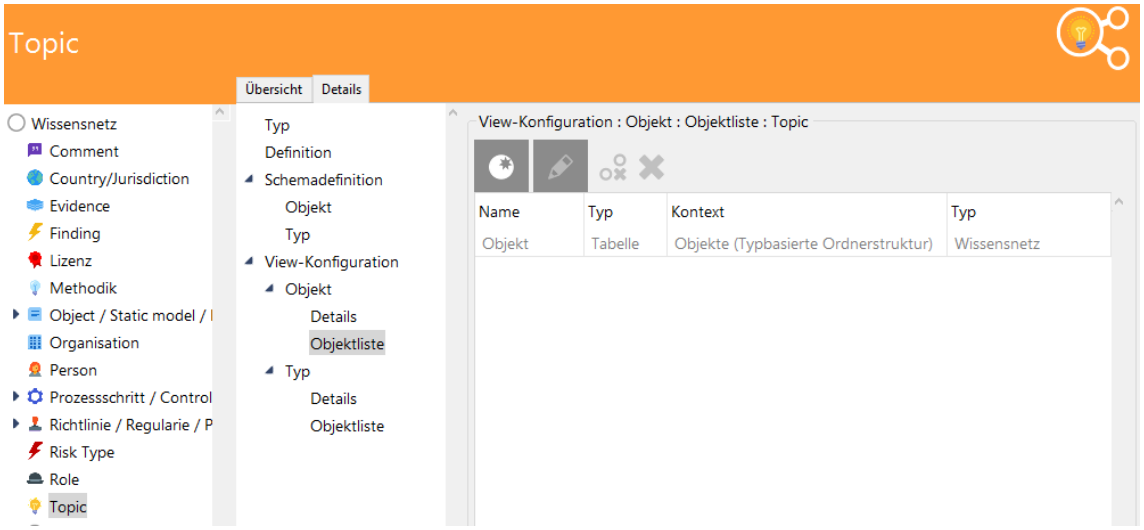
Im Knowledge-Builder gibt es zwei Stellen, an denen sich eine neue View-Konfiguration erstellen lässt:

1. Wissensnetzelement-orientierte Konfiguration

Die erste Stelle bietet sich an, wenn zu einem bestimmten Objekttyp eine View-Konfiguration erstellt werden soll: Unter dem Reiter "Details" kann die View-Konfiguration zu Detailansichten und Listen bearbeitet werden.

In der angezeigten Hierarchie gibt es Unterpunkt „View-Konfiguration“ mit vier weiteren Unterpunkten.

- Objekt -> Details: Hier kann die Detailansicht zu Objekten konfiguriert werden.
- Objekt -> Objektliste: Hier kann die Objektliste konfiguriert werden, die im Knowledge-Builder die Objekte des ausgewählten Typs anzeigt.
- Typ -> Details: Hier kann die Detailansicht zu Typen konfiguriert werden.
- Typ -> Objektliste: Hier kann die Objektliste der Untertypen des ausgewählten Typs konfiguriert werden, die im Knowledge-Builder zu sehen ist.



Am Objekttyp im Reiter „Details“ können View-Konfigurationen für diesen Typ oder Objekte diesen Typs erstellt werden.

Am Objekttyp im Reiter „Details“ können View-Konfigurationen für diesen Typ oder Objekte dieses Typs erstellt werden.

Mit einem Klick auf "Neu" können Sie eine neue View-Konfiguration anlegen. Bei Objektlisten erstellen Sie automatisch eine neue View-Konfiguration des Typs Tabelle. Bei Details, öffnet sich ein Dialog, in dem Sie das gewünschte View-Konfigurationslement auswählen können (siehe dazu Kapitel "View-Konfigurationselemente").

Mit einem Klick auf den Bearbeiten-Button oder einem Doppelklick auf die ausgewählte View-Konfiguration öffnen Sie den Editor, mit dem Sie die Ansicht konfigurieren können.

Hinweis: Unter dem Reiter "Kontext" der jeweiligen Konfiguration wird durch den Eintrag "anwenden in" festgelegt, in welcher Anwendung die Konfiguration angezeigt werden soll:

Anwendungskontext "anwenden in"	Resultat
Knowledge-Builder	Die Detailansicht oder die Liste zu einem Typ bzw. Objekt wird im Knowledge-Builder angezeigt.
Viewkonfiguration-Mapper	Die Detailansicht wird für das Web-Frontend verwendet.

Wenn kein Eintrag zum Anwendungskontext vorhanden ist und die View auch nicht anderweitig einen Anwendungskontext durch Vererbung von einem übergeordneten Element (View oder ein Panel) erhält, dann ist die View nicht zugeordnet und somit deaktiviert.

Sonderfall: Hierarchie + Objektliste

Ein möglicher Anwendungsfall für die Detailansicht des Knowledge-Builders ist das Anzeigen einer domänenspezifischen Hierarchie mit Objektdetails. In diesem Fall muss für für den Anwendungskontext in der Hierarchieansicht "Knowledge-Builder" eingetragen sein und für die Konfiguration der Details wiederum der Konfigurationsname der Hierarchieansicht. Eine anderweitige Vergabe des Anwendungskontextes in dieser Konstellation kann funktionsbedingt zu einem Endloszyklus in der Viewkonfiguration führen.

2. Ansichten-orientiert Konfiguration

Die zweite Stelle bietet sich an, wenn eine Anwendung von Grund auf zu erstellen ist und viele View-Konfigurationen am Stück erstellt werden wollen. Hierzu befinden sich unter *TECHNIK* > *View-Konfiguration* > *Objekttypen* alle View-Konfigurationselemente, die im Wissensnetz in Verwendung sind bzw. für eine View-Konfiguration neu angelegt werden können.

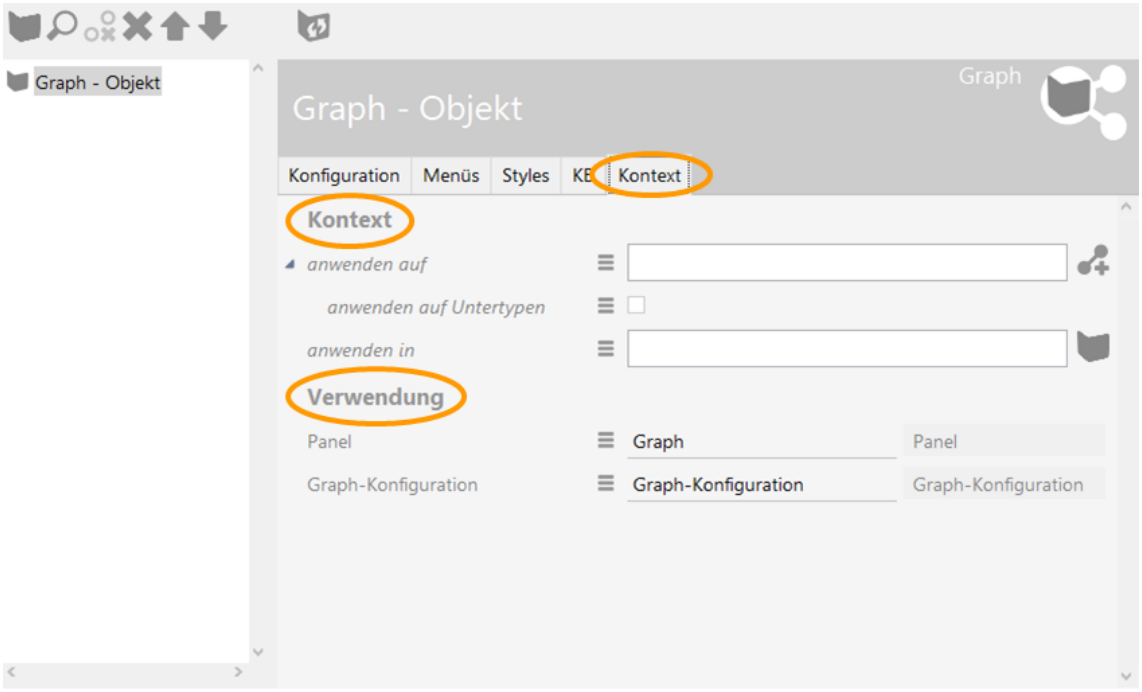
Für das Konfigurieren eines Web-Frontends ist die Panel-Konfiguration unter *TECHNIK* > *View-Konfiguration* > *Viewkonfiguration-Mapper* zu verwenden. Näheres hierzu unter Kapitel 3 "ViewConfig-Mapper".

Aktualisierung

Damit Änderungen an in der View-Konfiguration für die Anwendung übernommen werden, muss im Knowledge-Builder die View-Konfiguration durch Klick auf den Button "View-Konfiguration Aktualisieren" aktualisiert werden. Dieser Button befindet sich jeweils in der Menüleiste einer View-Konfiguration.

1.7.1.4 Kontext / Verwendung von View-Konfigurationen

In welchem Kontext ein View-Konfigurations-Element verwendet wird, wird im Eigenschaften-Editor unter dem Menü-Reiter "Kontext" angezeigt.



Kontext

Im Kontext-Bereich wird definiert für welche Wissensnetzelemente die View-Konfiguration gültig ist und wo, d.h. in welchen Anwendungen bzw. in welchen anderen View-Konfigurationen sie zur Anzeige kommt:

- "anwenden auf": Hier ist das Wissensnetzelement anzugeben, für das die View verwendet wird. Wenn die View-Konfiguration am Objekttyp definiert wird, wird der Objekttyp automatisch eingetragen. Es können nach Bedarf weitere Objekttypen angegeben werden
  Beispiel: Wenn die View eine Knoten-Kategorie des Net-Navigators ist, dann kann man bei "anwenden auf" den Objekttyp angeben, zu dem die Objekte dargestellt werden

sollen.

- "anwenden auf Untertypen": Wird gewählt, um den Typ selbst und seine Untertypen mit der Anwendung darzustellen.
- "anwenden in" spezifiziert den Anwendungskontext, d. h. in welcher Anwendung (meistens: Viewkonfiguration-Mapper oder Knowledge-Builder) oder Konfiguration die View angewendet wird.

Ist keine Anwendung als Verwendung der View-Konfiguration eingetragen, so wird die View-Konfiguration nicht angezeigt mit folgenden Ausnahmen. View-Konfigurationen werden als Baumstruktur definiert, in der das Prinzip der Vererbung gilt. Aus diesem Grund muss die Anwendung bei Unterkonfigurationen nicht extra angegeben werden. Sie werden als Teil der Oberkonfiguration mit angezeigt. Zum Beispiel wird eine Eigenschaftskonfiguration angezeigt, wenn diese Teil einer Gruppe ist, deren Verwendung angegeben wurde. Eine View-Konfiguration wird auch angezeigt, wenn sie Teil eines Panels ist, welches wiederum in einer Anwendung definiert wird.

Die folgenden Anwendungen stehen von Anfang an zur Verfügung:

- **Graph-Editor:** Die Konfigurationen haben Einfluss auf die Darstellung im Graph-Editor. Der Graph-Editor dient zur Visualisierung der semantischen Elemente und deren Zusammenhängen.
- **Knowledge-Builder:** Die View-Konfigurationen werden im Knowledge-Builder selbst angewendet. Hier stehen neben den Detailkonfigurationen auch die Objektlisten-Konfigurationen zur Verfügung.
- **Knowledge-Portal:** Das Knowledge-Portal ist eine Komponente von i-views, die als Frontend eingesetzt werden kann. Es stellt die Objekte des semantischen Netzes auf Detailseiten und in Kontextboxen basierend auf deren semantischen Kontext dar.
- **Net-Navigator:** Er dient zur Visualisierung von semantischen Elementen. Er kann im Gegensatz zum Graph-Editor der Teil des Knowledge-Builders ist, in den Anwendungen Knowledge-Portal und Viewkonfigurations-Mapper eingesetzt werden.
- **Topic-Chooser:** Er ermöglicht die Auswahl von Relationszielen in einem Fenster.
- **Viewkonfiguration-Mapper:** Der Viewkonfigurations-Mapper ist ein intelligentes Frontend, das im Gegensatz zum Knowledge-Portal die View-Konfigurationen verwendet. Mit ihm können einfach und schnell Sichten auf die Daten erstellt werden.

Darüber hinaus können auch eigene beliebige Anwendungen definiert werden, die an dieser Stelle mit der View-Konfiguration verknüpft werden können.

Verwendungen

"Verwendungen" bezieht sich auf die Wieder- und Weiterverwendung einer View-Konfiguration innerhalb einer anderen View-Konfiguration:

- "ist enthalten in Panel": Zeigt an, welche übergeordneten Panels in der Viewkonfigurations-Hierarchie vorhanden sind
- "beinhaltet Panel": Zeigt an, welche Panels in untergeordneten Hierarchiestufen vorhanden sind
- "Reihenfolge": Bestimmt die Reihenfolge des Panels, wenn das übergeordnete Panel ein lineares Layout (horizontal oder vertikal) hat

- "Sub-Konfiguration": Bezieht sich auf eine untergeordnete Konfiguration, welche die View (= konkrete Darstellung des Inhalts) enthält
- "Aktionen aktivieren aus Panel": Zeigt an, dass eine Aktion in diesem Panel durch die Aktion in einem anderen Panel beeinflusst wird (Bsp.: Anzeige des Suchergebnisses in einem Panel wird durch die Sucheingabe in einem anderen Panel beeinflusst)
- "Ergebnis anzeigen aus Aktion": Bestimmt, dass durch die Aktion eines anderen Panels in diesem Panel ein Ergebnis in bestimmter Form angezeigt wird (Bsp.: Net-Navigator zeigt die Elemente zu dem Objekt an, das im Suchergebnis-Feld eines anderen Panels angeklickt wurde)
- Weitere Relationen („Tabelle von", „Kontext von", „Konfiguration für Metaeigenschaften von", „Aktion von", ...) zeigen an in welchen Kontexten eine View-Konfiguration verwendet wird. Eine View-Konfiguration kann in beliebig vielen View-Konfigurationen verwendet werden.

1.7.1.5 Validity of View-Configurations

Im Kapitel *Die Verwendung von View-Konfigurationen* wurde bereits beschrieben, dass es für View-Konfigurationen ausschlaggebend ist, ob, in welcher Anwendung und für welche Objekte bzw. Typen die View angezeigt wird. Trotzdem ist es möglich, dass die View-Konfiguration nicht in der ausgewählten Anwendung angezeigt wird. Hier stellt sich die Frage: Wann ist eine View-Konfiguration gültig? Und für welche Objekt bzw. Typen ist die View-Konfiguration gültig?

Vererbung von View-Konfigurationen

View-Konfigurationen verhalten sich in Bezug auf die Vererbung wie Eigenschaften. View-Konfigurationen werden auf die Untertypen bzw. die Objekte der Untertypen vererbt.

Anwendung der konkretesten View-Konfiguration

Die Untertypen verwenden nach dem Prinzip der Vererbung die View-Konfiguration der Obertypen solange sie keine eigenen View-Konfigurationen besitzen. Es wird immer die konkreteste View-Konfiguration angewendet: Das ist die Konfiguration, die direkt am Typ definiert ist. Ist das nicht der Fall, so wird geprüft ob es am Obertyp eine View-Konfiguration gibt. Ist das ebenfalls nicht der Fall so wird in der Typenhierarchie jeweils eine Ebene nach oben gegangen und geprüft ob eine View-Konfiguration definiert ist. Es wird dann diejenige View-Konfiguration angewendet, die dem Objekttyp am nächsten steht. Wird keine View-Konfiguration an den Obertypen gefunden, wird für Administratoren die Default-Konfiguration verwendet.

Was passiert wenn zwei gleichwertige View-Konfigurationen existieren?

Gibt es zwei gleichwertige View-Konfigurationen, so wird keine View-Konfiguration angezeigt. Wurde bei einer View-Konfiguration die Anwendung oder der Objekttyp nicht definiert, zählt diese nicht zu den aktiven View-Konfigurationen. In diesem Fall wird die andere View-Konfiguration verwendet. Möchte man für unterschiedliche Benutzer jeweils andere Views anzeigen, kann im Detektorsystem eine Regel definiert werden. In diesem Fall wird dann die View-Konfiguration entsprechend der definierten Regel angewendet, solange die Regel abhängig von Nutzer nur eine gültige View-Konfiguration liefert.

1.7.2 Menus

Menü-Konfigurationen beinhalten Schaltflächen, sog. *Aktionen*, über welche der Benutzer unterschiedlichste Funktionen ausführen kann.

Die Menüs bedienen hauptsächlich zwei Funktionalitäten beim Umgang mit Aktionen. Zum Einen können mit ihnen Aktionen gegliedert werden, zum Anderen kann festgelegt werden, wo die Menüs zum Einsatz kommen. Im Knowledge-Builder und ViewConfigMapper gibt es viele Orte, an denen die Inhalte von Menüs angezeigt werden, beispielsweise Knöpfe am Kopf eines Editors oder das Kontextmenü an einer einzelnen Eigenschaften. Derzeit lassen sich noch nicht an alle Stellen, an denen Menüs theoretisch möglich sind, Menüs anbringen.

Im Folgenden werden die direkten Einstellungsmöglichkeiten an einem Menü und die bereits existierenden Menüarten und deren Verwendung beschrieben.

Name	Wert
Beschriftung	Ob die Beschriftung angezeigt wird, richtet sich nach der Menüart und dem Interface, das sich um die Anzeige kümmert.
Ersetzt Standardmenü	Dieser Parameter hat bisher nur Auswirkungen auf den Knowledge-Builder. Bei einigen Editoren, wie z.B. für eine Tabelle, werden Standardmenüs angezeigt. Mit Hilfe dieses Parameters können diese ausgeschaltet werden.
Menüart	Die Menüart beschreibt die Verwendung des Menüs in den einzelnen Komponenten. Die Menüarten werden weiter unten beschrieben.

Menüarten:

Menüleiste

Name	Wert
Standardaktionen hinzufügen	Dieses Icon wird nur angezeigt, wenn Standardaktionen hinzugefügt werden können. Dies ist aktuell bei einer Tabellen- und Suche-Konfiguration möglich. Diese Funktion bietet die Möglichkeit, beim gesetzten Parameter *Ersetzt Standardmenü* wieder einige oder alle Standardmenüeinträge zu reaktivieren und die Reihenfolge der einzelnen Aktionen zu ändern.

Anmerkungen

- Ist der Parameter *Ersetzt Standardmenü* nicht gesetzt, so werden die Aktionen, die in den Menüs enthalten sind, der Reihe nach hinten angefügt.
- Soll die Reihenfolge der Standardaktionen geändert werden, so muss der Parameter *Ersetzt Standardmenü* gesetzt sein. Anschließend können die Standardaktionen mit der Aktion *Standardaktionen hinzufügen* ergänzt werden. Die Standardaktionen können nun

beliebig sortiert und mit eigenen Aktionen gemischt werden.

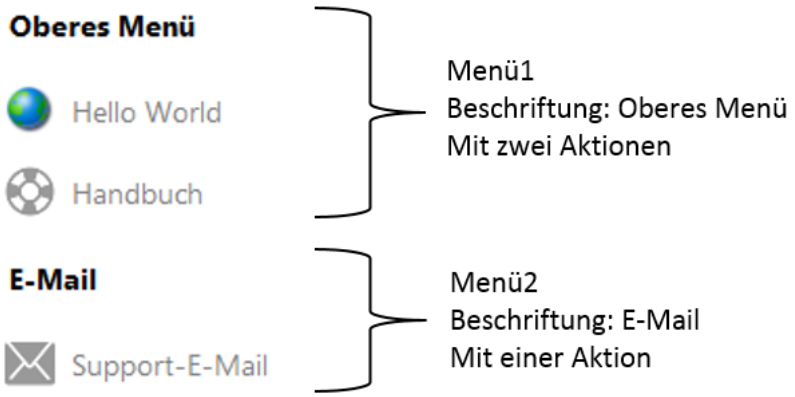
Kontextmenü

Icon	
Knowledge-Builder	Derzeit lassen sich Kontextmenüs für eine Tabellenzeile und einen Objekteditor erweitern oder neu definieren. **Objektkonfiguration**: In einer beliebigen Top-Konfiguration eines Elementes können unter dem Reiter *Menü* Menüs angelegt werden. Auch hier kann das Standardmenü durch das Setzen des Parameters *Ersetzt Standardmenü* ausgeschaltet werden. **Tabellen-Kofiguration**: Im Kontextmenü für eine Tabellenzeile gibt es zwei Abschnitte. Der erste bezieht sich auf das ausgewählte Element, der zweite bezieht sich auf die Tabelle. Für die beiden Abschnitte gibt es zwei unterschiedliche Konfigurationsorte. Für den ersten Fall muss das Menü für ein Element mit einer beliebigen, am besten neuen Konfiguration verknüpft werden, die wiederum über *anwenden in* an die Tabelle, die das Kontestmenü anzeigen soll, gehängt wird. Im zweiten Fall kann das Menü direkt an der Tabelle angebracht werden.
ViewConfigMapper	*Findet derzeit keine Verwendung im ViewConfigMapper.*

JSON	`"label" : "Menü (Kontext)",`
	`"actions" : [{...}],`
	`"type" : "contextMenu"`

Liste

Icon	

Knowledge-Builder	Findet nur Anwendung in der Startansicht-Konfiguration. Es werden die konfigurierten Aktionen in einer Liste dargestellt. Werden für die Menüs Beschriftungen vergeben, werden diese mit angezeigt und bieten somit eine Strukturierungsmöglichkeit. 
ViewConfigMapper	*Findet derzeit keine Verwendung im ViewConfigMapper.*

JSON	`"label" : "Menü (Liste)",`
	`"actions" : [{...}],`
	`"type" : "listMenu"`

Werkzeugliste

Icon	
Knowledge-Builder	Die Aktionen, die in den Menüs enthalten sind, werden der Reihe nach angefügt. Eine Unterteilung nach Menüs und eine Beschriftung der Menüs werden derzeit nicht berücksichtigt.
ViewConfigMapper	Die Aktionen, die in den Menüs enthalten sind, werden der Reihe nach angefügt. Eine Unterteilung nach Menüs und eine Beschriftung der Menüs werden derzeit nicht berücksichtigt.

JSON	`"label" : "Menü (Werkzeugleiste)",`
	`"actions" : [{...}],`
	`"type" : "toolbar"`

1.7.3 Actions

Die Aktionen von i-views sind in vorkonfigurierte Aktionsarten unterteilt. Diese Aktionsarten sind wie folgt kategorisiert:

- Universelle Aktionen (anwendbar in Knowledge und Viewconfiguration-Mapper)
- Knowledge-Builder spezifische Aktionen
- Viewconfiguration-Mapper spezifische Aktionen
- Interne Aktionen (nur für administrativen Gebrauch)

Je nach Aktionsart und Anwendungsfall sind zusätzliche Konfigurationen erforderlich, wie beispielsweise das Anlegen zusätzlicher Panels zur Anzeige der Ergebnisse einer Aktion.

1.7.3.1 General

Mit Hilfe von Aktionen lassen sich Funktionalitäten in der View-Konfiguration spezifizieren.

Im Knowledge-Builder werden die vollständig konfigurierten Aktionen als zusätzliche Schaltflächen angezeigt. Bei einer Selektion wird das enthaltene Skript ausgeführt.

Im Web-Frontend (Viewkonfiguration-Mapper) werden die konfigurierten Aktionen im Allgemeinen als Schaltflächen dargestellt. Aktionen können in einem Menü zusammengefasst oder direkt für eine View-Konfiguration definiert werden.

Aktion an einer Objektliste

Die Beschriftung wird als Tooltip im Knowledge-Builder angezeigt. Das ausgewählte Symbol (eine beliebige Bilddatei) wird auf Buttongröße skaliert.
Achtung: Ist kein Symbol angegeben, wird im Knowledge-Builder kein Button angezeigt.

In einer anderen Applikation sind Schaltflächen mit einer Beschriftung und/oder einer Symbolgrafik möglich. Zusätzlich kann ein Tooltip konfiguriert werden.

Wichtige Anmerkung: Aktionen jeglicher Art lassen sich an verschiedensten Stellen anbringen. In den meisten Fällen werden diese auch angezeigt. Ob diese Aktion, in dem Kontext in dem sie gerade eingesetzt wird, ausführbar ist, ist nicht immer gegeben.

Einstellungsmöglichkeiten

Name	Wert
Aktionsart	Die Art der Aktion. Die verschiedenen Arten werden weiter unten erklärt. Ein Skript überschreibt die durch die Aktionsart festgelegte Aktion.
ausführen in View (VCM-spezifisch)	View, in der die Aktion ausgeführt werden soll.

Benachrichtigung	Text der in einer Benachrichtigung angezeigt wird, die nach der Aktion eingeblendet wird.
Beschriftung	Hier lässt sich eine Beschriftung für die Schaltfläche der Aktion festlegen.
Ergebnis anzeigen in Panel (VCM-spezifisch)	Ein Panel in dem das Ergebnis der Aktion angezeigt werden soll.
Skript für Zielobjekt (VCM-spezifisch)	.
Frage vor der Ausführung	Hier lässt sich ein Text angeben, der dem Nutzer vor dem Ausführen der Aktion in einem Dialogfenster angezeigt werden soll. Der Dialog bietet die Möglichkeit die Aktion abzubrechen oder fort zu setzen.
Kontext von	.
Nachricht	.
Panel schließen (VCM-spezifisch)	Legt fest ob das Panel nach der Aktion geschlossen werden soll.
Skript	Das Skript das bei dieser Aktion ausgeführt werden soll. Nicht bei allen Aktionsarten verfügbar.
Skript (ActionResponse) (VCM-spezifisch)	Ein hier angegebenes Skript führt eine sog. *ActionResponse* nach der Aktion aus. Nicht bei allen Aktionsarten verfügbar.
Skript (enabled)	Hier kann über ein Skript ermittelt werden, ob die Schaltfläche der Aktion aktiviert und damit ausführbar sein soll.
Skript (visible)	Hier kann über ein Skript ermittelt werden, ob die Schaltfläche der Aktion angezeigt werden soll.
Skript für Benachrichtigung	Der Inhalt der Benachrichtigung kann hier über ein Skript ermittelt werden.
Skript für Beschriftung	Die Beschriftung kann hier über ein Skript festgelegt werden.
Skript für Frage vor der Ausführung	Der Text des Bestätigungs-Dialogs für die Aktion kann hier über ein Skript ermittelt werden. Wird eine leere Zeichenkette zurückgegeben, erscheint der Dialog nicht.
Skript für Nachricht	.
Skript für Tooltip	Hier über ein Skript der Inhalt des Tooltips der Aktion bestimmt werden, anstatt den Text der Beschriftung zu verwenden.
Start-Wissensnetzelement von	.
Symbol	Hier lässt sich ein Symbol auswählen, daß auf der Schaltfläche der Aktion angezeigt werden soll.

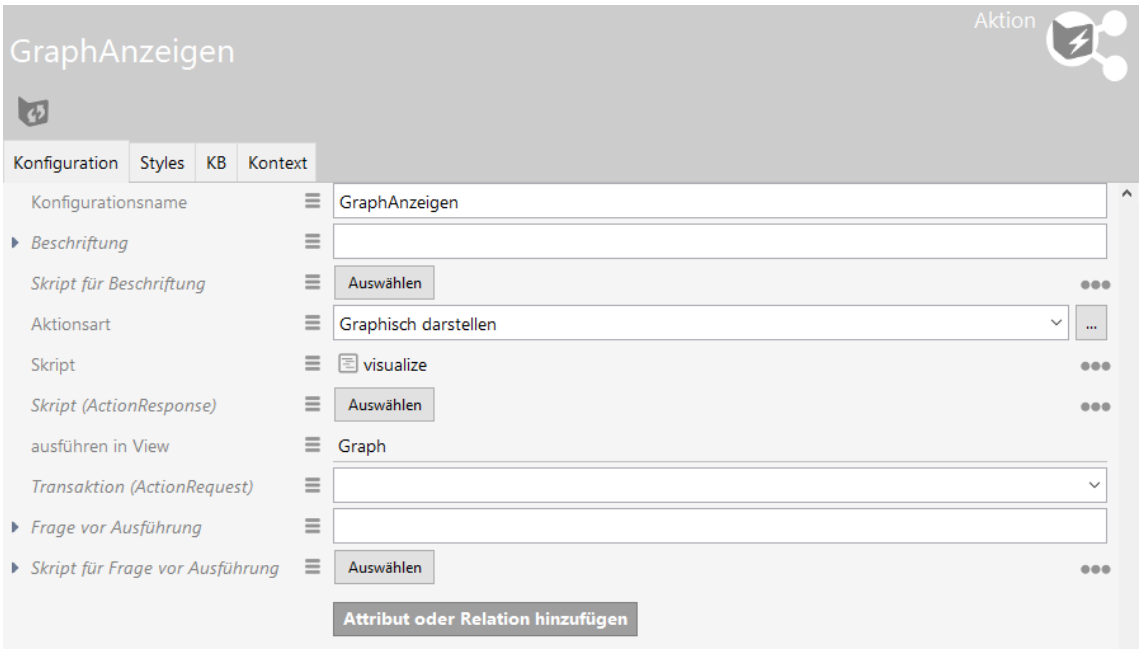
Tooltip	Hier kann der Inhalt des Tooltips der Aktion festgelegt werden, anstatt den Text der Beschriftung zu verwenden.
Ursprüngliche Position verwenden	.

1.7.3.2 Universell anwendbare Aktionen

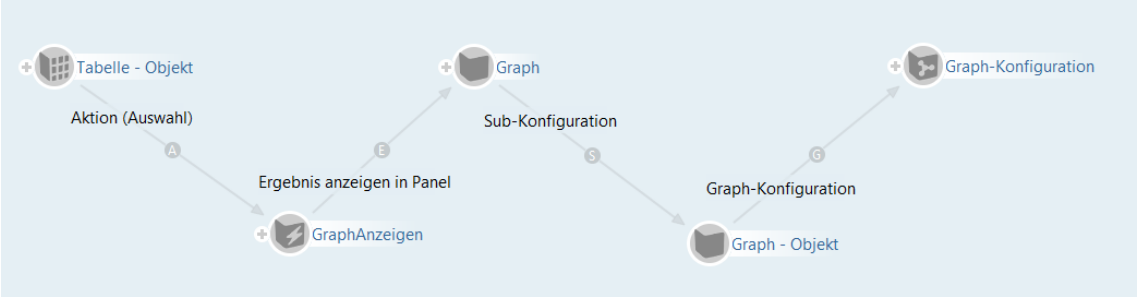
Universell anwendbare Aktionen können sowohl im Knowledge-Builder als auch per Viewconfiguration-Mapper im Web-Frontend angewendet werden. Hierzu zählen die Aktionsarten "Graphisch darstellen", "Löschen" und "Suchen".

1.7.3.2.1 Aktionsart "Graphisch darstellen"

Die Aktion "Graphisch darstellen" wird in einer View-Konfiguration dazu verwendet, um Objekttypen, Relationen und Objekte graphisch im Net-Navigator darzustellen. Die Konfiguration sieht dabei folgendermaßen aus:

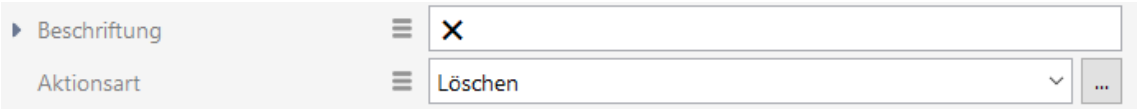


Hierfür muss unter "Ergebnis anzeigen in Panel" ein Panel angegeben werden, das als Sub-Konfiguration ein Graph-Objekt enthält. Das Graph-Objekt wiederum muss für die Definition der darzustellenden Elemente eine Graph-Konfiguration enthalten:

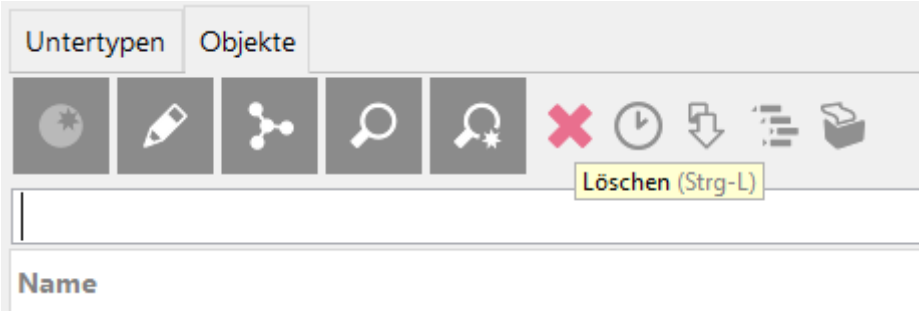


1.7.3.2.2 Aktionsart "Löschen"

Diese Aktionsart löscht das jeweilige Element.



Im Knowledge-Builder ist die Aktionsart "Löschen" für Objektlisten vorkonfiguriert:

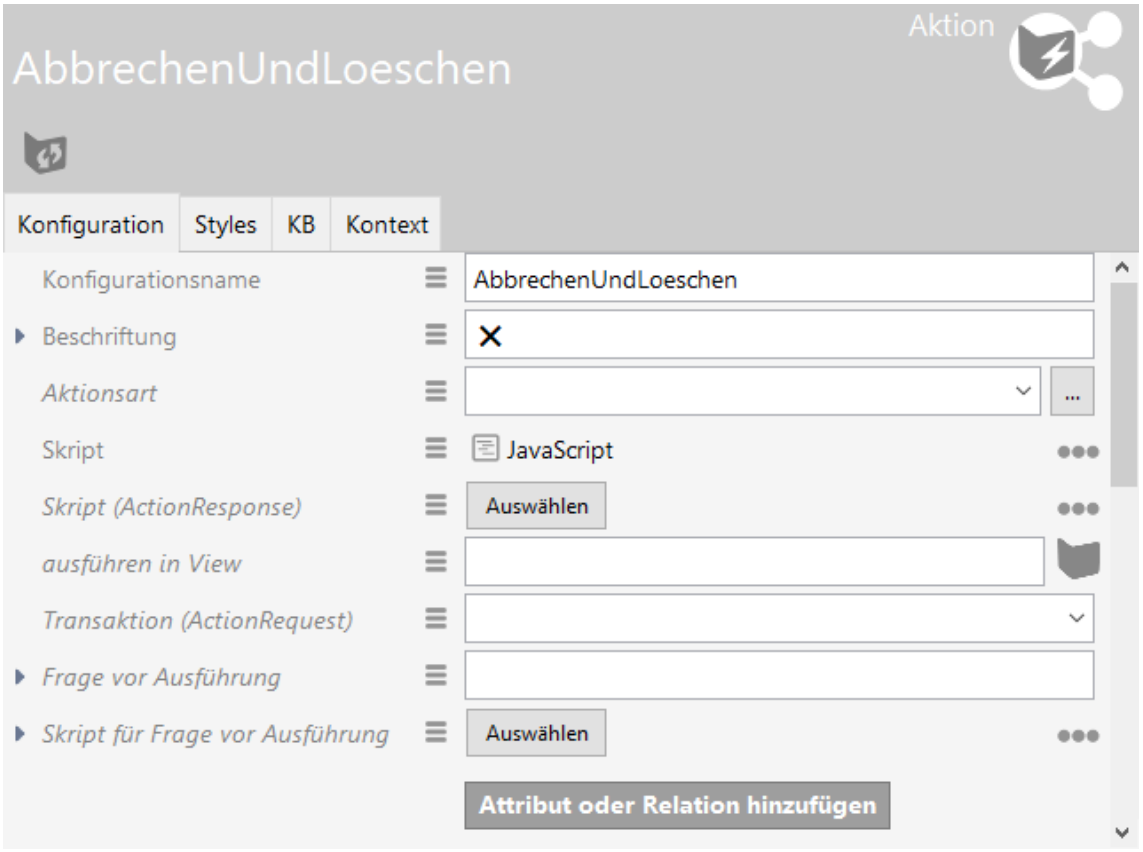


Wie jede Konfiguration im Knowledge-Builder kann auch die Standard-Konfiguration durch eine individualisierte Konfiguration ersetzt werden. In diesem Fall findet die Aktionsart "Löschen" ihre Anwendung.

Löschen durch Aktion mit Skript

Im Web-Frontend ist die Aktionsart "Löschen" unpraktikabel, weil danach das gelöschte Element angezeigt wird - also nichts mehr zu sehen ist. Löschen wird daher im Web-Frontend fast immer durch eine Aktion mit Skript realisiert.

Das Skript zum Löschen eines Elements wird unter dem Eintrag "Skript" der Aktion angelegt:



Die Syntax hierzu sieht bspw. folgendermaßen aus:

```
function onAction(element, context) {
    element.remove();
    return;
}
```

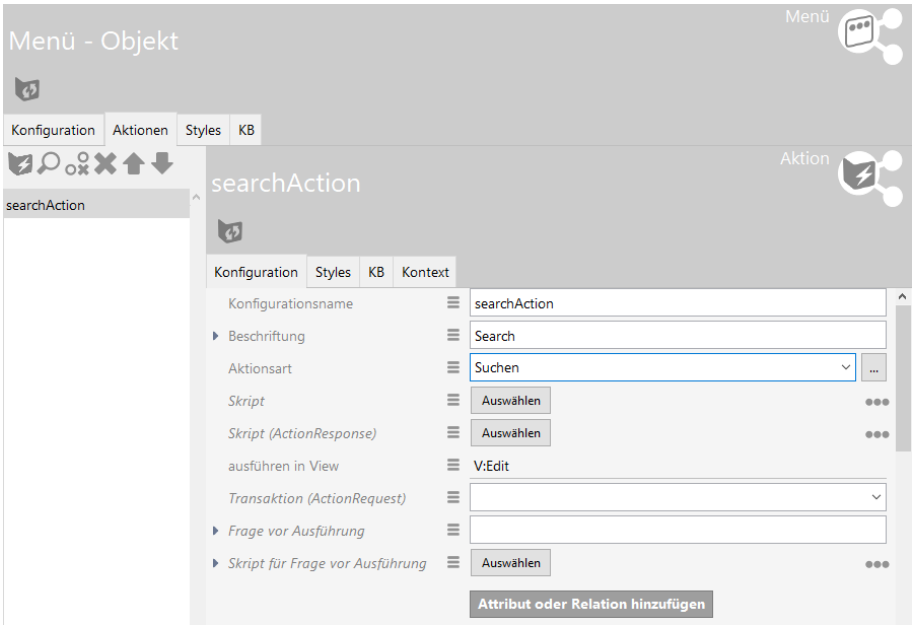
Eine Anwendungsmöglichkeit hierfür ist das Konfigurieren eines Dialog-Panels zum Anlegen neuer Objekte, dessen Abbrechen-Button das temporär erzeugte Objekt wieder löscht.

Zu beachten ist, dass mit diesem Skript keine Aktionsart für die Aktion ausgewählt werden muss, da das Skript die Aktionsart überschreibt.

1.7.3.2.3 Aktionsart "Suchen"

Diese Aktion löst die Suche aus. Im KB ist diese Funktion als Button in der Menüleiste von Objektlisten integriert (Shortcut Strg + S):

Bei Verwendung für die Konfiguration des Web-Frontends wird die Aktion mittels Dropdown-Auswahl unter dem Eintrag "Aktionsart" einer Aktion zugewiesen:



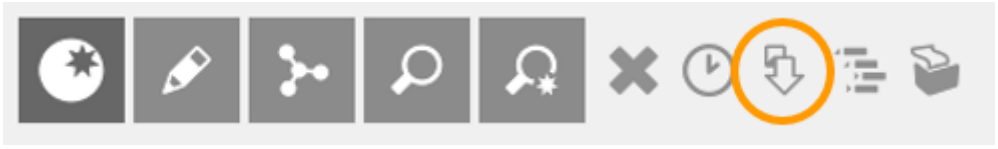
Tipp: Wird eine Such-Funktion mit Zeichenketten-Eingabe (Stichwortsuche) benötigt, so kann alternativ hierzu die Suchfeld-Ansicht in der Viewkonfiguration verwendet werden. Hier sind Eingabezeile und Suche-Button bereits vorkonfiguriert; das Suchergebnis kann in Kombination mit der Suchergebnis-Ansicht angezeigt werden.

1.7.3.3 Aktionen für den Knowledge-Builder

Diese Aktionsarten können ausschließlich für Konfigurationen im Knowledge-Builder verwendet werden.
Hinweis: Ab KB-Version 5.2.2 sind die KB-spezifischen Aktionsarten nur im Reiter "KB" einer Aktion verfügbar.

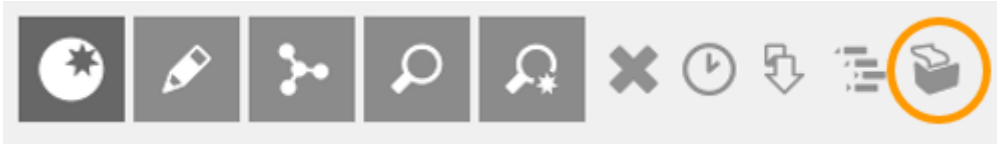
1.7.3.3.1 Aktionsart "Aktualisieren"

Im KB wird mit dieser Aktion der sichtbare Inhalt von Tabellenzellen neu berechnet. Verfügbar ist diese Option in der Menüleiste von Objektlisten unter dem Button "Aktualisieren" (Shortcut F5).

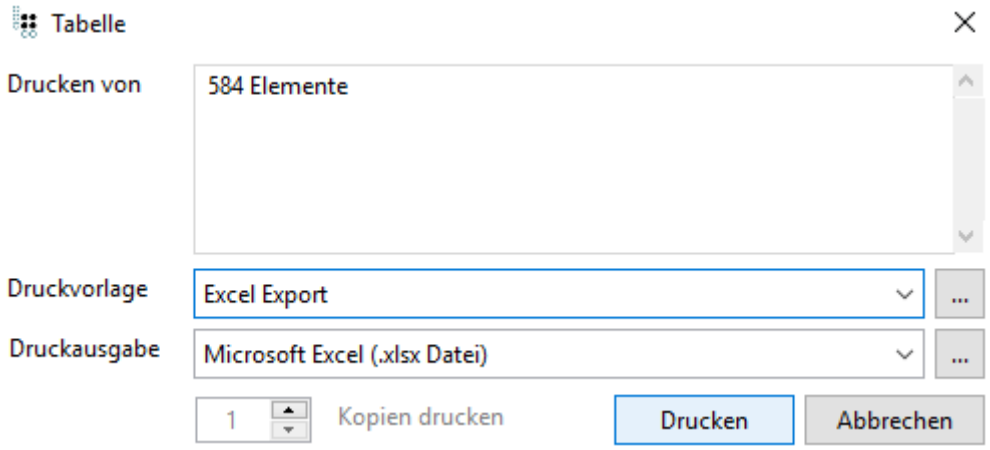


1.7.3.3.2 Aktionsart "Drucken"

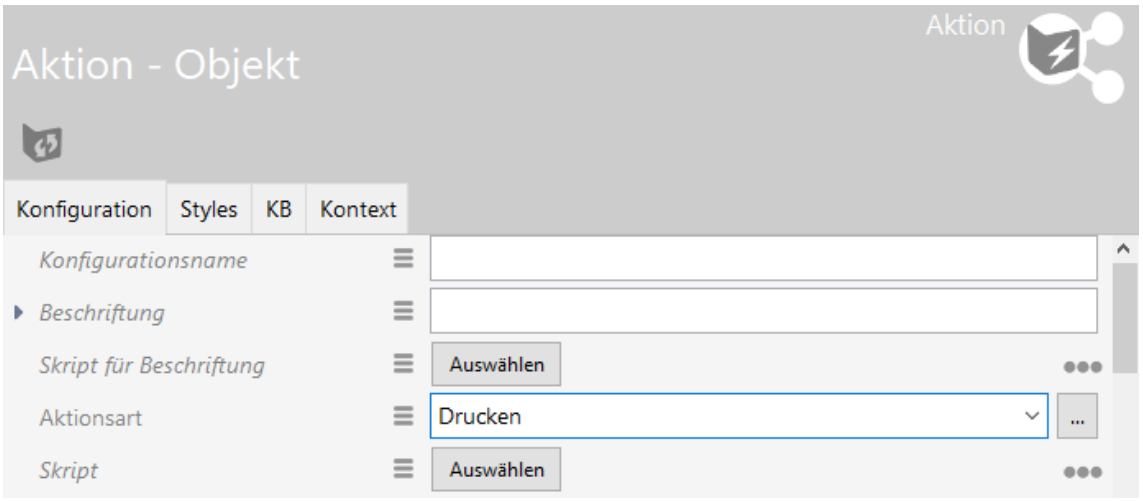
Diese Aktion findet in der Menüleiste von Listenansichten Verwendung. Mit der voreingestellten Konfiguration können Objektlisten ausgedruckt oder in Form einer Excel-Tabelle ausgegeben werden, ohne dass dafür ein Export-Mapping angelegt werden muss.



Die Aktion "Drucken" öffnet den Drucken-Dialog im Knowledge-Builder:



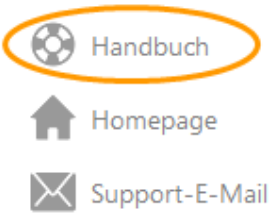
Die Drucken-Aktion ist des Weiteren in den Ergebnislisten von Strukturabfragen verfügbar. Für die Konfiguration individueller Ansichten im Knowledge-Builder ist die Aktion für die jeweilige View oder das Konfigurationselement hinzuzufügen:



Voraussetzung für die Anwendbarkeit der Aktionsart "Drucken" ist das Vorhandensein der Drucken-Komponente, welche bei Bedarf mithilfe des Admin-Tools nachinstalliert werden kann.

1.7.3.3.3 Aktionsart "Handbuch"

Bei dieser Aktion wird das i-views Web-Handbuch im Browser geöffnet.



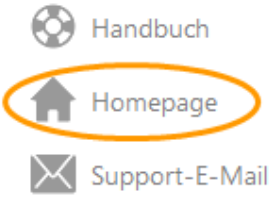
Im Gegensatz zur Aktionsart "Web-Link" verweist wie die Aktionsart "Handbuch" auf eine vorkonfigurierte Adresse.

Einstellungsmöglichkeiten

Name	Wert
URL	Voreingestellter Weblink zum i-views Handbuch.

1.7.3.3.4 Aktionsart "Homepage"

Diese Aktionsart ist für die Startansicht des KB verwendbar. Die Homepage wird im Browser geöffnet.



Einstellungsmöglichkeiten

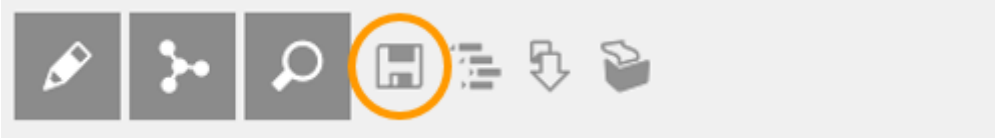
Name	Wert
URL	Link zu einer Webseite

1.7.3.3.5 Aktionsart "Im Baum anzeigen"

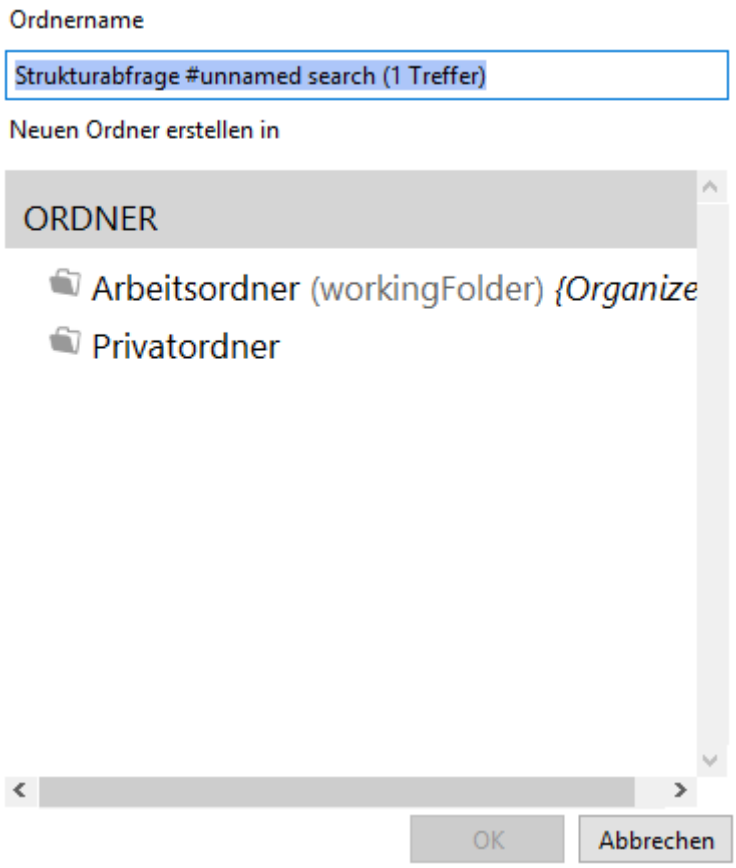
Mithilfe der Im-Baum-Anzeigen-Aktion kann die Verortung eines Elementes aus dem Semantischen Netz angezeigt werden. Das Ausführen der Aktion führt dazu, dass zum gewählten Element (bspw. Eintrag einer Listenansicht) die entsprechende Stelle im Strukturbaum des Organizers (linke Spalte des KB) markiert wird und sich die Detailansicht des Elements öffnet.

1.7.3.3.6 Aktionsart "Suchergebnis-Speichern"

Werden im Knowledge-Builder Suchen mittels einer Strukturabfrage ausgeführt, so können die Ergebnisse per Klick auf den Button in der Menüleiste abgespeichert werden:



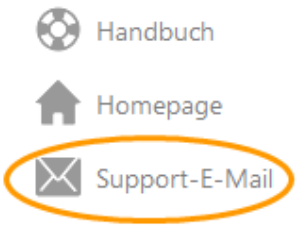
Diese Aktion speichert das Suchergebnis in einem wählbaren Ordner:

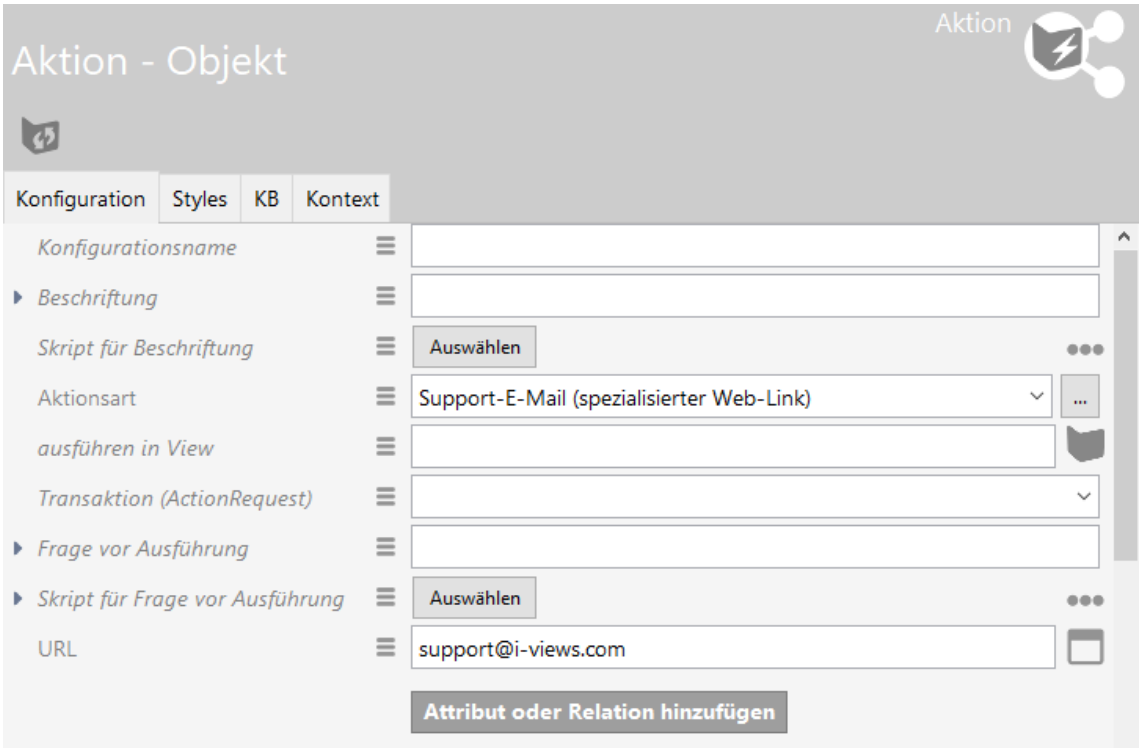


Hinweis: Die abgespeicherte Suche ist eine Objektliste, welche auf der Konfiguration einer Strukturabfrage zu aktuell vorhandenen Wissensnetzelementen basiert. Werden nach der Speicherung des Suchergebnisses Veränderungen an den entsprechenden Elementen vorgenommen, so wirkt sich dies auch auf die abgespeicherten Ergebnisse aus: Bei einer Löschung des jeweiligen Elementes ist dieses auch im abgespeicherten Suchergebnis nicht mehr vorhanden.

1.7.3.3.7 Aktionsart "Support-Email"

Diese Aktionsart ist für die Startansicht des KB verwendbar. Die daring enthaltene Aktion öffnet einen Dialog, in dem man eine E-mail an die konfigurierte Adresse senden kann.



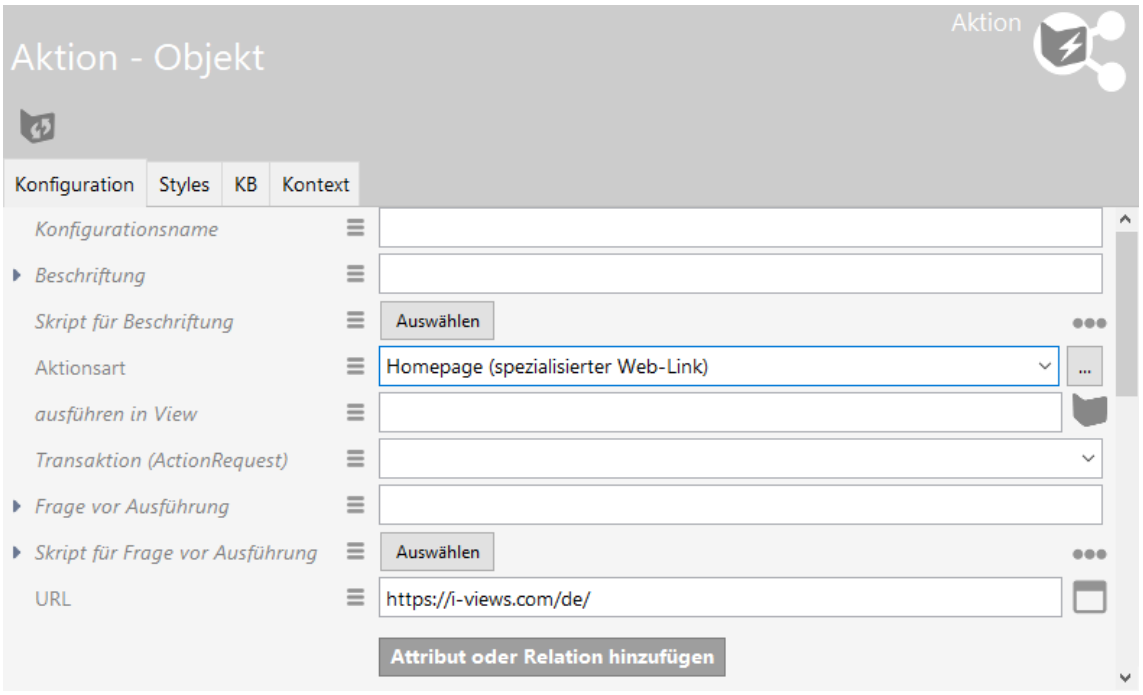


Einstellungsmöglichkeiten

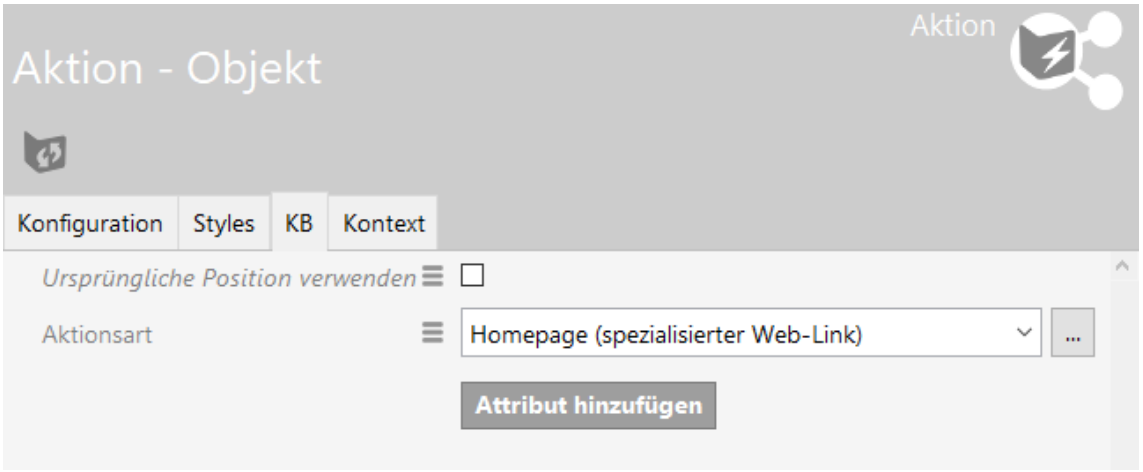
Name	Wert
URL	E-mail Link

1.7.3.3.8 Aktionsart "Web-Link"

Die Aktionsart "Web-Link" ist für die Startansicht des KB verwendbar. Der Unterschied zur Aktionsart "Handbuch" besteht darin, dass eine beliebige Web-Adresse als Hyperlink vergeben werden kann.



Hinweis: In späteren KB-Versionen (KB 5.2.2) ist die Aktionsart "Homepage" nur im Reiter "KB" verfügbar.



Einstellungsmöglichkeiten

Name	Wert
URL	Adresse des Weblinks.

1.7.3.3.9 Aktionsart "Zuletzt verwendete Objekte"

Zeigt die zuletzt verwendeten Objekt (Wissensnetzelemente) in der jeweiligen Tabelle an. Je nach Definition der Tabelle werden die Objekte ggf. gefiltert.

Diese Aktion ist im KB für Listenansichten vorkonfiguriert und kann mittels Tastenkürzel Strg-R aufgerufen werden.

1.7.3.3.10 Aktionsart "Neu"

Die Neu-Aktion legt neue Typen oder neue Objekte des Wissensnetzes an. Im Knowledge-Builder findet die Neu-Aktion bspw. in der Menüleiste von Objektlisten Anwendung.

Hinweis: Im Web-Frontend muss anstatt der Neu-Aktion ein Skript angewendet werden. Siehe hierzu das Kapitel "JavaScript-API".

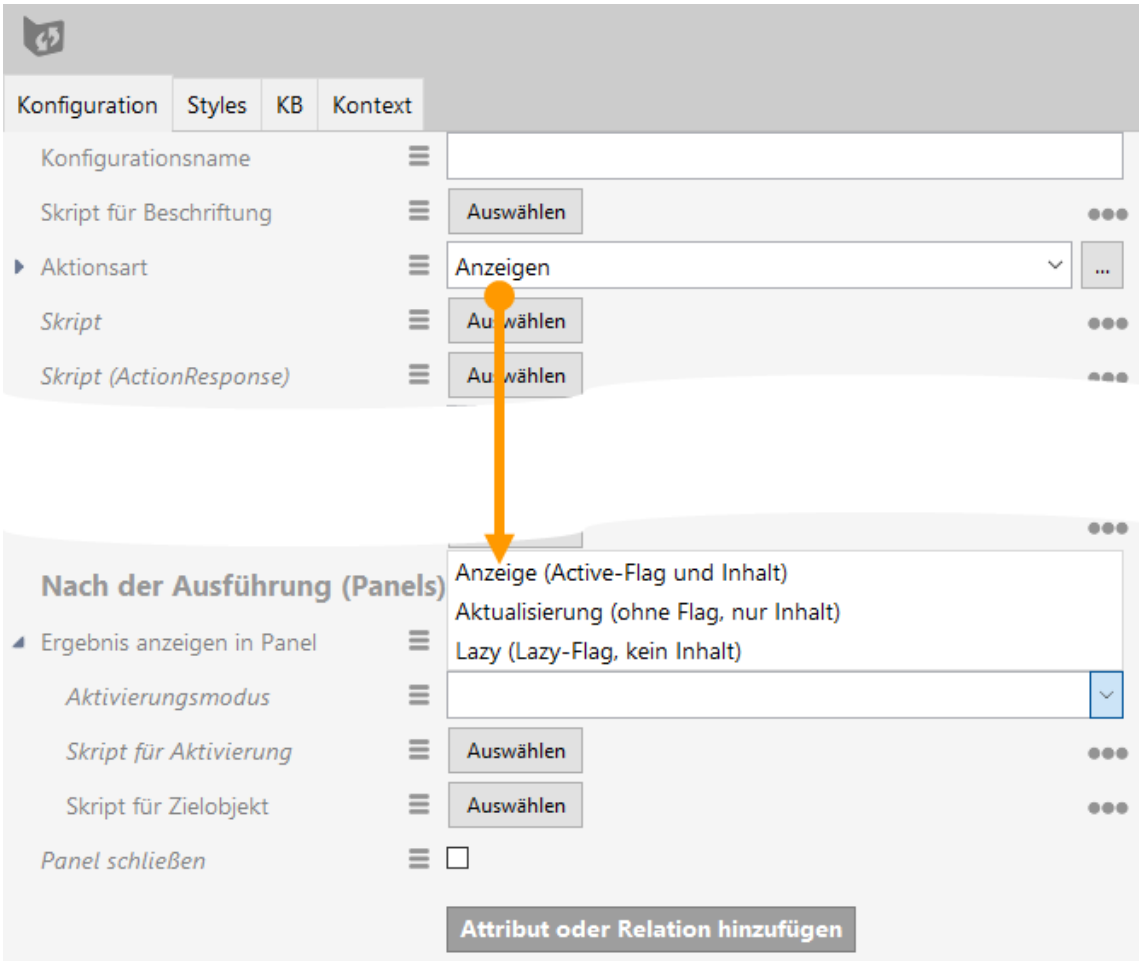
1.7.3.4 Aktionen für den Viewconfiguration-Mapper

Die Aktionen für den Viewconfiguration-Mapper können nur für das Web-Frontend verwendet werden und sie sind in unterschiedliche Aktionsarten eingeteilt.

1.7.3.4.1 Aktionsart "Anzeigen"

Diese Aktion initiiert eine Neu-Berechnung einer geeigneten View für das semantische Objekt, welches Ziel der Aktion ist. Typischerweise verwendet man diese Aktion, wenn man einen Wechsel der Ansicht bewirken möchte. Ergebnis der Aktion ist die neue View.

Mit "Ergebnis anzeigen in Panel" kann bestimmt werden, in welchem Panel die View angezeigt werden soll.



Der "**Aktivierungsmodus**" bestimmt das Aktualisierungsverhalten der Ansicht:

Anzeige (Active-Flag und Inhalt) = "Push-Verfahren"	Die View wird bei Beeinflussung durch die Anzeigen-Aktion oder bei jeglicher Änderung des Inhaltes neu berechnet, unabhängig davon ob das Panel aktiviert (= sichtbar) ist oder nicht. Dieser Modus ist beispielweise sinnvoll beim Aufrufen eines Dialog-Panels.

Aktualisierung (ohne Flag, nur Inhalt) = "Delta-Load"	Die View wird nur dann neu berechnet, wenn sich der Inhalt ändert: • Wenn das Panel erstmalig aufgerufen wird, dann wird die View neu berechnet. • Wenn eine View vorhanden ist, dann wird sie auch bei zwischenzeitlichem Aufrufen eines anderen Panels beibehalten. Die View wird für die Dauer der Sitzung solange "abgespeichert", bis sie durch eine Änderung des Inhalts neu berechnet werden muss. Ein Beispiel hierfür ist die Anzeige eines Graphen: Solange keine Änderungen am Graph vorgenommen werden, wird bei jedem Aufrufen des Graph-Panels der gleiche, unveränderte Graph angezeigt.
Lazy (Lazy-Flag, kein Inhalt) = "Pull-Verfahren"	Wenn der Inhalt sich ändert, so wird die View solange nicht neu berechnet, bis das Panel aufgerufen wird. Durch das Setzen des Lazy-Flag wird durch einen gesonderten Request die neue View nur bei Aktivierung des Panels nachgeladen. Ein Beispiel hierfür ist ein Warenkorb: Die Zusammensetzung eines Warenkorbes kann sich ständig ändern, das Anzeigen des Warenkorb-Inhaltes ist jedoch nur zu bestimmten Zeitpunkten wichtig.

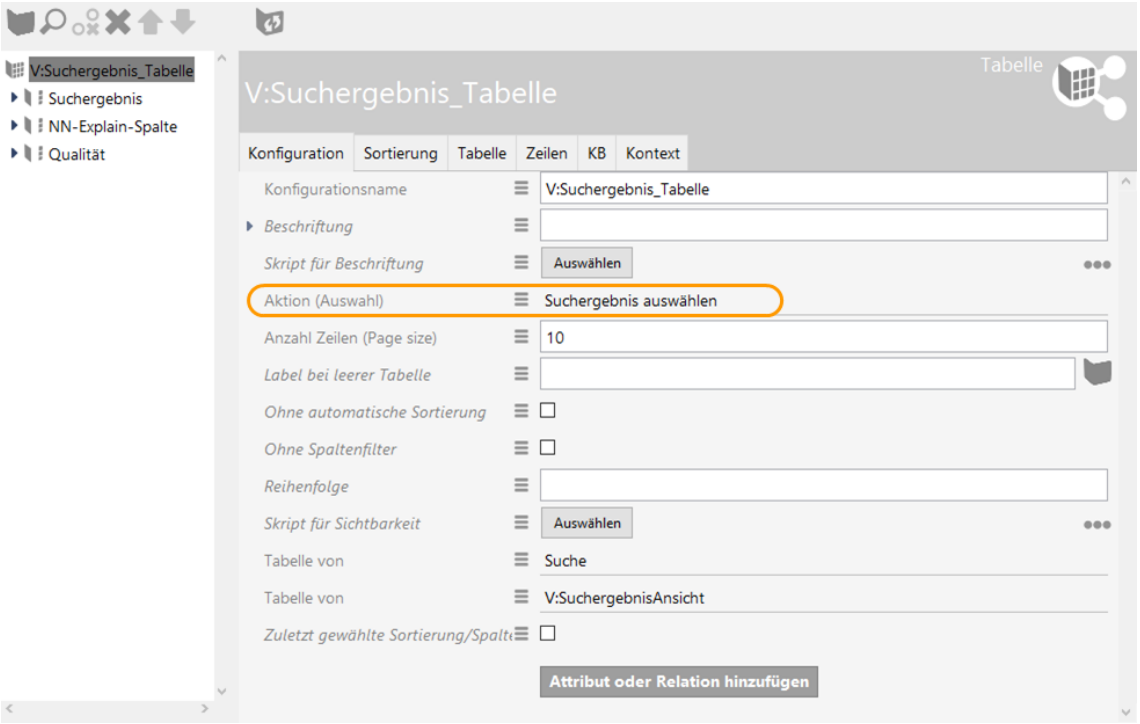
Wenn kein Aktivierungsmodus gewählt wurde, dann gilt per Default der Modus "Anzeige (Active-Flag und Inhalt)".

1.7.3.4.2 Aktionsart "Auswahl"

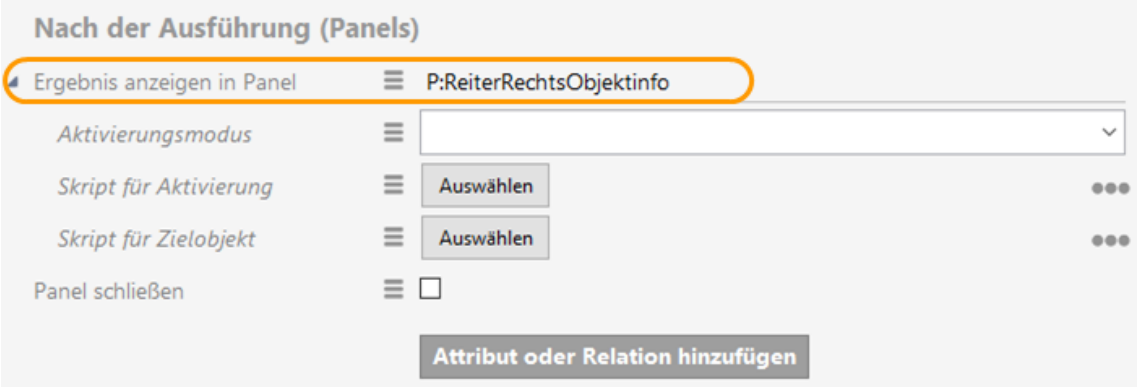
Diese Aktion entspricht der "Anzeigen"-Aktion mit dem einzigen Unterschied, dass die Aktion auf dem Parameter "selectionElement" - also auf einem ausgewählten Element ausgeführt wird. **Achtung:** Dieser Effekt gilt auch für ein ggf. vorhandenes Skript.

Die Aktion "Auswahl" wird ausschließlich (aber nicht zwingend) verwendet, um bei Klick auf einen Tabelleneintrag oder auf einen Listeneintrag aus einem Suchergebnis in einem anderen Panel eine Anzeige hervorzurufen. Eine häufiger Anwendungsfall ist das Anzeigen von Detailinformationen zu einem bestimmten semantischen Element.

Beispiel



Zu beachten ist, dass in der jeweiligen "Auswahl"-Aktion selbst angegeben ist, auf welches Panel sich die Aktion auswirken soll. Dies wird unter "Ergebnis anzeigen in Panel" angegeben.

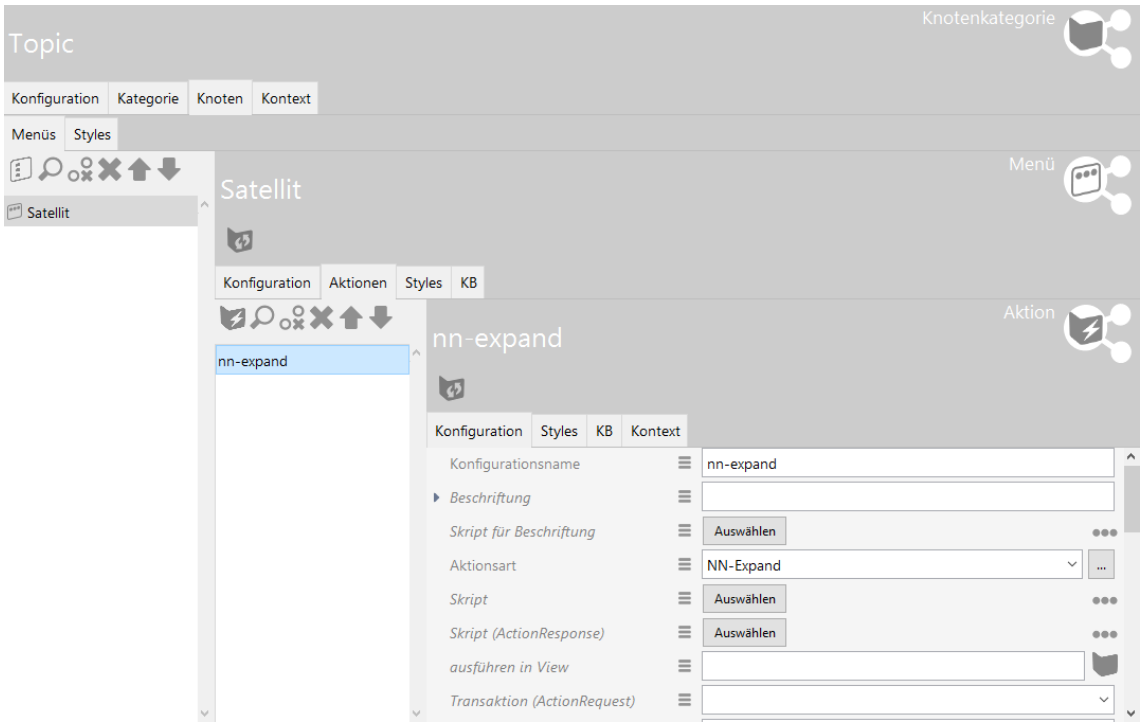


1.7.3.4.3 Aktionsart "NN-Expand"

Bei NN-Expand handelt es sich um eine Aktionsart, die das Aufklappen eines Graph-Knoten im Net-Navigator ermöglicht. D.h. es werden alle Knoten eingeblendet, die über eine Relation mit diesem Knoten verbunden sind und durch die Graph-Konfiguration zugelassen werden. Die betroffenen Relationen zwischen den Knoten werden ebenfalls angezeigt. Knoten, die bereits im Net-Navigator angezeigt wurden, zeigen nur zusätzlich die relevanten Relationen an.

Die Darstellung mit einem Plus-Symbol wie im Bild unten ist bereits voreingestellt. Ebenfalls bereits konfiguriert ist das Dialog-Fenster, dass sich nach Klick auf den Plus-Button öffnet, wenn zu viele Relationen davon betroffen sind. In diesem Dialog kann eine Auswahl getroffen werden, welche Knoten angezeigt werden sollen.

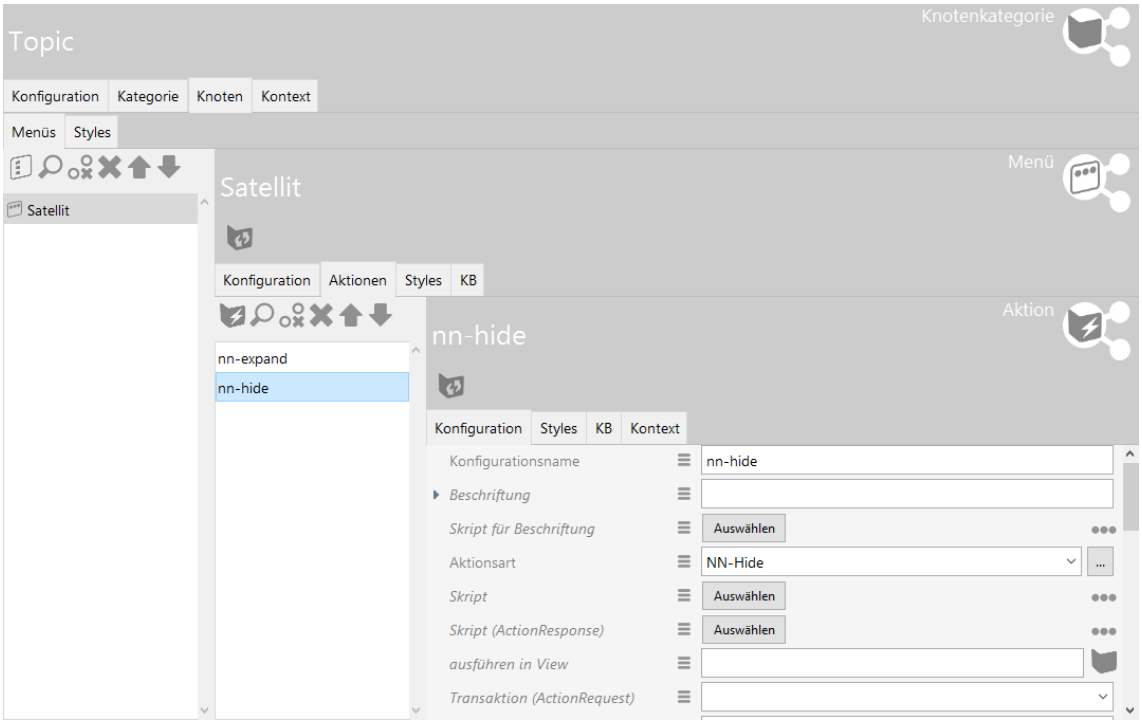
Die Aktion wird in der Graph-Konfiguration an allen Knotenkategorien angebracht, die sie besitzen sollen. Im Reiter "Knoten" wird ein Menü erstellt, das alle NN-Aktionen enthalten kann. In der Aktion selbst muss nur die Aktionsart "NN-Expand" ausgewählt werden, andere Angaben sind optional. Die Aktionsart wird möglicherweise nicht im Dropdown-Menü angeboten, weitere Aktionsarten sind über den "..."-Button daneben abrufbar.



1.7.3.4.4 Aktionsart "NN-Hide"

Mit der Konfiguration dieser Aktionsart wird an den Graph-Knoten ein Menü-Button bereitgestellt, der den ausgewählten Graph-Knoten und dessen angezeigte Relationen einmalig ausblendet (s. durchgestrichenes Auge im Bild). Der Knoten kann beispielsweise mit dem Ausklappen eines anderen verbundenen Knotens wieder angezeigt werden.

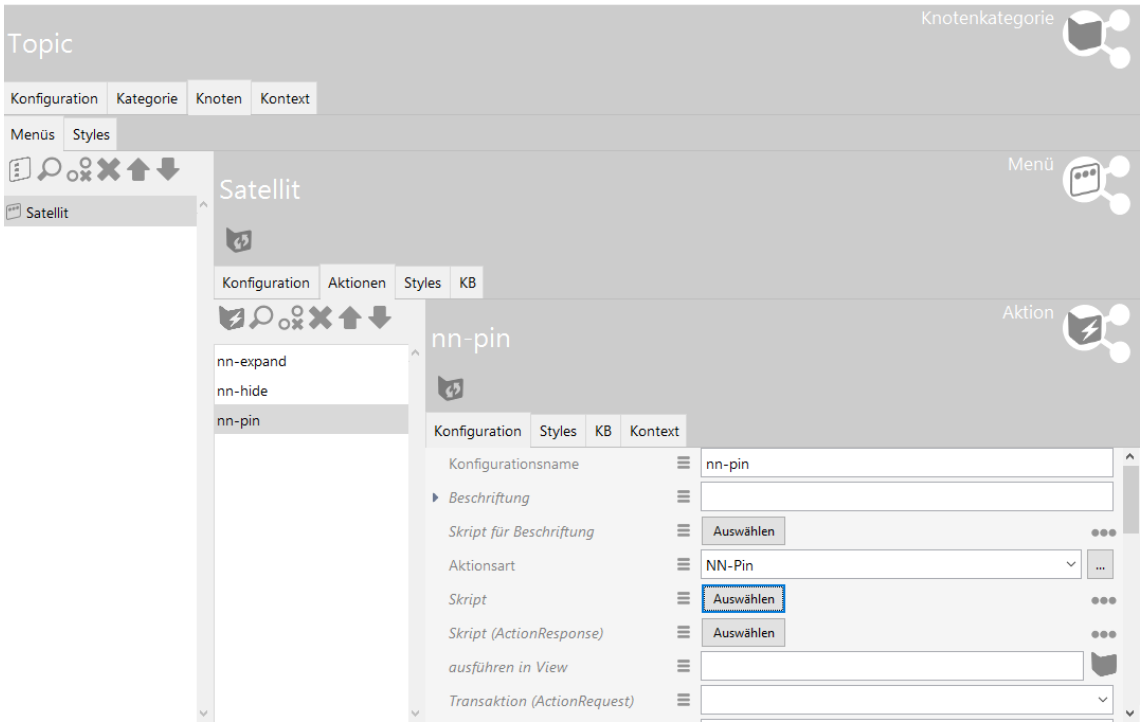
Die NN-Hide-Aktion wird wie die NN-Expand-Aktion konfiguriert, als Aktionsart wird statt "NN-Expand" allerdings "NN-Hide" ausgewählt. Um mehr als eine Aktionsart an einem Knoten zu konfigurieren, müssen mehrere Aktionen an einem Menü angelegt werden.



1.7.3.4.5 Aktionsart "NN-Pin"

Über die NN-Pin-Aktion wird ein Menü-Button konfiguriert, der das Festpinnen eines Knotens im Net-Navigator ermöglicht. Wenn der Graph sich automatisch neu ordnet, beispielsweise beim Ausklappen eines anderen Knotens, bleibt der festgepinnte Knoten an seiner Position. Der Knoten kann trotzdem manuell verschoben werden und der Pin löst sich wieder beim Neuladen des Graphen. Erneutes klicken auf den Pin löst diesen ebenfalls wieder. Der "gepinnt"-Status wird durch eine veränderte Grafik angezeigt (der Pin zeigt nach unten statt schräg zu liegen).

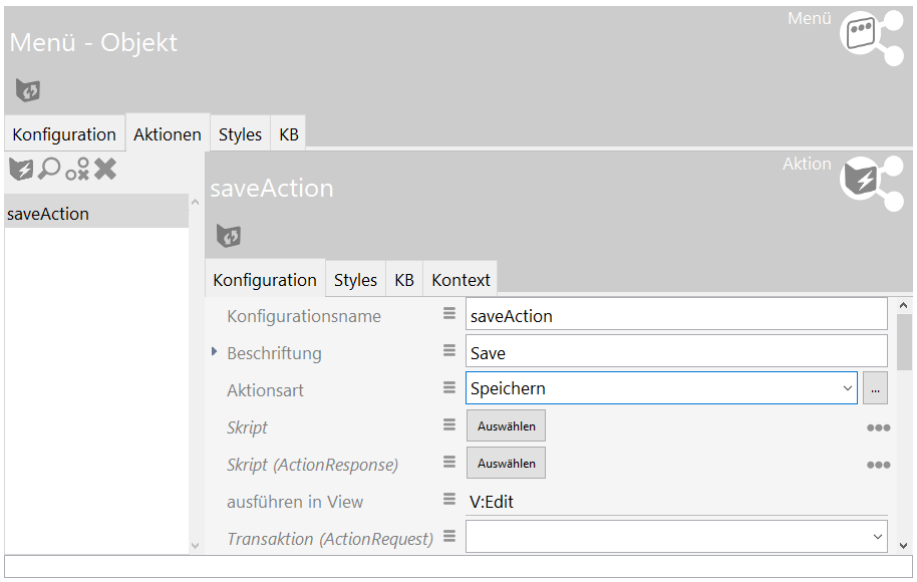
Die Konfiguration der Aktionsart erfolgt wie in der "NN-Expand-Aktion" beschrieben.



1.7.3.4.6 Aktionsart "Speichern"

Die Speichern- Aktion speichert die Formulardaten aus dem Web-Frontend im Wissensnetz. Das Web-Frontend erkennt die Aktionsart automatisch und schickt sie an die konfigurierte View. Ist keine View als Empfänger der Aktion konfiguriert, versucht das Web-Frontend eine passende View in einem benachbarten Panel zu finden.

Hierzu wird der Aktion in einem Menü die Aktionsart "Speichern" zugewiesen:



Die Speichern-Aktion kann beipielsweise dazu verwendet werden, um die einzelnen Speichern-Buttons mehrerer Edit-Felder in einem Dialog durch einen individualisierten Speichern-Button zu ersetzen.

1.7.3.5 Interne Aktionen

Der Gebrauch interner Aktionen setzt fachspezifisches Wissen voraus.
Bei Unklarheiten hierzu wenden Sie sich an den Support von i-views: support@i-views.com.

Die hier aufgeführten Aktionen sind lediglich aus Gründen der Vollständigkeit aufgeführt. Hierzu zählen Aktionen wie:

- Einblenden-Aktion
- Sortierung-Aktion
- Springen-Aktion
- Ziel-anlegen-Aktion
- Skript-Aktion: Das Vorhandensein eines Skriptes an einer Aktion bewirkt automatisch dessen Ausführung, überschreibt also die eingebaute Funktion der jeweiligen Aktionsart.

1.7.3.6 Scripts of actions

1.7.3.6.1 Skript (onAction)

Dieses Skript wird ausgeführt, wenn die Aktion ausgeführt wird. Der Rückgabewert wird an das optionale ActionResponse-Skript weitergereicht.

```
function onAction(element, context) { return element;
}
```

Argumente

element	Das semantische Element, in dessen Kontext die Aktion ausgeführt wird. Einzige Ausnahme bildet die "Auswahl"-Aktion - hier entspricht "element" dem ausgewählten Element, ist also identisch mit "context.selectedElement".
context	Weitere vordefinierte Variablen, die den Kontext der Aktion näher beschreiben

Das Skript einer Aktion kann auf folgende vordefinierte Variablen, in *context* enthalten, zugreifen:

Detaileditor

Variable	Wert
selectedElement	Ausgewähltes Objekt oder ausgewählter Typ
type	Objekttyp. Falls das Element ein Typ ist, wird der Typ selbst verwendet

Objektliste

Variable	Wert
selectedElement	Ausgewähltes Objekt oder ausgewählter Typ. Undefined, falls kein Element oder mehrere Elemente ausgewählt wurden.
selectedElements	Ausgewählte Elemente
elements	Alle Elemente der Objektliste
type	Typ der Objektliste

Transaktionen

Bei schreibenden Änderungen ist eine Transaktion erforderlich. Bei Ausführung über den ViewConfigMapper kann man über die Eigenschaft "Transaktion" steuern, ob und welche Art von Transaktion verwendet werden soll. Standardmäßig ist eine schreibende Transaktion aktiv.

Im Knowledge-Builder ist grundsätzlich keine Transaktion aktiv. Das Skript muss Transaktionen selber steuern.

Knowledge-Builder

Im Knowledge-Builder steht ein weitere Variable zur Interaktion mit dem Benutzer zur Verfügung:

Variable	Wert
ui	Objekt $k.UIObject

Beispielsweise kann eine Meldung anzeigen:

```
ui.alert("Aktuelles Element: " + element.name());
```

1.7.3.6.2 Skript (actionResponse)

Dieses Skript wird nach der Ausführung der Aktion ausgeführt. Hauptaufgabe ist es, das Ergebnis der Aktion für den ViewConfigMapper (oder andere Frontends) aufzubereiten. Das Skript muss ein Objekt vom Typ $k.ActionResponse liefern.

```
function actionResponse(element, context, actionResult) { var actionResponse = new $k.ActionRespon

    actionResponse.setData(actionResult);
    actionResponse.setFollowup("new");
    actionResponse.setNotification("Erledigt","warn");
```

```
    return actionResponse;
}
```

Argumente

element	Das semantische Element, in dessen Kontext die Aktion ausgeführt wird
context	Weitere vordefinierte Variablen, die den Kontext der Aktion näher beschreiben (siehe vorherigen Abschnitt)
actionResult	Der Rückgabewert des onAction-Skripts bzw. falls nicht definiert der Rückgabewert der konfigurierten Aktionsart.

ActionResponse

Die ActionResponse kann um Werte für *Followup / Data* und *Notification* erweitert werden. Diese Werte können von anderen Anwendungen wie z.B. dem ViewConfigMapper ausgewertet werden.

Im Knowledge-Builder sind folgende Werte von *Followup* in Tabellen möglich:

refresh	Rendert die aktuelle Tabelle neu, ohne die Liste neu zu berechnen
update	Berechnet die Tabelle neu
showElement	Selektiert das Element in *data* in der Tabelle an. Alternativ kann in data ein Objekt {"element": actionResult, "viewMode": "edit" } das Ergebnis in einem neuen Detaileditor geöffnet werden

In Detail-Editoren wird *Followup* nicht ausgewertet.

1.7.3.6.3 Skript (actionVisible)

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

Anhand des Rückgabewertes wird entschieden, ob der Knopf angezeigt werden soll oder nicht.

In Tabellen wird bei Aktionen auf den Elementen folgende Funktion aufgerufen, die einen Array von Elementen übergibt und einen Array von boolschen Werten erwartet. Dies kann dazu verwendet werden, die Sichtbarkeit für die Elemente effizienter am Stück zu berechnen.

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

1.7.3.6.4 Skript (actionEnabled)

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

Anhand des Rückgabewertes wird entschieden, ob der Knopf aktiv ist.

In Tabellen wird bei Aktionen auf den Elementen folgende Funktion aufgerufen, die einen Array von Elementen übergibt und einen Array von boolschen Werten erwartet:

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

1.7.3.6.5 Skript mit UI-spezifischen Aktionen

Das die Aktion realisierende Skript kann im Knowledge-Builder über *context.ui* auf UI-spezifische Funktionen zurückgreifen.

UI-Funktionen sollten nach Möglichkeit nicht innerhalb von Transaktionen ausgeführt werden, da sich die Anzeige innerhalb der Transaktion nicht aktualisiert.

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

Zeigt eine Meldung an.

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

Benutzer kann eine Zeichenkette eingeben.

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

Öffnet einen Abbrechen-Dialog.

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

Objekt aus einer Menge auswählen lassen.

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

Standardeditor für das Objekt öffnen.

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

Es wird ein Warte- bzw. Benachrichtigungsdialog geöffnet. Dieser kann, je nachdem wie er konfiguriert ist, abgebrochen werden.

Mögliche Parameter:

Parameter	Beschreibung	Standardwert
autoExpand	Ist der Anzeigebereich des Dialogs initial geöffnet.	true
canCancel	Kann der Dialog abgebrochen werden.	true
stayOpen	Bleibt der Dialog nach Beendigung der Funktion geöffnet.	true

Beispiel:

```
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 },
    "Ein Wartedialog"
)
```

Mit der folgenden Function *raiseNotification* können Meldungen auf dem Anzeigebereich ausgegeben werden.

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

Diese Benachrichtigung wird nur von der Function *notificationDialog* gefangen und die Nachricht wird nur dort im Anzeigebereich ausgegeben.

1.7.4 View configuration elements

Eine Viewkonfiguration beschreibt, wie Objekte oder Typen dargestellt werden sollen. Im folgenden werden die verschiedenen Elementarten, die der View-Konfiguration zur Verfügung stehen, beschrieben.

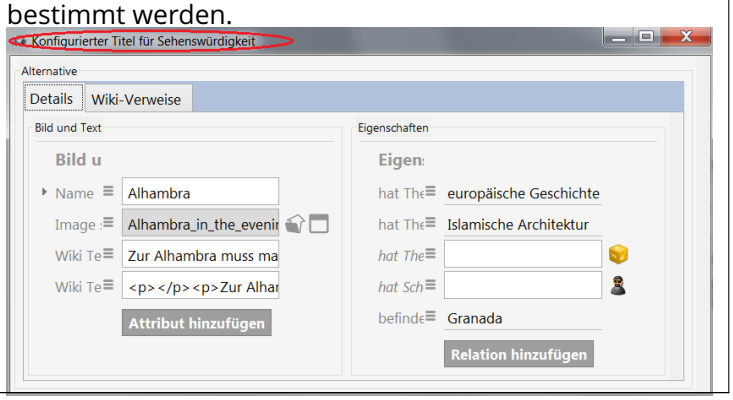
Die einzelnen Viewkonfigurationselemente lassen sich teilweise beliebig zusammenstecken. Ebenfalls können die Konfigurationen mehrfach als Unterkonfiguration verwendet werden.

Liste der verschiedenen Detailkonfigurationstypen

Konfigurationstyp	Top-Level-Konfiguration	Kann folgende Unterkonfiguration enthalten
Alternative	x	beliebig

Eigenschaft		
Eigenschaften	x	Eigenschaft
Gruppe	x	beliebig
Hierarchie	x	beliebig
Skriptgenerierter Inhalt	x	
Statischer Text		
Suche		Tabelle

Einstellungsmöglichkeiten, die alle Detailkonfigurationstypen gemeinsam haben

Name	Wert
Konfigurationsname	Findet keine Verwendung im Userinterface. Der Ersteller einer Konfiguration hat hier die Möglichkeit einen für ihn verständlichen Namen zu vergeben, um diese Konfiguration später besser wiederfinden und in anderen Konfigurationen wieder verwenden zu können.
Skript für Fenstertitel	Nur zur Verwendung im Knowledge-Builder. Öffnet man ein Objekt beispielsweise per Doppelklick in der Objektliste, öffnet sich ein Fenster mit den Eigenschaften dieses Objektes. Der Titel dieses Fensters kann durch ein Skript bestimmt werden. 

Anmerkung: In den folgenden Abschnitten werden die Einstellungsmöglichkeiten für die einzelnen Konfigurationstypen beschrieben. Die obligatorischen Parameter sind fett gedruckt.

1.7.4.1 Alternative

Eine Alternative wird verwendet, um beliebig viele verschiedene alternative Ansichten auf ein Objekt zu konfigurieren. Zwischen den Ansichten kann in der Anwendung mittels Reitern gewechselt werden.

Einstellungsmöglichkeiten

Name	Wert
Beschriftung	Eine Beschriftung findet nur ihre Anwendung, wenn diese Konfiguration in einer anderen Konfiguration wie z.B. einer weiteren *Alternative* eingebettet ist.
hat Default-Alternative	Die Unteransicht, die initial selektiert sein soll, kann hier festgelegt werden.

Darstellung in einer Anwendung

Wenn die Ansichten in JSON herausgeschrieben werden, so werden die einzelnen Unteransichten in einem ARRAY an den KEY *alternatives* gehängt.

Beispiel einer Alternative in einer Anwendung: Mittels der Reiter kann zwischen den Ansichten "Beispiel", "Branchen" und "Eigene Daten anlegen" gewechselt werden.

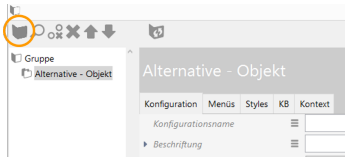
Darstellung im Knowledge-Builder

Im Knowledge-Builder werden dem Anwender die verschieden konfigurierten und mit der Alternative verknüpften Ansichten eines Objekts durch Reiter zur Verfügung gestellt.

Beispiel einer Alternative im Knowledge-Builder: Mittels der Reiter kann zwischen der Ansicht "Reiter 1" und der Ansicht "Reiter 2" gewechselt werden.

Konfiguration von Reitern

Wenn eine View-Konfiguration vom Typ "Alternative" erstellt wurde, kann über den Button "Objekte von Objektkonfiguration neu anlegen" ein neuer Reiter hinzugefügt werden.



Meistens empfiehlt es sich, als Reiter den View-Konfigurationstyp "Gruppe" zu verwenden, da sich hierin beliebige andere View-Konfigurationen platzieren lassen. Die Beschriftung der View-Konfiguration ist gleichzeitig auch die Beschriftung des Reiters.

1.7.4.2 Groups

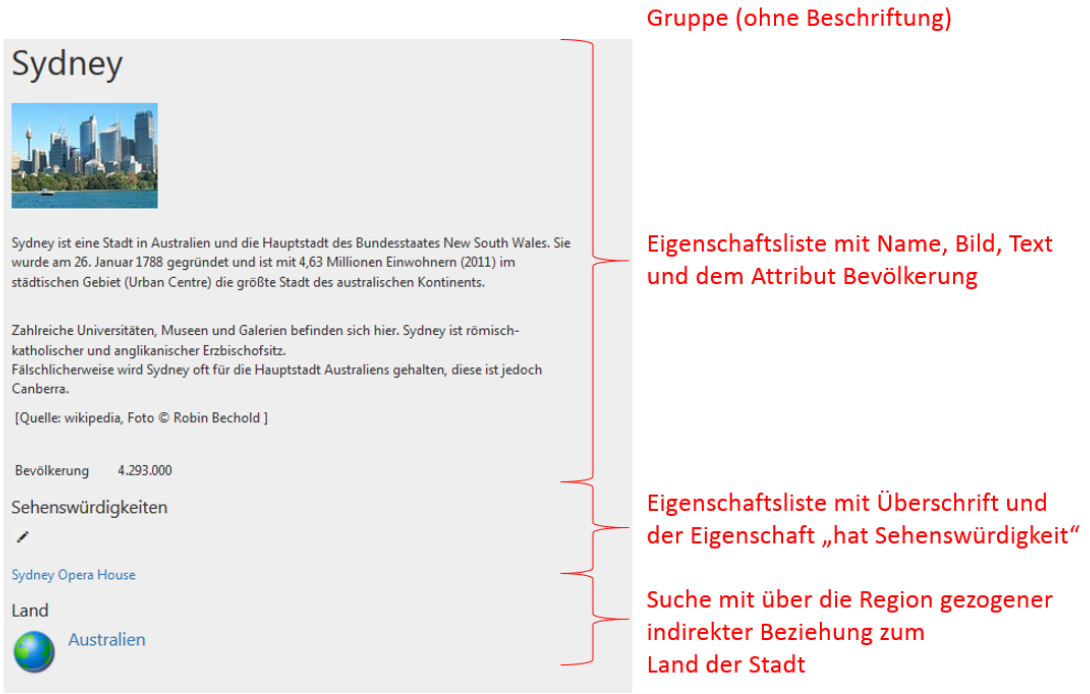
Mit Hilfe einer Gruppe lassen sich verschiedene Unterkonfigurationen in einer Ansicht zusammenfassen. Die Unterelemente werden dann der Reihe nach dargestellt. Es gibt jedoch Ausnahmen, die nur für das Front-End gelten: Das Konfigurationselement *Eigenschaft* kann keine direkte Unterkonfiguration von Gruppe sein. Hierfür braucht es zunächst die Konfiguration *Eigenschaften*.

Einstellungsmöglichkeiten

Name	Wert
Beschriftung	Eine Beschriftung findet nur ihre Anwendung, wenn diese Konfiguration in einer anderen Konfiguration wie z.B. *Alternative* eingebettet ist.

Darstellung in einer Anwendung

Wird die Ansicht in JSON herausgeschrieben, werden die einzelnen Unteransichten in einem ARRAY an den KEY *group* angehängt.



Darstellung im Knowledge-Builder

Im Knowledge-Builder wird um eine Gruppe ein Rahmen gezeichnet. Die Ansichten der Unterkonfigurationen werden dann in diesem Rahmen angezeigt.

Eine Gruppe mit folgenden Unterkonfigurationen: der Eigenschaftsliste "Bild und Text", der Eigenschaftsliste "Eigenschaften" und der Suche "Ähnliche Sehenswürdigkeiten"

1.7.4.3 Hierarchy

Der Konfigurationstyp "Hierarchie" stellt Elemente eines semantischen Modells hierarchisch in einer Baumstruktur dar, in der einzelne Äste auf- und zugeklappt werden können.

Es kann entweder mit Relationen oder Relationszielen gearbeitet werden. Der Aufbau der Hierarchie geschieht vom Startelement der View-Konfiguration aus, zu dem zunächst alle untergeordneten Relationen bzw. Objekte und deren Untergeordnete ermittelt werden. Danach werden für jedes Element die übergeordneten Relationen bzw. Objekte ermittelt. Diese Ergebnismenge von Elementen wird dann in der Hierarchie dargestellt.

Einstellungsmöglichkeiten

Name	Wert
Beschriftung	Eine Beschriftung findet nur ihre Anwendung, wenn diese Konfiguration in einer anderen Konfiguration wie z.B. *Alternative* eingebettet ist.
Skript für Beschriftung	Es ist auch möglich durch ein Skript eine Beschriftung festzulegen.
Banner der Hierarchiewurzel anzeigen	Betrifft nur den Knowledge-Builder: Banner wird angezeigt.
Aktion (Auswahl)	Verweis auf eine Aktion, die beim Anklicken eines Hierarchie-Elements aufgerufen wird.
Detailansicht ausblenden	Standardmäßig wird die Detailansicht eines ausgewählten Objektes angezeigt (Knowledge-Builder) oder ausgegeben (json, als *subview*). Durch Aktivieren dieser Option wird keine Detailansicht angezeigt bzw. ausgegeben.
Skript für Sichtbarkeit	
Unterelemente erzeugen ohne Frage nach Namen	Wenn neue Unterelemente in der Hierarchie erzeugt werden, wird standardmäßig gefragt, wie ihr Name lauten soll. Ein Häkchen hier, erzeugt ohne Frage nach Namen namenlose Objekte.

Verbiete manuelles Sortieren	Standardmäßig kann der Anwender im Knowledge Builder Elemente dem Schema entsprechend durch Drag&Drop umhängen. Wird diese Option aktiviert, ist dies nicht mehr möglich.

Einstellungsmöglichkeiten für die Sortierung

Name	Wert
Absteigend sortieren	Steuert, ob auf- oder absteigend sortiert wird. Ist dieser Parameter nicht gesetzt, wird aufsteigend sortiert.
Primäres Sortierkriterium	Auswahlmöglichkeit für das Kriterium, nach dem die Eigenschaften sortiert werden: • *Position*: Die in der Konfiguration festgelegte Reihenfolge wird verwendet (Default). • *Wert*: Inhalt des Attributes bzw. Anzeigename des Relationszieles wird verwendet. • *Skript für Sortierung*: Das in dem Attribut Skript zur Sortierung hinterlegte Skript wird zur Ermittlung des Sortierkriteriums verwendet.
Sekundäres Sortierkriterium	Sortierkriterium für Eigenschaften, die für das primäre Sortierkriterium den gleichen Wert haben. Einstellmöglichkeiten analog zum *Primären Sortierkriterium.*
Skript für Sortierung	Verweis auf ein registriertes Skript, das den Sortierschlüssel für das primäre bzw. sekundäre Sortierkriterium zurückgibt.

Ermittlungsmöglichkeiten der hierarchiebildenden Elemente

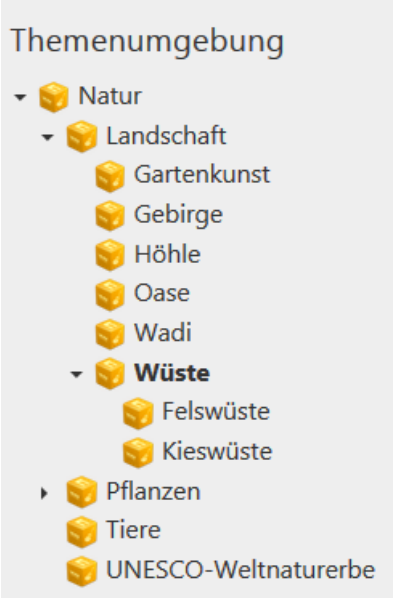
Name	Ermittlung von ...
Relation (absteigend)	Unterelementen
Relation (aufsteigend)	Oberelementen
Strukturabfrage (absteigend)	Unterelementen
Strukturabfrage (aufsteigend)	Oberelementen
Skript (absteigend)	Unterelementen
Skript (aufsteigend)	Oberelementen

Aktionen und Styles

Es lassen sich sowohl für die gesamte Hierarchie als auch für die einzelnen Knoten Aktionen und Styles anbringen. Ab Version 5.2 kann man auch automatisch Style-Klassen über ein Skript zuweisen lassen.

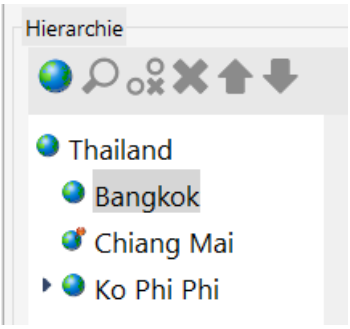
Darstellung in einer Anwendung

Erst ab Version 4.1 gibt es die JSON-Repräsentation einer Konfiguration vom Typ *Hierarchie*.



Darstellung im Knowledge-Builder

In der Detail-Anzeige eines Elements wird im linken Bereich eine Hierarchie eingeblendet. Im rechten Bereich wird das Element mit einer View-Konfiguration ohne Hierarchie angezeigt. Diese View-Konfiguration muss eigens definiert werden und unter *Verwendung >> anwenden in* muss der Konfigurationsname der Hierarchie angegeben werden. Die Subkonfiguration lässt sich alternativ auch direkt an der Hierarchie unter *Subkonfiguration* angeben.



Anmerkungen

- Elemente werden in Hierarchien immer mit ihrem Namen repräsentiert. Es ist nicht möglich etwas anderes als den Namen oder zusätzlich zum Namen Informationen direkt in der Hierarchie anzuzeigen.
- Die Werte aller Eigenschaften, die für die Hierarchiebildung ausgefüllt werden können, sind Relationen.

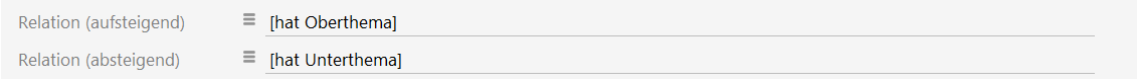
- Die einzelnen Attribute wie z.B. *Relation - absteigend* können mehrfach vergeben werden.
- Für jeden Attributtyp werden die Relation oder Relationen ermittelt und aufgesammelt. Sind verschiedene Attributtypen angegeben, wird mit den Teilmengen eine Schnittmenge gebildet.

Beispiel - Anwendungsfall

Typischerweise werden Hierarchien verwendet, um Ober-/Unterthema-Relationen oder Teil-von-Relationen darzustellen.

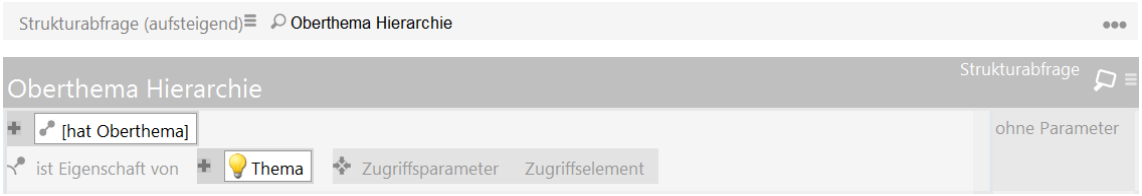
1. Hierarchiebildende Relation

Die direkteste Variante. Die Relationen, die die Hierarchie bilden, werden eingetragen.



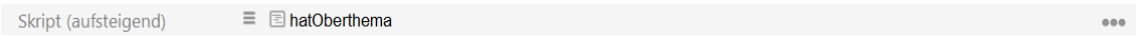
2. Hierarchiebildende Strukturabfrage

Die Relationen lassen sich ebenfalls über eine Strukturabfrage ermitteln.



3. Hierarchiebildendes Skript

Auch durch ein Skript lassen sich die möglichen hierarchiebildenden Relationen aufsammeln. Es bekommt das aktuelle Element als Parameter übergeben und muss eine Menge an Relationen zurückgeben. Statt auf Relationen kann man aber auch auf Elementen arbeiten.



Skript *hat Oberthema*

```
function relationsOf(element)
{
    return element.relations('hatOberthema');
}
function targetsOf (element)
{
    return element.relationTargets('hatOberthema');
}
```

1.7.4.4 Properties

Die Konfiguration *Eigenschaften* ist eine Liste von einzelnen Eigenschaften. Die Unterkonfigurationen können ausschließlich vom Typ *Eigenschaft* sein, welche jeweils mit einem Attribut oder einer Relation eines Wissensnetz-Objekts oder -Typs verknüpft ist.

Einstellungsmöglichkeiten

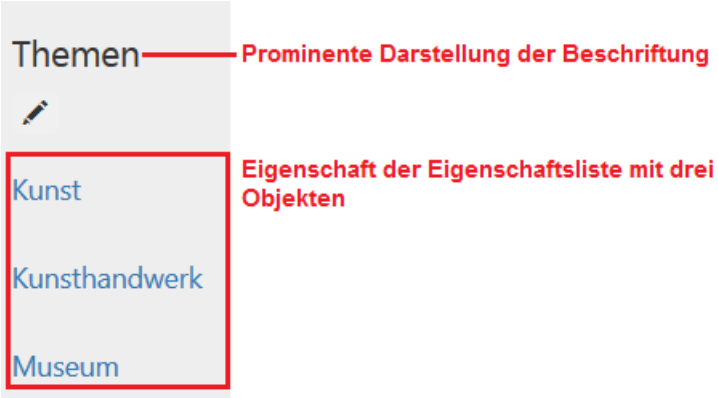
Name	Wert
Beschriftung	Anzeigename der Sammlung von Eigenschaften. Ist keine Beschriftung angegeben wird im Knowledge-Builder die Zeichenkette 'Eigenschaften' verwendet.
Skript für Beschriftung	Alternativ kann der Anzeigename auch über ein Skript ermittelt werden.
Skript für Sichtbarkeit	Steuerung der Sichtbarkeit der Eigenschaften durch ein Skript.
Initial ausgeklappt	Ist diese Konfiguration z.B. als Metakonfiguration eingehängt, kann mit diesem Parameter bestimmt werden, ob diese beim Öffnen des Knowledge-Builder-Editors bereits ausgeklappt sein soll. **Hinweis:** Das Web-Frontend stellt die betroffene Metaeigenschaft nicht dar, wenn der Haken hier nicht gesetzt ist.

Einstellungsmöglichkeiten für die Sortierung

Name	Wert
Absteigend sortieren	Steuert, ob auf- oder absteigend sortiert wird. Ist dieser Parameter nicht gesetzt, wird aufsteigend sortiert.
Primäres Sortierkriterium	Auswahlmöglichkeit für das Kriterium, nach dem die Eigenschaften sortiert werden: • *Position*: Die in der Konfiguration festgelegte Reihenfolge wird verwendet (Default). • *Wert*: Inhalt des Attributes bzw. Anzeigename des Relationszieles wird verwendet. • *Skript zur Sortierung*: Das in dem Attribut Skript zur Sortierung hinterlegte Skript wird zur Ermittlung des Sortierkriteriums verwendet.
Sekundäres Sortierkriterium	Sortierkriterium für Eigenschaften, die für das primäre Sortierkriterium den gleichen Wert haben. Einstellmöglichkeiten analog zum *Primären Sortierkriterium.*
Skript zur Sortierung	Verweis auf ein registriertes Skript, das den Sortierschlüssel für das primäre bzw. sekundäre Sortierkriterium zurückgibt.

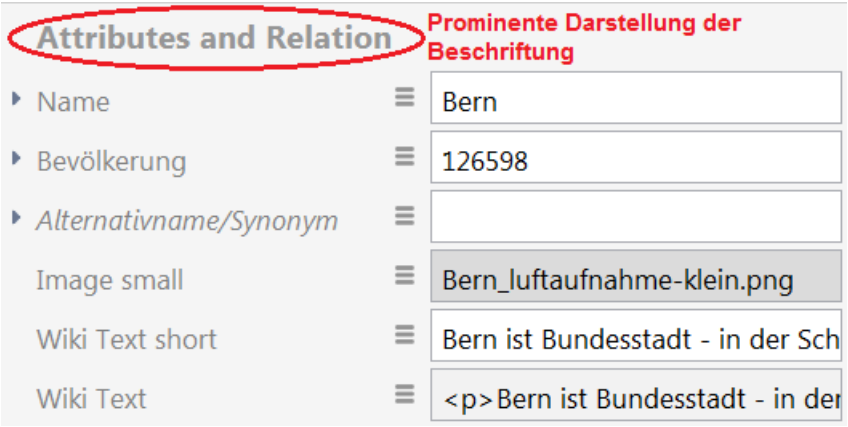
Darstellung in Anwendungen

Die Ansichten der Konfiguration einzelner Eigenschafts-Elemente werden beim Herausschreiben im JSON-Format in einem ARRAY abgelegt und mit dem KEY *properties* angehängt.



Darstellung im Knowledge-Builder

Die in der Konfiguration eingestellte Beschriftung wird prominent angezeigt. Ihm folgen die Ansichten der konfigurierten Eigenschaften.



Anmerkung

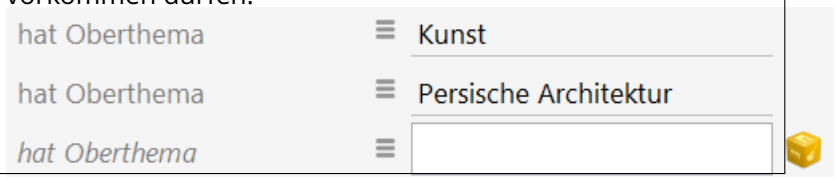
Meta-Eigenschaften werden mit dem gleichen Vorgehen angehängt.

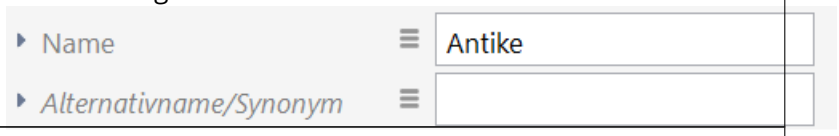
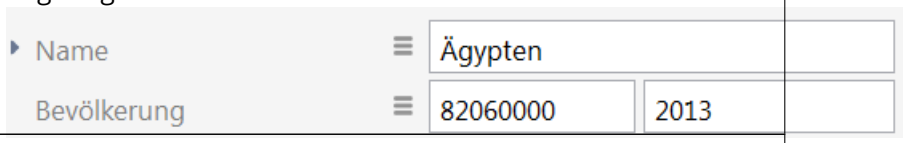
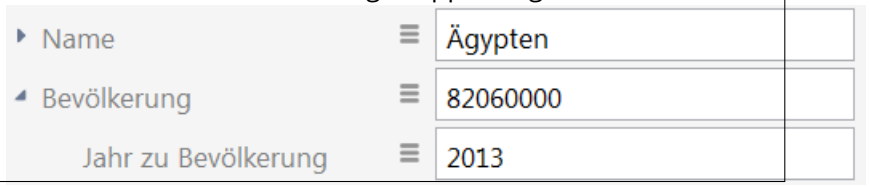
1.7.4.5 Property

Mit der View-Konfiguration *Eigenschaft* können einzelne Attribute oder Relationen definiert werden, die in einer Eigenschaften-Liste angezeigt werden sollen. Es kann auch eine abstrakte Eigenschaft benutzt werden, die eine Menge von Eigenschaften zusammenfasst.

Einstellungsmöglichkeiten

Name	Wert
Absteigend sortieren	Steuert, ob die Eigenschaften auf- oder absteigend nach ihrem Namen sortiert werden. Ist dieser Parameter nicht gesetzt, wird *aufsteigend* sortiert.

Anzeigeart	Steht in zwei Fällen zur Verfügung: **1.** Die Eigenschaft ist eine Relation: Auswahlmöglichkeit für die Darstellung der Beschriftung eines Relationsziels. Diese Einstellungsmöglichkeit steht nur dann zur Verfügung, wenn die Einstellung *Relationszielansicht* den Wert *Auswahl* oder *Relationsstruktur hat*. **2.** Die Eigenschaft ist ein Datei-Attribut: Auswahlmöglichkeit für die Darstellung des Wertes eines Datei-Attributs. Auswahlmöglichkeiten: • *Symbol (topic*Icon): Icon des Relationsziels/die Datei als Icon • *Symbol und Zeichenkette* • *Zeichenkette (Namensattribut)*: Name des Relationsziels/Name der Datei
Beschriftung	Anzeigename der Eigenschaft. Ist keine Beschriftung angegeben wird der Name des Eigenschaftstyps ausgegeben.
Eigenschaft	Verknüpfung zu einem Eigenschaftstyp, der angezeigt werden soll.
Skript für virtuelle Eigenschaften	Alternativ zu 'Eigenschaft': Skript zur Berechnung anzuzeigender Werte
Einblendung des Relationsziels	Steht nur bei Relationen zur Verfügung. Standardmäßig wird lediglich der Name des Relationsziels angezeigt. Wird der Name angeklickt, öffnet sich das Relationsziel in einem weiteren Editor. Wird hingegen die Option *Einblendung des Relationsziels* gewählt, werden die Relationsziele direkt angezeigt, d.h. nicht nur ihr Name, sondern alle ihre Eigenschaften.
Einblendung zusätzlicher Eigenschaften	Nur bei der Ansicht zum Bearbeiten von Objekten relevant: Ist diese Option gesetzt, wird zu der Eigenschaft eine weitere Eigenschaft eingeblendet, sodass diese bequem und schnell ausgefüllt werden kann. Die Eigenschaft muss mehrfach vorkommen dürfen. 

Einblendung neuer Eigenschaften	Nur bei der Ansicht zum Bearbeiten von Objekten relevant: Ist diese Option gesetzt, wird die Eigenschaft nur dann eingeblendet, wenn die Eigenschaft noch nicht angelegt wurde. So kann sie bequem und schnell ausgefüllt werden und wird nicht so leicht vergessen. 
Einblendungsfilter	Nur bei der Ansicht zum Bearbeiten von Objekten relevant: Mit dieser Option kann eine Verknüpfung zur einer Abfrage angelegt werden, die entscheidet, ob diese Konfiguration angezeigt wird. Die Abfrage wird mit dem Objekt dieser Eigenschaft gefüllt. Nur, wenn die Abfrage ein Ergebnis enthält, wird die Eigenschaft zum Bearbeiten angezeigt.
Konfiguration für eingebettete Metaeigenschaften	Angabe der Konfiguration, die verwendet werden soll, um Metaeigenschaften anzuzeigen. Die Metaeigenschaften werden eingebettet, d.h. hinter dem Wert der Eigenschaft angezeigt. Der Name des Eigenschafttyps wird nicht angezeigt. 
Konfiguration für Metaeigenschaften	Angabe der Konfiguration, die verwendet werden soll, um Metaeigenschaften anzuzeigen. Die Metaeigenschaften werden unter dem Wert der Eigenschaft angezeigt. Für die Anzeige im Web-Frontend müssen die Eigenschaften mit den Metas auf "initial ausgeklappt" eingestellt sein. 

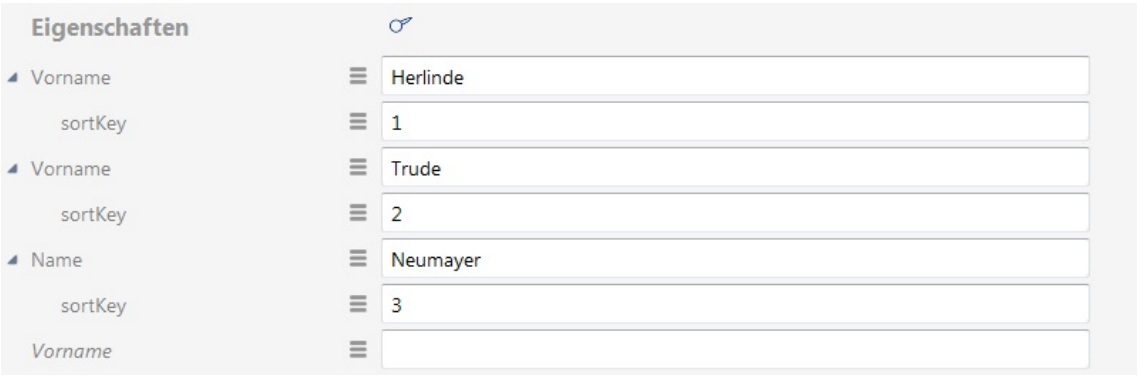
Relationszielansicht	Wird als Eigenschaft eine Relation gewählt, kann mit diesem Parameter die Ansicht der Relationsziele definiert werden: • *Auswahl*: Alle Relationsziele werden aufgelistet und mit einer vorangestellten Checkbox angezeigt. Bei bestehenden Relationen ist die CheckBox mit einem Häkchen versehen. • *Drop Down*: Diese Einstellung ist nur sinnvoll, wenn die Relation nur einmal vorkommen darf. Es wird eine Drop Down-Liste mit allen möglichen Relationszielen zur Auswahl angezeigt. • *Relationsstruktur*: Alle Relationsziele werden im linken Bereich aufgelistet, ähnlich einer Hierarchie. Im rechten Bereich ist dann die Detailansicht des selektierten Relationsziels zu sehen. Diese Ansicht kommt nur zur Geltung, wenn die Konfiguration direkt einer Top-Level-Konfiguration untergeordnet ist. • *Tabelle*: Tabellenansicht der Relationen. Die Tabellenansicht kann *nicht* im Knowledge-Builder angewendet werden. Für die Tabellenansicht muss die Einstellung *Tabelle* ausgefüllt werden. • *Tabelle (Relationsziele)*: Tabellenansicht der Relationsziele. Diese Tabelle kann im Knowledge-Builder angewendet werden.
Skript für Beschriftung	Die Beschriftung kann durch ein hier angegebenes Skript ermittelt werden.
Skript für Relationszielbezeichner	Über dieses Skript kann für Relationen gesteuert werden, wie das Relationszeil dargestellt wird. Wenn kein Skript eingetragen ist, wird der Objektname des Relationsziels zur Darstellung verwendet. Anwendungsbeispiel: Eine Person gehört zu einer Abteilung mit Namen 'Abt. IV'. Durch ein passendes Skript kann erreicht werden, dass in der Anzeige bei der Person statt 'Abt. IV' die Angabe 'Stadtverwaltung Darmstadt, Abt. IV' erscheint.
Skript für Sortierung	Anhand des Skriptes wird ein Wert, nach dem sortiert wird, ermittelt. Siehe Beispiel unten.
Suche zur Zielauswahl	Die Suche bestimmt, welche Wissensnetzelemente als mögliche Relationsziele angeboten werden. Zur Konfiguration der Suche, siehe auch Kapitel Suchen/Abfragen.
Tabelle	Steht nur zur Verfügung, wenn *Relationszielansicht* den Wert *Tabelle*, bzw. *Tabelle (Relationsziele)* hat und ist dann obligatorisch. Die hier angegebene Tabellenkonfiguration gibt an, welche Eigenschaften der Relationsziele tabellarisch ausgegeben werden sollen. Um das Relationsziel anzuzeigen muss mindestens sein Name in der Tabelle konfiguriert werden. Zur Konfiguration einer Tabelle, siehe Kapitel Tabelle.

Beispiel für *Skript für Sortierung*

Vorbedingung: An allen Eigenschaften kann das Attribut *sortKey* angebracht werden.

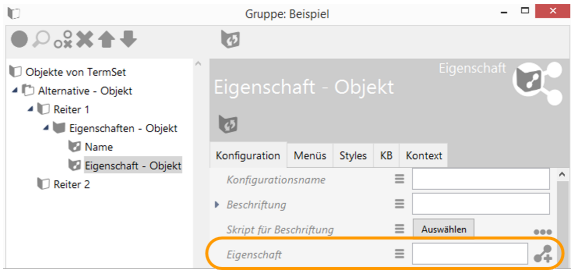
```
function sortKey(element)
{
    if (element instanceof $k.Property)
    {
        var attribute = element.attribute('sortKey')
        if (attribute)
        {
            return attribute.value();
        };
    };
    if (element instanceof $k.Domain)
    {
        var attribute = element.type().attribute('sortKey');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

Eigenschaftsanzeige einer Person



Konfiguration einer Eigenschaft

Eine Eigenschaft kann nur als Teil einer Eigenschaftenliste konfiguriert werden. Es ist in Ordnung, wenn die Liste nur eine Eigenschaft enthält.



In diesem Beispiel enthält die Eigenschaften-View-Konfiguration bereits die Eigenschaft "Name". Eine zweite Eigenschaft wird angelegt, wenn bei "Eigenschaft" (orange markiert) ein Attribut oder

eine Relation ausgewählt wird.

Anmerkung

Beim Attributtyp *Name* ist für *sortKey* der Wert 3 eingetragen, daher steht dieser vorläufige Wert am Ende der Liste.

1.7.4.6 Edit

Dieser Konfigurationstyp wird benutzt um Attribute und Relationen einer *Eigenschaften*-Konfiguration editierbar zu machen. Dazu wird er dem jeweiligen *Eigenschaften*-Element übergeordnet. Neben einem Knopf zum Speichern der Änderungen, wird neben jeder Eigenschaft, bei der dies möglich ist, ein Löschen-Knopf angezeigt.

Einstellungsmöglichkeiten

Name	Wert
Beschriftung	Eine Beschriftung findet nur ihre Anwendung, wenn diese Konfiguration in einer anderen Konfiguration wie z.B. *Alternative* eingebettet ist.
Editiermodus umschaltbar	Wird diese Option ausgewählt, werden die Eigenschaften zunächst nur als normale Liste angezeigt. Zusätzlich wird jedoch ein Schalter angeboten, mit dem man zwischen der normalem und der Editieransicht umschalten kann.
Nur benutzerdefinierte Schaltflächen	Ist diese Option gesetzt, wird der Speichern-Knopf nicht angezeigt.

1.7.4.7 Tabelle

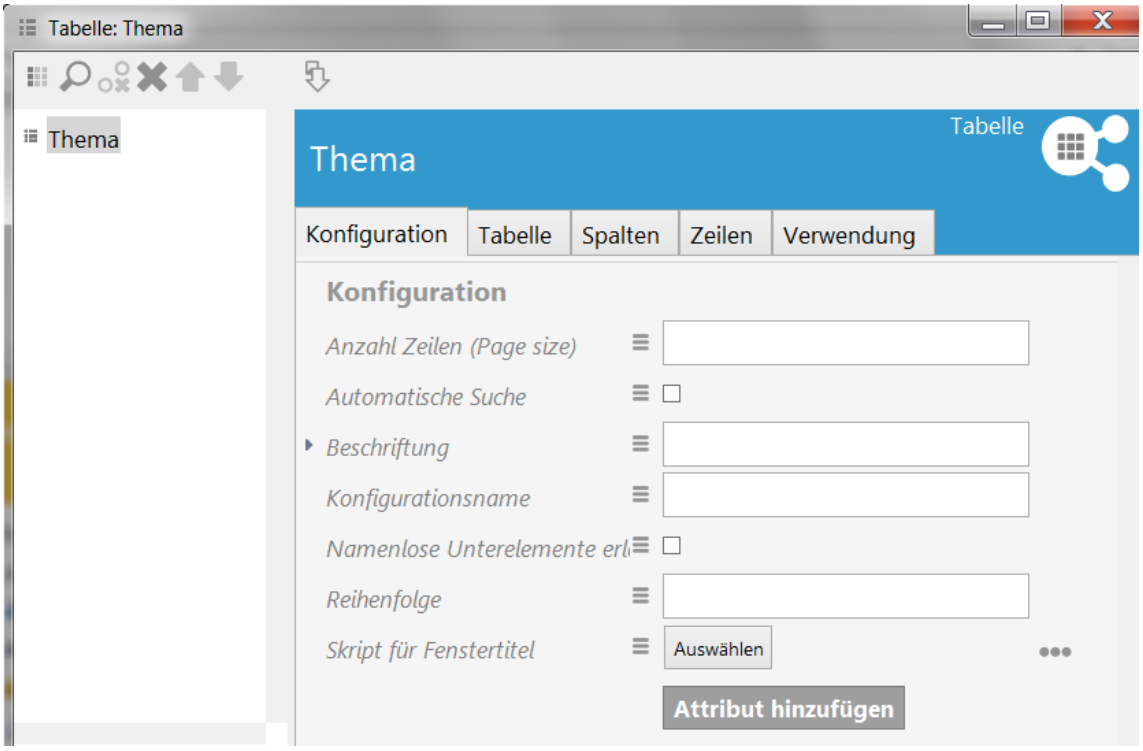
Tabellen können als Unterkonfiguration für die Ergebnisanzeige von Abfragen des Konfigurationstyps "Suche" oder als eigenständige Konfiguration zur Darstellung der Objektlisten im Knowledge-Builder verwendet werden.

Eine Tabelle listet konkrete Objekte, Eigenschaften oder Untertypen eines bestimmten Typs auf. Ob alle Objekte, Eigenschaften oder Untertypen oder nur eine Auswahl angezeigt werden, lässt sich über die Eingabe in den Spaltenköpfen steuern. Mit den eingegebenen Werten wird eine Strukturabfrage nach passenden Objekten, Eigenschaften oder Untertypen ausgeführt und das Ergebnis tabellarisch dargestellt. Außerdem kann bei Objektlisten nach Eingabe von Werten in die Spaltenköpfe ein neues Objekt, ein neuer Eigenschaftswert oder ein neuer Untertyp mit den ausgefüllten Eigenschaften erzeugt werden.

Bestandteile der Konfiguration *Tabelle* sind *Spaltenkonfigurationen*. Diese wiederum beinhalten *Spaltenelemente*. Diese Aufteilung dient der Trennung von spaltenrelevanten Eigenschaften, wie Reihenfolge und Benennung der Spalte in der Tabelle, und der Zuordnung, welche Inhalte in der Spalte angezeigt werden sollen. *Spaltenelemente* wiederum erlauben die Zuordnung von Eigenschaften, zusätzlich können Skript-Bausteine und Strukturabfrage-Bausteine eingebettet werden.

Seit Version 5.1. lassen sich in eine *Tabellen*-Konfiguration nicht nur *Spaltenkonfigurationen* einfügen, sondern auch weitere *Tabellen*. Dies bietet die Möglichkeit Spalten, die öfter Verwendung finden, in einer *Tabellen*-Konfiguration zusammenfassen und diese komplett in eine

andere *Tabelle* einzuhängen. Bei der Ermittlung der Gesamttabelle werden die Zwischen-Tabellen entfernt. Es gibt nur eine Ebene von Spalten.

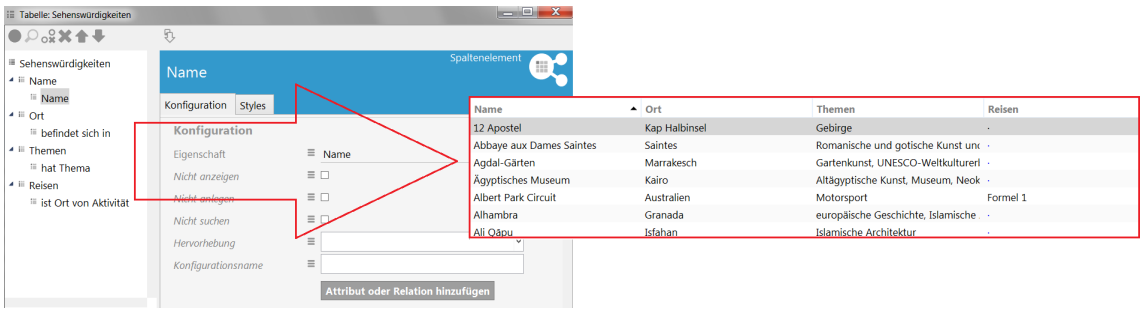


Die hierarchische Darstellung aller Unterkonfigurationselemente der Tabellenkonfiguration weist eine Menü-Zeile auf, die wie folgt mit Aktionen belegt ist:

- Neues Unterelement anlegen und verknüpfen.
- Bereits vorhandene mögliche Unterelemente durchsuchen und verknüpfen
- Verknüpfung wieder löschen. Das Unterelement bleibt dabei als Objekt erhalten und kann in anderen Konfigurationen wieder verwendet werden.
- Gewähltes Unterelement komplett löschen. Falls es in anderen Konfigurationen verwendet wird, öffnet sich vor der Löschung eine Warnung, die alle vorhandenen Verknüpfungen aufzeigt.
- Gewähltes Unterelement in der Liste nach oben schieben.
- Gewähltes Unterelement in der Liste nach unten schieben.
- Aktualisierung. Erst nach einer Aktualisierung werden alle Änderungen für die Anwendungen übernommen.

Beispiel einer einfachen Tabellenkonfiguration

Zu einer Liste von Sehenswürdigkeiten soll der Ort, an dem sie sich befinden, Themen, mit denen sie zu tun haben und Reisen, bei denen sie besucht werden in einer Tabelle zu sehen sein. Nicht zu vergessen ist das Namensattribut, durch das die Sehenswürdigkeiten in der ersten Spalte repräsentiert werden.



Einstellungsmöglichkeiten (Tabelle)

Name	Wert
Anzahl Zeilen (Page size)	Gibt an, wie viele Zeilen auf einer Seite angezeigt werden sollen. Standardwert: 20
Keine automatische Suche	Es wird keine automatische Suche durchgeführt.
Namenlose Unterelemente erlauben	Namenlose Unterelemente werden ebenfalls angezeigt.
Ohne Sortierung	Es findet keine Sortierung statt. Standardverhalten: es wird nach der ersten Spalte sortiert.
Reihenfolge	Durch Angabe einer Ganzzahl lässt sich steuern an welcher Stelle, falls mehreren Konfigurationen vom Typ *Tabelle* angezeigt werden sollen, die aktuelle Konfiguration angezeigt wird. Die Sortierung wird nach zwei Kriterien durchgeführt, die in der folgenden Reihenfolge überprüft werden: 1. Attribut *Reihenfolge* vorhanden, wenn ja, dann wird dieses als Sortierkriterium verwendet, wenn nein, werden erst die Konfigurationen für Typen und dann die für Objekte angezeigt. 2. Sortierung nach Anzeigename

Aktionen und Styles

Aktionen und Styles lassen sich für die gesamte Tabelle, aber auch für Zeilen festlegen.

Verwendung

Wo die Tabelle zur Verwendung kommt, wird auf dem Reiter *Verwendung* angegeben.

Unter *anwenden auf* wird der Objekttyp angegeben, auf den die Tabelle angewendet werden soll. Tabellen können in anderen View-Konfigurationen wieder verwendet werden. Falls die Tabelle Baustein einer anderen View-Konfiguration ist, wird dies unter *[inverse] anwenden in* angezeigt.

Die Eigenschaft *anwenden in* verweist auf eine Anwendung. Mehrere Verknüpfungen sind möglich.

Beispiele:

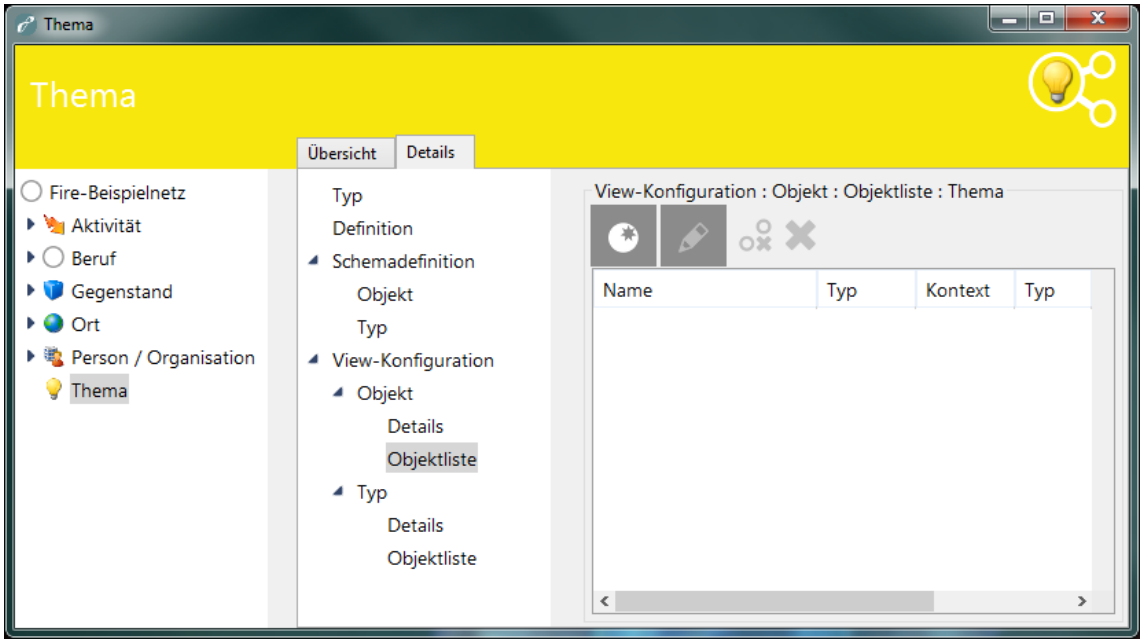
- Soll die Tabelle im Knowledge-Builder rechts im Hauptfenster bei der Navigation durch die Ordnerstruktur verwendet werden, dann muss die Tabellenkonfiguration mit dem entsprechenden Ordnerstrukturelement verknüpft sein.
- Sollen mögliche Relationsziele im Knowledge-Builder tabellarisch dargestellt werden, dann muss die Tabelle mit der Anwendung Knowledge-Builder verknüpft sein.

Tabellen / Objektslisten im Knowledge-Builder

Für die Konfiguration der tabellarische Darstellung von Objekten oder Typen im Knowledge-Builder findet sich im Reiter *Details* beim jeweiligen Typen der Abschnitt *View-Konfiguration -> Objekt/Typ -> Objektliste*. Das Erstellen und Pflegen der Tabellen-Konfiguration wird am Beispiel der Objekte des Typs *Thema* erläutert.

Objekte vom Typ *Thema* bieten selbst als zusätzliche Eigenschaften Synonym und die Relation *ist Thema von* an. Diese Eigenschaften, zusammen mit dem Namen des Themas sollen die Bestandteile der zu bauenden Tabelle ergeben.

Der Typ *Thema* wird bearbeitet, in der Ansicht wird der Reiter *Details* gewählt und hier der Punkt *View-Konfiguration -> Objekt -> Objektliste* angeklickt.



Noch wurde keine Tabellenkonfiguration mit diesem Typ verknüpft. Durch Klicken auf den Knopf *Neu*  wird eine neue, leere Konfiguration erzeugt. Diese kann dann selektiert und wie oben beschrieben bearbeitet werden.

Einstellungsmöglichkeiten für die Sortierung

Name	Wert
Absteigend sortieren	Steuert, ob auf- oder absteigend sortiert wird. Ist dieser Parameter nicht gesetzt, wird aufsteigend sortiert.

Primäres Sortierkriterium	Auswahlmöglichkeit für das Kriterium, nach dem die Eigenschaften sortiert werden: • *Position*: Die in der Konfiguration festgelegte Reihenfolge wird verwendet (Default). • *Wert*: Inhalt des Attributes bzw. Anzeigename des Relationszieles wird verwendet. • *Skript zur Sortierung*: Das in dem Attribut Skript zur Sortierung hinterlegte Skript wird zur Ermittlung des Sortierkriteriums verwendet.
Sekundäres Sortierkriterium	Sortierkriterium für Eigenschaften, die für das primäre Sortierkriterium den gleichen Wert haben. Einstellmöglichkeiten analog zum *Primären Sortierkriterium*
Skript zur Sortierung	Verweis auf ein registriertes Skript, das den Sortierschlüssel für das primäre bzw. sekundäre Sortierkriterium zurückgibt.

1.7.4.7.1 Spaltenkonfigurationen

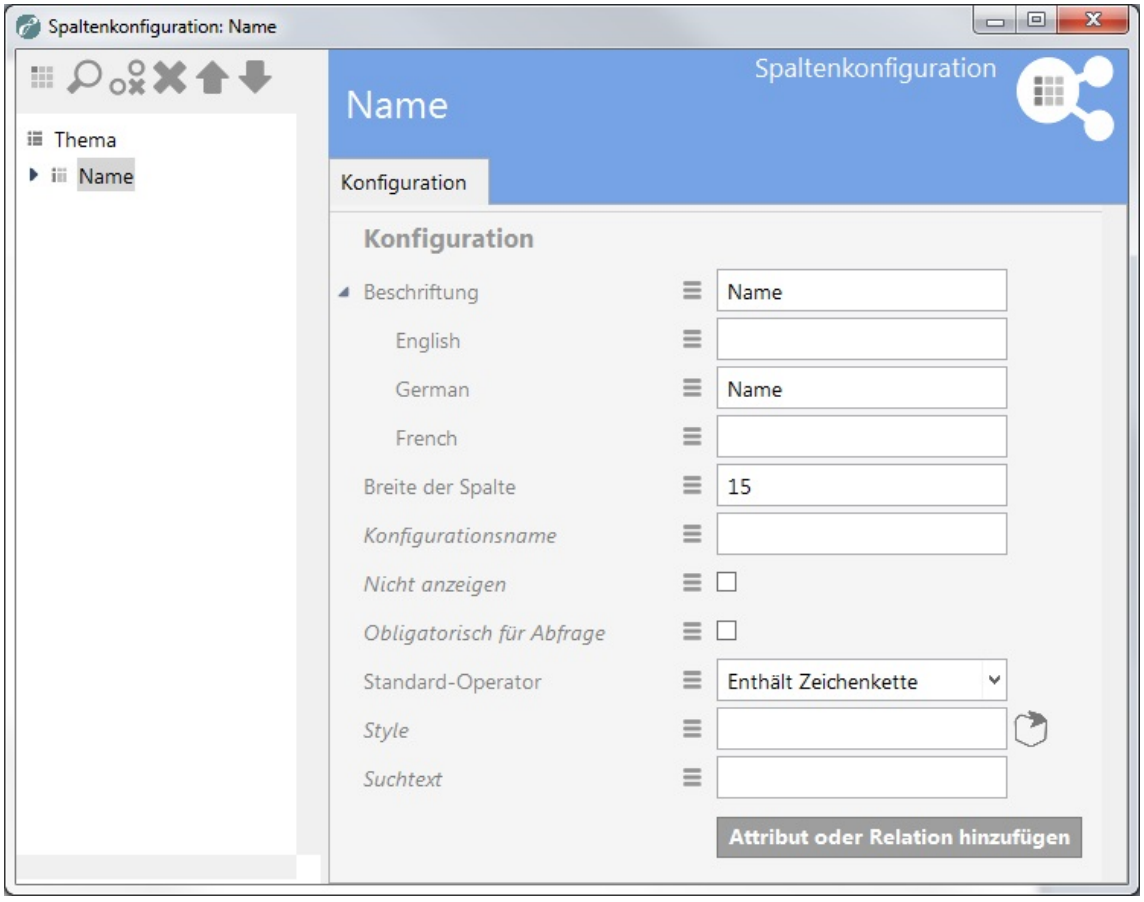
Wie bereits erwähnt, tragen *Spaltenkonfigurationen* Eigenschaften, die der Festlegung der Darstellung und des Verhaltens der Spalte in der Tabelle dienen. Erst wenn Eigenschaften an den in der Spaltenkonfiguration enthaltenen Spaltenelementen konfiguriert werden, wird die Spalte angezeigt.

Einstellungsmöglichkeiten

Name	Wert
Beschriftung	Wird in der Titelzeile der Spalte angezeigt. Hierbei ist zu beachten, dass *Beschriftung* der Anzeige in der Tabelle dient, die Spaltenkonfiguration aber zusätzlich noch das Attribut *Konfigurationsname* enthält. Dieser Name dient allein der Verwaltung und dem Auffinden der Konfiguration in der semantischen Graph-Datenbank und wird nicht angezeigt oder ausgegeben.
Breite der Spalte (%)	Für die Breite der Spalte wird hier ein prozentualer Wert erwartet (für 60% muss also "60" eingegeben werden).
Nicht anzeigen	Ist dieser Wert gesetzt, wird die komplette Spalte nicht angezeigt. Dies dient dazu z.B. eine Suche mit den Werten dieser Spalte durchzuführen. Der Anwender soll aber nicht die Möglichkeit haben Veränderungen an dieser Spalte vorzunehmen.
Obligatorisch für Abfrage	Ist dieser Wert gesetzt, muss die Spalte ausgefüllt sein, um suchen zu dürfen.

Sortierpriorität	Nach der Spalte mit Sortierpriorität 1 wird primär sortiert. Bei gleichen Werten wird nach der Spalte mit der nächst höheren Sortierpriorität sortiert usw. Die Standardsortierreihenfolge ist aufsteigend. Um absteigend zu sortieren negiert man die Sortierpriorität oder setzt das Meta-Attribut "Absteigend sortieren".
Standard-Operator	Operator, der initial bei der Suche für einen Suchtext angewendet wird.
Suchtext	Initial kann eine Spalte mit einem Suchtext versehen werden.

Beispiel



Spaltenkonfiguration für die Spalte Name

1.7.4.7.2 Spaltenelemente

Ein *Spaltenelement* dient der Zuweisung, welche Inhalte eine Tabellenspalte darstellen soll und wie dies zu geschehen hat. Es können entweder an den semantischen Objekten definierte Eigenschaften wie Attribute und Relationen spezifiziert oder Strukturabfrage-Bausteine oder Skript-Bausteine verwendet werden.

Einstellungsmöglichkeiten

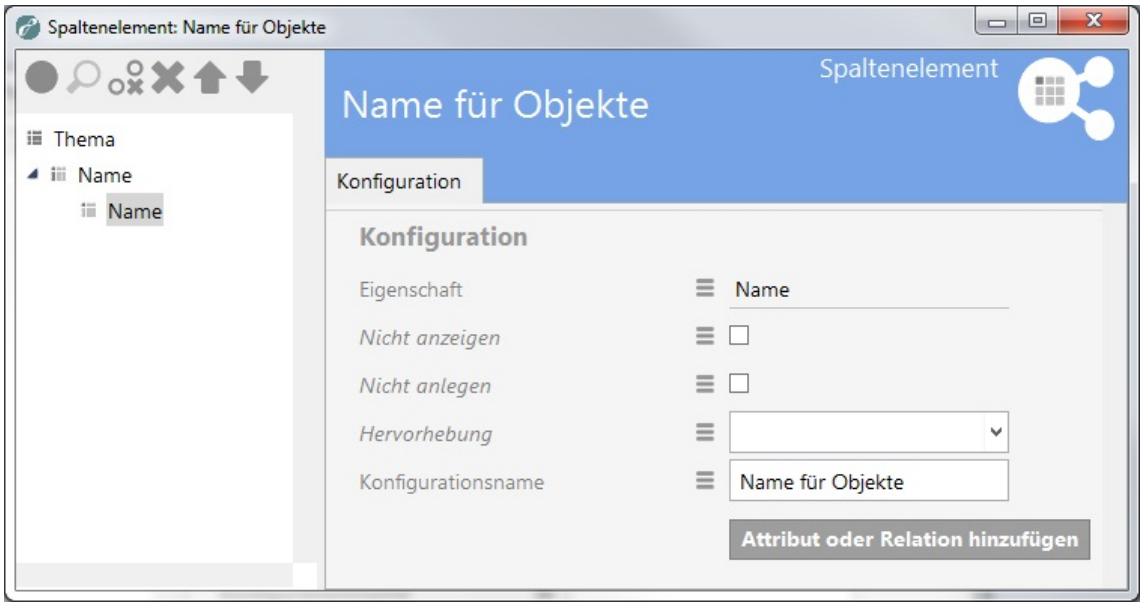
Name	Wert
Eigenschaft (obligatorisch oder Skript)	Verknüpfung zu einem Eigenschaftstyp, der angezeigt werden soll.
Skript (obligatorisch oder Eigenschaft)	Ausführen des Skripts *cellValues* zur Ermittlung der Werte, die angezeigt werden sollen.
Nicht anzeigen	Über dieses boolesche Attribut kann gesteuert werden, ob Werte der ausgewählten Eigenschaft angezeigt werden sollen. Standardmäßig werden alle Eigenschaften angezeigt.
Nicht anlegen	Dieses Attribut steuert, ob diese Eigenschaft beim Erzeugen eines neuen Objekts erzeugt werden soll, falls das entsprechende Eingabefeld der Spalte einen Wert enthält. Standardmäßig werden neue Eigenschaften erzeugt.
Nicht suchen	Hier kann eingestellt werden, dass die konfigurierte Eigenschaft nicht in die Suche übernommen wird. D.h. eingegebene Suchwerte werden nicht über diese Eigenschaft gesucht. Achtung: Wenn alle Spaltenelemente einer Spalte auf "Nicht suchen" geschaltet werden, hat dies denselben Effekt wie "Nicht anzeigen"!
Hervorhebung	Hier können für das Anzeigen von Werten Formatierungsvorgaben gemacht werden, zur Wahl steht derzeit nur *Unterstreichen*.
Hits verwenden	Standardmäßig werden Objekte erzeugt. Möchte man allerdings in dem Skript *cellValues* die Hits weiterverarbeiten, muss *Hits verwenden* eingeschaltet werden.
Relationszielansicht	Derzeit steht nur die Alternative *Drop down* zur Verfügung. Wird diese ausgewählt, so werden die möglichen Werte, die sich für die Filterung in der Tabelle für diese Spalte eintragen lassen, aus den möglichen Relationszielen gemäß Schema als Dropdown-Liste zusammengestellt, so dass ein möglicher Wert schnell spezifiziert werden kann. Dies empfiehlt sich für überschaubare Mengen an möglichen Relationszielen. Anmerkung: Dieser Parameter steht nur zur Verfügung, wenn eine Eigenschaft vom Typ Relation gewählt wurde.

Es ist möglich für eine Spaltenkonfiguration mehrere Spaltenelemente zu definieren. Das ist z.B. dann sinnvoll, wenn mehrere Attribute in der Suche berücksichtigt werden sollen wie beispielsweise das Attribut Name und Synonym, aber nur eines davon angezeigt werden soll.

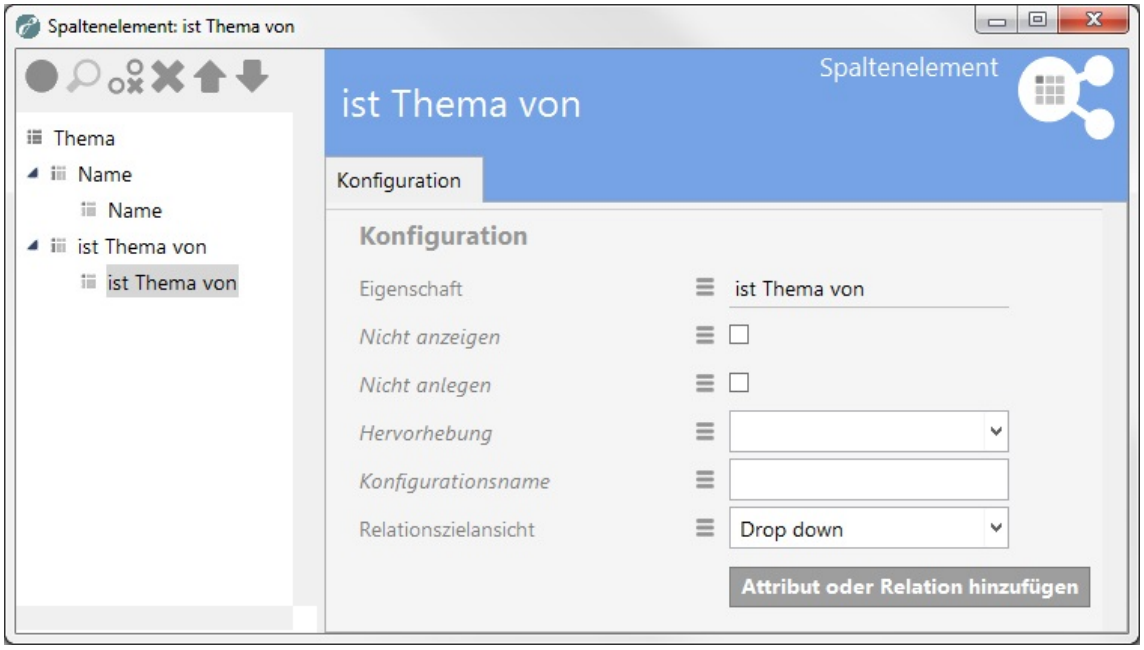
Beispiel

Im ersten Spaltenelement der Spaltenkonfiguration *Name* wurde das Attribut *Name* hinter-

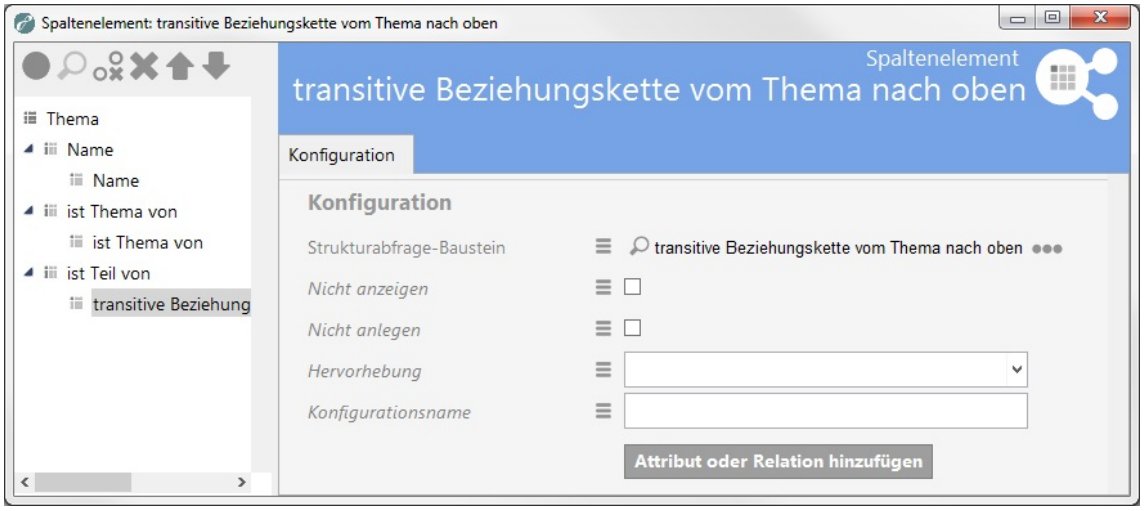
legt.



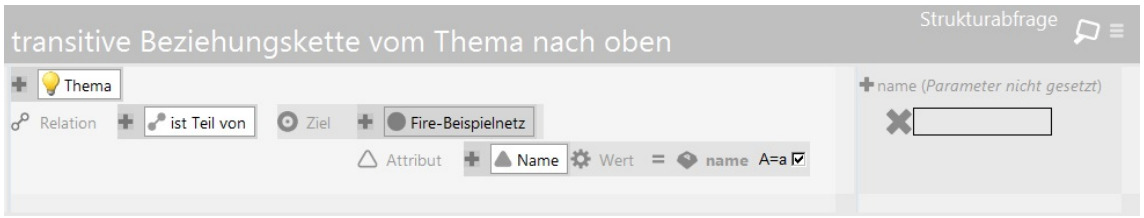
Auf der zweiten Spalte wurde im Spaltenelement die Beziehung *ist Thema von* hinterlegt.



Auf der dritten Spalte wurde im Spaltenelement der Strukturabfrage-Baustein *transitive Beziehungskette vom Thema nach oben* hinterlegt.



Hinterlegte Struktur-Abfrage



Um Werte aus dem Eingabefeld der Spalte übernehmen zu können, muss die hinterlegte Strukturabfrage konfigurierte Parameter haben. Es können mehrere Parameter angebracht werden, diese sind beim Auswerten der Strukturabfrage alle mit demselben Wert belegt.

Vorsicht: Hier liegt ein Unterschied zu sonstigen Fällen, in denen die Strukturabfrage verwendet wird. Normalerweise bestimmt das Ausgangsobjekt (in diesem Fall wäre dies "Thema") die Ergebnisse, hier sind es jedoch die Objekte oder Eigenschaften, an denen der Parameter angebracht ist (in diesem Fall das Namensattribut).

Der in der Spalte gezeigte Wert ist, wenn keine weiteren Anpassungen vorgenommen werden, der Wert des zum Filtern verwendeten Attributes. Wenn sich der angezeigte Wert nicht aus dem zur Filterung verwendeten Attribut ergibt, so gibt es zwei Möglichkeiten:

- der Bezeichner "*renderTarget*" kann an Relationszielen angebracht werden. Die hiermit markierten Objekte werden als Spaltenwert in der Tabelle angezeigt.
  "renderTarget" bewirkt außerdem, dass bei einer Ausgabe über die JavaScript API die Eigenschaften zur Darstellung als Link mit ausgegeben werden.
- der Bezeichner "*renderProperty*" kann an Attributen angebracht werden. Die hiermit markierte Eigenschaften werden als Spaltenwerte in der Tabellenspalte angezeigt.

Wird der Suchbaustein nicht zur Filterung verwendet, so muss das anzuzeigende Element mittels renderTarget oder renderProperty bestimmt werden!

Die Strukturabfragen, die in den Baustein des Spaltenelements eingefügt werden, können aus einer Liste bereits registrierter Strukturabfragen ausgewählt werden, sie können aber auch für genau diesen Baustein neu angelegt werden, was auch die Vergabe eines Registrierungsschlüssels mit sich bringt. Die Eigenschaft *Nicht anlegen* hat auf Spalten, die mit einem Strukturabfrage-Baustein belegt sind, keine Wirkung.

Auf die vierte Spalte wurde ein Script-Baustein abgebildet

Es sollen die Verantwortlichen für die Objekte, mit denen das in der Tabelle gelistete Thema mit *ist Thema von* verknüpft ist, angezeigt werden. Wie bei der Strukturabfrage kann das zugeordnete Skript aus einer Liste von bereits registrierten Skripten ausgewählt oder aber im Dialog neu angelegt (und registriert) werden. Der Skript-Editor öffnet sich nach Klicken auf den Skript-Baustein-Namen.

```
/*
 * Returns matching elements for colum search value "objectListArgument"
*  Note: "elements" may be undefined if no partial query result is available.
 * Return undefined if the script cannot provide any partial result itself.
 */
function filter(elements, queryParameters, objectListArgument) {
    return elements;
}

// Returns cell values for the given element
function cellValues(element, queryParameters) {
    var result = new Array();
    var firstTargets = element.relationTargets("istThemaVon") ;
    if ( firstTargets.length == 0 ) { return result ;
    }
    else {
        for (var i = 0; i < firstTargets.length; i++) {
        var secondTargets = firstTargets[i].relationTargets("hatVerantwortlichen");
        for (var j = 0; j < secondTargets.length; j++) {
            result.push(secondTargets[j].name());};
            };
        };
    return result.join(', ');
}
```

In diesem Fall ist die Sprache des Skriptbausteins JavaScript. Hier müssen zwei Teile gepflegt werden, der obere Teil dient der Filterung aller Elemente der Tabelle anhand des in der Spalte eingetragenen Wertes *objectListArgument*, der zweite Teil gibt an, wie für ein Element ein auszugebender Wert berechnet wird. Der erste Teil ist im Moment nicht ausgeführt. Zu bei-

den Teilen wird ein Code-Muster beim Erzeugen eingefügt, auf das beim Erstellen aufgebaut werden kann.

Wenn KScript als Sprache im Skript-Baustein gewählt wurde, um die Ausgabe einer Spalte zu steuern, dann muss das ausgewählte (registrierte) Skript zu jedem Objekt, das eine Zeile bildet, einen Rückgabewert für die Spalte liefern.

Da in KScript im Prinzip nur eine Ausgabe vorgesehen ist, wurde für die Filterung folgende Konvention getroffen:

Wenn es in dem ausgewählten Skript eine Funktion mit Namen *objectListScriptResults* und einem deklarierten Parameter gibt, so wird diese Funktion mit dem Argument der zugehörigen Sucheingabe aufgerufen, um die Menge der passenden Objekte zurückzuliefern. Die Funktion wird auf dem Wurzelbegriff oder der bisherigen Treffermenge als Ausgangsobjekt aufgerufen - je nach dem, wie die Suche am besten gelöst werden kann. Damit diese Variante wirklich effizient wird, ist es empfehlenswert, die Sucheingabe entsprechend auszuwerten und mit dem Ergebnis eine registrierte Strukturabfrage aufzurufen, um deren Ergebnis an die Objektliste weiterzuleiten.

1.7.4.8 Search

Mit dem View-Konfigurationselement "Suche" können dem Nutzer Suchmöglichkeiten für das Wissensnetz eingerichtet werden. Die Suche kann entweder eine vordefinierte Suche mit Parametern sein oder eine Suchfeld-Eingabemaske für den Nutzer.

Die "Suche" kann als Unterkonfiguration einer *Alternative* oder einer *Gruppe* ausgewählt werden. Eine beliebige Abfrage ist hier obligatorisch, deren Ergebnisse angezeigt werden. Auch Suchen zur Benutzereingabe können konfiguriert werden; für die View-Konfiguration wird hier anstatt dem Konfigurationslement "Suche" (Objektkonfiguration) das Konfigurationselement "Suchfeld-Ansicht" verwendet. Beispiele für die Panel-Konfiguration für das Web-Frontend finden sich hierfür in Kapitel 3 "ViewConfig-Mapper".

Wenn eine Suche mit Facetten konfiguriert werden soll, dann ist bei der Panel-Beeinflussung die Wirkungskette zu beachten:
Suchfeld-Ansicht –> Facette –> Suchergebnis.

Einstellungsmöglichkeiten

Name	Wert
Abfrage	Hier kann die Suche ausgewählt werden, die sofort bei der Anzeige des Konfigurationselements ausgeführt wird. Das semantische Objekt, für welches die View-Konfiguration angezeigt wird, kann als Zugriffselement in der Abfrage verwendet werden.
Beschriftung	Eine Beschriftung findet nur ihre Anwendung, wenn diese Konfiguration in einer anderen Konfiguration wie z.B. *Alternative* eingebettet ist.
Tabelle	Hier wird eine Tabellenkonfiguration angegeben, die zur Darstellung des Suchergebnisses dient.

Skript für Beschriftung	Die Beschriftung lässt sich alternativ auch über ein Skript ermitteln.
Skript für Sichtbarkeit	Über ein Skript lässt sich steuern, ob das Konfigurationselement angezeigt werden soll.
Skript für Tabellenkonfiguration	Die Tabelle kann auch über ein Skript ermittelt werden.

Einstellungsmöglichkeiten für eine Abfrage

Die folgenden Parameter werden als Meta-Eigenschaften für eine *Abfrage* gepflegt.

Name	Wert
Parametername	Angabe eines Parameternamens, wie er in der Abfrage verwendet wird.

Einstellungsmöglichkeiten für einen Parameternamen

Die folgenden Parameter werden als Meta-Eigenschaften für einen *Parameternamen* gepflegt:

Name	Wert
Skript	Das Skript *parameterValue* wird zur Ermittlung des Suchwertes für den angegebenen Parameternamen verwendet.
Wertermittlung	Angabe über den Weg der Wertermittlung *Skript*: Der Wert wird aus dem Skript ermittelt und darf nicht von einem Benutzer überschrieben werden. *Skript, überschreibbar*: Das Skript ermittelt Wert. Der Benutzer darf überschreiben. *Benutzereingabe*: Keine Skriptauswertung. Nur Eingabe durch einen Benutzer.
Typ	xsd-Typ
Beschriftung	Beim Herausschreiben nach JSON landet dieser Wert in *label.*

Darstellung in einer Anwendung

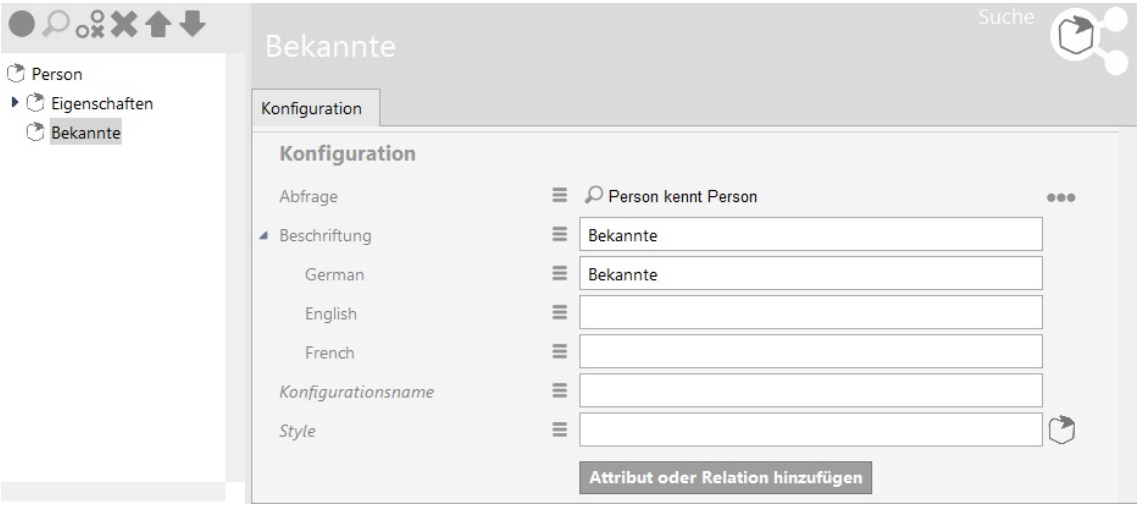
Eine Ausgabe von Suchergebnissen erfolgt standardmäßig in einer Tabelle.

In diesem Beispiel werden Suchergebnisse als Tabellen-View mit dem Style renderMode "mediaList" ausgegeben. Der renderMode "mediaList" wandelt die typische Tabellenansicht in eine ansehnliche Liste mit Icon und Link auf die Objekte um. Zusätzliche Eigenschaften des Objektes könenn angegeben werden (hier der Ort der Sehenswürdigkeit).

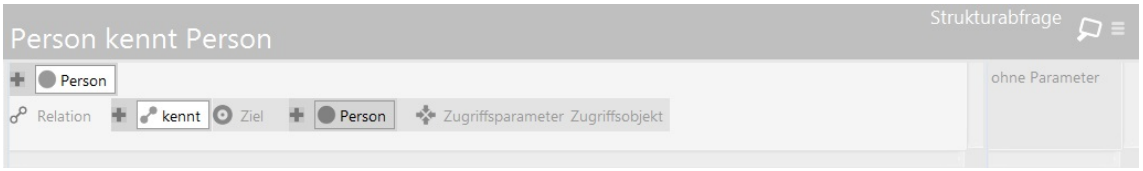
Darstellung im Knowledge-Builder

Die Ergebnisse einer beliebigen Abfrage werden im Knowledge-Builder stets in einer Objektliste angezeigt.

Beispiel:



In der View-Konfiguration werden die Reiter "Eigenschaften" und "Bekannte" definiert. "Bekannte" ist ein Konfigurationselement des Typs "Suche. Unter "Abfrage" kann eine Suche ausgewählt oder direkt neu angelegt werden.



Definition der Suche

Das Ergebnis der Abfrage wird im Reiter "Bekannte" bei Objekten des Typs "Person" im Knowledge-Builder angezeigt.

Statt eines einzelnen Konfigurationselements, lassen sich Suchen auch in mehrere seperate Konfigurationen aufteilen.

1.7.4.8.1 Search field view

Standarddarstellung eines Suchfeldes.

Dieses Konfigurationselement bietet eine separate Eingabemaske um Suchparameter für eine Abfrage eingeben zu können.

Name	Wert
Abfrage	Hier kann eine Abfrage festgelegt werden.
Parametername	Angabe eines Parameternamens, wie er in der Abfrage verwendet wird.
Beschriftung	Hier kann eine Beschriftung für die Elementkonfiguration angegeben werden.
Skript für Beschriftung	Alternativ lässt sich die Beschriftung auch über eine Skript ermitteln.

Einstellungsmöglichkeiten für Parameternamen

Die folgenden Parameter werden als Meta-Eigenschaften für einen *Parameternamen* gepflegt:

Name	Wert
Skript	Das Skript *parameterValue* wird zur Ermittlung des Suchwertes für den angegebenen Parameternamen verwendet.
Wertermittlung	Angabe über den Weg der Wertermittlung *Skript*: Der Wert wird aus dem Skript ermittelt und darf nicht von einem Benutzer überschrieben werden. *Skript, überschreibbar*: Das Skript ermittelt Wert. Der Benutzer darf überschreiben. *Benutzereingabe*: Keine Skriptauswertung. Nur Eingabe durch einen Benutzer.
Typ	Datentyp des Parameters.
Beschriftung	Beim Herausschreiben nach JSON landet dieser Wert in *label*.
Reihenfolge	Mehrere Parameter können hier eine Reihenfolge zugewiesen bekommen.

1.7.4.8.2 Search result view

Dieses Konfigurationselement beinhaltet eine separate Ergebnisansicht einer Suche.

Name	Wert
Abfrage	Hier kann eine Suchabfrage ausgewählt werden, die beim Anzeigen der Konfiguration ausgeführt wird.
Beschriftung	Eine Beschriftung findet nur ihre Anwendung, wenn diese Konfiguration in einer anderen Konfiguration wie z.B. *Alternative* eingebettet ist.
Skript für Beschriftung	Alternativ kann die Beschriftung kann auch über ein Skript ermittelt werden.
Tabelle	Hier wird eine Tabellenkonfiguration ausgewählt, die zur Darstellung der Suchergebnisse dient.
Skript für Tabellenkonfiguration	Alternativ kann die Tabellenkonfiguration über ein Skript generiert werden.

1.7.4.9 Graph

In einem Graph werden die Inhalte der semantischen Datenbank mit ihren Objekten und Verbindungen graphisch dargestellt (siehe *Knowledge-Builder* > *Grundlagen* > *Graph-Editor*).

Einstellungsmöglichkeiten

Name	Wert
Beschriftung	Eine Beschriftung wird nur ausgegeben, wenn diese Konfiguration in einer anderen Konfiguration wie z.B. *Alternative* eingebettet ist.
Skript für Beschriftung	Ein Skript, welches die Beschriftung zurückliefert.
Graph-Konfiguration	Hier wird ein Graph-Konfigurations-Objekt festgelegt.
Legende ausblenden	Legt fest ob die Legende zu den Knotentypen angezeigt werden soll.
Breite / Höhe	Legt die Breite und Höhe des Konfigurationselements, entweder prozentual oder pixelgenau, fest.
Skript für Sichtbarkeit	In einem hier referenzierten Skript lässt sich die Sichtbarkeit des Konfigurationselements festlegen.
Strukturabfrage für Startelemente	Eine Abfrage, welche die anzuzeigenden Objekte ermittelt.

1.7.4.9.1 Graph-Konfiguration

Die Graph-Konfiguration ermöglicht es nur bestimmte Typen und Relationen im Graphen anzuzeigen. So kann verhindert werden, dass unerwünschte Typen und Relationen im Graphen zu sehen sind. Die Graph-Konfiguration kann ebenfalls über JavaScript-Funktionen angefragt werden. Sie findet beispielsweise Verwendung im Net-Navigator.

Einer Graph-Konfiguration werden Knotenkategorie-Elemente untergeordnet.

Einstellungsmöglichkeiten

Name	Wert
Beschriftung	Eine Beschriftung findet nur ihre Anwendung, wenn diese Konfiguration in einer anderen Konfiguration wie z.B. einer weiteren *Alternative* eingebettet ist.
maximale Pfadlänge	.
Schritte bis Knotenausblendung	.

1.7.4.9.2 Knotenkategorie

Knotenkategorien werden Graph-Konfigurationen untergeordnet.

Ihnen werden Verknüpfung-Elemente untergeordnet.

Einstellungsmöglichkeiten

Name	Wert
Beschriftung	Eine Beschriftung findet nur ihre Anwendung, wenn diese Konfiguration in einer anderen Konfiguration wie z.B. einer weiteren *Alternative* eingebettet ist.
Erweiterungen initial anzeigen	.
Farbe	Farbgebung der Knoten dieser Kategorie
In Legende anzeigen	*Immer*, *Bei Bedarf* oder *Nie*
Knotengröße	.
Nur das Icon malen	.

1.7.4.9.3 Verknüpfung

Verknüpfungen sind **was genau?**

Sie werden einer Knotenkategorie untergeordnet.

Einstellungsmöglichkeiten

Name	Wert
Abfrage für Verknüpfung	.
Beschriftung	Eine Beschriftung findet nur ihre Anwendung, wenn diese Konfiguration in einer anderen Konfiguration wie z.B. einer weiteren *Alternative* eingebettet ist.
Bevorzugt ausklappen	.
Farbe	.
Initial ausgeklappt	.
Relation	.
Skript für Beschriftung	Hier lässt sich ein Skript referenzieren welches die Beschriftung ermittelt.
Skript für Verknüpfung	Über ein hier referenziertes Skript lässt sich die Verknüpfung festlegen.

1.7.4.10 Text

Dieses Konfigurationselement gibt einen einfachen Text aus. Dieser wird entweder fest konfiguriert oder über ein Skript ermittelt.

Einstellungsmöglichkeiten

Name	Wert
Beschriftung	Eine Beschriftung wird nur ausgegeben, wenn diese Konfiguration in einer anderen Konfiguration wie z.B. *Alternative* eingebettet ist.
Skript für Beschriftung	Ein Skript, welches die Beschriftung zurückliefert.
Text	Text, der ausgegeben werden soll.
Skript für Text	Ein Skript, welches den anzuzeigenden Text zurückliefert.

1.7.4.11 Image

Mit Hilfe dieses Konfigurationselements kann eine statische Grafik eingebunden werden.

Name	Wert
Beschriftung	Eine Beschriftung wird nur ausgegeben, wenn diese Konfiguration in einer anderen Konfiguration wie z.B. *Alternative* eingebettet ist.
Skript für Beschriftung	Alternativ kann hiermit die Beschriftung durch ein Skript ermittelt werden.
Bild	Die Bilddatei, welche ausgegeben werden soll.
Skript für Bild	Alternativ kann die Grafik durch ein Skript zurückgegeben werden.
Breite / Höhe	Skaliert die Bilddatei auf die angegebenen Maße.
Skript für Sichtbarkeit	Durch ein Skript lässt sich bestimmen ob die Grafik angezeigt werden soll.

1.7.4.12 Script generated view/HTML

Skriptgenerierte View

Eine mit Hilfe eines im Wissensnetz hinterlegten Skripts erstellte Sicht. Dieses ist in Javascript geschrieben und kann ein eigenes Template (ein Ractive.js "Partial") verwenden. Hierüber lassen sich komplexe Sichten erstellen, die über die Funktionalitäten der Standard-View-Konfigurationen hinaus gehen.

Einstellungsmöglichkeiten

Name	Wert
Beschriftung	Eine Beschriftung findet nur ihre Anwendung, wenn diese Konfiguration in einer anderen Konfiguration wie z.B. *Alternative* eingebettet ist.
Skript	Skript zur Generierung der View.
Skript für Sichtbarkeit	Skript zum Ermitteln der Sichtbarkeit.
Skript für Beschriftung	Skript zum ermitteln der Beschriftung.
viewType	Name des Partials.

Skriptgeneriertes HTML

Diese View-Konfiguration zeigt ein HTML-Fragment an, welches mit Hilfe eines im Wissensnetz hinterlegten Skripts generiert wird. Hierin wird über die Javascript-API von i-views auf Wissensnetz-Elemente und ihre Eigenschaften zugegriffen und über ein XML-Writer-Objekt eine HTML Struktur erzeugt und mit Daten befüllt.

Einstellungsmöglichkeiten

Name	Wert
Beschriftung	Eine Beschriftung findet nur ihre Anwendung, wenn diese Konfiguration in einer anderen Konfiguration wie z.B. *Alternative* eingebettet ist.
Skript	Skript zur Generierung einer HTML-Ausgabe.
Skript für Sichtbarkeit	Skript zum Ermitteln der Sichtbarkeit.
Skript für Beschriftung	Skript zum ermitteln der Beschriftung.

Beispiel für ein Skript, das eine einfache HTML-Ausgabe erzeugt:

```
function render(element, document) {
  var writer = document.xmlWriter();
  writer.startElement("div");
    writer.startElement("h2");
      writer.cdata(element.name());
    writer.endElement();
  writer.endElement();
}
```

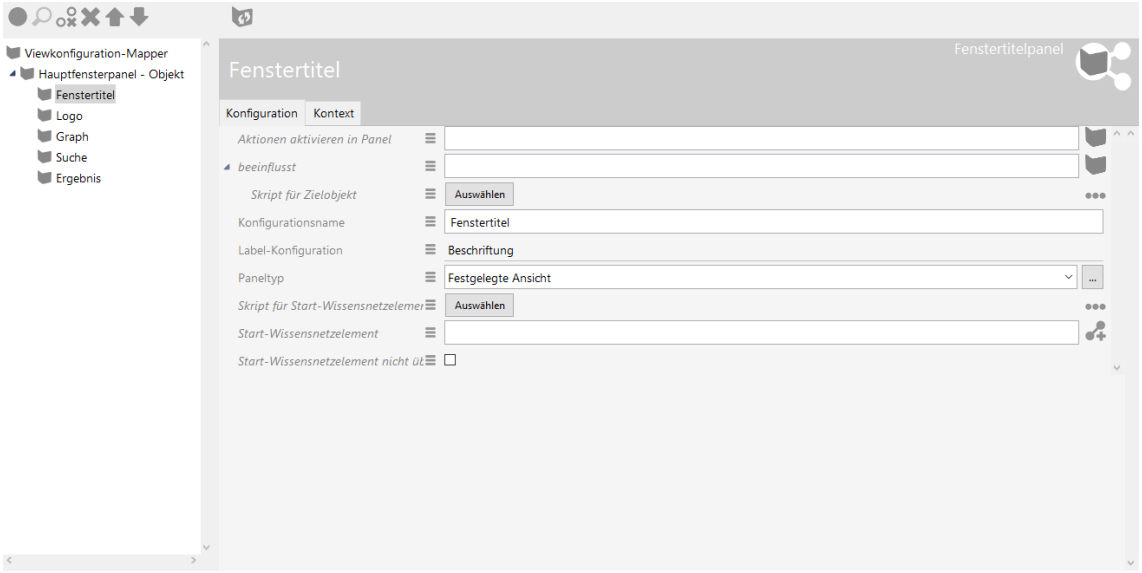
Ausgabe:

```
<div>
  <h2>Hermann</h2>
```

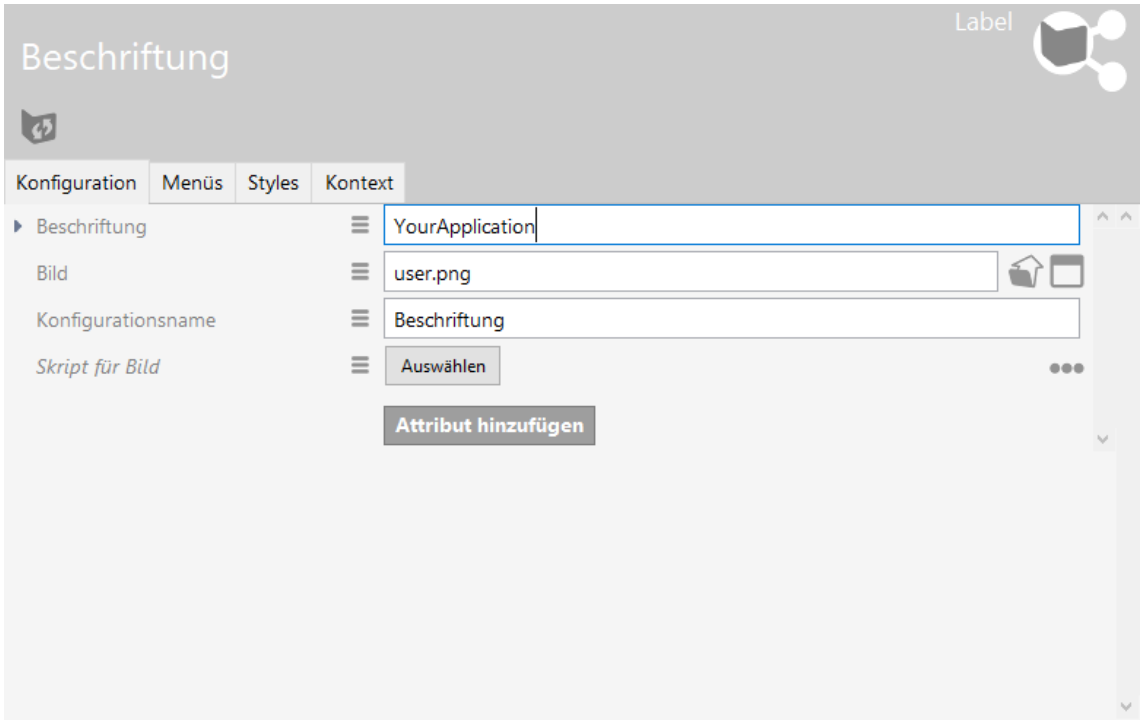
```
</div>
```

1.7.4.13 Label

Mit der Label-Konfiguration kann beispielsweise die Beschriftung einer Webseite oder die Beschriftung einer Tabelle konfiguriert werden. Im Knowledge-Builder werden die Label-Konfigurationen unter der Kategorie „Nachgeordnete Konfiguration“ verwaltet.
Label finden im Fenstertitel-Panel Verwendung; hierzu muss ein neues Objekt unter „Label-Konfiguration“ angelegt werden:



Unter „Beschriftung“ und „Bild“ können dann die Einträge vorgenommen werden, die sich später auf dem Browser-Tab der Webseite wiederfinden:

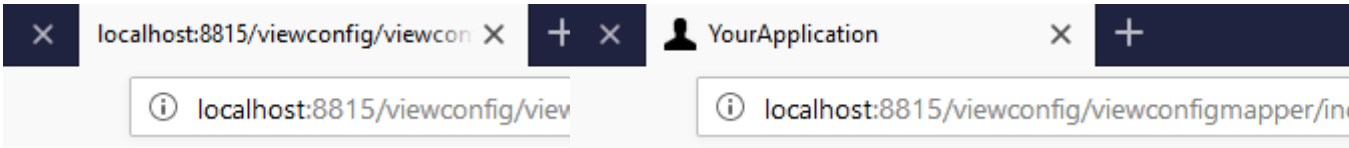


Hinweis: Im Knowledge-Builder wird das View-Konfigurationselement ("Label") standardgemäß mit "Label - Objekt" betitelt. Wenn unter "Beschriftung" eine Zeichenkette eingetragen wird, so erscheint diese als Elementname des View-Konfigurationselements. Wenn ein Konfigurationsname ("Beschriftung") vergeben wird, so erscheint dieser als Elementname. Konfigurationsnamen sollten generell vergeben werden, damit Konfigurationselemente wiedergefunden und besser voneinander unterschieden werden können.

Bezüglich des Web-Frontends, welches durch den Viewkonfiguration-Mapper erzeugt wird, entspricht dies dem Element <title> in der Sektion <head>:

```
<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>
```

Ein Vergleich zeigt die unterschiedlichen Zustände der Webseite ohne Label (Titel = Pfad der Webseite) oder mit Label (Titel = Beschriftung):



1.7.5 Panels

Panels sind Konfigurationselemente, welche die Anwendungsoberfläche in Bereiche aufteilen. Mit ihnen wird das grundsätzliche Layout einer Anwendung aufgebaut.

Panels beinhalten weitere Panels oder View-Konfigurationen und können ineinander verschachtelt werden. Sie können sich gegenseitig beeinflussen.

Panels erhalten immer genau ein Startelement (ein Objekt oder einen Typ) bei ihrer Aktivierung, welches sie an ihre Unterkonfigurationen weiter geben.

Panels selbst haben ansonsten keinerlei Funktion. Diese werden erst mit Hilfe von Aktionen und View-Konfigurationen festgelegt.

Es gibt verschiedene Arten von Panels:

- Hauptfensterpanel
- Dialogpanel
- Fenstertitelpanel
- Fußzeilenpanel
- Normale Panel

Für jede Anwendung muss genau ein sogenanntes *Hauptfensterpanel* existieren, welches durch untergeordnete Panels aufgeteilt werden kann. Zusätzlich kann ihm ein *Fenstertitelpanel* zugeordnet werden, welches den Titel und das Logo (*Favicon*) der Anwendung festlegt.

Weiterhin können einer Anwendung weitere *Dialogpanel* zugewiesen werden, die als Pop-Up über dem Hauptfenster angezeigt werden können. Diese können neben weiteren Panels auch Fenstertitel- und Fußzeilenpanel enthalten.

Für jedes Panel muss ein bestimmter Paneltyp ausgewählt werden.

- Layout-Panels (enthalten weitere Panels):
  - Lineares Layout (alle untergeordnete Panels werden horizontal oder vertikal angeordnet dargestellt)
  - Wechselndes Layout (nur eins der untergeordneten Panels wird zur gleichen Zeit angezeigt)
- Ansicht-Panels (enhalten View-Konfiguration(en)):
  - Festgelegte Ansicht (enthält ein einziges festgelegtes Konfigurationselement)
  - Flexible Ansicht (mehrere Ansichten je nach Typ des Startelements möglich)

Einstellungsmöglichkeiten

Name	Wert
Aktionsergebnisse anzeigen in Panel	Alle Aktionen, die im Quell-Panel aktiviert werden, führen dazu, dass das Ziel-Panel mit dem jeweilig übergebenen Objekt angezeigt werden (Beispiel: Jeder Klick im Panel Objektliste führt dazu, dass im Panel Detailansicht das Ergebnis angezeigt wird).
beeinflusst	Hier kann ein Panel festgelegt werden, dass vom aktuellen Panel beeinflusst wird (Beispiel: Je nachdem welche Objekte bei Suchergebnis angezeigt werden, beeinflusst das welche Facetten dazu angezeigt werden).

Skript für Zielobjekt	Mithilfe von Skripten können hier nicht einfach nur Panels, sondern auch Bedingungen angegeben werden unter denen bestimmte Panels durch das aktuelle Panel beeinflusst werden.

Einstellungsmöglichkeiten Layout

Name	Wert
class	CSS-Klassen für das Panel (wird nur für Web-Anwendungen bzw. im ViewConfig-Mapper berücksichtigt)
Breite / Höhe	Die exakten Maße des Panels können hier jeweils entweder prozentual oder pixelgenau gesetzt werden.
Maximale Breite / Höhe	Alternativ lassen sich hier die Höchstmaße des Panels angeben. Das Panel nimmt sich so viel Platz wie es benötigt, ohne diese Werte zu überschreiten.
Flex-grow / -shrink	Hier lassen sich die Werte für die jeweilige CSS-Eigenschaft für den Wachstums- bzw. Schrumpffaktor des Panels angeben. Ein Element mit einem Wert von 2 für flex-grow zum Beispiel, erhält doppelt so viel Platz wie ein Element mit Wert 1.
overflow-x / -y (Scrollbar)	Hierüber lässt sich die Darstellung von Scrollbars in der Applikation festlegen, wenn der Inhalt des Panels nicht in seine horizontale (x) und vertikale (y) Abmessungen passt. Zur Auswahl stehen *auto*, *scroll* und *hidden*.
Style	CSS-Styling-Regeln für das Panel (wird nur in Web-Anwendungen bzw. im ViewConfig-Mapper berücksichtigt)

1.7.5.1 Aktivierung von Panels

Panels kennen zwei grundsätzliche Zustände: "aktiv" und "inaktiv". Ein Panel ist sichtbar, wenn es aktiv ist.

Die Aktivierung von Panels funktioniert über folgende Mechanismen:

A. Zum Start einer Anwendung ist immer das Haupfenster-Panel der Anwendung aktiv

B. Beim Ausführen einer Aktion bestimmt der Ausführungsort, welches Panel aktiv wird

Ausgehend von A/B gibt es Folge-Aktivierungen nach diesen Regeln:

1. Beeinflusste Panels werden aktiviert
2. Panels mit einer spezialisierten Funktion (z.B. Fenstertitel) werden aktiviert - und zwar von allen Panels in der entsprechenden Hierarchie aus
3. Unterpanels werden aktiviert

4. Im Falle eines Panels mit wechselndem Layout: Geschwister-Panels des aktiven Unterpanels werden deaktiviert
5. Fortfahren bei 1. bis keine weiteren Panels mehr aktiviert werden können (eine eingebaute Zyklenprüfung verhindert Endlosschleifen)

Folge-Aktivierungen transportieren jeweils das angezeigte Modell. Wenn also beispielsweise Panel A das Objekt "Herr Meier" anzeigt, dann zeigt das aktivierte Unterpanel B ebenso "Herr Meier" an.

Zuletzt wird sichergestellt, dass alle Panels oberhalb der aktivierten Panels ebenso aktiv sind. Dabei wird deren Inhalt aber nicht neu berechnet.

Fortgeschrittene Aktivierungsmechanismen (ab Version 5.2):

In Schritt A (Aktionsaktivierung) sowie in Schritt 1 (Beeinflussung) kann über den sogenannten "Aktivierungsmodus" die Berechnung der Panel-Inhalte optimiert werden.

Auf diese Weise kann vermieden werden, dass Panel-Inhalte neu berechnet werden, die aktuell nicht angezeigt werden, weil sie trotz Aktivierung nicht im Sichtbarkeitsbereich liegen (z.B. ein Warenkorb). Für diesen Fall gibt es die Option "Lazy".

Analog dazu kann mit der Option "Aktualisierung" eine Auslösung der Aktivierungskette vermieden werden. Hier wird nur der Inhalt des Panels neu berechnet.

Die Option "Anzeige" ist der Standard, wenn keine der beiden obigen Optionen gewählt wurde.

1.7.5.2 Layout-Panels

Mit Layout-Panels wird die Anwendung in verschiedene Bereiche unterteilt. Lineare Layouts ordnen untergeordnete Panels entweder nebeneinander oder untereinander an. Wechselnde Layouts erlauben alternative Darstellungen auf der gleichen Visualisierungsfläche, bei denen nur eines der untergeordneten Panels gleichzeitig angezeigt wird.

Einstellungsmöglichkeiten Konfiguration

Name	Wert
Standardmäßig erstes aktivieren (nur bei *wechselndes Layout*)	Wird hier der Haken gesetzt, heißt das, dass das erste untergeordnete Panel standardmäßig aktiviert ist (im Beispiel unten ist das die Startseite)

1.7.5.3 View-Panels

Ansicht-Panels dienen als Container für einzelne Ansichten. Sie können dafür keine weiteren Panels enthalten.

Einstellungsmöglichkeiten

Name	Wert

Kontextelement	Hier kann ein konkretes Objekt oder ein konkreter Typ angegeben werden, der als Ausgangselement dient, von dem aus weitere Wege durch das Netz gegangen werden können.
Nicht überschreibbar durch äußeres Kontextelement	Wenn diese Option aktiviert ist, wird immer das konfigurierte Kontextelement verwendet. Die Beeinflussung durch andere Panels hat dann keine Auswirkung. Falls kein Kontextelement konfugiert ist, bleibt das Kontextelement leer.
Skript für Kontextelement	Das Script bestimmt das Startelement. Als Argument wird das äußere Kontextelement übergeben. Die Option "Nicht überschreibbar durch äußeres Kontextelement" hat keinen Einfluss, das Skript wird immer ausgeführt.
Sub-Konfiguration (nur bei *festgelegte Ansicht*)	Hier kann die eine View-Konfiguration angegeben werden, die für die festgelegte Ansicht genutzt wird.

1.7.5.4 Dialog-Panels

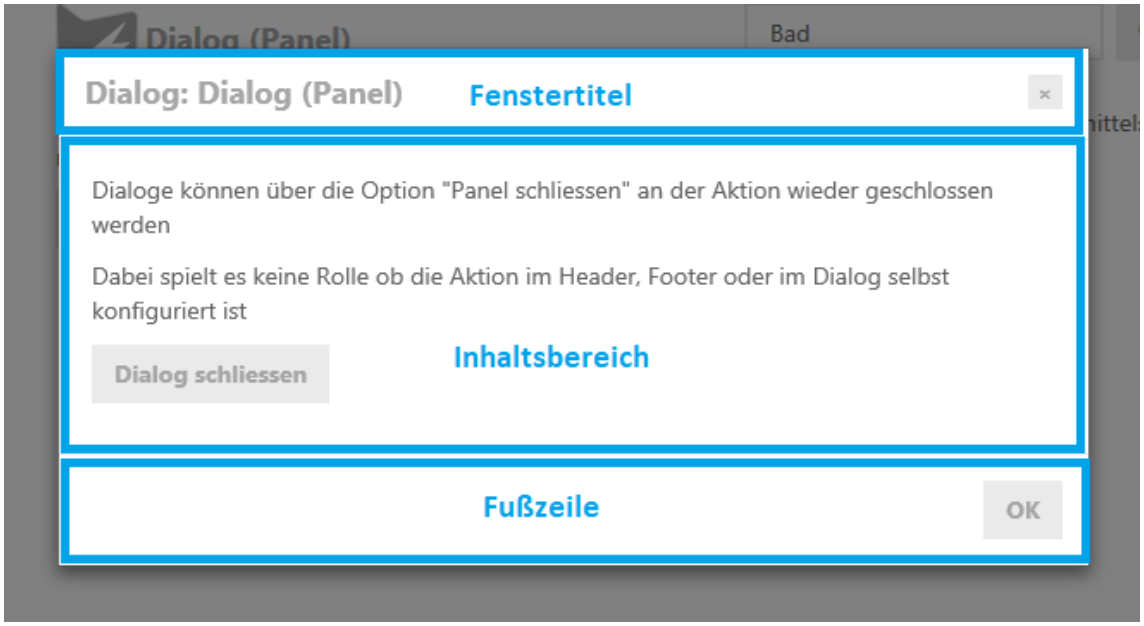
Dialog-Panels sind spezielle Anzeigebereiche, deren Inhalte in einem Dialogfenster angezeigt werden. Die Dialogfenster werden automatisch sichtbar, wenn das entsprechende Dialog-Panel aktiviert wird. Die Aktivierung kann so wie bei anderen Panels auch gezielt über bestimmte Aktionen erfolgen (siehe Relation "Ergebnis anzeigen in Panel" in Aktionskonfigurationen) oder generell bei Aktivierung bzw. Aktualisierung anderer Panels (siehe Relationen "Aktionen aktivieren in Panel" und "beeinflusst" in anderen Panel-Konfigurationen).

Zum Ausblenden ("Schließen") von Dialogfenstern müssen ebenfalls Aktionen verwendet werden. Ist in einer Aktionskonfiguration das Attribut "Panel schließen" angehakt, so führt die Ausführung dieser Aktion in einem Dialogfenster dazu, dass das Fenster ausgeblendet wird. Die Aktion muss dau also mit einem Menü verknüpft sein, welches im Dialog-Panel selbst oder einem seiner untergeordneten Panels angezeigt wird.

Dialogfenster werden inhaltlich in die folgenden drei Bereiche unterteilt:

- Fenstertitel
- Inhaltsbereich
- Fußzeile

Die Inhalte und das Layout innerhalb der drei Bereiche können jeweils über eine eigene Panel-Konfiguration festgelegt werden. Das Dialog-Panel selbst steht dabei stellvertretend für den Inhaltsbereich. Zur Konfiguration von Fenstertitel und Fußzeile muss am Dialog-Panel eine Unterkonfiguration vom Typ Fenstertitel- oder Fußzeilen-Panel angelegt werden (siehe Beispiel unten).



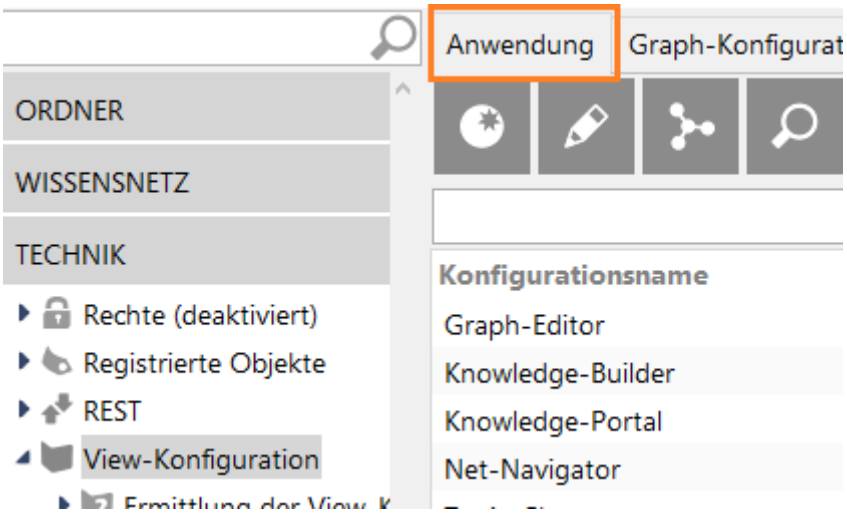
Über das Attribut "Paneltyp" am Dialog-Panel selbst sowie an dessen Fenstertitel- und Fußzeilen-Panels kann bestimmt werden, ob das jeweilige Panel Layout- oder Ansichtsfunktionalitäten bereitstellt. Details zu den verschiedenen Paneltypen sind in den vorangehenden Kapiteln beschrieben.

Dialog-Panels können im Knowledge-Builder folgendermaßen angelegt werden:

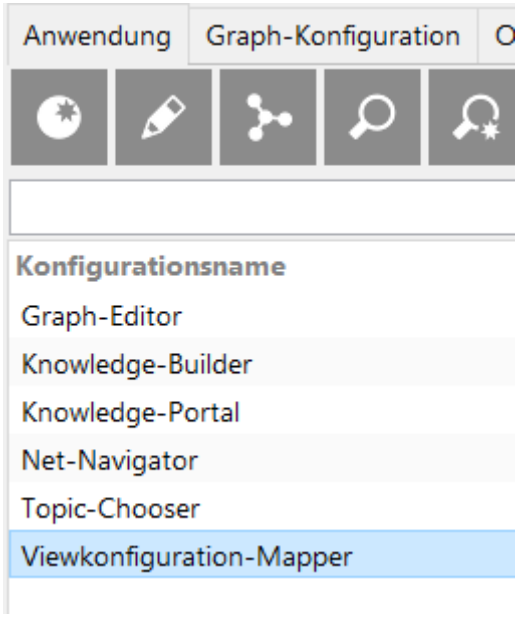
1. Melden Sie sich mit einem Benutzerkonto im Knowledge-Builder an, das über Administrator-Berechtigungen verfügt

2. Öffnen Sie im Navigationsbereich auf der linken Seite die Rubrik "Technik" und wählen Sie den Unterpunkt "View-Konfiguration" aus.

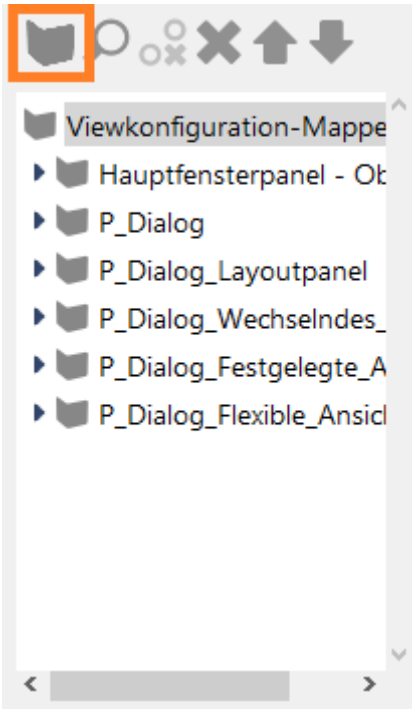
3. Wählen Sie den Reiter "Anwendung" auf der rechten Seite aus.



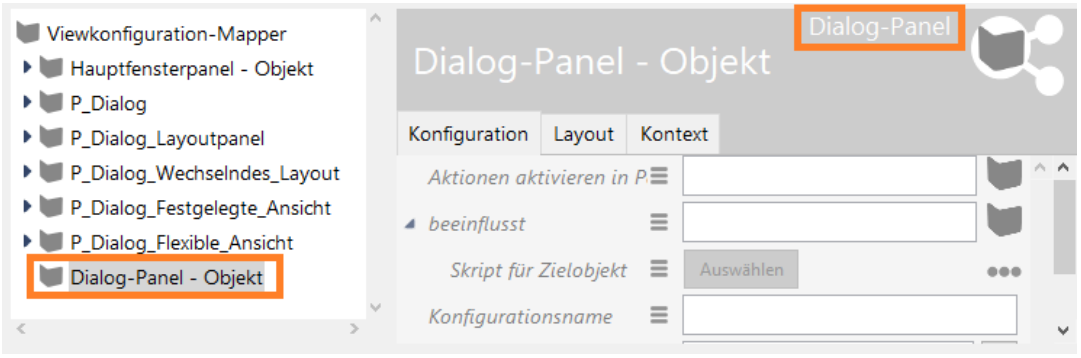
4. Wählen Sie in der Liste darunter die Anwendung, zu der Sie das Dialog-Panel hinzufügen möchten (Normalerweise "Viewkonfiguration-Mapper").



5. Wählen Sie das oberste Element im Panel-Baum unten aus und Klicken Sie auf das Anlegen-Symbol



6. Das neu angelegte Dialog-Panel wird im Panel-Baum automatisch ausgewählt und die Detailansicht rechts nebem dem Panel-Baum angezeigt



Zum Anlegen eines Fenstertitel- oder Fußzeilen-Panels muss das Dialog-Panel im Panel-Baum ausgewählt und das Symbol zum Anlegen von Unterkonfigurationen  angeklickt werden. Es erscheint daraufhin ein Auswahlfenster, in dem der Eintrag "Fenstertitel" oder "Fußzeile" ausgewählt werden kann. Je nach Paneltyp des Dialog-Panels können auf diesem Weg auch noch andere Unterelemente angelegt werden, die sich dann jedoch auf den Inhaltsbereich des Dialogfensters beziehen.

1.7.6 Configuration of the Knowledge-Builder

Die hier beschriebenen View-Konfigurationen betreffen ausschließlich den Knowledge-Builder. Weitere View-Konfigurationen, die den Knowledge-Builder betreffen, finden sich auch an anderen Stellen in Kapitel 7, können dann aber auch zusätzlich jeweils die Ausgabe in JSON betreffen.

1.7.6.1 Folder structure

Der linke Teil des Hauptfensters im Knowledge-Builder dient der Navigation durch das semantische Modell. Dazu wird dort eine hierarchische Ordnerstruktur angezeigt. Diese lässt sich in mehrere Hauptbereiche gliedern, die dann als Balken angezeigt werden. Klickt man einen solchen Balken an, dann klappt die darunter liegende Ordnerstruktur auf, über die man Inhalte (Elemente, Abfragen, Import/Export-Abbildungen usw.) erreichen kann. Die Inhalte werden auf der rechten Seite aufgelistet und können dort bearbeitet werden.

1.7.6.1.1 The Standard Folder Structure

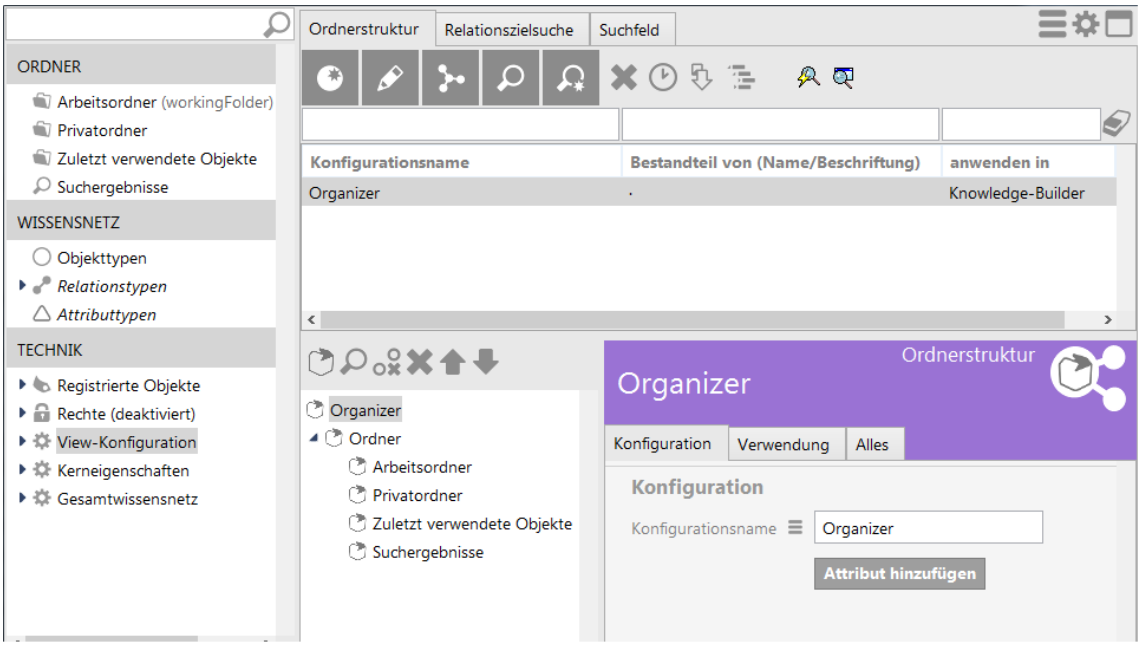
Die Konfiguration der Standard-Ordnerstruktur stellt Ordner zur Verfügung, so dass im semantischen Modell navigiert und Inhalte abgelegt werden können. Für Administratoren werden drei Hauptbereiche zur Verfügung gestellt.

Der obere Hauptbereich **"Ordner"** stellt Ordner für die Anlage weiterer Ordner und das Verwalten von Inhalten zur Verfügung. Das sind der Arbeitsordner, der Privatordner, der Ordner "Zuletzt verwendete Objekte" und der Ordner "Suchergebnisse".

Der zweite Hauptbereich **"Wissensnetz"** ermöglicht die Navigation zu den Elementen über die Hierarchie der Typen. Die hier zu erreichenden Elemente sind Typen, Objekte und auch Attribute und Relationen. Dafür enthält der Bereich drei Ordner:

- Objekttypen für die Hierarchie der Objekttypen und ihrer konkreten Objekte
- Relationstypen für die Hierarchie der Relationen
- Attributtypen für die Hierarchie der Attribute

Der dritte Hauptbereich **"Technik"** ermöglicht es Administratoren Änderungen, Einstellungen und Konfigurationen verschiedenster Art im sem. Netz vorzunehmen. Dazu gehören u.a. Registrierte Objekt, das Rechtesystem und Trigger.



Die Konfiguration dieser Standard-Ordnerstruktur kann im Technik-Bereich >> View-Konfiguration überprüft, verändert und den Bedürfnissen der Anwender angepasst werden.

Anmerkung: Für Administratoren wird immer die Standard-Ordnerstruktur angezeigt. Konfiguriert man eine View-Konfiguration für Ordner, so werden diese nur für Nicht-Administratoren angezeigt. Möchte man als Administrator auch die konfigurierte Sicht der Ordnerstruktur angezeigt bekommen, so kann das in den persönlichen Einstellungen des Knowledge-Builder auswählt werden: Unter "Einstellungen" > "Persönlich" > "View-Konfiguration" die Auswahl "Konfiguriert" wählen.

1.7.6.1.2 Configuration Of The Folder Structure

Die Ordnerstruktur wird im Technik-Bereich unter *View-Konfiguration* >> *Objekttypen* >> *Knowledge-Builder-Konfiguration* >> *Ordnerstruktur* konfiguriert. Einen schnellen Zugriff auf die Konfigurationen erhält der Admin, wenn er im Technik-Ast den Knoten *View-Konfiguration* selektiert und im rechten Teilfenster auf dem Reiter *Ordnerstruktur* das Objekt *Organizer* auswählt.

In der Konfiguration werden Ordnerstrukturelemente hierarchisch miteinander verknüpft. Der Wurzelknoten dieser Hierarchie ist ein Objekt des Typs *Ordnerstruktur*. Initial ist eine Ordnerstruktur mit Namen *Organizer* enthalten. Alle Unterknoten und deren Unterknoten sind vom Typ *Ordnerstrukturelemente*. Die Hierarchie in der Konfiguration zeigt direkt die im Hauptfenster dargestellte Hierarchie. Die direkten Unterknoten des Wurzelknotens werden im Hauptfenster als Balken dargestellt, so dass sich eine optische Abgrenzung der verschiedenen Ordnerhierarchien voneinander ergibt.

Beschriftung ist ein Parameter, den alle Konfigurationstypen gemein haben. Ein Knoten, der durch eine Konfiguration beschrieben wird, wird mit diesem Wert beschriftet. Was im rechten Teil des Hauptfensters angezeigt wird, wenn man einen Knoten selektiert, hängt von den Parametern des Ordnerstrukturelements ab. Dazu muss der Parameter **Ordnertyp** belegt werden, für den eine Auswahl an Typen zur Verfügung steht. Diese Ordnertypen und deren zusätzliche Parameter werden in der folgenden Tabelle aufgeführt.

Ordnertyp (obligatorisch)	**Parameter**	**Beschreibung**
Attributtypen	Typ	Der angegebene Attributtyp und alle seine Untertypen werden in einem hierarchischen Baum angezeigt.
Privatordner	-	Anzeige des Ordners, den nur der Benutzer selbst sehen darf und der für jeden Benutzer unterschiedlich ist.
Relationstypen	Typ	Der angegebene Attributtyp und alle seine Untertypen werden in einem hierarchischen Baum angezeigt.
Strukturordner	Strukturordner	Ein beliebiger *Strukturordner* kann hier eingehängt werden.
Suchergebnisordner	-	Jeder Benutzer hat einen eigenen Suchergebnisordner, der die letzten Suchergebnisse des Benutzers speichert.

Typbasierte Ordnerstruktur	Ansicht "Ohne Vererbung", Typ	Der angegebene *Typ* und seine Untertypen werden tabellarisch aufgelistet. Ist der Parameter *Ansicht "Ohne Vererbung"* gesetzt, wird nur der angegebene Typ angezeigt. Anmerkung: Zur Steuerung welche Tabellenkonfigurationen auf der rechten Seite Anwendung finden, muss dort die Relation *anwenden in* mit diesem *Ordnerstrukturelement* verknüpft werden.
Virtueller Ordner	-	Ein Ordner, der zur Strukturierung der Ordner dient.
Zuletzt verwendete Objekte	-	Jeder Benutzer hat einen eigenen Ordner, in dem die zuletzt verwendeten Objekte für einen schnelleren Zugriff gespeichert werden.

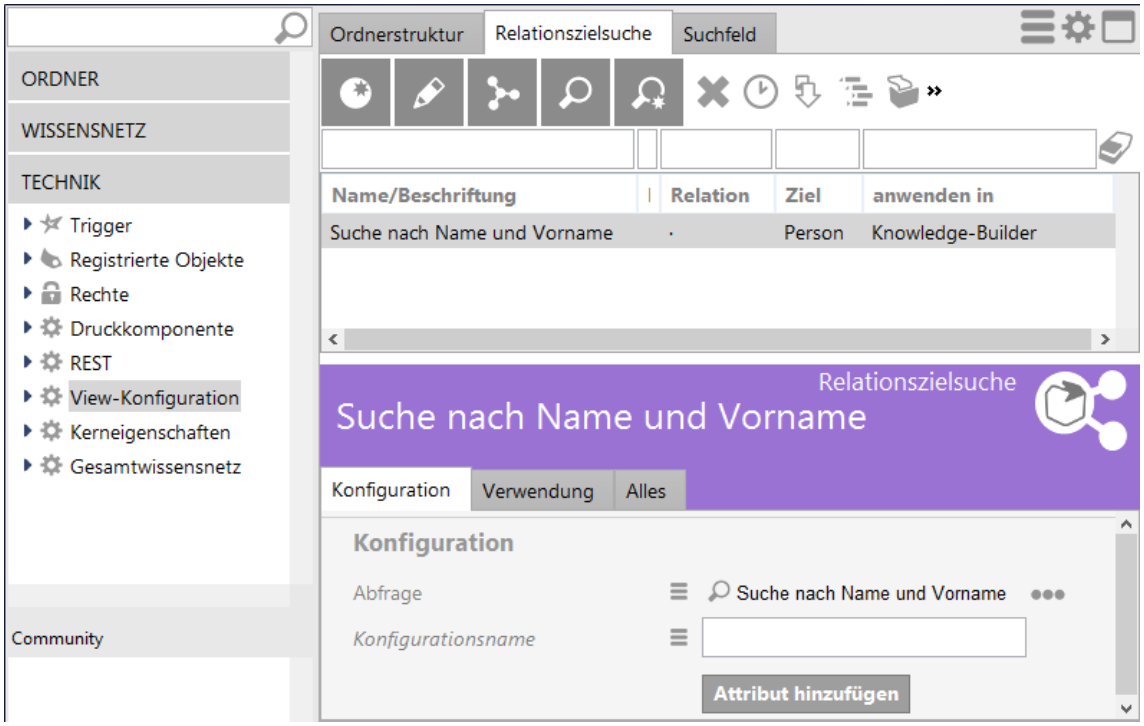
Nur der Konfigurationstyp *Virtueller Ordner* kann weitere Unterkonfigurationen enthalten bzw. nur beim ihm machen Unterkonfigurationen Sinn.

Anmerkung: Bei dem Ordnertyp Attributtypen, Relationstypen und Typenbasierte Ordnerstruktur dient der Parameter Typ zur Angebe des Attribut-, Relations- oder Objekttyp der und dessen Untertypen in dem Ordner angezeigt werden sollen.

1.7.6.2 Relation target search

Die Konfiguration der Relationszielsuche ermöglicht es, auf die Strategie einzuwirken, mit der mögliche Relationsziele gesucht werden. Enthält ein semantisches Modell keine Relationszielsuche, dann wird auf eine Eingabe von "Egon" immer nach einem Objekt mit Namen "Egon" gesucht (d.h. das jeweilig definierte Namensattribut wird verwendet). Durch Angabe einer zuvor definierten Abfrage kann dieses Verhalten verändert werden. Beispielsweise könnte man für die Suche nach Personen eine Abfrage definieren, die sowohl den Vornamen als auch den Nachnamen durchsucht. Für gewöhnlich werden zu diesem Zweck gewöhnliche Abfragen und nicht etwa Strukturabfragen verwendet.

Sucht man dann nach einem Ziel für eine Relation, deren Zieldomäne Person ist, dann werden die Nachnamen und Vornamen von Personen nach der Eingabe von "Egon" durchsucht. Sinnvoll ist eine angepasste Relationszielsuche auch, wenn man gleichzeitig Namen und Synonyme von Objekten durchsuchen möchte, sodass beispielsweise das Objekt "Architektur" auch gefunden wird, wenn der Anwender "Baukunst" eingibt.



Relationszielsuche konfiguriert für die Suche nach Personen

Wie bei allen Konfigurationen muss der Kontext angegeben werden, in dem die Relationszielsuche verwendet werden soll. Hierzu muss bei "anwenden auf Relation" die Relation eingetragen werden, bei der die Relationszielsuche angewendet werden soll. Die Eigenschaften "anwenden auf Ziel" und "anwenden in" werden ab Version 5.2 nicht mehr berücksichtigt.

1.7.6.3 Start view

Mit der Konfiguration *Startansicht (KB)* (zu finden als Reiter im View-Konfiguration-Bereich) lässt sich definieren, welches Hintergrundbild und welche Aktionen auf dem Startbildschirm im Knowledge-Builder auf der rechten Seite angezeigt werden sollen. Die Anzeige lässt sich jederzeit durch Deselektion (Klick auf bestehende Selektion im linken Navigationsbaum) hervorrufen.

Einstellungsmöglichkeiten

Name	Wert
Hintergrundbild	Ein Bild
Farbwert für Schriftart einer Aktion	Je nach ausgewähltem Bild muss eine andere Farbe für die Beschriftung der Aktionen gewählt werden, um den Text lesen zu können.

Darüber hinaus lassen sich Aktionen definieren. Siehe dazu Kapitel Aktion. Zusätzlich kann eine Aktionsart festgelegt werden. Hier stehen folgende Einträge zur Verfügung:

Aktionsart	Aktion
Handbuch (spezialisierter Web-Link)	Web-Handbuch wird im Browser geöffnet
Homepage (spezialisierter Web-Link)	Die Homepage wird im Browser geöffnet.
Support-E-Mail (spezialisierter Web-Link)	Ein Fenster für eine neue E-Mail wird mit der Support-E-Mail-Adresse geöffnet.
Web-Link	Frei definierbarer Web-Link
<keine Aktionsart>	Konfigurierte Aktion (mit Skript) ausführen

Ein Web-Link muss vollständig konfiguriert sein, sonst wird er nicht angezeigt.

Abweichend dazu muss dies bei den drei oberen Aktionsarten (spezialisierte Web-Links) nicht so sein. Diese verwenden Standardwerte, falls eine Eigenschaft fehlt. Es besteht die Möglichkeit, die Standardwerte zu überschreiben.

Konfigurationsmöglichkeit Web-Link

Name	Wert
Beschriftung	Anzeigename hinter dem Icon
Symbol	Icon, welches vor der Beschriftung angezeigt wird
URL	URL die geöffnet werden soll

1.7.6.4 Search field

Das Schnellsuchfeld findet sich in der linken oberen Ecke des Hauptfensters. Dieses Feld ermöglicht den schnellen Zugriff auf Abfragen. Diese werden vom Administrator zur Verfügung gestellt oder auch vom Anwender hinzugefügt. Alle Abfragen, die hier verwendet werden, dürfen nur eine Suchzeichenkette oder keine Sucheingabe erwarten.

Keine Sucheingabe macht bei solchen Abfragen Sinn, deren Ergebnis sich von Zeit zu Zeit ändert. Das Ausführen einer solchen Suche im Schnellsuchfeld zeigt dann das aktuelle Ergebnis, ohne dass man die entsprechende Abfrage jedes Mal beispielsweise in einem Ordner aufsuchen muss. Beispielsweise könnte es eine Suchabfrage geben, die alle Lieder anzeigt, die der aktive Anwender schon gehört hat.

1.7.6.4.1 Search field configuration for administrators

Die "Suchfeld"-Konfiguration legt fest, welche Abfragen vom Administrator im Schnellsuchfeld des Knowledge-Builders zur Verfügung gestellt werden.

Neu angelegte Netze verfügen über eine Suchfeld-Konfiguration, die für alle Nutzer gleich

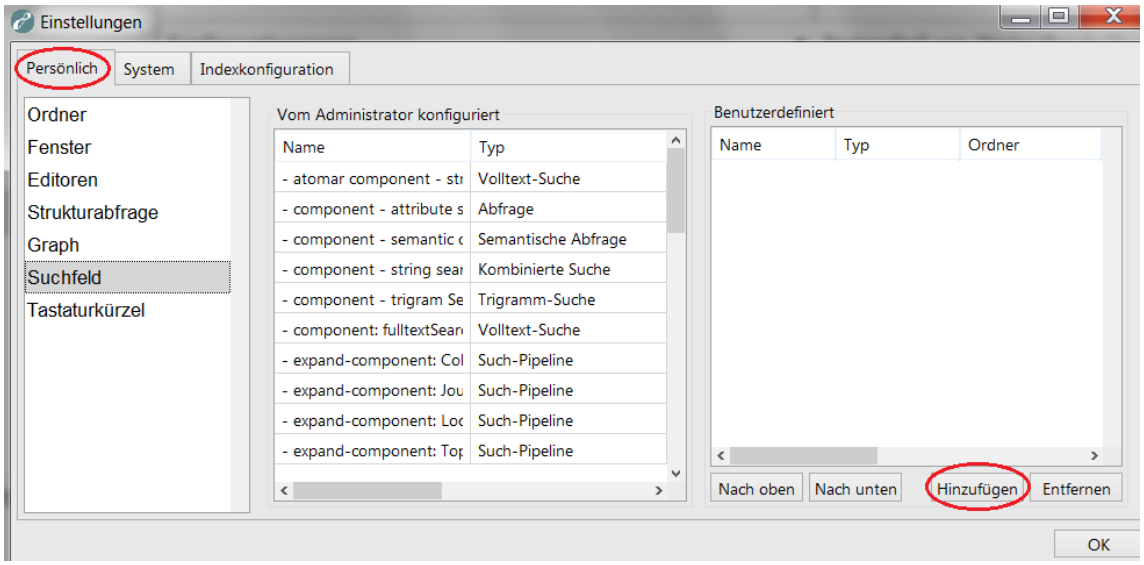
ist. Der Administrator kann diese Suchfeld-Konfiguration erweitern, um allen Nutzern weitere Abfragen zugänglich zu machen. Zusätzlich kann jeder Nutzer seinem Schnellsuchfeld weitere Abfragen hinzufügen, die dann aber nur für ihn persönlich sichtbar sind.

Eine Suchfeld-Konfiguration besteht aus "Schnellsuchelementen", die eine Referenz auf eine Abfrage enthalten müssen und optional mit einer Beschriftung versehen werden können. Die Reihenfolge der Schnellsuchelemente bestimmt die Reihenfolge der Menüeinträge am Schnellsuchfeld.

1.7.6.4.2 Search field configuration for user

Der Anwender kann Abfragen durch ziehen einer existierenden Abfrage auf das Schnellsuchfeld hinzufügen.

Das Hinzufügen kann ebenfalls über die *Einstellungen* erfolgen. Auf dem Reiter *Persönlich* befindet sich der Punkt *Suchfeld*. Im rechten Bereich im Abschnitt *Benutzerdefiniert* steht neben dem *Hinzufügen* auch die Operationen *Entfernen* und die Möglichkeit die Reihenfolge zu ändern zur Verfügung.



1.7.7 Style

Aufgabe der View-Konfiguration ist die strukturelle Aufbereitung von Elementen des semantischen Modells für die Anzeige. Geht es darüber hinaus um die Festlegung rein optischer Eigenschaften bzw. kontextloser Informationen, wird das sogenannte "Style"-Element verwendet.

Es gibt eine Reihe von Style-Elementen, die bereits in i-views definiert sind. Um welche Elemente es sich handelt und wie diese Style-Elemente im Knowledge-Builder angelegt werden, sodass sie dann mit einzelnen Elementen der View-Konfiguration einer Anwendung oder des Knowledge-Builders verknüpft werden können, wird im Folgenden erläutert.

Zunächst muss das Element der View-Konfiguration ausgewählt werden, mit dem ein oder mehrere Style-Elemente verknüpft werden sollen. Fast jeder View-Konfigurations-Typ hat einen "Styles"-Reiter. Dort kann entweder ein neues Style-Element definiert oder ein bereits vorhandenes Style-Element verknüpft werden. Wenn ein neues Style-Element definiert wird, muss diesem zuerst ein Konfigurationsnamen gegeben werden. Auf der

rechten Seite des Editors kann daraufhin die Konfiguration vorgenommen werden.

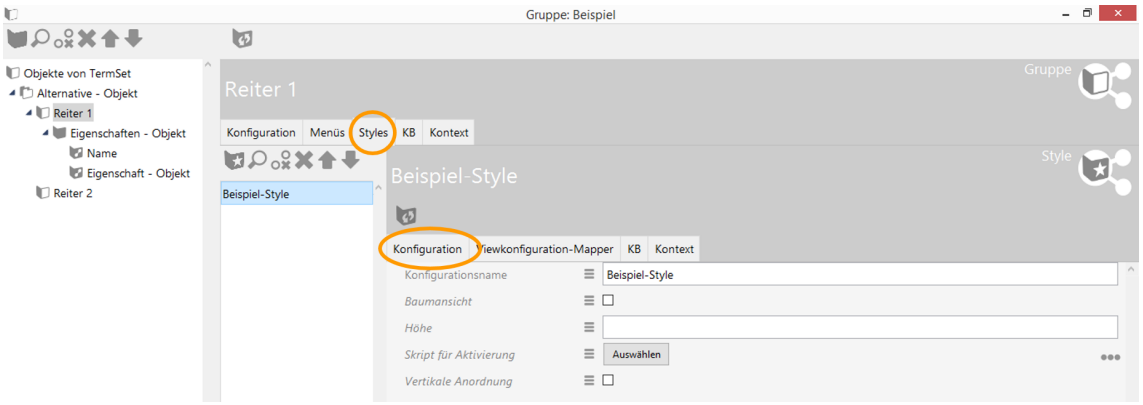
Ein Style-Element kann mit beliebig vielen Style-Eigenschaften befüllt werden. Die Style-Eigenschaften sind auf mehrere Reiter aufgeteilt, die in den folgenden Unterkapiteln beschrieben sind.

Anmerkung

Nicht alle Eigenschaften eines Styles ergeben für alle Konfigurationen Sinn. Die Tabellen der folgenden Unterkapitel führen darum noch die Spalte "Konfigurationstyp", welcher zeigt, welcher View-Konfigurationstyp von der jeweiligen Eigenschaft unterstützt wird. Der Effekt wird in der letzten Spalte beschrieben.

1.7.7.1 Style-Eigenschaften in Anwendungen und im Knowledge-Builder

Der Reiter "Konfiguration" eines Style-Elements ist in diesem Kapitel beschrieben und enthält die Style-Eigenschaften, die sowohl im Knowledge-Builder als auch im Viewkonfiguration-Mapper verwendet werden.



Style-Eigenschaft	Konfigurationstyp	Effekt
Baumansicht	Gruppe	Elemente der Gruppe als Baum darstellen. Standard ist *nein*, d.h. die Gruppenelemente werden nebeneinander oder untereinander angezeigt.
Höhe	Eigenschaft	Höhe in Zeilen bei Zeichenkettenattributen
Höhe	Gruppe	Höhe eines Gruppenelements in Pixeln bei nebeneinander dargestellten Gruppenelementen.
Skript für Aktivierung	Alle	Der Style kann über ein Skript abhängig vom aktiven Element aktiviert werden.
Vertikale Anordnung	Gruppe	Elemente der Gruppe nebeneinander darstellen. Standard ist *untereinander* anzeigen.

1.7.7.2 Style properties in applications

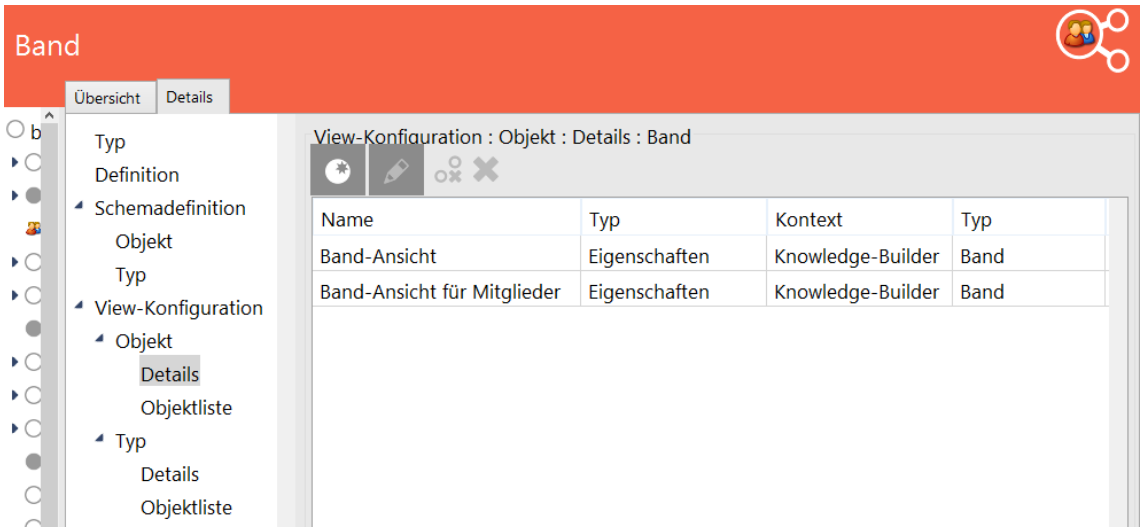
Der Reiter "Viewconfiguration-Mapper" wird nur angezeigt, wenn die Komponente "Viewconfiguration-Mapper" installiert ist. Die verfügbaren Style-Eigenschaften für diese Komponente sind im Kapitel Style des Viewconfig-Mapper (Kaiptel 3) enthalten.

1.7.7.3 Style properties in Knowledge-Builder

1.7.8 Detector System

Mit Hilfe des Detektorsystems können View-Konfigurationen an Bedingungen geknüpft werden. Das Detektorsystem bestimmt wann welche Konfiguration angezeigt werden soll. Im Folgenden wird die Funktionsweise des Detektorsystems und das Zusammenspiel mit View-Konfigurationen an einem Beispiel erläutert.

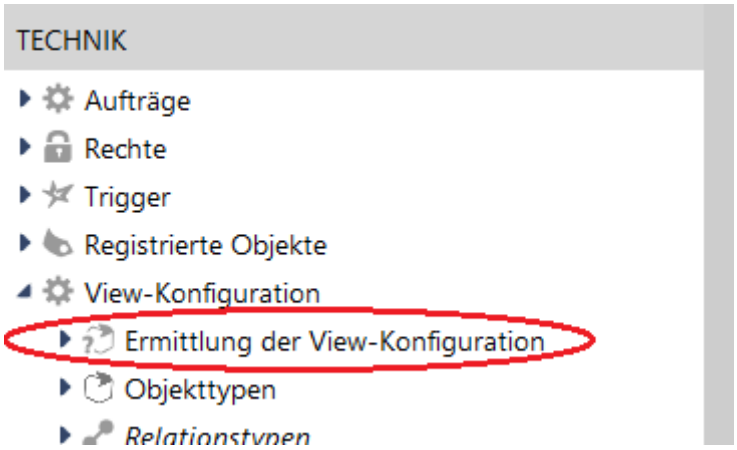
Für Objekte eines Objekttyps können, über Einstellungen in der View-Konfiguration, mehrere Anzeigen erstellt werden. Mit Hilfe des Detektorsystems können diese an Bedingungen geknüpft werden - wie beispielsweise an einen bestimmten Benutzer. Für das hier beschriebene Beispiel wurden für die Objekte eines beliebigen Typs mit Hilfe der View-Konfiguration zwei Ansichten konfiguriert.



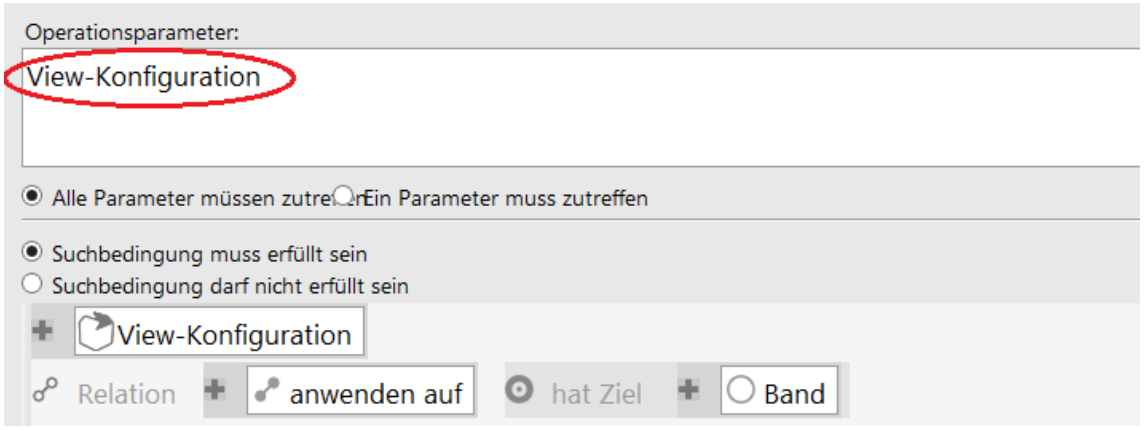
Benutzer, die Mitglied in der Band sind, auf die sie zugreifen wollen, sollen die "Band-Ansicht für Mitglieder" sehen. Alle Benutzer, die nicht Mitglied in der Band sind, auf die sie zugreifen wollen, sollen die "Band-Ansicht" sehen. Die Bedingungen, nach denen die Ansichten benutzt werden sollen, werden im Detektorsystem definiert.

Erstellung einer View-Konfigurations-Ermittlung

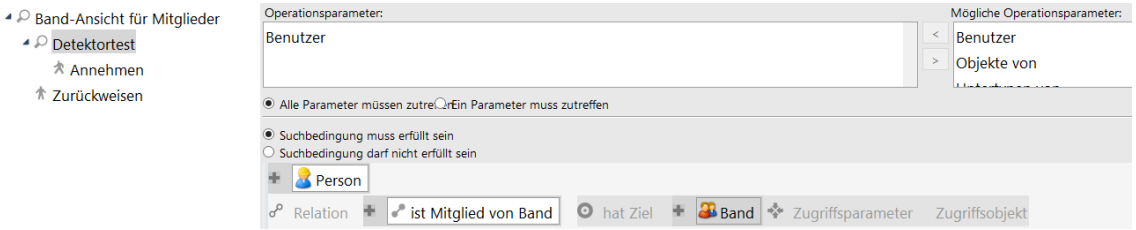
Das Detektorsystem befindet sich in der linken Ordnerhierarchie unter dem Abschnitt "Technik" und ist mit der Bezeichnung "Ermittlung der View-Konfiguration" unter "View-Konfiguration" versehen.



Im erste Schritt muss, über das Anlegen eines neuen Suchfilters (siehe Kapitel Suchfilter), der Ausgangspunkt definiert werden - das heißt, es muss definiert werden, wofür die noch folgenden Einstellungen gelten sollen. In diesem Beispiel ist unser Ausgangspunkt daher eine View-Konfiguration (hier: "Band-Ansicht für Mitglieder"), für die gleich eine Bedingung angelegt wird. Als Operationsparameter muss "View-Konfiguration" aus der Liste ausgewählt und eingetragen werden. Der Suchfilter sieht dann folgendermaßen aus:



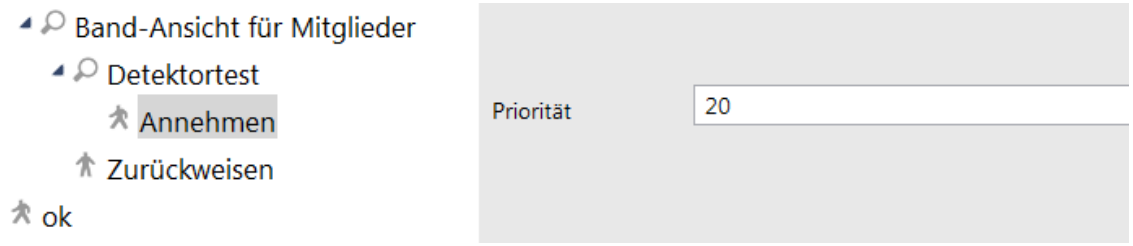
Unter dem Suchfilter, der nach der View-Konfiguration "Band-Ansicht für Mitglieder" sucht, muss nun ein neuer Suchfilter angelegt werden, der die Bedingung für diese View-Konfiguration beschreibt: Die View-Konfiguration "Band-Ansicht für Mitglieder" soll nur für Benutzer sichtbar sein, die Mitglieder der Band sind, die sie sich gerade ansehen. Der zweite Suchfilter prüft also, ob der aktive Benutzer ein Mitglied der Band ist. Über einen Klick auf wird der Menge der Suchergebnisse dann erlaubt, die Konfiguration "Band-Ansicht für Mitglieder" einzusehen. Die folgende Abbildung zeigt, den Suchfilter nach Benutzern, die Mitglied der Band sind, die sie sich gerade ansehen und die Ordnerhierarchie, die auf der linken Seite bisher aufgebaut wurde.



Die View-Konfiguration "Band-Ansicht" wird automatisch für diejenigen Nutzer verwendet, die nicht Mitglied der Band sind, die sie sich gerade ansehen.

Gewichtung der Konfigurationen im Detektorsystem

Die Konfigurationen im Detektorsystem "Ermittlung der View-Konfiguration" werden in der Anwendung von oben nach unten gewichtet. Das heißt, Zugangseinstellungen die weiter oben vorgenommen wurden, wiegen mehr als jene weiter unten. Um diese Standardeinstellung zu umgehen, können den Berechtigungen bzw. Verweigerungen Prioritäten gegeben werden.



Dabei ist Priorität 1 die höchste Priorität. Gibt es bei den Bedingungsanweisungen Überschneidungen, so wird die Berechtigungs- bzw. Verweigerungsbedingung mit der höchsten Priorität durchgesetzt. Sind keine Prioritätsangaben gemacht oder haben alle Prioritätszahlen den gleichen Wert, so wird die frühere Bedingungen im Detektorbaum durchgesetzt.

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:

www.k-infinity.de/api/4.0/index.html

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" }
);
```

Transaktionen, wie oben beschrieben, dürfen nicht geschachtelt werden. Es gibt allerdings Fälle, in denen eine Schachtelung nicht vermeidbar ist, weil beispielsweise eine Skriptfunktion sowohl von Funktionen aufgerufen wird, die bereits in einer Transaktion gekapselt sind als auch von Funktionen, für die das nicht gilt. Hier kann eine sogenannte "optimistische Transaktion" eingesetzt werden. Dieses Konstrukt verwendet die äußere Transaktion - sofern vorhanden, oder startet einen neue Transaktion.

```
$k.optimisticTransaction(function() {
    return $k.Registry.type("Article").createInstance();
});
```

Solche Konstruktionen sollten vermieden werden, weil eine Transaktion eine sinnvolle Arbeitseinheit darstellt, welche ganz oder gar nicht ausgeführt wird. Entweder ist die eingebettete für sich alleine sinnvoll und vollständig oder nicht.

Achtung: Eine Fehlerbehandlungsfunktion bei Fehlschlag der optimistischen Transaktion steht nicht zur Verfügung. Sofern eine äußere Transaktion existiert, wird bei Fehlschlag deren Fehlerbehandlung ausgeführt.

1.8.1.6 Modifying elements

1.8.1.6.1 Creating 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 Creating and modifying 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 Creating 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 Deleting 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 Queries created at runtime

Die Javascript-API erlaubt es auch, Abfragen dynamisch zu generieren. Hier einige Beispiele aus einem Filmnetz:

Suche nach Filmen mit Jahr + Name

```
var query = new $k.StructuredQuery("imdb_film");
query.addAttributeValue("imdb_film_year", "year");
query.addAttributeValue("name", "name");
query.findElements({year: "1958", name: "Vert*"});
```

Dem Constructor wird die Domain übergeben. Bei internen Namen wird automatisch nach Objekten dieses Types gesucht. Mehr Möglichkeiten bietet die Funktion setDomains()

Jahr + Anzahl Regisseure >= 3

```
var query = new $k.StructuredQuery("imdb_film");
query.addAttributeValue("imdb_film_year", "year");
query.addCardinality("imdb_film_regisseur", 3, ">=");
query.findElements({year: "1958"});
```

Jahr + Name des Regisseurs

```
var query = new $k.StructuredQuery("imdb_film");
query.addAttributeValue("imdb_film_year", "year", ">=");
var directorQuery = query.addRelationTarget("imdb_film_regisseur").targetQuery();
directorQuery.addAttributeValue("name", "director");
query.findElements({year: "1950", director: "Hitchcock, Alfred"});
```

Alternativen (Oder-Bedingungen)

```
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*"});
```

Mögliche Operatoren sind:

Operator-Name	Kurzbezeich-nung	Beschreibung
containsPhrase		Enthält Phrase
covers		enthält
distance		Abstand
equal	==	Gleich
equalBy		Entspricht
equalCardinality		Kardinalität gleich
equalGeo		Gleich (Geo)
equalMaxCardinality		Kardinalität kleiner gleich
equalMinCardinality		Kardinalität größer gleich
equalPresentTime		gleich jetzt (Gegenwart)
equalsTopicOneWay		filtern mit
fulltext		Enthält Zeichenkette
greater	>	Grösser als
greaterOrEqual	>=	Grösser/Gleich
greaterOverlaps		überschneidet von oben
greaterPresentTime		nach jetzt (Zukunft)
isCoveredBy		ist enthalten in
less	<	Kleiner als
lessOrEqual	<=	Kleiner/Gleich
lessOverlaps		überschneidet von unten
lessPresentTime		vor jetzt (Vergangenheit)
notEqual	!=	Ungleich
overlaps		überschneidet

range		Zwischen
regexEqual		Regulärer Ausdruck
regexFulltext		Enthält Zeichenkette (Regulärer Ausdruck)
unmodifiedEqual		Exakt gleich
words		Enthält Zeichenkette

1.8.2.3 Creating and modifying 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)
        relation[r].isolate();
    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 Sending eMails

E-Mails können mit dem MailMessage-Objekt verschickt werden. Dazu muss im Netz ein SMTP-Server konfiguriert werden (Einstellungen -> 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();
```

Das Benutzerkonto "kinf" wird für die Authentifizierung verwendet. Das Passwort wird in den SMTP-Einstellungen hinterlegt.

Es können auch Anhänge verschickt werden:

```
var mail = new $k.MailMessage();
var city = $k.Registry.elementAtValue("RDF-ID", "Darmstadt");
var blob = city.attributeValue("picture");
var attachment = new $k.NetEntity();
attachment.setContents(blob);
mail.attach(attachment);
mail.setSubject("Mail with an attachment");
mail.setText("Here is a picture of Darmstadt");
mail.setSender("kinfinity@example.org");
mail.setReceiver("developers@example.org");
mail.setUserName("kinf");
mail.send();
```

In HTML-Mails kann man auch Anhänge einbinden. Dazu muss das Feld **content-id** des Anhangs gesetzt werden. Im HTML-Code kann der Anhang dann mit **cid:**<**id**> referenziert

werden:

```
var mail = new $k.MailMessage();
var city = $k.Registry.elementAtValue("RDF-ID", "Darmstadt");
var blob = city.attributeValue("picture");
var attachment = new $k.NetEntity();
// Set id for referencing the image
attachment.setHeaderField('content-id', 'pic')
attachment.setContents(blob);
mail.attach(attachment);
mail.setSubject("Mail with a picture");
mail.setContentType("text/html")
mail.setText("<html><body>Here is a picture of Darmstadt: <p><img src='cid:pic'/></body></htm
mail.setSender("kinfinity@example.org");
mail.setReceiver("developers@example.org");
mail.setUserName("kinf");
mail.send();
```

1.8.2.8 Views

JSON-Strukturen können auch anhand der View-Konfiguration generiert werden, sowohl für einzelne Objekte als auch Objektlisten.

Im einfachsten Fall wird ein Objekt anhand der Standard-Konfiguration ohne weiteren Kontext in JSON umgewandelt:

```
var data = element.renderJSON();
```

Es werden dann alle durch die Konfiguration definierten Strukturen in JSON umgesetzt:

```
{
    "viewType" : "fieldSet",
    "label" : "Bern",
    "elementType" : "instance",
    "modNum" : 26,
    "elementId" : "ID17361_141538476",
    "type" : {
        "elementType" : "instance",
        "typeId" : "ID10336_319205877",
        "internalName" : "City",
        "typeName" : "Stadt"
    },
    "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" : "Alternativname/Synonym",
            "elementType" : "attribute",
            "internalName" : "alternativeName",
            "attributeType" : "string",
            "rdf-id" : "alternativeName",
            "viewId" : "ID20839_64952366",
            "typeId" : "ID4900_79689320",
            "minOccurrences" : 0
        }
    }, {
        "values" : [{
                "target" : {
                    "typeId" : "ID10336_493550611",
                    "label" : "Kunstmuseum Bern",
                    "elementId" : "ID17362_205182965"
                },
                "propertyId" : "ID17361_395925739"
            }, {
                "target" : {
                    "typeId" : "ID10336_493550611",
                    "label" : "Schweizerische Nationalbibliothek",
                    "elementId" : "ID20401_126870015"
                },
                "propertyId" : "ID17361_9264966"
            }
        ],
        "schema" : {
            "targetDomains" : [{
                    "elementType" : "instance",
                    "typeId" : "ID10336_493550611",
                    "internalName" : "point_of_interest",
                    "typeName" : "Sehenswürdigkeit"
                }
            ],
            "label" : "beherbergt Sehenswürdigkeit",
            "elementType" : "relation",
            "internalName" : "contains_poi",
            "viewId" : "ID20840_182208894",
            "typeId" : "ID2052_332207092",
            "minOccurrences" : 0
```

```
            }
        }
    ]
}
```

Zusätzlich kann ein Kontext in Form eines Anwendungs- oder Konfigurationsobjekts angegeben werden. Es wird dann ein zu diesem Kontext passende Konfiguration gewählt. Im folgenden Beispiel wird die Anwendung "Android" vorgegeben:

```
var application =  $k.Registry.elementAtValue("viewconfig.configurationName", "Android");
var data = element.renderJSON(application);
```

Es ist aber auch möglich, eine Konfiguration vorzugeben und diese das Element umwandeln zu lassen. Dazu erzeugt man eine $k.ViewConfiguration aus dem Konfigurationsobjekt.

```
var configurationElement = $k.Registry.elementAtValue("viewconfig.configurationName", "Android Art
var data = $k.ViewConfiguration.from(configurationElement).renderJSON(element);
```

Da die JSON-Struktur recht umfangreich ist, kann man auch bestimmte Properties bei der Umwandlung weglassen, indem man die Schlüssel als zusätzlichen Parameter angibt:

```
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": "Stadt"
    },
    "properties": [
        {
            "values": [
                {
                    "value": "Bern"
                }
            ],
            "schema": {
                "elementType": "attribute",
                "label": "Name",
                "internalName": "name",
                "attributeType": "string",
                "maxOccurrences": 1
            }
        },
```

```
        {
            "schema": {
                "elementType": "attribute",
                "label": "Alternativname/Synonym",
                "internalName": "alternativeName",
                "attributeType": "string"
            }
        },
        {
            "values": [
                {
                    "target": {
                        "label": "Kunstmuseum Bern",
                        "elementId": "ID17362_205182965"
                    }
                },
                {
                    "target": {
                        "label": "Schweizerische Nationalbibliothek",
                        "elementId": "ID20401_126870015"
                    }
                }
            ],
            "schema": {
                "elementType": "relation",
                "targetDomains": [
                    {
                        "elementType": "instance",
                        "internalName": "point_of_interest",
                        "typeName": "Sehenswürdigkeit"
                    }
                ],
                "label": "beherbergt Sehenswürdigkeit",
                "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 Defining 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 Using 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

Um JavaScript-Bibliotheken einzubinden, die den AMD-Standard unterstützen, muss man vorher define() und require() global definieren:

```
this.define = $k.define;
this.define.amd = {};
this.require = $k.require;
```

Falls eine Bibliothek ein Modul mit einer bestimmten ID definiert und man diese Bibliothek unter einem anderen Namen registrieren möchte, kann man Module-IDs auf Registratur-IDs abbilden:

```
 $k.mapModule("underscore", "lib.underscore");
```

Nun kann man underscore.js als "lib.underscore" registrieren und das fort definierte Modul "underscore" verwenden.

1.8.4 Debugger

1.8.4.1 Debugging in the Knowledge Builder

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

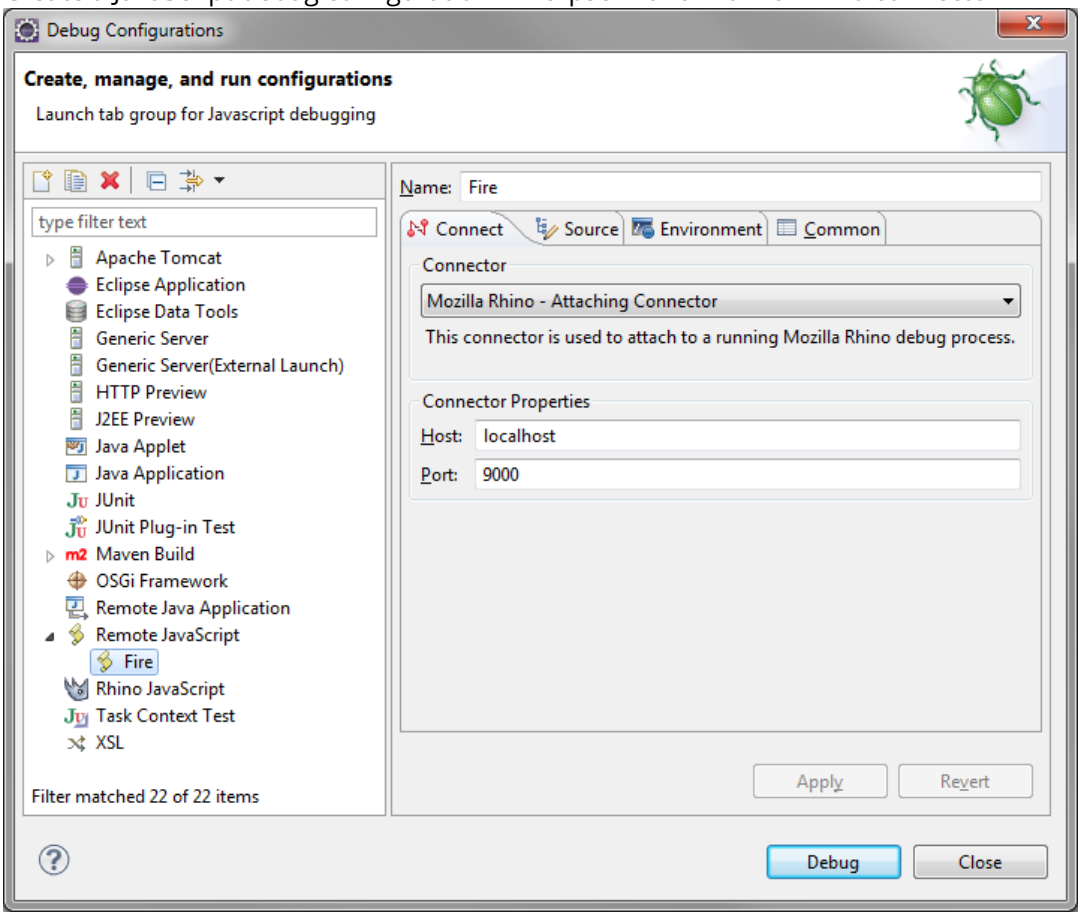
Restlet scripts can be also be debugged with a remote debugger.

1. Enable a debug port in the REST bridge .ini file

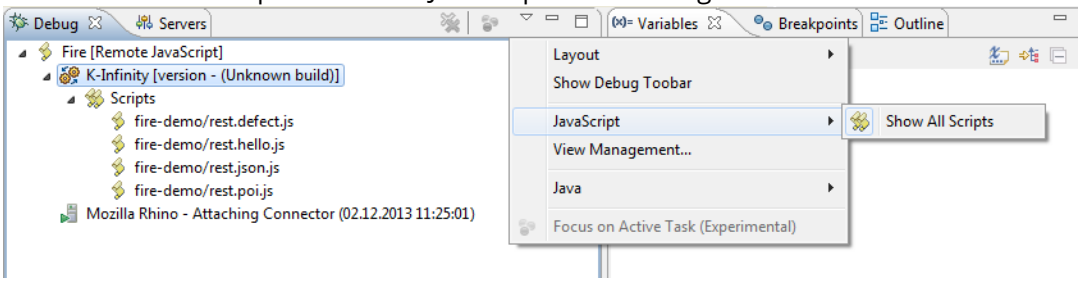
   ```
   [KRemoteDebuggerBridge]
   ```

```
port=9000
```

2. Install Eclipse and JavaScript Development Tools
3. Create a JavaScript debug configuration in Eclipse with a Mozilla Rhino connector



4. Start debugging
5. Enable "Show all scripts" to see all Javascripts of the bridge



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 Extending the API

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 Defining custom 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");
```

1.9 REST services

Über die REST-Schnittstelle kann lesend und schreibend auf das Wissensnetz zugegriffen werden. Dazu definiert man im Wissensnetz **Ressourcen** (beschreiben das Verhalten der Schnittstelle beim Zugriff auf eine Ressource) und **Services** (fassen mehrere Ressourcen zusammen).

Das Verhalten einer Ressource wird über Skripte gesteuert. Zusätzlich können auch vordefinierte Ressourcen verwendet werden.

Der Zugriff erfolgt über HTTP-Requests, die nach dem Muster

```
https://<hostname>:<port>/<service-id>/<resource-pfad-und-parameter>
```

aufgebaut sind.

1.9.1 Configuration

Im Wissensnetz muss die REST-Komponente hinzugefügt werden. Diese definiert das notwendige Schema, das man im Knowledge-Builder im Bereich "Technik "-> "REST" findet.

Die REST-Schnittstelle wird normalerweise vom Bridge-Dienst bereitgestellt. Diese beantwortet HTTP-Anfragen anhand der REST-Konfiguration im Netz. In der Tryout-Variante des Knowledge-Builders ist die Schnittstelle bereits enthalten, man benötigt keinen Bridge-Dienst.

Konfigurationsänderungen im Wissensnetz wirken sich nicht automatisch auf bereits laufende Schnittstellen aus. Dies passiert, wenn man im Hauptmenü des Knowledge-Builders den Menüpunkt "Administrator -> REST-Schnittstelle aktualisieren" ausführt.

Der Bridge-Dienst benötigt eine passende Konfigurationsdatei (bridge.ini). In diese trägt man die Namen des Servers (host), des Wissensnetzes (volume) und der REST-Service-IDs ein. Die Zeile mit "services" kann auch komplett weggelassen werden, dann werden die Ressourcen aller vorhandenen Service-Objekte automatisch aktiviert.

```
[Default]
host=localhost
loglevel=10

[KHTTPRestBridge]
volume=demo
port=8086
services=core,extra
```

1.9.2 Services

Services fassen mehrere Ressourcen zusammen. Ressourcen können in mehreren Services enthalten sein.

Der Services-Editor im Knowledge-Builder zeigt in seiner Strukturansicht die Ressourcen an. Mit "Neues verknüpfen" wird eine neue Ressource angelegt und zum Service hinzugefügt. Mit "Bestehendes verknüpfen" wird eine bereits definierte Ressource zum Service hinzugefügt.

1.9.3 Resources

Ressourcen beschreiben das Verhalten bei einer HTTP-Anfrage an die Schnittstelle. Es gibt folgende Arten von Ressourcen:

Ressource	Beschreibung
Script Resource	Durch Skripte definierbare Ressource.
Built-In Resource	Vordefinierte Ressource, deren Verhalten vom System definiert ist. Diese Ressourcen werden von der Komponente angelegt.

Static File Resource	Liefert Dateien aus dem Dateisystem aus.

Eine Ressource hat folgende konfigurierbare Eigenschaften:

Eigenschaft	**Beschreibung**

Path pattern	Definiert die URL der Ressource relativ zur Adresse des Services. Der Pfad kann parametrisiert werden, indem man Parameter in geschweiften Klammern hinzufügt:
	
	`albums/{genre}`
	
	Es können mehrere Parameter angegeben werden. Jeder Parameter muss aber ein kompletter durch "/" getrennter Teil sein:
	
	`albums/{genre}-{year}`
	
	ist nicht gültig,
	
	`albums/{genre}/{year}`
	
	schon

Part of service	Services, die diese Ressource verwenden
Description	Beschreibung zu Dokumentationszwecken
Requires authentication	Für den Zugriff auf die Ressource ist eine Authentifizierung notwendig

1.9.3.1 Methods

Eine Ressource wird mit einer oder mehreren **Methoden** verknüpft. Diese definiert das Verhalten sowie die unterstützten Ein- und Ausgabetypen (Content-Type). Anhand der Methoden und Typen der HTTP-Anfrage wird eine passende konfigurierte Methode ausgewählt.

In der Strukturansicht werden Methoden als Unterelemente von Ressourcen angezeigt und können dort angelegt/gelöscht werden.

Methode	**Beschreibung**
HTTP Method	Unterstützte HTTP-Methoden (GET, POST, PUT, DELETE). Mehrfachangaben sind möglich.

Input media type	Nur POST/PUT: Erwarteter Content-Type des Inhalts der Anfrage.
Output media type	Content-Type der Antwort. Falls die Anfrage per "Accept" erwartete Content-Typen vorgibt, muss der Output media type dazu passen.
Script	Registriertes Skript zur Definition des Verhaltens (nur bei Script-Ressourcen relevant)
Transaction	Transaktionssteuerung (nur bei Script-Ressourcen relevant)

Die Transaktionssteuerung ist für schreibenden Zugriffe auf das Wissensnetz relevant, da diese nur innerhalb einer Transaktion möglich sind.

Transaktionssteuerung	Beschreibung
Automatic	Bei GET nur lesender Zugriff, bei POST/PUT/DELETE wird das Skript in einer Transaktion ausgeführt. Dies ist die Standardeinstellung.
Controlled by script	Keine Transaktion, das Skript muss diese selbst steuern.
Read	Nur lesender Zugriff, das Skript kann keine Transaktion starten.
Write	Das Skript wird in einer Transaktion ausgeführt.

1.9.3.2 Script-Ressource

Durch ein Skript wird bei einer Methode einer Script-Ressource die Antwort auf eine HTTP-Anfrage definiert. Von der Schnittstelle wird dazu die Funktion respond(request, parameters, response) aufgerufen, die im Skript definiert werden muss.

Argument	Typ	Beschreibung
request	$k.HttpRequest	Anfrage (URL, Header, usw.)
parameters	Object	Aus dem Request extrahierte Parameter
response	$k.HttpResponse	Antwort

Die Funktion füllt dann Header und Inhalt der Antwort. Einen Rückgabewert gibt es nicht.

Wenn für einen Parameter ein Typ definiert wurde (z.B. xsd:integer), wird der konvertierte Wert übergeben, ansonsten eine Zeichenkette. Bei Parametern, die laut Definition mehrfach vorkommen können, werden diese immer als Array übergeben.

Wenn in der Methode ein Output Content-Type für die Antwort definiert wurde, wird dieser automatisch gesetzt. Alternativ kann der Content-Type auch im Skript definiert werden.

Das folgende Skript sucht Alben und wandelt diese in JSON-Objekte um. Die Parameter der Ressource werden an als Suchparameter an die Abfrage weitergereicht.

```
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");
}
```

Dieses Skript könnte man z.B. mit bei einer Ressource

```
albums/{genre}/{year}
```

verwenden und in der Abfrage "albums" die Suchparameter "genre" und "year" als Suchbedingungen verwenden.

1.9.3.3 Built-In Resources

Built-In Resourcen sind vordefinierte Ressourcen, deren Verhalten vom System vorgegeben ist. Jedes vordefinierte Verhalten kann durch einen zugeordneten Wert des Zeichenketten-Attributes *Rest resource ID* zugeordnet werden.

Rest resource ID	**Methode**	**Beschreibung**
BlobResource	GET	Gibt den binären Inhalt eines bestehenden Blob-Attributes zurück. Das Blob-Attribut wird über den Query-Parameter "blobLocator" identifiziert. Optional kann über den Parameter "allowRedirect" festgelegt werden, das Blobs nicht direkt vom Blobservice geholt werden dürfen (Fixed-Value: false).

BlobResource	POST, PUT	Ändert den binären Inhalt eines Blob-Attributes. Das Blob-Attribut wird über den Query-Parameter "blobLocator" identifiziert. Je nach Typ des blobLocators wird ein neues Attribut angelegt oder ein bestehendes verändert.
EditorConfigResource	GET, POST, PUT	Ausgeben und Einlesen einer XML-Repräsentation eines semantischen Elementes.
ObjectListResource	GET	Gibt eine Tabelle von Instanzen oder Untertypen von dem angegebenen Typ zurück. Optional kann gefiltert, sortiert oder direkt die Menge der Objekte definiert werden.
ObjectListPrintTemplateResource	GET	Gibt eine Tabelle von Instanzen oder Untertypen in gedruckter Form zurück. Das Drucktemplate muss angegeben sein.
ObjectListPrintTemplate ResourceWithFilename	GET	Gibt eine Tabelle von Instanzen oder Untertypen in gedruckter Form zurück. Das Drucktemplate muss angegeben sein. Der Parameter {filename} wird nicht ausgewertet und dient allein der besseren Handhabung im Browser.
TopicIconResource	GET	Gibt das Icon oder Bild des angegebenen semantischen Elementes zurück.

Ab i-views 4.1 kann noch ein Java-Script (*rest.preprocessScript*) an die Ressource angebracht werden. Die darin enthaltene Funktion **preprocessParameters (parameters, request)** kann die Parameter ergänzen. Aus den übergebenen Parametern kann so etwa der noch fehlende blobLocator (bzw. das zugehörige Blobattribut) ermittelt werden, was sonst einen zusätzlichen Script-Ressource-Aufruf benötigen würde.

BlobResource

Diese eingebaute Resource ermöglicht das Laden und Speichern der Inhalte von Datei-Attributen.

Download

Über die Methode "GET" kann man den binären Inhalt eines bestehenden Datei-Attributes herunterladen. Das Datei-Attribut wird über den Query-Parameter "blobLocator" identifiziert.

Upload

Beim Upload identifiziert der Parameter "blobLocator" entweder ein existierendes Datei-Attribut oder ein potentielles (d.h. neu anzulegendes) Datei-Attribut. Die Syntax für ein potentielles Attribut hat die Form: "PP∼ID1_115537458∼ID36518_344319903", wobei die erste ID das Wissensnetzelement und die zweite ID den Attribut-Prototyp repräsentiert.

Die Binärdaten können wahlweise als Multi- oder Singlepart übertragen werden. Bei Multi-

part können potentiell mehrere Dateien gleichzeitig hochgeladen werden, was natürlich nur Sinn macht, wenn jede Datei in ein neu anzulegendes Datei-Attribut geschrieben wird. In jedem Fall ist zu jeder übertragenden Datei der Dateiname zu setzen.

Der optionale Parameter "binaryKey" definiert den form-key, unter dem die Binärdaten im MultiPart übertragen werden.

Setzt man den optionalen booleschen Parameter "uploadOnly" auf "true", dann werden nur die Binärdaten hochgeladen jedoch nicht ins Datei-Attribut geschrieben. Dieser Modus wird im Zusammenspiel mit dem ViewConfig-Mapper verwendet. Rückgabe ist in diesem Fall der JSON-Wert (fileName, fileSize, binaryContainerId), der dann in einem zweiten Schritt über den Mapper in das Attribut geschrieben werden kann. Der Content-Type der Rückgabe des JSON-Werts ist normalerweise "application/json", kann aber über den Parameter "overrideContentType" auf einen anderen Wert gesetzt werden, falls der Browser (z.B. IE) Probleme damit hat.

Topic Icon

Über den folgenden Pfad kann man die Bilddatei zu einem gegebenen Topic laden. Wenn ein Individuum keine eigene Bilddatei hat, wird auf die Bilddatei des Typs zurückgegriffen, welche wiederum vererbbar ist. Über den optionalen Parameter "size" kann man die Bilddatei mit der am ehesten passenden Größe selektieren, vorausgesetzt im Wissensnetz sind mehrere Bildgrößen hinterlegt.

http://{server:port}/baseService/topicIcon/{topicID}?size=10

Objektliste

Über den folgenden Pfad kann eine Objektliste im JSON-Format angefordert werden:

http://{server:port}/baseService/{conceptLocator}/objectList

Der Typ der Objektliste wird über den Parameter "**conceptLocator**" referenziert, dem Format für Topic-Referenzen in der Rest-URL folgt. (siehe Verknüpfung)

Alternativ kann der "conceptLocator" auch den einen Prototyp (Individuum oder Typ) des zu verwendenden Typs referenzieren.

Der optionale Parameter "**name**" bestimmt die Objektliste, die für die Ausgabe verwendet werden soll.

Filter

Über den optionalen und mehrwertigen Query-Parameter "**filter**" kann die Objektliste gefiltert werden. Ein Filter hat zwei mögliche Formen:

1. <Spalten-Name/Spalten-Nr.> ~ <Operator> ~ <Wert>
2. <Spalten-Name/Spalten-Nr.> ~ <Wert>

Mögliche Operatoren sind: equal, notEqual, greater, less, greaterOrEqual, lessOrEqual, equalCardinality, containsPhrase, covers, isCoveredBy, distance, fulltext, equalGeo, equalPresentTime, greaterOverlaps, greaterPresentTime, lessOverlaps, lessPresentTime, equalMaxCardinality, equalMinCardinality, overlaps, unmodifiedEqual.

Sortierung

Über den optionalen und mehrwertigen Query-Parameter "**sort**" kann die Objektliste sortiert werden. Die Reihenfolge der Sortierparameter bestimmt die Sortierpriorität. Die Angabe der Sortierung hat zwei mögliche Formen:

1. <Spalten-Name>
2. {-}<Spalten-Nr.>

Stellt man in Variante 2 ein Minus vor, wird absteigend sortiert, sonst aufsteigend.

Startmenge der Liste setzen

Über den optionalen QueryParameter "**elements**" kann eine Komma-separierte Liste von Topic-Referenzen übergeben werden, die als Listenelemente verwendet werden sollen.

Da die Liste der Element ggf. sehr lang ist, kann der Request auch als POST geschickt und die Parameter als Form-Parameter übergeben werden.

Startmenge der Liste über KPath setzen

Über die optionalen Query-Parameter "**elementsPath**" und "**startTopic**" können die initialen Elemente der Liste berechnet werden. Sind diese Parameter nicht gegeben, besteht die Ausgangsmenge aus allen Individuen bzw. allen Untertypen (im Falle einer Typ-Objektliste) des per "conceptLocator" festgelegten Typs.

Dabei ist "elementsPath" ein KPath-Ausdruck und "startTopic" eine Referenz auf das Topic, mit dem die Auswertung des KPath gestartet werden soll. Die Form des Parameters "startTopic" entspricht der des "conceptLocator".

Vererbung

Über den optionalen Query-Parameter "**disableInheritance**" kann die Vererbung unterdrückt werden. Der Paramater macht nur Sinn, wenn kein "elementsPath" gesetzt ist.

JSON-Ausgabeformat (Beispiel)


```
{
 rows: [{
  topicID: "ID123_987654321",
  row: ["MM",
  "Mustermann",
  "Max",
  "111",
  "m.mustermann@email.net",
  "10",
  "6",
  "2000-01-01",
  "Projekt A, Projekt B"]
 },
 {
  topicID: "ID987_123456789",
  row: ["MF",
  "Musterfrau",
  "Maxine",
  "222",
  "m.musterfrau@email.net",
  "10",
  "8",
  "2000-01-01",
  "Projekt X, Projekt Y, Projekt Z"]
 }],
 columnDescriptions: [{
```

```
 label: "Login",
 type: "string",
 columnId: "1"
},
{
 label: "Nachname",
 type: "string",
 columnId: "2"
},
{
 label: "Vorname",
 type: "string",
 columnId: "3"
},
{
 label: "Telefondurchwahl",
 type: "string",
 columnId: "4"
},
{
 label: "Email",
 type: "string",
 columnId: "5"
},
{
 label: "Verfügbarkeit",
 type: "number",
 columnId: "6"
},
{
 label: "Aufwand",
 type: "string",
 columnId: "7"
},
{
 label: "erstellt am",
 type: "dateTime",
 columnId: "8"
},
{
 label: "Projekt",
 type: "string",
 columnId: "9"
}]
}
```


Objektlistendruckvorlage

Über den folgenden Pfad kann eine Objektliste in eine 'Druckvorlage für Liste' gefüllt und das Resultat heruntergeladen werden:

http://{server:port}/baseService/{conceptLocator}/objectList/printTemplate/

{templateLocator}/{filename}

Der Service funktioniert genau wie der Abruf einer Objektliste, trägt aber als zuätzlichen Pa-

rameter eine Referenz auf das Individuum des Typs 'Druckvorlage für Liste' im Wissensnetz.

"**templateLocator**" muss eines der unter "Allgemeines" beschriebenen Formate haben

Der optionale Pfad-Parameter "filename" wird nicht ausgewertet und dient dem besseren Browser-Handling.

Über das Header-Field "**Accept**" wird gesteuert, in welches Ausgabeformat konvertiert werden soll. Fehlt das Header-Field oder ist der Wert "***/***", findet keine Konvertierung statt. Mehrwertige Accept werden nicht unterstützt und resultieren in einer Fehlermeldung.

Über den optionalen Query-Parameter "**targetMimeType**" kann der Wert des "Accept" Header-Fields überschrieben werden. Dies ist notwendig, wenn man den Request aus einem Browser aufrufen möchte und dort keinen Einfluss auf die Header-Fields hat.

Topic drucken

Über den folgenden Pfad kann ein Topic in ein Drucklistentemplate gefüllt und das Resultat heruntergeladen werden:

http://{server:port}/baseService/{topicLocator}/printTemplate/

{templateLocator}/{filename}

"**templateLocator**" muss eines der unter "Allgemeines" beschriebenen Formate haben

Der optionale Pfad-Parameter "filename" wird nicht ausgewertet und dient dem besseren Browser-Handling.

Über das Header-Field "**Accept**" wird gesteuert, in welches Ausgabeformat konvertiert werden soll. Fehlt das Header-Field oder ist der Wert "***/***", findet keine Konvertierung statt. Mehrwertige Accept werden nicht unterstützt und resultieren in einer Fehlermeldung.

Über den optionalen Query-Parameter "**targetMimeType**" kann der Wert des "Accept" Header-Fields überschrieben werden. Dies ist notwendig, wenn man den Request aus einem Browser aufrufen möchte und dort keinen Einfluss auf die Header-Fields hat.

Dokumentformatkonvertierung

Über den folgenden Pfad kann ein Dokument in ein anderes Format umgewandelt werden (z.B. odt in pdf):

http://{server:port}/baseService/jodconverter/service

Der Service bildet den JOD-Konverter (siehe http://sourceforge.net/projects/jodconverter/) ab und dient der Abwärtskompatibilität für Installationen, die bisher mit dem JOD-Konverter betrieben wurden.

Damit der Service funktioniert muss Open/LibreOffice (ab Version 4.0) installiert sein und die Konfigurationsdatei "bridge.ini" muss einen Eintrag enthalten, der auf die Datei "soffice" verweist:

```
[file-format-conversion] sofficePath="C:\Program Files (x86)\LibreOffice 4.0\program\soffice.exe"
```

1.9.3.4 Static File Resource

Liefert Dateien aus dem Dateisystem aus.

Bei dieser Art von Ressource legt man lediglich per *Path pattern* das Verzeichnis fest, unterhalb dessen Dateien ausgeliefert werden. Die Adressierung des Verzeichnisses erfolgt relativ zum Installationsverzeichnis der REST-Bridge.

Beispiel:

Gegeben sei ein Verzeichnis *icons* mit der Datei *bullet.png*. Das Path-Pattern der Ressource lautet *icons*, der dazugehörige Service hat die *Service ID test*. Der Zugriff auf die Datei *bullet.png* lautet dann:

http://localhost:8815/test/icons/bullet.png

1.9.3.5 Parameter

Unterhalb von Methoden kann man die **Parameter** der Ressource definieren. Dies ist nicht zwingend erforderlich, hat aber einige Vorteile:

- Durch Typangaben kann man Parameter prüfen und konvertieren (z.B. in Zahlen oder Objekte)
- Dokumentation für Kunden

Folgende Parameter-Eigenschaften können konfiguriert werden:

Parameter name	Name des Parameters
Style	Art des Parameters • path (Teil des Pfads der URL) • query (Query-Parameter der URL) • header (HTTP-Header)
Type	Datentyp des Parameters. Parameter werden validiert und umgewandelt an das Skript übergeben.
Repeating	Parameter darf mehrfach vorkommen. Wenn aktiviert wird immer eine Array von Werten an das Skript übergeben, selbst wenn nur ein Parameterwert in der Anfrage vorhanden ist.
Required	Parameter muss angegeben werden
Fixed value	Standardwert, falls kein Parameter angegeben wurde.

1.9.4 CORS

Bei OPTIONS-Requests antwortet die REST-Schnittstelle standardmäßig mit

```
Access-Control-Allow-Origin: *
Access-Control-Allow-Headers: Origin, X-Requested-With,Content-Type, Accept
```

In der Konfigurationsdatei (bridge.ini) können diese Header konfiguriert werden:

```
[KHTTPRestBridge]
accessControlAllowOrigin=http://*.i-views.de
```

```
accessControlAllowHeaders=Origin, X-Requested-With,Content-Type, Accept
```

1.10 Reports and Printing

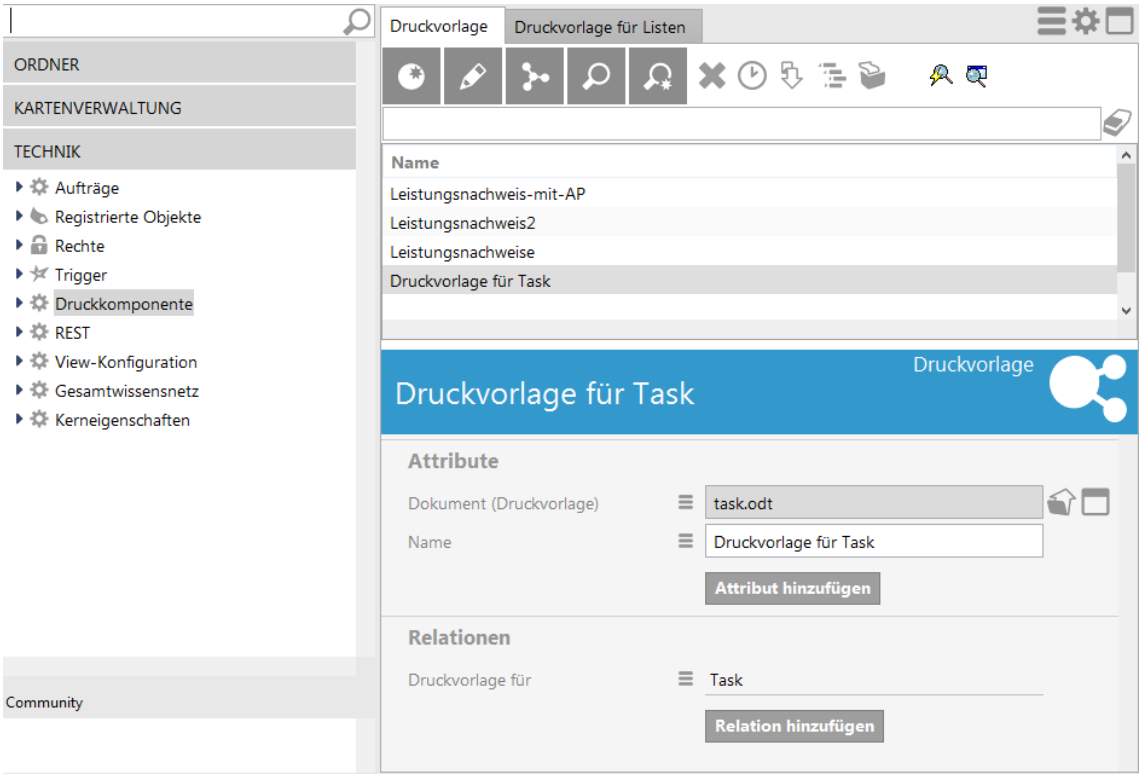
Mit Hilfe der Druckkomponente kann man Dokumentvorlagen (ODT/DOCX/XLSX/RTF-Dateien) mit KPath-Ausdrücken auf Objekten oder Objektlisten anwenden und daraus eine angepasste Ausgabe-Datei generieren, die entweder gedruckt oder gespeichert werden kann.

Das Hinzufügen der Druckkomponente über das Admin-Tool legt im Wissensnetz Konfigurationsschema für Objekte ("Druckvorlage") und Listen ("Druckvorlage für Listen") an. Die Existenz dieser Komponente ist Voraussetzung dafür, dass die Druckfunktionalität im Knowledge-Builder bzw. über die REST-Schnittstelle zur Verfügung steht.

1.10.1 Configuring of print templates

Druckvorlagen werden im Knowledge-Builder im Bereich "TECHNIK/Druckkomponente" angelegt. Jedes Druckvorlagen-Objekt enthält ein Druckvorlagen-Dokument (ODT,DOCX,RTF) und eine Relation, die angibt auf welche Objekte die Druckvorlage angewendet werden soll.

Das folgende Beispiel zeigt eine ODT-Druckvorlage für Objekte des Typs "Task".



Die folgenden Kapitel erläutern das Erstellen der Druckvorlagen-Dokumente.

1.10.1.1 Creation of RTF templates

Die RTF-Vorlagedateien können auswertbare KPath-Ausdrücke mit den Schlüsselworten **KPATH_EXPAND** und **KPATH_ROWS** sowie Aufrufe registrierter KSkripte mit den Schlüsselworten **KSCRIPT_EXPAND**

und **KSCRIPT_ROWS** enthalten. Die Pfadausdrücke bzw. der Name des aufzurufenden Skriptes stehen immer zwischen spitzen Klammern und nach dem Schlüsselwort durch ein Leerzeichen getrennt.

KPATH_EXPAND

Der KPath-Ausdruck nach diesem Schlüsselwort sollte ein einzelnes semantisches Objekt oder einen einfachen Wert (Datum, Zeichenkette etc.) zurückliefern. Bei der Auswertung wird der ursprüngliche Ausdruck durch das Ergebnis ersetzt. Die Formatierung des Ausdrucks bleibt erhalten, Umbrüche des Wertes werden in Zeilenumbrüche umgesetzt.

Beispiel:
Die Vorlage sei:

```
Absender:
<KPATH_EXPAND @$adresse$/rawValue()>
```

Nach Auswertung steht in der Ausgabedatei:

```
Absender:
intelligent views gmbh
Julius-Reiber-Str. 17
64293 Darmstadt
```

KSCRIPT_EXPAND

Alternativ zum Pfadausdruck kann mit KSCRIPT_EXPAND ein registriertes KSkript aufgerufen werden. Die Ausgabe dieses Skriptes (Skriptelemente mit <Output>) wird in das Dokument übernommen. Die Registrierung von Skripten erfolgt im Knowledge-Builder im Ordner TECHNIK/Registrierte Objekte/Skri.

Beispiel:
Die Vorlage sei:

```
<KSCRIPT_EXPAND einSkriptDas1bis9Ausgibt>
```

Nach Auswertung steht in der Ausgabedatei:

```
123.456.789
```

KPATH_ROWS

Dieser Ausdruck muss in einer Tabelle stehen. Der KPath-Ausdruck nach diesem Schlüsselwort muss eine Liste semantischer Objekte liefern. Bei der Auswertung wird die Tabellenzeile des KPATH_ROWS Ausdrucks für jedes Ergebnis des KPath-Ausdrucks einmal ausgewertet. Somit können Tabellen dynamisch ergänzt werden. Es spielt übrigens keine Rolle, in welcher Spalte der KPATH_ROWS Ausdruck steht.

Beispiel:
Die Vorlage sei:

Teile (<KPATH_EXPAND topic()/~$hatTeile$/size()> Stück)	**Bemerkung**

<KPATH_EXPAND topic()><KPATH_ROWS topic()/~$hatTeil$/target()/sort(@$name$, true)>	<KPATH_EXPAND topic()/@$bemerkung$>

Nach Auswertung in der Ausgabedatei:

Teile (3 Stück)	Bemerkung
RTF-Druck	
ODT-Druck	Ersetzt den RTF-Druck
Konvertierungsservice	Optionaler Dienst

KSCRIPT_ROWS

Bei KSCRIPT_ROWS werden die Objekte für die Tabellenzeilen durch ein registriertes KSkript ermittelt. Der Name des registrierten Skriptes wird direkt hinter KSCRIPT_ROWS angegeben. Das Skript muss vom Typ KSkript sein und die auszugebenden Objekte zurückgeben.

Beispiel:
Die Vorlage sei:

Spalte1	Spalte2
<KSCRIPT_ROWS allePersonen><KPATH_EXPAND @$nachname$>	<KPATH_EXPAND @$Vorname$>

Nach Auswertung in der Ausgabedatei:

Spalte1	Spalte2
Meier	Peter
Schulze	Helmut

1.10.1.2 Creating ODT (OpenOffice) documents

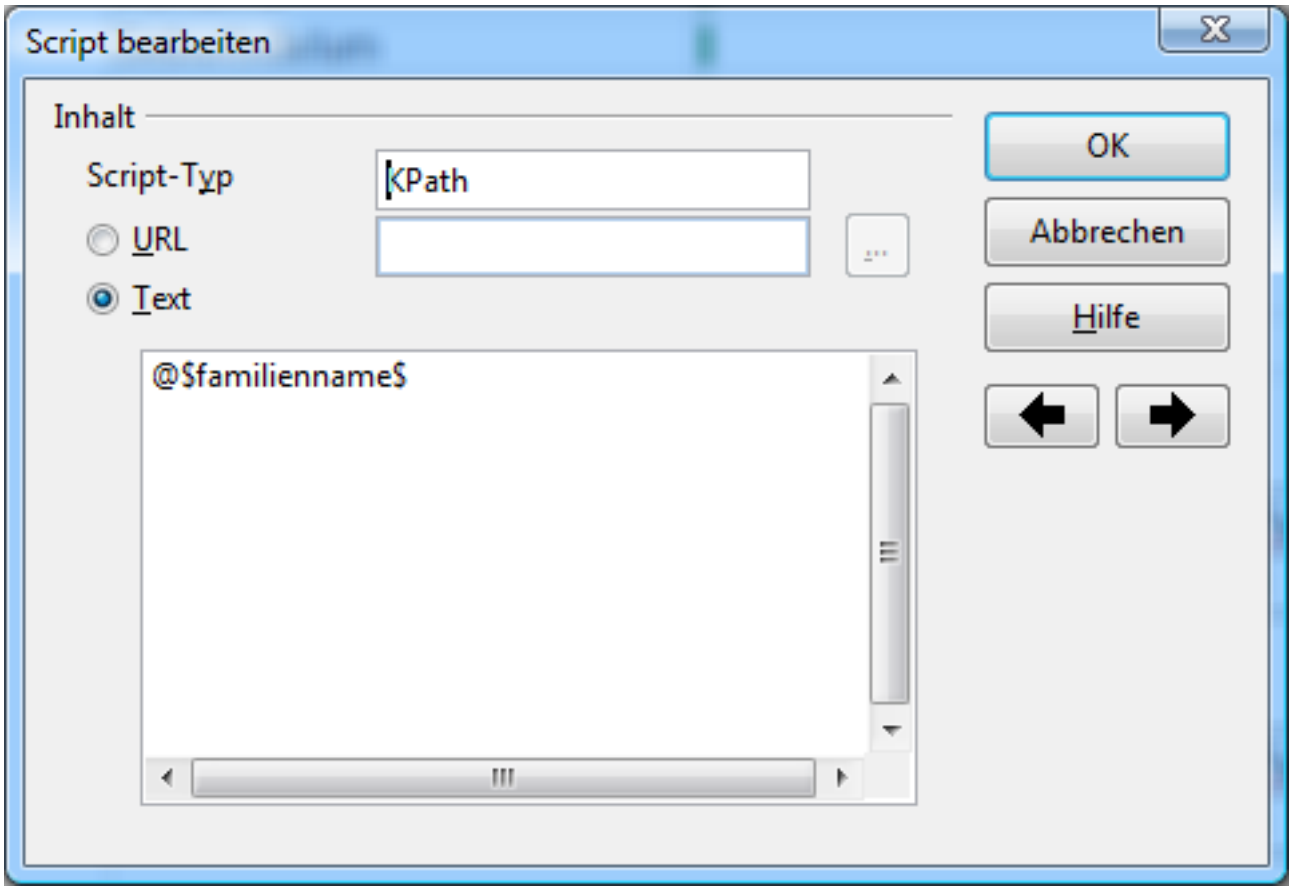
Der Druck über das ODT-Format (Open Document Text, offener Standard) hat viele Vorteile gegenüber dem RTF-Format:

- Die eingebetteten Skript-Anweisungen sind nicht Teil des Textes sondern werden in speziellen Script-Elementen abgelegt. Somit macht man sich seine Formatierung nicht durch längliche Skripte kaputt.
- Das ODT-Format unterstützt eine große Menge an Formatanweisungen (vergleichbar

mit MS-Word), die RTF nicht kennt.

- RTF hat als Format keine einheitliche Normierung (MS-Word kann z.B. "mehr" RTF als der Standard).
- Die Bearbeitung der RTF-Vorlagen ist sehr fragil. Vor allem MS-Word neigt dazu, die Vorlagen mit Steuerelementen (wie z.B. die aktuelle Cursorposition bei der letzen Bearbeitung) zu ergänzen , sodass die Skripte nicht mehr verlässlich erkannt werden können.

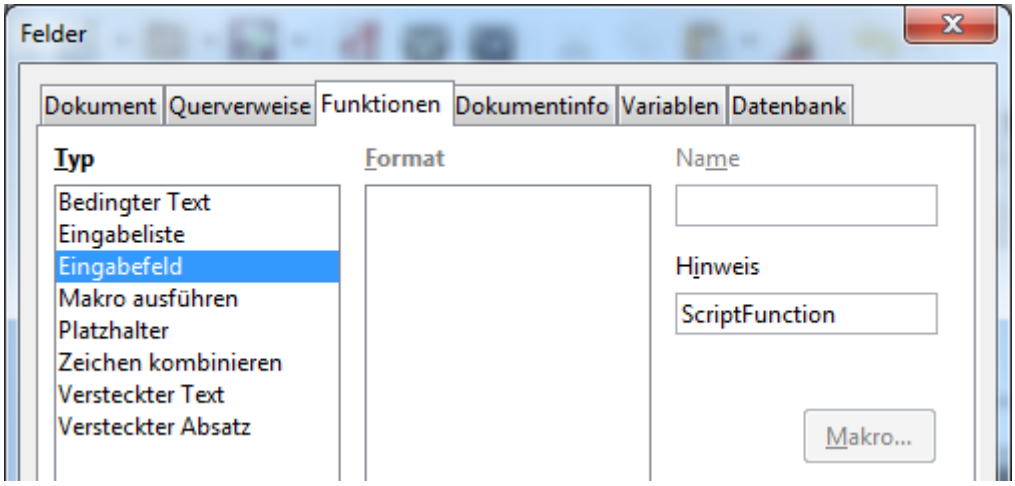
ODT-Vorlagen können mit OpenOffice oder LibreOffice erstellt werden. Die Erstellung erfolgt analog zu der Erstellung von RTF-Vorlagen mit dem Unterschied, dass die Path-/Script-Anweisungen in Script-Elementen abgelegt werden, wie in der folgenden Abbildung gezeigt.



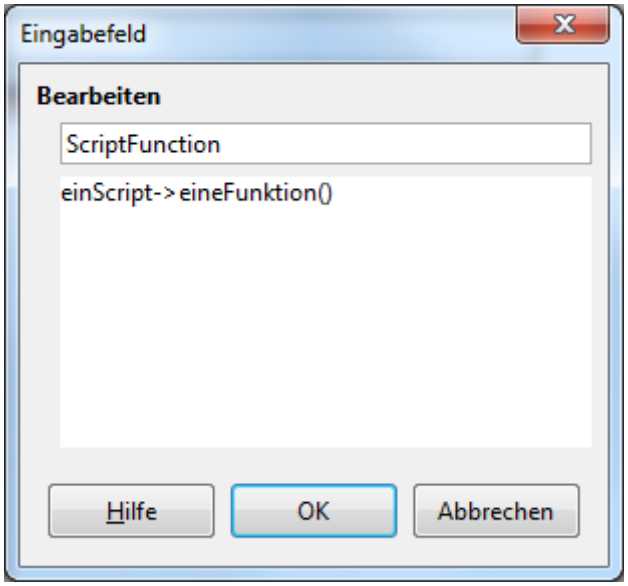
In LibreOffice 5 lässt sich das Skriptfeld nicht länger einbinden. Als Alternative hierzu kann das Feld "Eingabefeld" benutzt werden:

Einfügen > Feldbefehl > Weitere Feldbefehle (alternativ Tastenkombination Strg+F2)

Dort findet sich im Reiter "Funktionen" das Eingabefeld.



"Hinweis" entspricht dem vorherigen "Script-Typ"; nach einem Klick auf Einfügen öffnet sich ein weiteres Fenster, in das das Skript eingetragen werden kann.



Verfügbare Skript-Typen

Als Skript-Typen gibt es:

- **KPath** : analog zu **KPATH_EXPAND**
- **KScript** : analog zu **KSCRIPT_EXPAND**
- **KPathRows** : analog zu **KPATH_ROWS**
- **KPathImage** : zur Einbettung von Bildern
- **ScriptFunction**: Aufruf einer Funktion eines registrierten Scripts. Als Text wird eine Zeichenkette mit folgendem Format erwartet:

  `ScriptID->Funktionsname()`
  Der Funktionsaufruf wird automatisch um zwei Argumente erweitert: das semantisch Elemente und die durch die Umgebung vorgegebenen Variablen

Ein Beispiel für ein aufgerufenes Skript:

```
function headerLabel(element, variables)
{
   return element.name().toLocaleUpperCase();
}
```

- **ScriptRowsFunction**: Analog zu ScriptFunction. Für die zurückgegebenen Objekte werden analog zu KPathRows Tabellenzeilen erzeugt.
- **ScriptImageFunction:** zum Einfügen von Bitmap-Images
- **ScriptSVGImageFunction:** zum Einfügen von SVG-Zeichnungen
- **DataPath:** An der Druckvorlage muss das "Skript zur Erzeugung von JSON-Inhalten" gesetzt sein. Auf die Werte des JSON-Objekts kann nun über den entsprechenden Schlüssel zugegriffen werden.

  Beispiel zum Erzeugen des JSON-Objekts:

  ```
  function templateData(element)
  {
      return {
              name: element.name(),
              idNumber: element.idNumber(),
              someData: { idString: element.idString() }
      }
  }
  ```
  Um zum Beispiel auf den Wert idString zuzugreifen, muss als Text

  ```
  someData.idString
  ```

  gesetzt sein.
- **DataRowsPath:** Analog zu DataPath. Als Text wird ein Schlüssel erwartet, dessen Wert im JSON ein Array von Objekten ist. Nachfolgende DataPath-Felder beziehen sich auf die Objekte in diesem Array.

Zur Einbettung von Bildern können Datei-Attribute oder URLs verwendet werden. Bei der Verwendung von URLs wird versucht, ein Bild von der angegebenen Adresse zu laden.

Eingebettete Bilder werden immer auf ihre Originalgröße (bei 96d dpi) gezogen. Möchte man im Ausdruck eine andere Größe erhalten, muss man einen Rahmen mit den gewünschten Ausmaßen (unbedingt absolute Maße in cm verwenden!) um das Skript-Element bauen. Das resultierende eingebettete Bild wird dann so in den Rahmen eingepasst, dass das Rahmenmaß unter Beibehaltung der Bild-Seitenverhältnisse nicht überschritten wird.

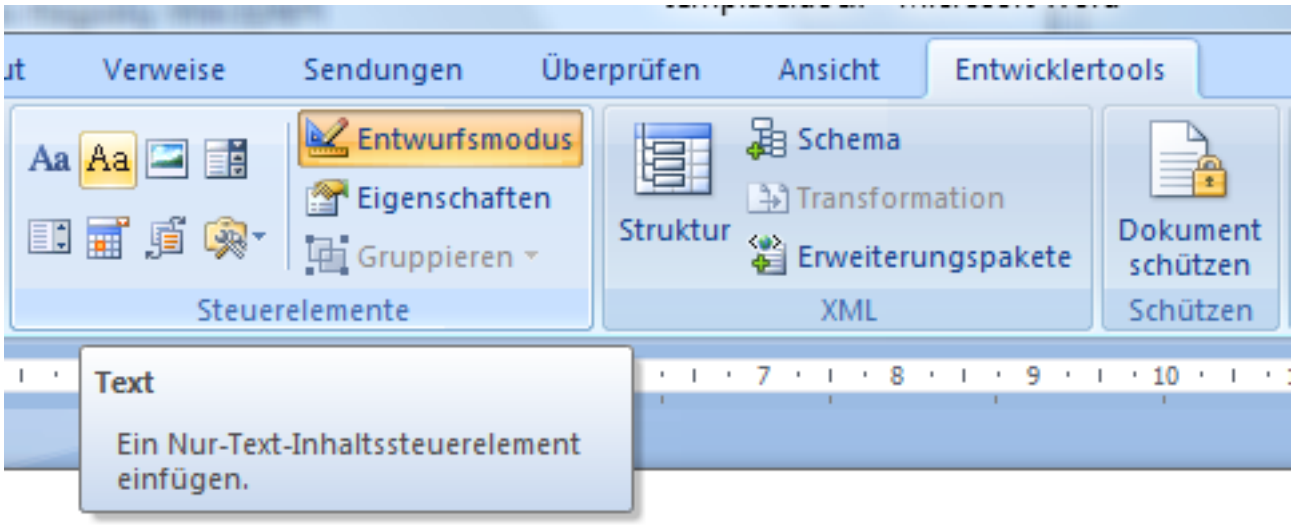
1.10.1.3 Creating DOCX documents (Microsoft Word)

DOCX-Vorlagen können mit Microsoft Word 2007 oder neuer erstellt werden.

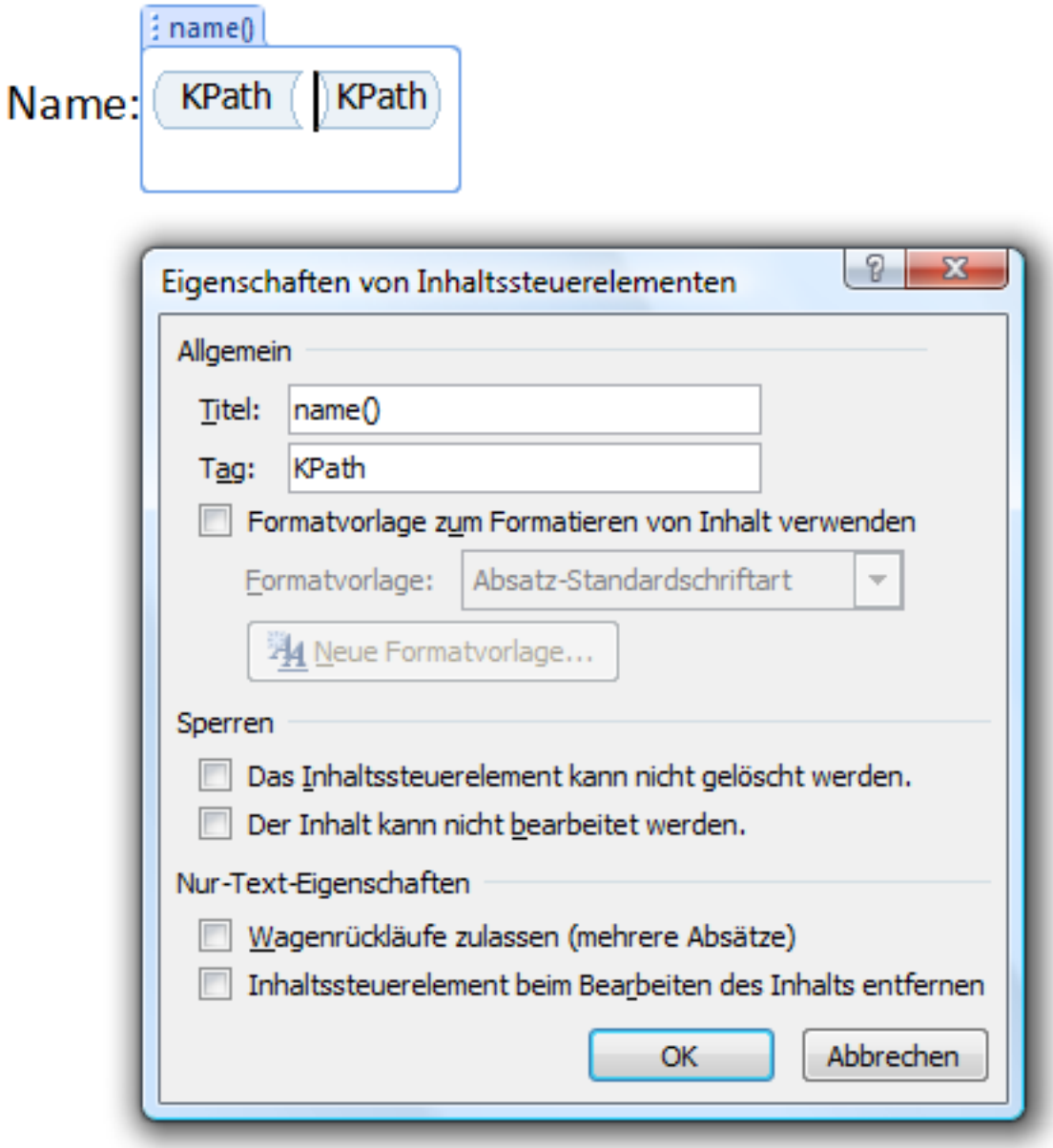
Die Erstellung erfolgt analog zu der Erstellung von RTF-Vorlagen mit dem Unterschied, dass die Path-/Skript-Anweisungen in Text-Inhaltssteuerelementen abgelegt werden.

Zum Einfügen der Steuerelement müssen in Word zuerst die Entwicklertools aktiviert werden. Dazu im Office-Menü die **Word-Optionen** öffnen und in der Kategorie **Häufig verwendet** die Option **Entwicklerregisterkarte in der Multifunktionsleiste anzeigen** aktivieren.

Nun aktiviert man auf der Registerkarte **Entwicklertools** den **Entwurfsmodus**.



Um KSkript/KPath-Ausdrücke einzufügen, fügt man ein **Nur-Text-Inhaltssteuerelement** ein. Der Text des Steuerelements wird durch den berechneten Text ersetzt. Bei den Eigenschaften des Steuerelements (erreichbar über das Kontextmenü auf der schließenden Klammer) gibt man bei **Titel** das KSkript bzw. den KPath an. Falls man den Titel leer lässt, wird stattdessen der Text des Steuerelements verwendet. Als **Tag** gibt man den Skript-Typ an. Als Skript-Typen gibt es alle Typen, die bei ODT zur Verfügung stehen, mit Ausnahme von KPathImage**.**



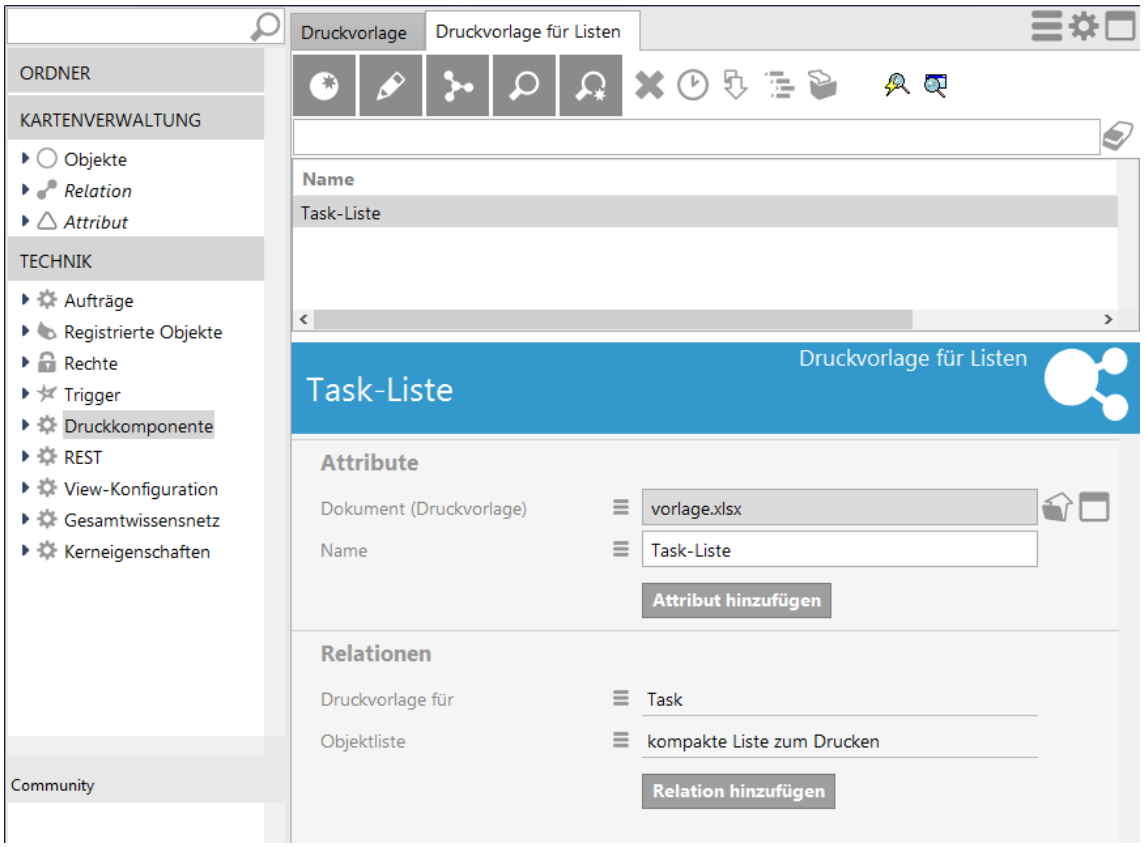
1.10.2 Konfiguring list templates

Druckvorlagen für Listen werden im Knowledge-Builder im Bereich "TECHNIK/Druckkomponente" angelegt. Jedes "Druckvorlagen für Listen"-Objekt enthält ein Druckvorlagen-Dokument (XSLX) und eine Relation, die angibt auf welche Objekte die Druckvorlage angewendet werden soll. Optional kann eine Objektliste angegeben werden, die zur Generierung der Ausgabe verwendet werden soll. Auf diese Weise kann man bewirken, dass sich das Format der Liste, die der Anwender am Bildschirm sieht und das der ausgegebenen Liste unterscheiden.

Wenn man das Attribut "Dokument (Druckvorlage)" nicht angelegt hat, so wird bei der Dokumentgenerierung eine Excel-Datei erzeugt, die ein Arbeitsblatt mit den Daten der Objektliste und den Spaltenüberschriften aus der Objektlistenkonfiguration enthält, d.h. man muss

nicht zwangsläufig eine Excel-Datei als Druckvorlage angeben.

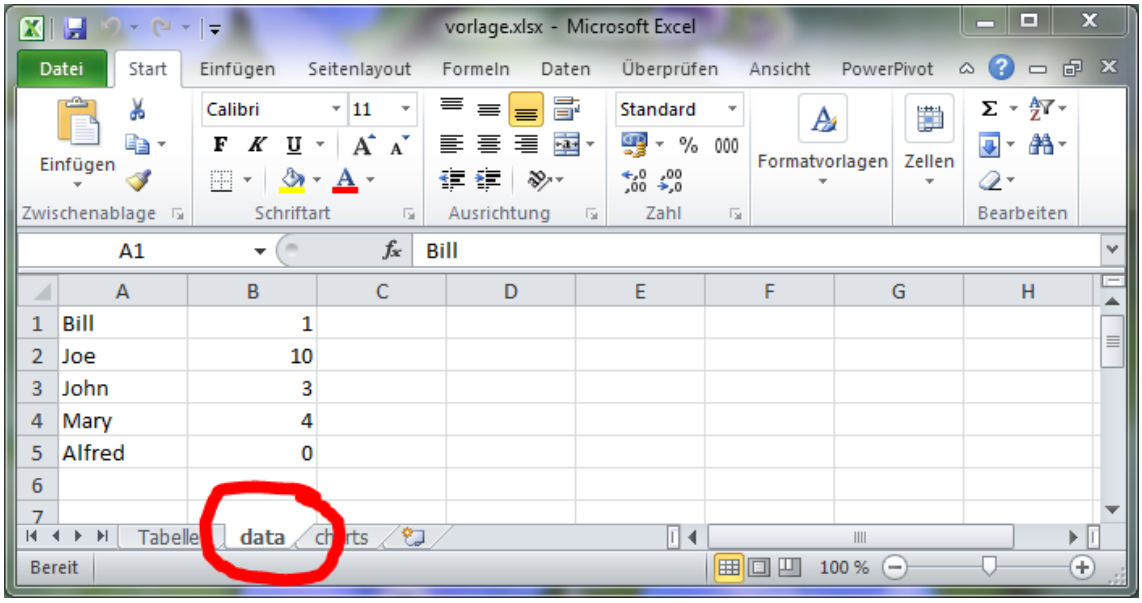
Das folgende Beispiel zeigt eine Druckvorlage für Listen mit Objekten des Typs "Task".



XLSX-Vorlagen können mit Microsoft Excel 2007 oder neuer erstellt werden. Diese Vorlagen funktionieren nur mit Objektlisten.

Erstellen der Excel-Datei

Als Vorlage dient eine gewöhnliche Excel-Datei, die ein zusätzliches Arbeitsblatt namens "data" enthalten muss. Die Objektlistendaten werden später dann in dieses Arbeitsblatt gefüllt und zwar ohne Überschriften und beginnend mit der Zelle A1.



Die anderen Arbeitsblätter können Daten aus dem Blatt "data" in Formeln referenzieren. i-views sorgt dafür, dass alle Formeln neu berechnet werden, sobald die ausgefüllte Excel-Datei das nächste Mal mit Excel geöffnet wird.

1.10.3 File Format Conversion using Open/LibreOffice

The output format of the printing process corresponds to that of the template used. If you want to getanother output format, you must set up a converter.

This requires an installation of Libre or OpenOffice version 4.0 or higher on the machine that is to perform the conversion - usually where the bridge or the job client is running, which also performs the printing process.

In addition, in the configuration file (bridge.ini, jobclient.ini, etc.) the path to the "soffice" program must be specified, which is part of the Libre / OpenOffice installation and is located there in the subdirectory "program". This specification must be made as 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 or cannot maintain a Libre / Office installation on all workstations or server installations from which format conversion is to be carried out, an appropriately configured REST bridge can perform the conversion.

The .ini file of the REST bridge must have the following format:

```
[Default]
host=localhost

[KHTTPRestBridge]
port=3040
volume=nameOfTheVolume
```

services=jodService

[file-format-conversion]

sofficePath="C:\Program Files (x86)\LibreOffice 4.0\program\soffice.exe"

In the Admin Tool enter under System Configuration / Components / Conversion Service the address via which the conversionservice can be reached.

Example:

http://localhost:3040/jodService/jodconverter/service

Documents formats

For the output formats to be available, appropriately configured objects of the "converter document format" type must be available in the network.

It is important that not all formats can be converted into each other. The most important are:

Name	Extension	Mime-Type
Portable Document Format	pdf	application/pdf
OpenDocument Text	odt	application/vnd.oasis.opendocument.text
Microsoft Word	doc	application/msword

1.11 Tagging

Die Tagging-Komponente ermöglicht es, Objekte aus dem Wissensnetz (Personen, Themen usw.) in Dokumenten zu finden oder neu anzulegen.

Für das Tagging benötigt man

- eine konfigurierte Tagging-Komponente im Wissensnetz
- eine Tagging-Software (Intrafind, OpenNLP), die potentielle Objekte in einem Text findet

Das Tagging erfolgt in drei Schritten

1. Der zu taggende Text eines Dokuments wird bestimmt (z.B. der Wert eines Textattributs)
2. Der Text wird an die Tagging-Software weitergereicht, die den Text analysiert und eine Reihe von Tags liefert
3. Anhand der Konfiguration werden für jedes Tag vorhandene Objekte im Wissensnetz gesucht und potentiell auch neue Objekte angelegt. Die Objekte werden mit dem Dokument per Relation verknüpft.

1.11.1 Configuration

Für das Tagging wird die gleichnamige Komponente benötigt, die im Admin-Tool hinzugefügt werden kann. Diese richtet das benötigte Schema ein.

Anschließend kann im Knowledge-Builder unter "Technik" > "Tagging" die Konfiguration vorgenommen werden.

Jede Tagging-Konfiguration besteht aus

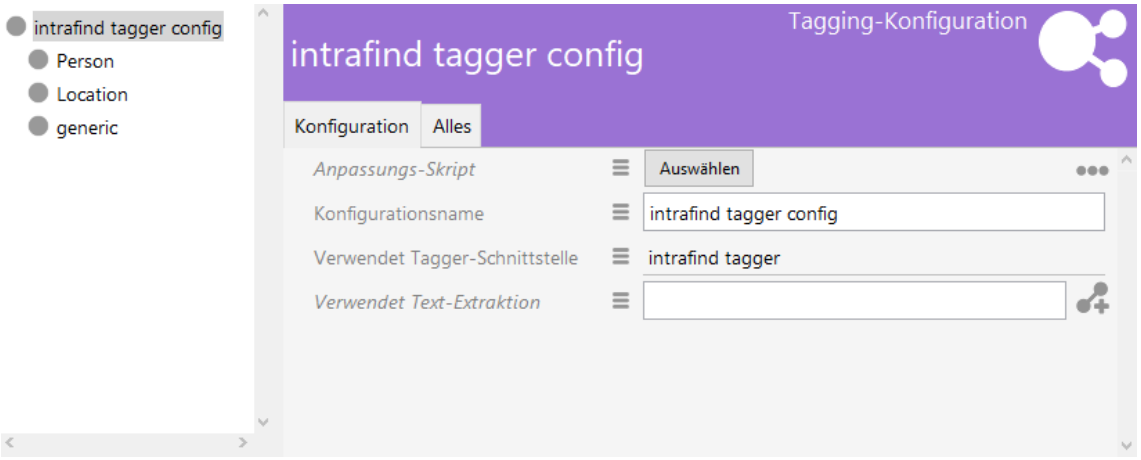
- einer Schnittstellen-Konfiguration der zu verwendenden Tagging-Software (intrafind, OpenNLP)
- der Konfiguration der Text-Extraktion, die den zu taggenden Text eines Dokuments ermittelt
- Tag-Konfigurationen, die bestimmen, wie Objekte im Wissensnetz gefunden, angelegt und verknüpft werden

1.11.1.1 Tagging configuration

Die Tagging-Konfiguration bündelt alle Informationen, die man für das Tagging benötigt.

Zwingend erforderlich ist es, die zu verwendende Tagger-Schnittstelle anzugeben.

Die Angabe der zu verwendenden Text-Extraktion ist optional. Alternativ kann diese auch dynamisch ermittelt werden (siehe das dazugehörige Unterkapitel).



Weiterhin kann ein Anpassungs-Skript angebeben werden, mit dem Einfluss auf das Tagging genommen werden kann. Zusätzliche Anpassungen können auch bei den Konfigurationen für Tags und für Text-Extraktion vorgenommen werden.

Neu angelegte Anpassungs-Skripte enthalten auskommentierte Funktionsrümpfe. Um diese zu aktivieren braucht man nur die Kommentarzeichen zu entfernen.

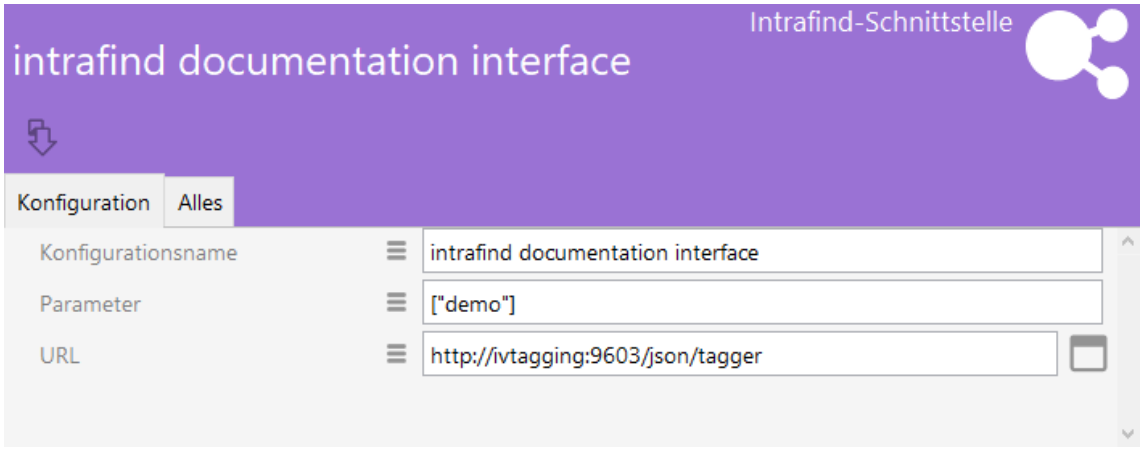
1.11.1.2 Interface configuration

Die Intrafind-Schnittstelle hat folgende Einstellungen:

Konfigura-tionsname	Frei wählbarer Name
Parameter	Wird über die Schnittstelle an Intrafind weitergegeben, steuert das Tag-ging
URL	URL des Intrafind-Taggers

Bei OpenNLP wird neben dem optionalen Namen nur die URL des REST-Services benötigt.

Die Schnittstelle "Interner Tagger" ist nur zu Testzwecken / interne Demos gedacht, bei denen man kein externes System anbinden möchte. Dieser Tagger hat keinerlei Anspruch, sinnvolle Ergebnisse zu liefern.



1.11.1.3 Text extraction

Wenn der zu taggende Text nicht dynamisch ermittelt wird, weil z.B. nur der Text eines ganz bestimmten Attributtyps oder der Text eines Dokuments extrahiert werden soll, muss eine Text-Extraktion konfiguriert werden.

Diese Konfiguration kann unter dem Reiter "Text-Extraktion" hinzugefügt werden.

Konfigura-tionsname	Frei wählbarer Name
anwenden auf	Objekt-Typ, für den diese Konfiguration gilt. Wird verwendet, wenn bei einer Tagging-Konfiguration keine explizite Text-Extraktion vorgegeben wird.
Skript zur Textex-traktion	Optionales Skript zur Bestimmung des Texts

Um die zu taggenden Attributtypen anzugeben, werden der Text-Extraktion ein oder mehrere (auf der linken Seite hierarchisch geordnete) Textteil-Extraktionen hinzugefügt. In jeder Textteil-Extraktion wird unter "extrahiert Text aus" der zu taggende Attributtyp hinterlegt.

Als Textteil-Extraktionen können neben Zeichenketten auch Blobs verwendet werden. Aus diesen wird der Text extrahiert und an die Tagging-Schnittstelle weitergereicht. Dazu muss im Client (Bridge oder KB) die Textextraktion konfiguriert werden (siehe Kapitel i-views-Dienste > Text-Extraktion).

Das optionale Skript hat drei Argumente

textDocument	$k.TextDocument	Ausgabe des zu taggenden Texts
element	$k.SemanticElement	Das Element, dessen Text extrahiert werden soll
attributes	$k.Attribute[]	Array der Attribute des Elements. Es werden die Attribute gemäß der Konfiguration aufgesammelt.

Folgendes Beispiel schreibt die Werte der Attribute hintereinander:

```
function extractText(textDocument, element, attributes)
{
    attributes.forEach(function(attribute) {
        textDocument.println(attribute.valueString());
    });
}
```

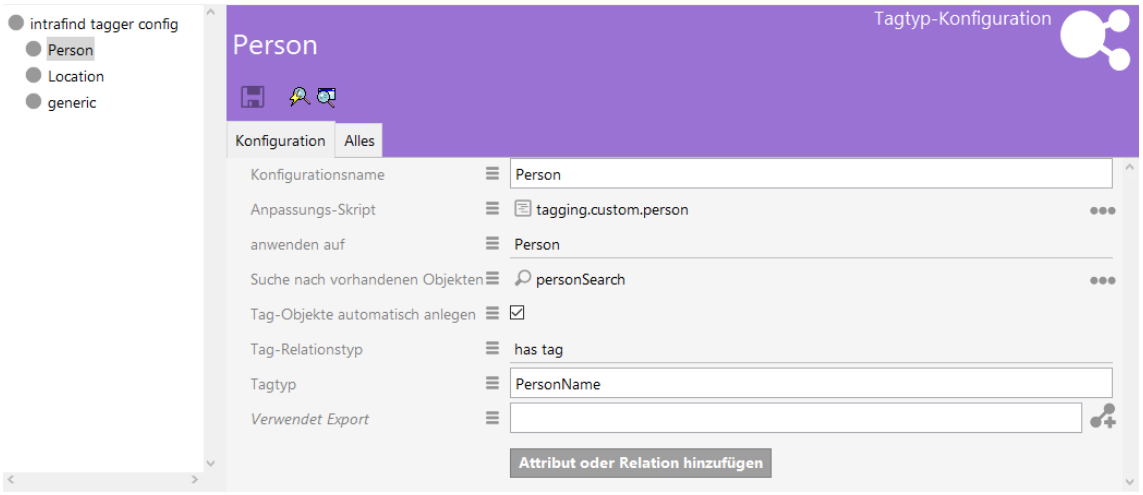
1.11.1.4 Tag types

Die Konfiguration eines Tagtyps definiert, wie Objekte im Wissensnetz gefunden, angelegt und verknüpft werden sollen. Dazu kann man für jeden von der Tagging-Schnittstelle bereitgestellten Tagtyp eine eigene Konfiguration angeben. Eine neue Konfiguration kann man bei der Tagging-Konfiguration in der Hierarchie-Ansicht auf der linken Seite anlegen.

Standardmäßig liefern die Schnittstellen folgende Tagtypen:

Intrafind	PersonName, Location, TFIDF
OpenNLP	NP

Ein Tagkonfiguration kann für einen oder mehrere Tagtypen gelten.



Die Konfiguration bietet folgende Einstellungen:

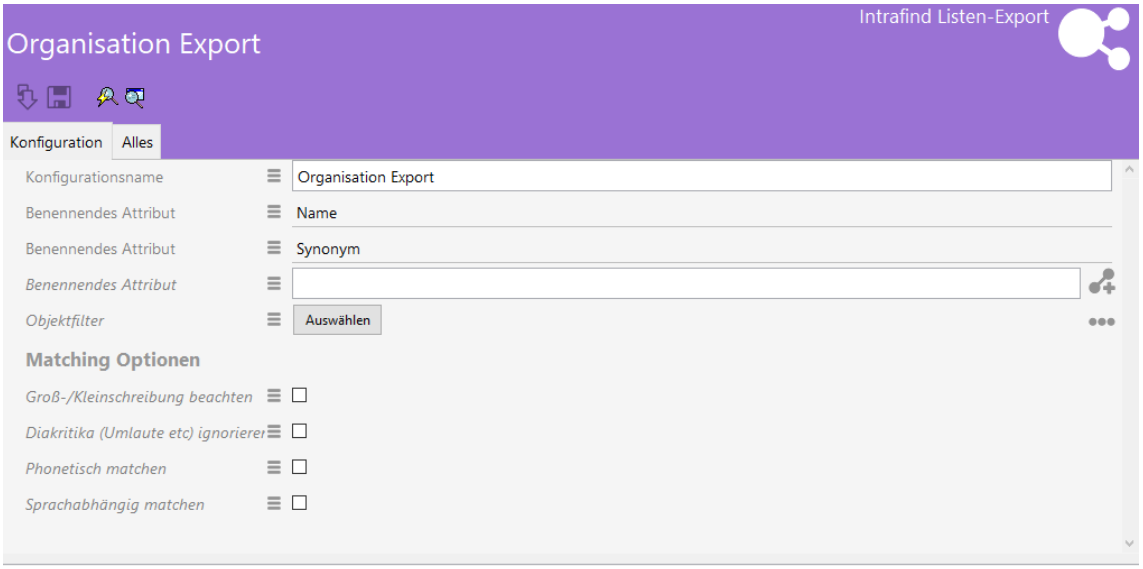
Anpassungs-Skript	Skript, um in das Tagging einzugreifen. Das Template enthält eine Reihe auskommentierter Funktionen, die aktiviert werden können.
anwenden auf	Typ im Wissensnetz, der dem Tagtyp entspricht. Falls Objekte gesucht/angelegt werden sollen und keine weiteren Konfigurationsangaben gemacht wurden, wird dieser Typ verwendet.
Konfigurationsname	Frei wählbarer Name
Suche nach vorhandenen Objekten	Suche, die als Eingabe den Text des Tags als Parameter *searchString* erhält und *ein* passendes Objekt im Wissensnetz sucht. Es können mehrere Suchen angegeben werden, um z.B. die einzelnen Suchen kompakter zu halten. Es ist Aufgabe der Suche, bei mehreren Suchtreffern den geeigneten Treffer zu liefern. Falls mehrere Treffer mit unterschiedlicher Qualität gefunden werden, wird der Treffer mit der besten Qualität verwendet. Falls sich kein bester Treffer bestimmen lässt wird kein Objekt zugeordnet.

Tag-Objekte automatisch anlegen	Falls kein Objekt gefunden wurde und diese Option aktiviert wurde, werden neue Objekt angelegt. Es ist darauf zu achten, dass die Suche nach vorhandenen Objekten diese findet, da ansonsten bei jedem Tagging neue Objekte entstehen. Falls kein Anpassungsskript hier eingreift wird ein Objekt vom bei "anwenden auf" angegeben Typ angelegt und als dessen Name der Text des Tags gesetzt.
Tag-Relationstyp	Mit diesem Relationstyp werden Dokumente mit den durch den Tagger gefundenen Objekten verknüpft.
Tagtyp	Die oben angegeben Tagtypen. Falls kein Tagtyp definiert ist gilt die Konfiguration für alle Arten von Tags.
Verwendet Export	Hier kann eine Export-Konfiguration angegeben werden, mittels der sich alle Tags des Typs oder eine Untermenge davon exportieren lassen. Details hierzu im nächsten Abschnitt.

1.11.1.5 Export der bekannten Tags

Um in einem Tagging-Service, z.B. Intrafind, Information aus dem Netz zu hinterlegen gibt es eine Export-Möglichkeit. Derzeit wird diese nur für Intrafind unterstützt und hat dort folgendes Verhalten:

Es lassen sich eine oder mehrere Listen generieren, die dann bei der Tagging-Schnittstelle abgelegt werden. Jeder Listen-Export ordnet den zu exportierenden Wissensnetz-Elementen benennende Attribute (z.B. Name, Synonym) zu. Diese Benennungen sucht der Tagger dann in Texten und kann das passende Wissensnetz-Element dazu liefern. Beispielsweise lässt sich so die Liste der bekannten Organisationen exportieren und der Tagger kann diese zuverlässig identifizieren.



Der *Intrafind Listen-Export* wird für jeden Tagtyp konfiguriert und wird auch von der *Tagtyp-Konfiguration* beeinflusst. Allgemeine Konfigurationsoptionen:

Konfigurationsname	Frei wählbarer Name
Benennendes Attribut	(optional) Attribut, das das Objekt identifiziert. Mehrfache Angabe möglich. Wenn kein Attribut angegeben ist, wird standardmäßig das Namensattribut exportiert.
Objektfilter	(optional) Hier lässt sich eine Suche angeben, die die Menge der Objekte spezifiziert. Wenn keine Suche angeben ist, werden alle Typen die in der Tagtyp-Konfiguration mittels *anwenden auf* zugeordnet sind exportiert.

Intrafind spezifische *Matching Optionen*. Diese beeinflussen direkt das Verhalten des Tagging-Services:

Groß-/Kleinschreibung beachten	Standardmäßig wird case-insensitiv gematcht. Hier lässt sich die case-sensitivität aktivieren.
Diakritika (Umlaute etc) ignorieren	[vermutlich] Mittels dieser Option werden Zeichen mit Accents oder Umlauten ignoriert, z.B. Geräte matcht auch Gerate.

Phonetisch matchen	[vermutlich] Auch z.B. "Photographie" mit "Fotografie" matchen.
Sprach-abhängig matchen	Diese Option schaltet die linguistische Verarbeitung der übergebenen Benennungen ein. Hierbei ist es wichtig, dass die Daten korrekt nach Sprache im Wissensnetz gepflegt sind, da jede Sprache mit einer ihr eigenen Linguistik verarbeitet werden muss.

1.11.1.6 Überlappungsfiltergruppe

Es kann vorkommen, dass der Tagger zu einer Textstelle mehrere Tags liefert. In manchen Fällen will man diese Überlappung bewusst zulassen und mehrere Tags anzeigen.

Das Verhalten der Überlappungsfiltergruppe ist wie folgt:

- Alle Tagtypen die in einer solcher Gruppe zusammengefasst sind, müssen überlappungsfrei sein
- Innerhalb einer Gruppe lässt sich über ein Skript eine Priorisierung angeben, um die Entscheidung, welcher Tag am Ende angezeigt wird, zu beeinflussen
- Um Überlappungen zuzulassen müssen mind. zwei solcher Gruppen existieren
- Alle Tagtypen ohne Gruppe werden in der "default"-Überlappungsfiltergruppe zusammengefasst

Priorisierung mit Skript

```
/**
 * 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 ];
}
```

Das Skript muss eine Liste von Integern zurückgeben, wobei das erste Element dieser Liste den höchsten Einfluss hat. Im Prinzip funktioniert es wie bei der Sortierung nach mehreren Spalten, also das zweite Element wird nur dann hinzugezogen, wenn im Ersten der gleiche Wert vorliegt.

Default-Priorisierung

Wenn kein Skript angegeben ist, oder der Tagtyp in der impliziten "default"-Gruppe gruppiert ist, dann wird folgende Priorisierung angewendet:

- Reihenfolge der Tagtypen - höhere Priorität zuerst
- Längere Tags bevorzugt
- Position innerhalb der Überlappung (also bei "eine rote Wand" wird "eine rote" vor "rote Wand" bevorzugt, weil es weiter vorn steht)

Vergleiche auch das Skript-Template.

1.11.2 View configuration

Zur Anzeige sind zwei Views verfügbar:

- Markup-View
- Tag-Liste

Der Markup-View ist sowohl im Knowledge-Builder als auch im ViewConfig-Mapper verwendbar. Der View kann überall dort verwendet werden, wo auch andere Views wie z.B. Eigenschaften oder Hierarchien eingesetzt werden können.

Im Knowledge-Builder hat der View bereits einen fest eingebauten Tag-Button. Im ViewConfig-Mapper gibt es eine eingebaute Aktionsart "Tag", die man auch in einem eigenen Button verwenden kann.

Die Tag-Liste ist nur im ViewConfig-Mapper verfügbar und ist dort Inhalt eines Panels (z.B. als Subkonfiguration eines Panels mit festangelegter Ansicht). Falls man in einem anderen Panel einen Markup-View mit Tag-Buttons konfiguriert hat, sollte man dessen Panel per Relation "beeinflusst" mit dem Tag-Listen-Panel verknüpfen, damit nach dem Tagging die Tagliste aktualisiert wird.

Beide Views haben die obligatorische Konfigurationseinstellung "Verwendet Tagging-Konfiguration", die den View mit der Tagging-Konfiguration verbindet.

1.11.3 Tagging scripts

Taggen ist auch per Script möglich. Dazu erstellt man ein Objekt vom Typ $k.TaggingConfiguration.

Mit der Funktion tag(context) wird das Tagging durchgeführt. Das Tagging wird durch ein Objekt vom Typ $k.TaggingContext gesteuert. Dieses muss bei jedem Aufruf von tag() neu erstellt werden, da es zustandsbehaftet ist. Das TaggingConfiguration-Objekt ist zustandslos und kann wiederverwendet werden.

```
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

Der Intrafind-Tagger muss separat erworben und installiert werden.

Die OpenNLP-Anbindung erfolgt über eine von i-views bereitgestellte REST-Schnittstelle zu OpenNLP.

1.12 Development support

1.12.1 Dev-Tools

Es stehen verschiedene Tools zur Verfügung um die Entwicklung zu erleichtern.

- K-Infinity-Plugin: Bietet Unterstützung für Jetbrains Produkte. Dazu gehört die Synchronisierung von Quelldateien, KJavascript- und KPath-Unterstützung.

1.12.2 Dev-Service

Der Knowledge-Builder bietet die Möglichkeit, Zugriff aus externen Anwendungen zu ermöglichen. Dies ermöglicht z.B. die Synchronisierung mit Entwicklungsumgebungen oder das öffnen bestimmter Elemente einer Applikation aus dem Browser.

Hierfür muss im Knowledge-Builder der Dev-Service gestartet werden. Dazu ruft man zunächst die *Einstellungen* auf und geht dann im Reiter *Persönlich* auf *Dev-Tools*. Hier lässt sich nun ein Port angeben unter denen der Dienst erreichbar sein soll. Über die nebenstehenden Schaltflächen kann der Dienst manuell gestartet und angehalten werden. Ist die Checkbox "Automatisch starten" gesetzt, wird der Service automatisch mit dem Knowledge-Builder gestartet.

Hat der Knowledge-Builder eine INI-Datei (der Standardname ist "kb.ini") kann er die Einstellungen dauerhaft speichern. In der INI-Datei können die Einstellungen aber auch von Hand eingetragen werden:

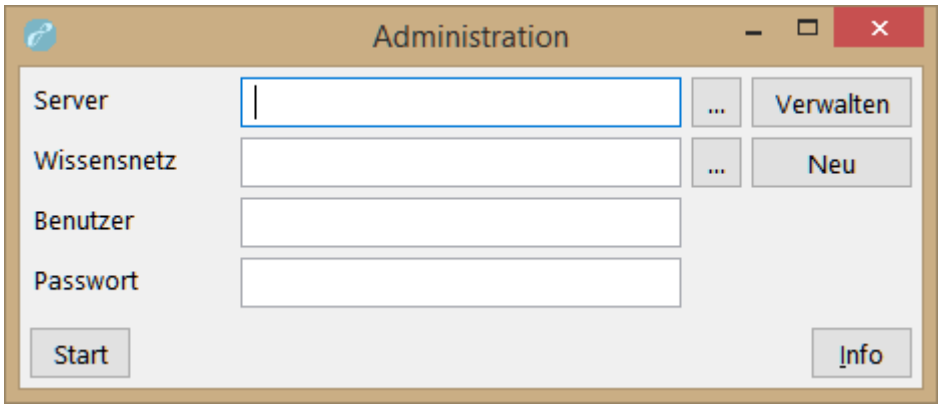
```
[DevService]
autostart=true
port=3050
```

2 Admin-Tool

Mit dem Admin-Tool können neue Wissensnetze angelegt, alle Wissensnetze eines Mediators verwaltet und einzelne Wissensnetze konfiguriert werden.

2.1 Startfenster

Nach dem Start des Admin-Tools (Windows: *admin.exe*, Mac OS: *admin*, Linux: *admin-64.im*) erscheint das **Startfenster**.



2.1.1 Server

Im Freitextfeld **Server** wird die URL des Servers eingegeben. (Falls kein Protokoll angegeben wird, wird das Protokoll cnp:// verwendet). Gültige URLs verwenden eines der Protokolle *[cnp://,cnps://,http:// oder https://] gefolgt von [Rechnername oder IP-Adresse]:[Portnummer]*. Dieses Format entspricht der interface einstellung am Mediator.

Läuft der Mediator, über den Wissensnetze administriert werden sollen, auf dem gleichen Rechner wie das Admin-Tool, kann er auch über den Rechnernamen *localhost* angesprochen werden.

Bleibt das Feld leer, wird auf die Wissensnetze zugegriffen, die relativ zur Position des Admin-Tools im direkten Unterordner *volumes* liegen. Für diese Art des Zugriffes ist kein Mediator notwendig.

Einmal eingegebene Einträge im Freitextfeld werden gespeichert. Über die Schaltfläche **...** können sie in einem separaten Fenster aus einer Liste ausgewählt werden.

Mit der Schaltfläche **Verwalten** gelangt man zur **Serververwaltung**, bei der eine Authentifizierung mit dem Serverpassword benötigt wird.

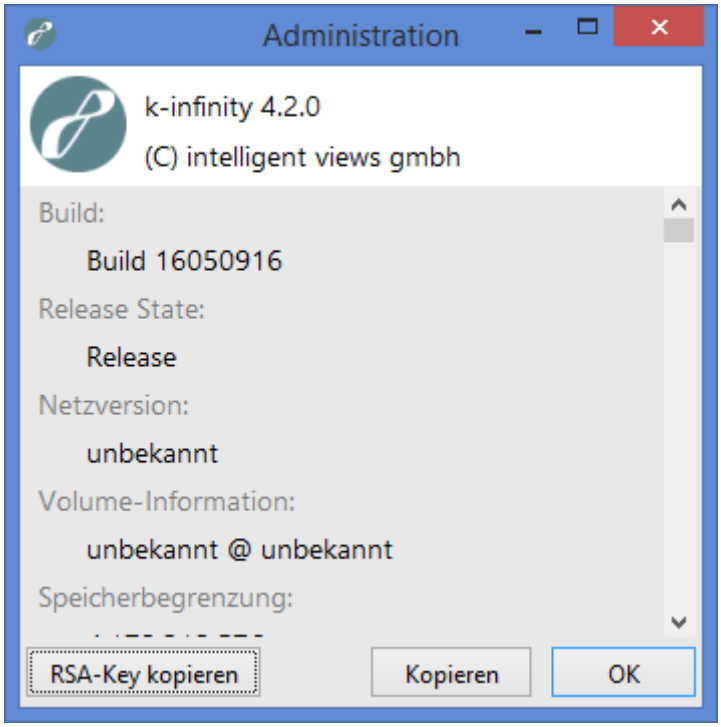
2.1.2 Wissensnetz

Im Freitextfeld **Wissensnetz** wird das Wissensnetz angegeben, das administriert werden soll.

Einmal eingegebene Einträge im Freitextfeld werden gespeichert. Über die Schaltfläche **...** können sie in einem separaten Fenster aus einer Liste ausgewählt werden. Zur Anzeige aller Wissensnetze wird man gegebenenfalls aufgefordert, das Serverpasswort einzugeben.

2.1.3 Info

Über die Schaltfläche **Info** lassen sich in einem eigenen Fenster versionsspezifische Informationen über das Admin-Tool abrufen.



Konkret handelt es sich dabei um

- die Versionsnummer des Admin-Tools (*Build*),
- den Veröffentlichungsstatus des Admin-Tools (*Release State*),
- die vom Admin-Tool maximal nutzbare Menge an Systemarbeitsspeicher in Byte (*Speicherbegrenzung*),
- die Versionsnummer und der digitale Fingerabdruck der vom Admin-Tool verwendeten Ausführungsumgebung (*VM Version*),
- die im Betriebssystem aktive Spracheinstellung (*Locale*),
- die im Admin-Tool verwendeten, mitgelieferten Schriftarten (*Fonts*),
- die mit dem Admin-Tool ausgelieferten Wissensnetzkomponenten inklusive Versionsnummer (*Softwarekomponenten*) und
- die im Admin-Tool verwendeten Smalltalk-Pakete inklusive Versionsnummer (*Pakete*).

Die Angaben zur *Netzversion* und zur *Volume-Information* sind hierbei nicht einschlägig.

Die Informationen werden in einem unsichtbaren Textfeld ausgegeben, welches über ein Kontextmenü verfügt, das per Rechtsklick aktiviert werden kann:

- **Select All** markiert den gesamten Text. Alternativ kann mit dem Mauszeiger ein beliebiger Textausschnitt markiert werden.
- **Copy** kopiert den gewählten Textbereich in die Zwischenablage des Betriebssystems.
- **Find Again** sucht nach dem gewählten Textbereich und findet sein nächstes Auftreten gemäß der Leserichtung.
- **Find** erlaubt die Eingabe einer zu suchenden Zeichenkette in einem separaten Fenster und findet ihr nächstes Auftreten gemäß der Leserichtung relativ zur Position der per Mausklick setzbaren Schreibmarke. Bei der Suche wird Groß- und Kleinschreibung unterschieden.

Über die Schaltfläche **Kopieren** werden alle Informationen in die Zwischenablage des Betriebssystems kopiert.

Die Schaltfläche **RSA-Key kopieren** kopiert den für jedes kompilierte Admin-Tool einzigartigen Schlüssel in die Zwischenablage des Betriebssystems. Dieser Schlüssel kann in die Initialisierungsdatei eines Mediators (Standarddateiname *mediator.ini*) eingetragen werden und beschränkt dadurch den Zugang zu diesem Mediator über ein Admin-Tool auf Admin-Tools mit genau diesem Schlüssel.

Die Schaltfläche **Ok** ermöglicht eine Rückkehr zum Startfenster.

2.1.4 Verwalten, Neu und Weiter

Neu leitet weiter zur Wissensnetzerzeugung.

Verwalten leitet weiter zur Serververwaltung.

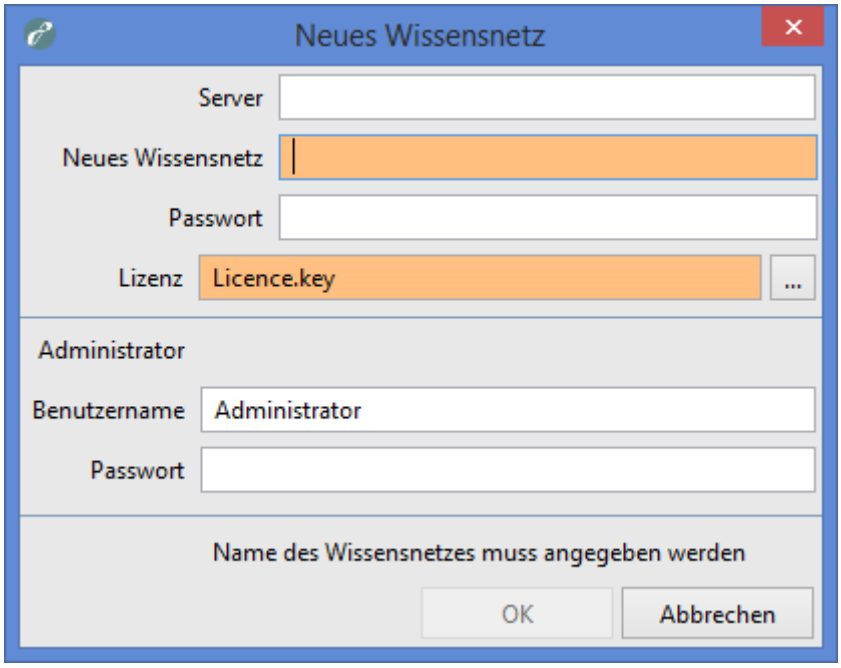
Weiter leitet weiter zur Einzelnetzverwaltung. Hierfür werden die Einträge **Benutzername** und **Passwort** zur Anmeldung mit einem Administratorkonto verwendet.

2.1.5 Ende

Die Schaltfläche **Ende** schließt das Admin-Tool.

2.2 Wissensnetzerzeugung

Das Anlegen eines neuen Wissensnetzes erfolgt über ein eigenes **Netzerzeugungsfenster**. Es kann über die Schaltfläche **Neu** im **Startfenster** erreicht werden. Etwaige Eingaben in den Freitextfeldern **Server** und **Wissensnetz** werden ignoriert.



2.2.1 Server

Im Freitextfeld **Server** wird der Name oder die IP-Adresse des Rechners angegeben, auf dem der Mediator läuft, über den das neue Wissensnetz angelegt werden soll. Sollte dieser nicht über den Standard-Port erreichbar sein, muss außerdem die korrekte Port-Nummer genannt werden. Die Eingabeform hierzu lautet *[Rechnername oder IP-Adresse]:[Portnummer]*.

Läuft der Mediator, über den das neue Wissensnetz angelegt werden soll, auf dem gleichen Rechner wie das Admin-Tool, kann er auch über den Rechnernamen *localhost* angesprochen werden.

Bleibt das Feld leer, wird das Wissensnetz im relativ zur Position des Admin-Tools direkten Unterordner *volumes* erzeugt.

2.2.2 Neues Wissensnetz

Im Freitextfeld **Neues Wissensnetz** wird der Name des Wissensnetzes festgelegt. Die dafür erlaubten Zeichen werden über das Dateisystem des Betriebssystems vorgegeben, auf dem das Wissensnetz gespeichert werden soll. Damit die Daten auch auf unterschiedlichen Dateisystemen gespeichert werden können, gilt:

- maximal 64 Zeichen
- keine Leerzeichen am Anfang oder Ende
- erlaubte Zeichen: große und kleine lateinische Buchstaben, Ziffern, Leerzeichen, !@#$%&'()+-.[]^_'{}~Œœ sowie ASCII-Zeichen 160-255
- nicht erlaubte Zeichenfolgen sind: AUX, CON, NUL, PRN sowie COM0-COM9 und LPT0-LPT9

Die Vergabe eines Namens ist zwingend.

Der Name lässt sich später nur bei Kopiervorgängen des Wissensnetzes oder über Umbenennungen der Datei- und Verzeichnisnamen ändern. Bei einer Änderung ist zu beachten, dass der Name des Wissensnetzes eventuell in Initialisierungsdateien verwendet wird und möglicherweise die Lizenz darauf angepasst wurde.

2.2.3 Passwort (Mediator)

Der Mediator unterstützt eine Authentifizierung über ein Passwort. Ist für den Mediator, über den das neue Wissensnetz angelegt werden soll, ein Passwort gesetzt, muss es im Freitextfeld **Passwort**, welches sich zwischen den Feldern **Neues Wissensnetz** und **Lizenz** befindet, eingegeben werden. Ist kein Passwort vergeben, muss das Freitextfeld leer bleiben.

2.2.4 Lizenz

Ein Wissensnetz muss eine gültige Lizenz besitzen, damit der Knowledge-Builder und andere Software-Komponenten (mit Ausnahme des Admin-Tools) damit arbeiten können. Über die Schaltfläche **...** kann auf das Dateisystem des Betriebssystems zugegriffen werden, um einen Lizenzschlüssel (Dateiname: *[Lizenzname].key*) zu laden.

2.2.5 Benutzername

Im Freitextfeld **Benutzername** wird der Name des ersten im Wissensnetz registrierten Nutzers festgelegt. Die Art und Menge der dafür erlaubten Zeichen ist nicht beschränkt.Die Voreinstellung Administrator ist lediglich ein Vorschlag. Dieses Feld darf nicht leer bleiben.

Der Name kann im Admin-Tool oder im Knowledge-Builder nachträglich geändert werden. Der hierüber angelegte Nutzer besitzt automatisch Administratorrechte.

2.2.6 Passwort (Benutzer)

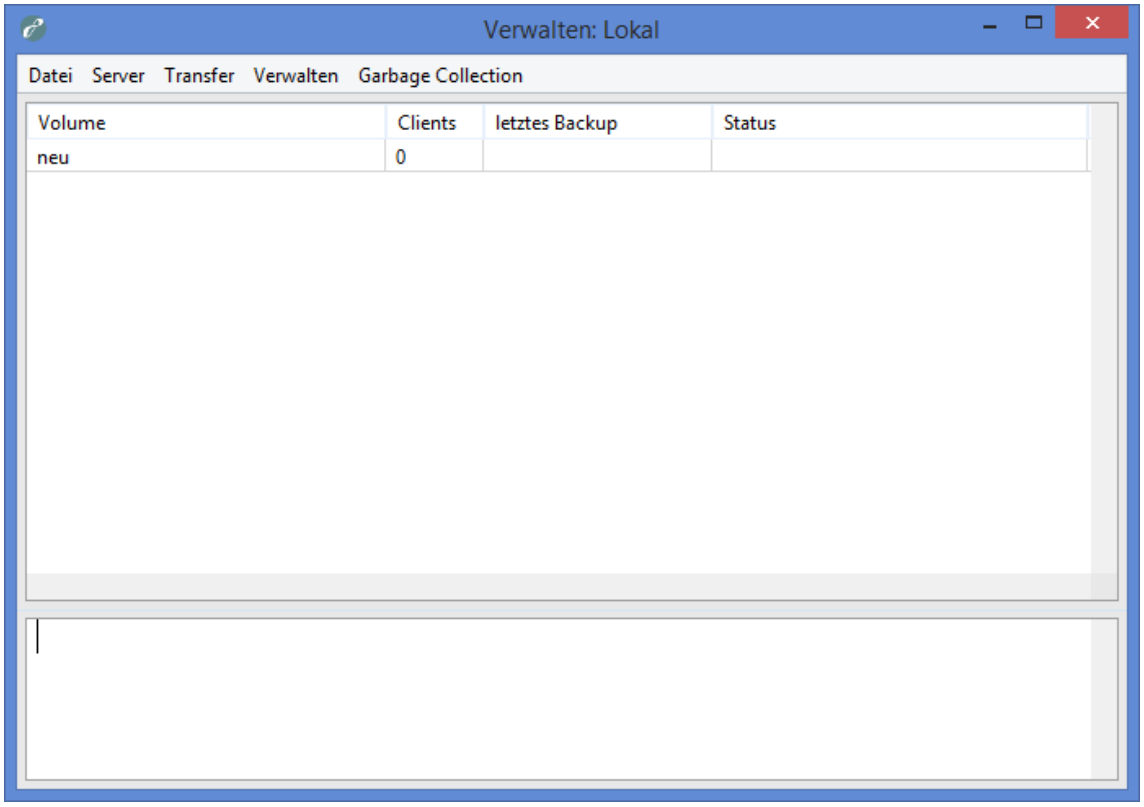
Im Freitextfeld **Passwort** kann ein Passwort für den ersten im Wissensnetz registrierten Nutzer vergeben werden. Dieses Passwort wird später gebraucht, wenn dieser Nutzer sich im Knowledge-Builder oder im Admin-Tool für das Wissensnetz anmelden will.

2.2.7 OK und Abbrechen

Die Schaltfläche **Ok** erzeugt das Wissensnetz unter Einbeziehung der eingegebenen Daten. Die Schaltfläche **Abbrechen** bricht den Vorgang ab. In beiden Fällen erfolgt eine Rückkehr zum **Startfenster**.

2.3 Server administration

Die Gesamtnetzverwaltung erlaubt die Administration aller Wissensnetze eines Mediators beziehungsweise des lokalen Unterordners *volumes*. Sie kann über die Schaltfläche **Verwalten** im **Startfenster** erreicht werden. Erforderlich ist hierzu eine entsprechende Eingabe im Feld **Server** des **Startfensters**. Etwaige Eingaben im Feld **Wissensnetz** des **Startfensters** werden ignoriert. Werden die zu administrierenden Wissensnetze über einen Mediator angesprochen, muss außerdem das korrekte Mediator-Passwort in einem eigenen Fenster angegeben werden.



Das **Gesamtnetzverwaltungsfenster** besteht aus einer tabellarischen **Netzübersicht**, einem **Nachrichtenfeld** und einer **Menüzeile**.

2.3.1 Netzübersicht

Die tabellarische **Netzübersicht** gibt Aufschluss über

- den Namen (*Volume*),
- die Anzahl an gegenwärtig aktiven Nutzern (*Clients*),
- das Datum und die Uhrzeit der letzten Sicherung (*letztes Backup*) sowie
- die letzte Statusmeldung (*Status*) des jeweiligen Netzes.

Die einzelnen Spalten sind über einen Klick auf den Spaltenkopf sortierbar.

Die Daten werden nur beim Auslösen von Operationen aktualisiert und sind deswegen nicht immer aktuell. Eine manuelle Aktualisierung kann jederzeit über den Menüpunkt **Server** –> **Aktualisieren** forciert werden.

2.3.2 Nachrichtenfeld

Das **Nachrichtenfeld** gibt alle Statusmeldungen aller Netze aus. Statusmeldungen werden durch das Auslösen von Aktivitäten im Admin-Tool erzeugt. Sie gehen beim Schließen des Admin-Tools verloren. Um dies zu verhindern, können sie über den Menüpunkt **Datei** –> **Administrations-Log** exportiert werden. Das **Nachrichtenfeld** ist zwar editierbar, Änderungen werden beim Export aber ignoriert.

2.3.3 Menüzeile

Die **Menüzeile** besteht aus den folgenden Menüreitern:

2.3.3.1 Datei

Administrations-Log speichern speichert alle Einträge im Nachrichtenfenster in einer Textdatei (Dateistandardname: *admin.log*) ab. Name und Speicherort können in einem Speicherdialog frei gewählt werden. Diese Operation setzt voraus, dass das Admin-Tool mit einem Mediator verbunden ist.

Abmelden schließt die Serververwaltung und öffnet wieder das Anmeldefenster-

Beenden schließt die Serververwaltung

2.3.3.2 Server

Aktualisieren ermittelt die in der **Netzübersicht** im **Gesamtnetzverwaltungsfenster** erhobenen Daten neu.

Ini-Datei neu einlesen veranlasst den Server, seine ini-Datei neu einzulesen. Dabei können nicht alle Optionen im laufenden Betrieb aktualisiert werden. Der Server leifert eine Mitteilung über aktualisierte Optionen.

Log herunterladen erzeugt eine Kopie der üblicherweise im Ordner des verbundenen Mediators liegenden Mediator-Protokolldatei (Standarddateiname: *mediator.log*). Name und Speicherort der Kopie können in einem Speicherdialog frei gewählt werden. In der Mediator-Protokolldatei wird ein Protokoll über alle Aktivitäten des Mediators seit seiner ersten Inbetriebnahme geführt.

Serververbindungen gibt im **Nachrichtenfeld** die Nummer und die IP-Adresse aller gegenwärtig über den verbundenen Mediator in Wissensnetzen angemeldeten Software-Komponenten (außer dem Blob-Service) aus und gruppiert diese nach Wissensnetzen. Die Nummer wird vom Mediator fortlaufend generiert und bei jeder Neuanmeldung einer Software-Komponente neu vergeben.

2.3.3.3 Administrate

Volume herunterladen erzeugt eine Kopie des in der **Netzübersicht** ausgewählten Wissensnetzes und speichert sie lokal im relativ zur Position des Admin-Tools liegenden Unterordner *volumes*. In einem separat erscheinenden Freitextfeld kann ein neuer Name für diese Kopie vergeben werden.

Volume kopieren erzeugt eine Kopie des in der **Netzübersicht** ausgewählten Wissensnetzes und speichert sie im gleichen Ordner wie das Originalnetz. In einem separat erscheinenden Freitextfeld muss ein neuer Name für diese Kopie vergeben werden.

Volume hochladen erzeugt eine Kopie eines ausgewählten lokalen Wissensnetzes und speichert sie im relativ zum verbundenen Mediator liegenden Unterordner *volumes*. In einem separat erscheinenden Freitextfeld kann ein neuer Name für diese Kopie vergeben werden. Die Auswahl des lokalen Wissensnetzes, das im relativ zur Position des Admin-Tools liegenden Unterordner *volumes* abgelegt sein muss, erfolgt über ein separates Auswahlfenster.

Volume austauschen erzeugt eine Kopie eines ausgewählten lokalen Wissensnetzes und überschreibt damit das in der **Netzübersicht** ausgewählte Wissensnetz. Die Kopie erhält

dabei den Namen des überschriebenen Wissensnetzes. Die Auswahl des lokalen Wissensnetzes, das im relativ zur Position des Admin-Tools liegenden Unterordner *volumes* abgelegt sein muss, erfolgt über ein separates Auswahlfenster.

In Folge der über Transfer-Operationen ausgelösten Kopierprozesse wird die Blockbelegung der Cluster und Blobs innerhalb der Wissensnetzkopien neu festgelegt und dabei deren Platzverbrauch optimiert. Der hierdurch bewirkte Komprimierungseffekt ist der gleiche wie über die Operation **Verwalten** -> **Volume komprimieren**.

Mit Ausnahme der Operation **Volume kopieren** setzen alle diese Operationen voraus, dass das Admin-Tool mit einem Mediator verbunden ist.

2.3.3.4 Verwalten

Admintool öffnen meldet sich im ausgewählten Volume mit dem Admintool an. Hiefür ist keine Authentifizierung im Volume nötig - die Mediator-Authentifizierung genügt.

Auf diese Art kann bei einem vergessenen Administrator-Passwort auf die Benutzerverwaltung des Volumes zugegriffen werden.

Backup erstellen erzeugt eine Sicherungskopie des in der **Netzübersicht** ausgewählten Wissensnetzes und speichert sie im relativ zur Position dieses Wissensnetzes liegenden Parallelordner *backup*. Für jede Sicherungskopie wird dort ein eigener Unterordner angelegt, in dessen Bezeichnung der Zeitpunkt der Kopieerstellung auf eine Sekunde genau eingearbeitet ist.Jede Sicherungskopie ist eine vollständige Kopie des Originalnetzes.

Vor Erzeugung der Sicherungskopie wird über ein eigenes Fenster erfragt, ob der Nutzer bis zum Abschluss des Kopiervorganges warten will. Gegebenenfalls wird bis zu diesem Zeitpunkt die weitere Nutzung des Admin-Tools blockiert. Andernfalls startet der Kopiervorgang im Hintergrund und eine Nachricht über den Fortgang des Kopiervorganges oder dessen Fertigstellung unterbleibt.

Backup wiederherstellen erzeugt eine Kopie einer ausgewählten Sicherungskopie und speichert sie im gleichen Ordner wie die in der **Netzübersicht** dargestellten Wissensnetze. In einem separat erscheinenden Freitextfeld muss ein neuer Name für diese Kopie vergeben werden. Die Auswahl der Sicherungskopie, die in einem Unterordner des Ordners *backup* abgelegt sein muss, der parallel zur Position der in der **Netzübersicht** dargestellten Wissensnetze liegt, erfolgt über zwei separate Auswahlfenster, in denen zuerst das Wissensnetz und danach die nach Erstellungszeiten sortierte Version ausgewählt werden.

Backup löschen löscht eine ausgewählte Sicherungskopie. Die Auswahl dieser Sicherungskopie, die in einem Unterordner des Ordners *backup* abgelegt sein muss, der parallel zur Position der in der **Netzübersicht** dargestellten Wissensnetze liegt, erfolgt über zwei separate Auswahlfenster, in denen zuerst das Wissensnetz und danach die nach Erstellungszeiten sortierte Version ausgewählt werden.

Die Blockbelegung der Cluster und Blobs innerhalb des Originalwissensnetzes wird bei der Erstellung einer Wissensnetzkopie nicht verändert. Der durch Backup-Operationen ausgelöste Kopiervorgang erzeugt deswegen keinen Komprimierungseffekt.

Volume löschen löscht das in der **Netzübersicht** ausgewählte Wissensnetz.

Volume komprimieren verringert den Platzbedarf des in der **Netzübersicht** ausgewählten Wissensnetzes. Dies geschieht über die Beseitigung ungenutzter Binnenblöcke. Durch Umkopierprozesse der Cluster und Blobs werden zunächst alle ungenutzten Blöcke an das Dateiende verschoben und dann im Dateisystem des Betriebssystems freigegeben.

Volume-Storage aktualisieren aktualisiert die Version des Blockdateisystems des in der

Netzübersicht ausgewählten Wissensnetzes. Wird das Wissensnetz über einen Mediator angesprochen, wird die darin enthaltene Version verwendet, ansonsten wird die im Admin-Tool mitgelieferte Version verwendet. Die Aktualisierung ermöglicht eine schnellere Speicherung von Indexstrukturen. Sie ist möglich für Wissensnetze, deren i-views-Core-Komponente älter als 4.2 ist.

2.3.3.5 Garbage Collection

Die Garbage-Collection ist ein Verfahren, das nicht mehr referenzierte Objekte (nach programmierterminologischer Lesart) in einem Wissensnetz löscht und damit den Speicherverbrauch des Wissensnetzes minimiert. Die Nutzung der Garbage-Collection setzt voraus, dass das zu bereinigende Wissensnetz über einen Mediator angesteuert wird.

Start startet eine neue Speicherbereinigung für das in der **Netzübersicht** ausgewählte Wissensnetz oder setzt eine pausierte Speicherbereinigung fort. Es erfolgt keine Rückmeldung, wann der Vorgang abgeschlossen ist. Der Stand des Fortschritts kann über den Menüpunkt **Status** in Erfahrung gebracht werden.

Pause unterbricht die Durchführung der aktiven Speicherbereinigung für das in der **Netzübersicht** ausgewählte Wissensnetz.

Anhalten bricht die Durchführung der aktiven Speicherbereinigung für das in der **Netzübersicht** ausgewählte Wissensnetz ab.

Status schreibt den aktuellen Zustand der Speicherbereinigung für das in der **Netzübersicht** ausgewählte Wissensnetz in die Statusspalte der **Netzübersicht** und in das **Nachrichtenfeld**. Ist eine Speicherbereinigung aktiv, erfolgt zusätzlich eine Rückmeldung über den Stand des Fortschritts in Form einer Prozentangabe.

2.4 Einzelnetzverwaltung

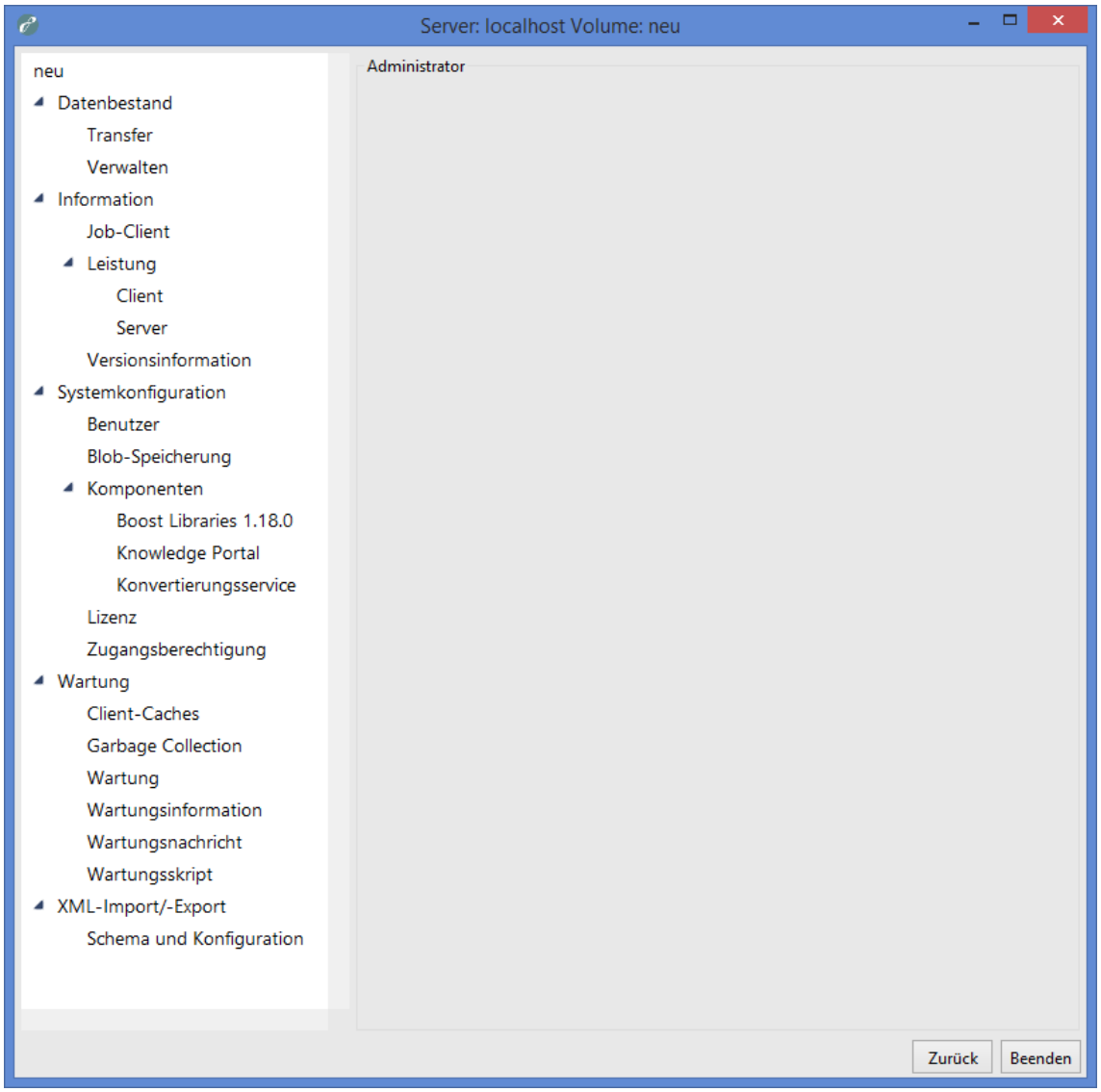
Die Einzelnetzverwaltung erlaubt die Administration eines einzelnen Wissensnetzes. Sie kann über die Schaltfläche **Start** im Startfenster erreicht werden. Erforderlich sind hierzu entsprechende Eingaben in den Feldern **Server**, **Wissensnetz**, **Benutzer** und **Passwort** des Startfensters.

2.4.1 Nutzerauthentifizierung

Für den Zutritt zum **Einzelnetzverwaltungsfenster** ist die Anmeldung eines Nutzers mit Administratorrechten notwendig.

Falls man keinen Zugang mehr zum Wissensnetz mehr hat, kann man mit Hilfe der Anmeldung in der **Serververwaltung** über die Authentifizierung am Server Zugang zum Wissensnetz bekommen.

2.4.2 Einzelnetzverwaltungsfenster



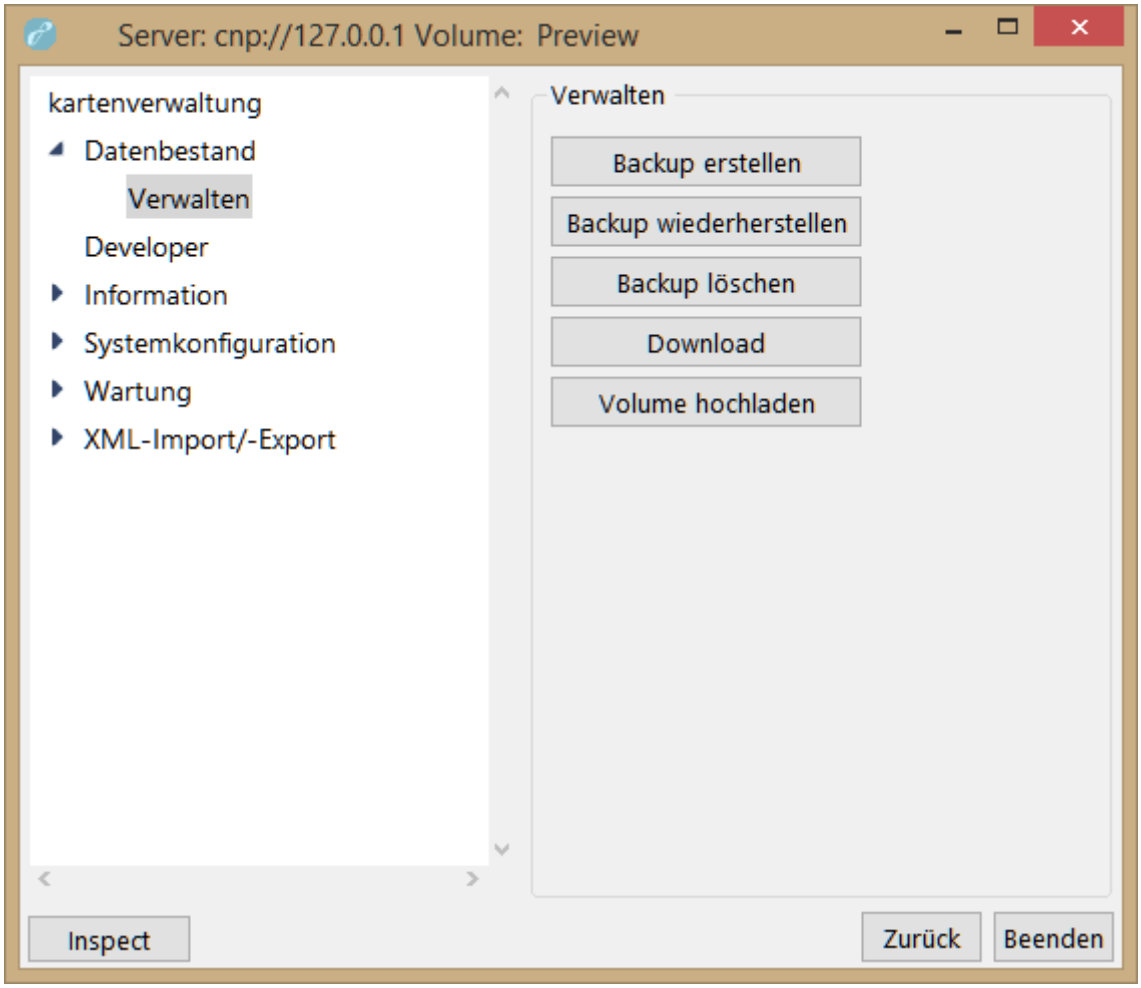
Das **Einzelnetzverwaltungsfenster** verfügt über eine mehrstufig gegliederte Menüliste links und ein Operationsfenster rechts. Der Inhalt des Operationsfensters richtet sich nach dem gewählten Menüpunkt in der Menüliste.

Die Schaltfläche **Zurück** wechselt zum Startfenster zurück.

Die Schaltfläche **Beenden** schließt das Admin-Tool.

Wird das zu administrierende Wissensnetz ohne Mediator angesprochen, ist ein Zutritt anderer Nutzer zum Wissensnetz über den Knowledge-Builder oder eine weitere Instanz des Admin-Tools nicht möglich, solange das **Einzelnetzverwaltungsfenster** geöffnet ist.

2.4.2.1 Administrate data



Backup erstellen erzeugt eine Sicherungskopie des Wissensnetzes und speichert sie (auf dem Server) im relativ zur Position dieses Wissensnetzes liegenden Parallelordner *backup*. Für jede Sicherungskopie wird dort ein eigener Unterordner angelegt, in dessen Bezeichnung der Zeitpunkt der Kopierstellung auf eine Sekunde genau eingearbeitet ist.Jede Sicherungskopie ist eine vollständige Kopie des Originalnetzes.

Vor Erzeugung der Sicherungskopie wird über ein eigenes Fenster erfragt, ob der Nutzer bis zum Abschluss des Kopiervorganges warten will. Gegebenenfalls wird bis zu diesem Zeitpunkt die weitere Nutzung des Admin-Tools blockiert. Andernfalls startet der Kopiervorgang im Hintergrund und eine Nachricht über den Fortgang des Kopiervorganges oder dessen Fertigstellung unterbleibt.

Backup wiederherstellen ersetzt das aktuelle Wissensnetz durch eine Sicherungskopie (danach wird man automatisch abgemeldet). Die Auswahl der Sicherungskopie erfolgt über die Zeitpunkte der jeweiligen Backups.

Backup löschen löscht eine einzelne Sicherungskopie dieses Wissensnetzes.

Die Blockbelegung der Cluster und Blobs innerhalb des Originalwissensnetzes wird bei der Erstellung einer Wissensnetzkopie nicht verändert. Der durch Backup-Operationen ausgelöste Kopiervorgang erzeugt deswegen keinen Komprimierungseffekt.

Download erzeugt eine Kopie des Wissensnetzes und speichert sie lokal im relativ zur Posi-

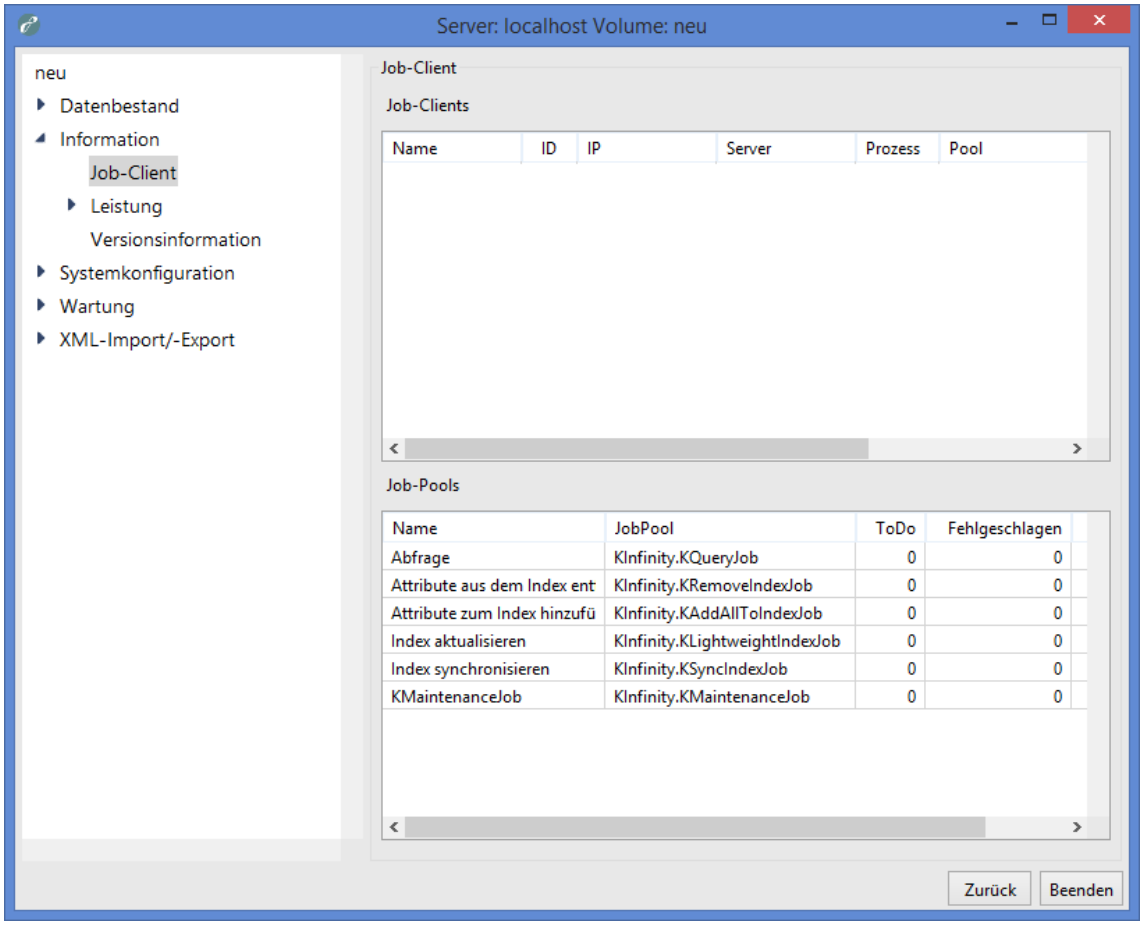
tion des Admin-Tools liegenden Unterordner *volumes*. In einem separat erscheinenden Freitextfeld kann ein neuer Name für diese Kopie vergeben werden.

Volume hochladen überträgt ein lokal gespeichertes Netz und ersetzt das aktuelle Wissensnetz durch dieses (danach wird man automatisch abgemeldet)

2.4.2.2 Information

2.4.2.2.1 Job-Client

Um den Knowledge-Builder von bestimmten rechenintensiven Prozessen wie der Indizierung und der Abfrage von Wissensnetzen sowie der Ausführung von Skripten zu entlasten, können diese Prozesse teils wahlweise, teils exklusiv als Aufgaben (Jobs) von Job-Clients (einem Software-Dienst) übernommen werden. Dazu muss über die Benutzeroberfläche des Knowledge-Builders oder per Skript eine Aufgabe ausgelöst oder es müssen die Bedingungen für ihre Auslösung festgelegt werden. Außerdem ist mindestens ein Job-Client zu konfigurieren und zu starten, der Aufgaben dieses Aufgabentyps (Job Pool) übernehmen kann. Das Admin-Tool übernimmt hierbei überwiegend eine Beobachtungsfunktion. Unerledigte Aufgaben erscheinen im Knowledge-Builder unter dem Eintrag *Aufträge* in der Rubrik *Technik*. Die Verwaltung von Job-Clients über das Admin-Tool setzt voraus, dass das Admin-Tool mit einem Mediator verbunden ist.



Die tabellarische **Job-Clients-Übersicht** zeigt für jeden aktuell laufenden Job-Client

- seinen Namen im Format [Job-Client-Name]@[Mediator-Name] (*Name*),
- seine Job-Client-Nummer (*ID*),

- seine IP-Adresse (*IP*),
- den Namen des mit ihm verbundenen Mediators (*Server*),
- seine vom Betriebssystem vergebene Prozessnummer (*Prozess*),
- die ihm zugeordneten Aufgabentypen (*Pool*),
- seinen Arbeitsstatus (*Status*) und
- die Anzahl von ihm erledigter Aufgaben (*Erledigt*).

Die Job-Client-Nummer wird vom Mediator fortlaufend generiert und bei jeder Neuanmeldung neu vergeben. Der Job-Client-Name und die dem Job-Client zugeordneten Aufgabentypen werden in der Initialisierungsdatei des jeweiligen Job-Clients (Standarddateiname: *jobclient.ini*) unter dem Schlüssel *name* respektive dem Schlüssel *jobPools* festgelegt. In der Job-Client-Übersicht wird jeder Aufgabentyp eines Job-Clients in einer eigenen Zeile dargestellt, so dass ein Job-Client regelmäßig mehrere Zeilen belegt.

Die einzelnen Spalten der **Job-Clients-Übersicht** sind über einen Klick auf den Spaltenkopf sortierbar. Per Rechtsklick auf eine Zeile kann außerdem ein Kontextmenü geöffnet werden:

- **Informationen anzeigen** stellt alle in der ausgewählten Zeile gelisteten Daten mit Ausnahme des Aufgabentyps und der erledigten Aufgabenanzahl in einem neuen Fenster dar. Ergänzt werden
  - Datum und Uhrzeit des letzten Startzeitpunkts des Job-Clients (*startUpTime*),
  - die ihm zur Verfügung stehende maximal nutzbare Menge an Systemarbeitsspeicher in Byte (*max Memory*),
  - der Name seiner Protokolldatei (*logFileName*) und
  - sein spezieller Name, unter dem er zum Beenden gezwungen werden kann (eine Verkettung der Zeichenkette „jobclient“ und der Job-Client-Nummer) (*shutDownString*).
- Die Daten können dort in die Zwischenablage des Betriebssystems kopiert (Schaltfläche **In Zwischenablage kopieren**) oder über einen Speicherdialog als frei benennbare Textdatei an eine beliebige Stelle exportiert werden (Schaltfläche **Speichern**).
- Die über den Menüpunkt **Informationen anzeigen** ausgelöste Operation kann alternativ über einen Doppelklick auf eine Zeile in der Job-Clients-Übersicht erwirkt werden.
- **Job-Client entfernen** beendet den in der **Job-Clients-Übersicht** ausgewählten Job-Client.
- **Alle Job-Clients** entfernen beendet alle in der **Job-Clients-Übersicht** gelisteten Job-Clients.

Die tabellarische **Job-Pools-Übersicht** listet alle Aufgabentypen, die in der **Job-Clients-Übersicht** mindestens einem Job-Client zugeordnet sind. Für jeden Aufgabentyp werden

- seine Bezeichnung (*Name*),
- seine in der Job-Clients-Initialisierungsdatei verwendete technische Bezeichnung (*JobPool*),
- die Anzahl unerledigter Aufgaben dieses Aufgabentyps (*ToDo*),
- die Anzahl fehlgeschlagener Aufgaben dieses Aufgabentyps (*Fehlgeschlagen*) und
- die Anzahl der ihm zur Verfügung stehender Job-Clients (*Job-Clients*)

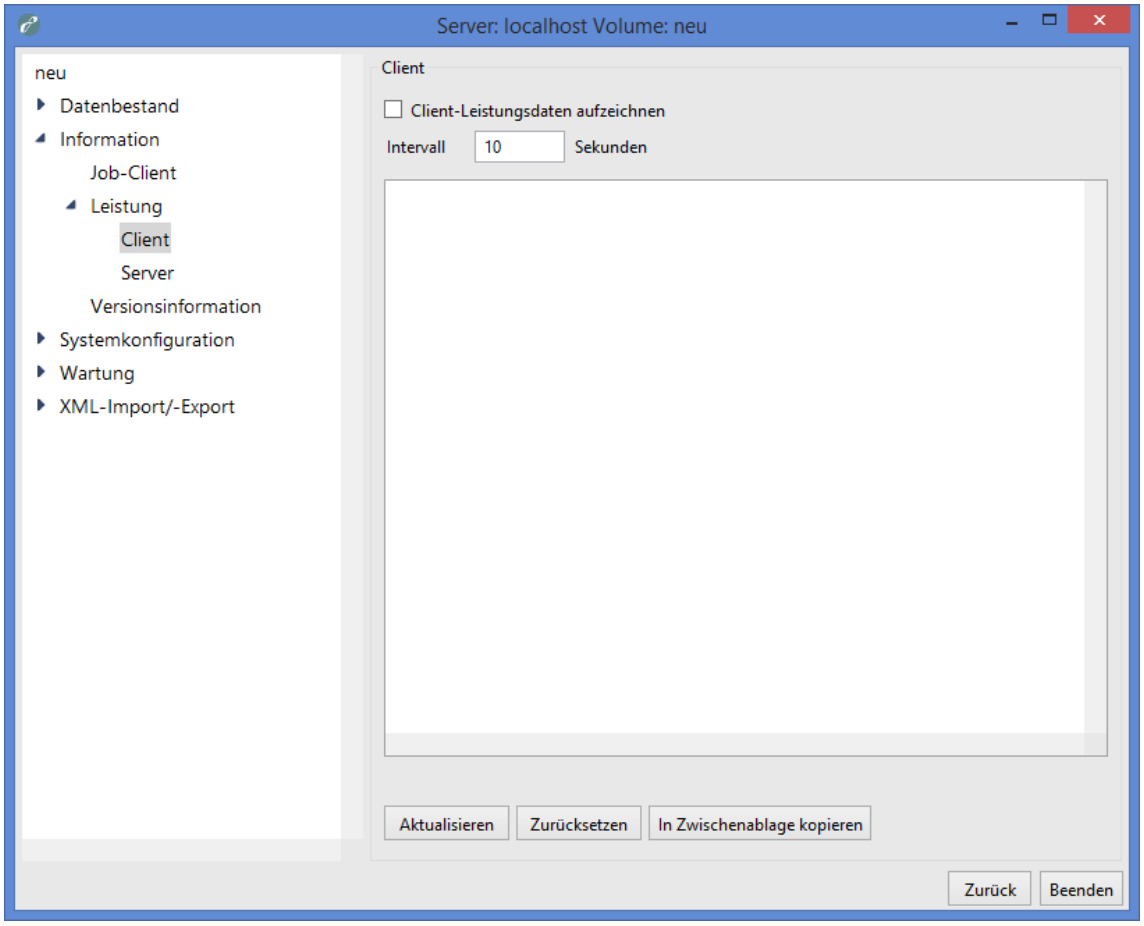
genannt.

Die einzelnen Spalten der **Job-Pools-Übersicht** sind über einen Klick auf den Spaltenkopf sortierbar. Per Rechtsklick auf eine Job-Client-Zeile kann außerdem ein Kontextmenü geöffnet werden:

- **Job-Pool leeren** löscht alle unerledigten und fehlgeschlagenen Aufgaben des in der **Job-Pools-Übersicht** ausgewählten Aufgabentyps. Diese Operation ist nur möglich, wenn kein Job-Client läuft.
- **Zu ignorierende Fehlermeldungen konfigurieren** ermöglicht die Unterdrückung bestimmter Fehlermeldungen bei der Ausführung von Aufgaben des in der **Job-Pools-Übersicht** ausgewählten Aufgabentyps. Wird eine Fehlermeldung auf diese Weise unterdrückt, bleibt die dem Fehler zugehörige Aufgabe bei der Ermittlung der Anzahl fehlgeschlagener Aufgaben in der **Job-Pools-Übersicht** unberücksichtigt. Diese Operation ist nur möglich, wenn bereits Aufgaben des in der **Job-Pools-Übersicht** ausgewählten Aufgabentyps auf ihre Bearbeitung warten oder bereits bearbeitet wurden.
- Die Verwaltung der zu unterdrückenden Fehlermeldungen erfolgt in einem separaten Fenster:
  - In der alphabetisch sortierten **Fehlermeldungsliste** werden alle zu unterdrückenden Fehlermeldungen gelistet. Eine Fehlermeldung wird unterdrückt, wenn ihr Ausgabetext mit einem Text in der **Fehlermeldungsliste** übereinstimmt.
  - **+** erlaubt die Eingabe einer zu unterdrückenden Fehlermeldung über ein eigenes Fenster. Die eingegebene Fehlermeldung erscheint in der **Fehlermeldungsliste**.
  - **...** erlaubt die Änderung der in der **Fehlermeldungsliste** ausgewählten Fehlermeldung.
  - **-** löscht die in der **Fehlermeldungsliste** ausgewählte Fehlermeldung.

2.4.2.2.2 Leistung

Client



Client-Leistungsdaten aufzeichnen startet und beendet die Erhebung diverser Leistungskennzahlen, die an Aktivitäten der mit dem Wissensnetz verbundenen Software-Komponenten gekoppelt sind. Diese Leistungskennzahlen können zur Performanzanalyse verwendet werden.

Intervall setzt die notwendige Zeitspanne in Sekunden, bis eine Software-Komponente erneut ein Datenpaket mit Leistungskennzahlen an das Admin-Tool sendet. Es kann nach dem Start der Aufzeichnung nicht mehr verändert werden. Die Vorerinstellung liegt bei 10 Sekunden.

In der **Leistungskennzahlenübersicht** werden die Leistungskennzahlen in geschachtelten Listenpunkten ausgegeben. Per Klick auf die links neben den Rubriken befindlichen Dreiecksymbole können Listenunterpunkte ein- und ausgeklappt werden. Alternativ lässt sich dies über ein Kontextmenü realisieren, das über einen Klick mit der rechten Maustaste auf einen Listenpunkt aufgerufen werden kann:

- **Expand** klappt alle direkten Listenunterpunkte des gewählten Listenpunkts aus.
- **Expand fully** klappt alle direkten und alle indirekten Listenunterpunkte des gewählten Listenpunkts aus.
- **Contract fully** klappt alle Listenunterpunkte des gewählten Listenpunkts wieder ein.

Mit einem Doppelklick auf einen Listenpunkt lassen sich alle darunter abgelegten Leistungskennzahlen in einem separaten Fenster auf einen Blick darstellen. Dort können sie in die Zwischenablage des Betriebssystems kopiert (Schaltfläche **In Zwischenablage kopieren**) oder über einen Speicherdialog als frei benennbare Textdatei an eine beliebige Stelle ex-

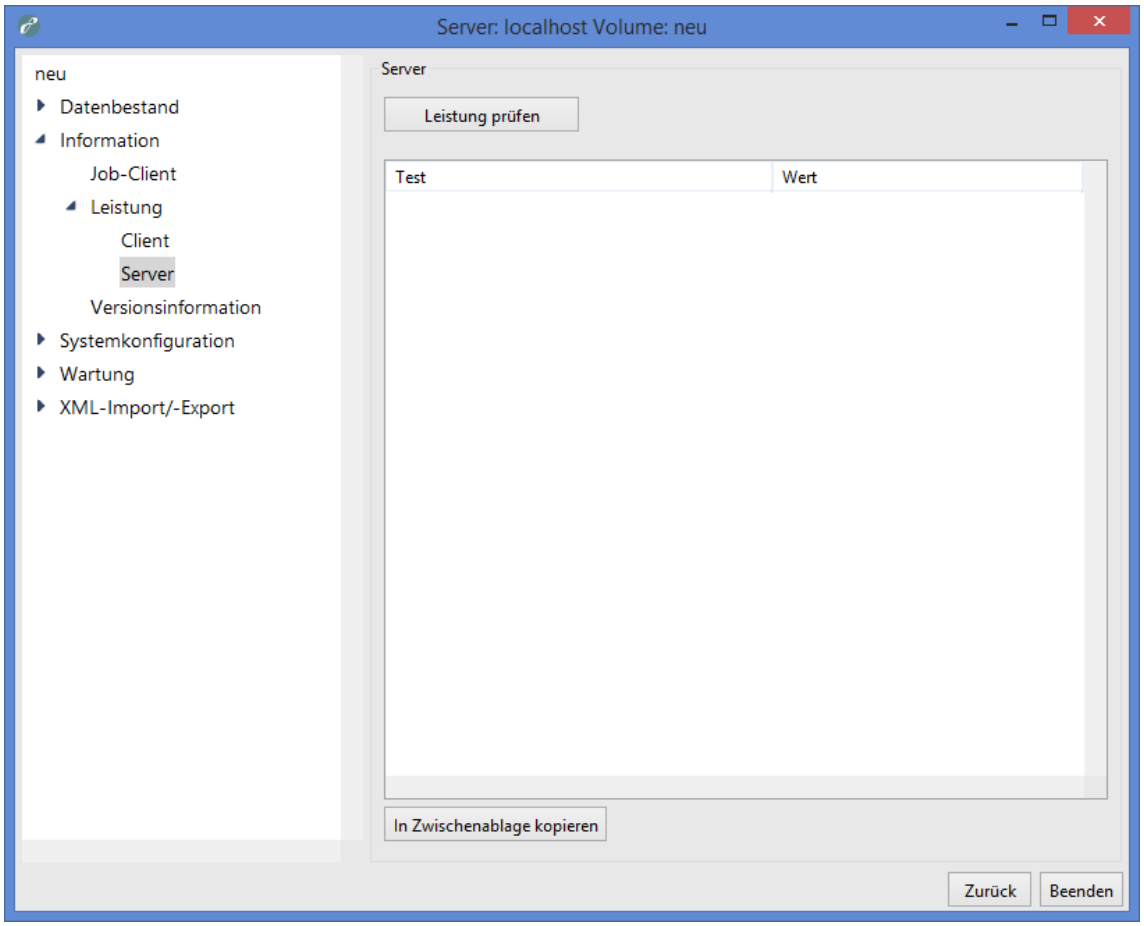
portiert werden (Schaltfläche **Speichern**).

Aktualisieren aktualisiert die in der **Leistungskennzahlenübersicht** dargestellten Leistungskennzahlen.

Zurücksetzen löscht die in der **Leistungskennzahlenübersicht** dargestellten Leistungskennzahlen.

In Zwischenablage kopieren kopiert diein der **Leistungskennzahlenübersicht** dargestellten Leistungskennzahlen in die Zwischenablage des Betriebssystems.

Server



Leistung prüfen startet einen Testvorgang, der die Performanz des angeschlossenen Mediators auswertet. Dabei werden vier Anfragen an den Mediator geschickt und die an das Admin-Tool gesendeten Antworten ausgewertet. Gemessen werden

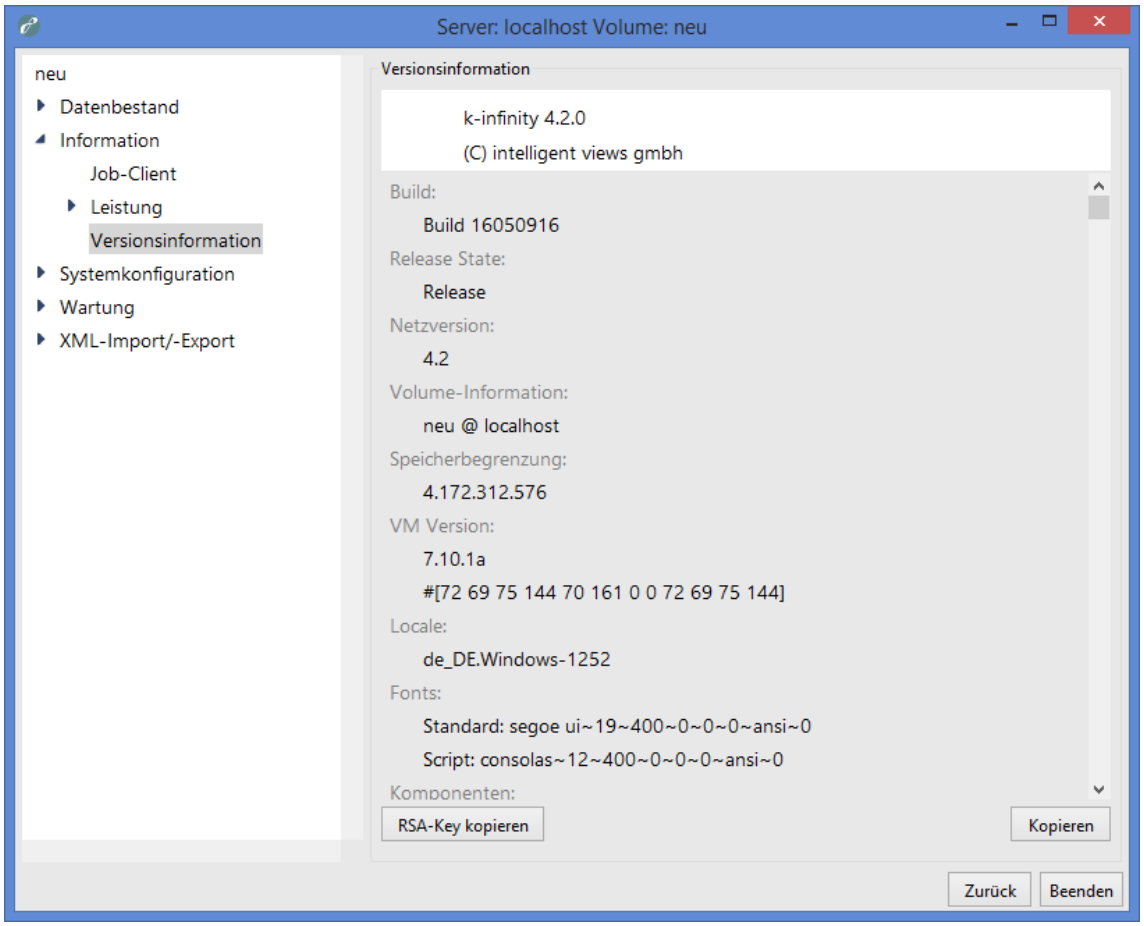
- die Zeiten bis zur Rücksendung einer kleinen Datei (*Roundtrip: Blob*) und
- des Ergebnisses einer Indexsuchanfrage (*Roundtrip: RPC*) sowie
- die durchschnittliche Übertragungsgeschwindigkeite beim Versand mehrerer Dateien der Größe 1 MB (*Throughput: Blob (1.0 MB)*) und
- die durchschnittliche Übertragungsgeschwindigkeite beim Versand mehrerer Dateien der Größe100KB (*Throughput: Blob (100.0 KB)*).

Die Testergebnisse werden in die bereitgestellte **Ergebnisliste** geschrieben. Die einzelnen Spalten der Tabelle sind über einen Klick auf den Spaltenkopf sortierbar.

In Zwischenablage kopieren kopiert die Testergebnisse der **Ergebnisliste** als Reintext in die Zwischenablage des Betriebssystems.

2.4.2.2.3 Versionsinformation

Über diesen Menüpunkt lassen sich diverse versionsspezifische Informationen über das Wissensnetz und das Admin-Tool abrufen.



Konkret handelt es sich dabei um

- die Versionsnummer des Admin-Tools (*Build*),
- den Veröffentlichungsstatus des Admin-Tools (*Release*),
- die Versionsnummer des Wissensnetzes (Netzversion), die Namen des Wissensnetzes und des verwendeten Mediators (*Volume-Information*),
- die vom Admin-Tool maximal nutzbare Menge an Systemarbeitsspeicher in Byte (*Speicherbegrenzung*),
- die Versionsnummer und der digitale Fingerabdruck der vom Admin-Tool verwendeten Ausführungsumgebung (*VM Version*),
- die im Betriebssystem aktive Spracheinstellung (*Locale*),
- die im Admin-Tool verwendeten, mitgelieferten Schriftarten (*Fonts*),
- die im Wissensnetz installierten Wissensnetzkomponenten inklusive Versionsnummer (*Softwarekomponenten*) und

- die im Admin-Tool verwendeten Smalltalk-Pakete inklusive Versionsnummer (*Pakete*).

Die Informationen werden in einem unsichtbaren Textfeld ausgegeben, welches über ein Kontextmenü verfügt, das per Rechtsklick aktiviert werden kann:

- **Select All** markiert den gesamten Text. Alternativ kann mit dem Mauszeiger ein beliebiger Textausschnitt markiert werden.
- **Copy** kopiert den gewählten Textbereich in die Zwischenablage des Betriebssystems.
- **Find Again** sucht nach dem gewählten Textbereich und findet sein nächstes Auftreten gemäß der Leserichtung.
- **Find** erlaubt die Eingabe einer zu suchenden Zeichenkette in einem separaten Fenster und findet ihr nächstes Auftreten gemäß der Leserichtung relativ zur Position der per Mausklick setzbaren Schreibmarke. Bei der Suche wird Groß- und Kleinschreibung unterschieden.

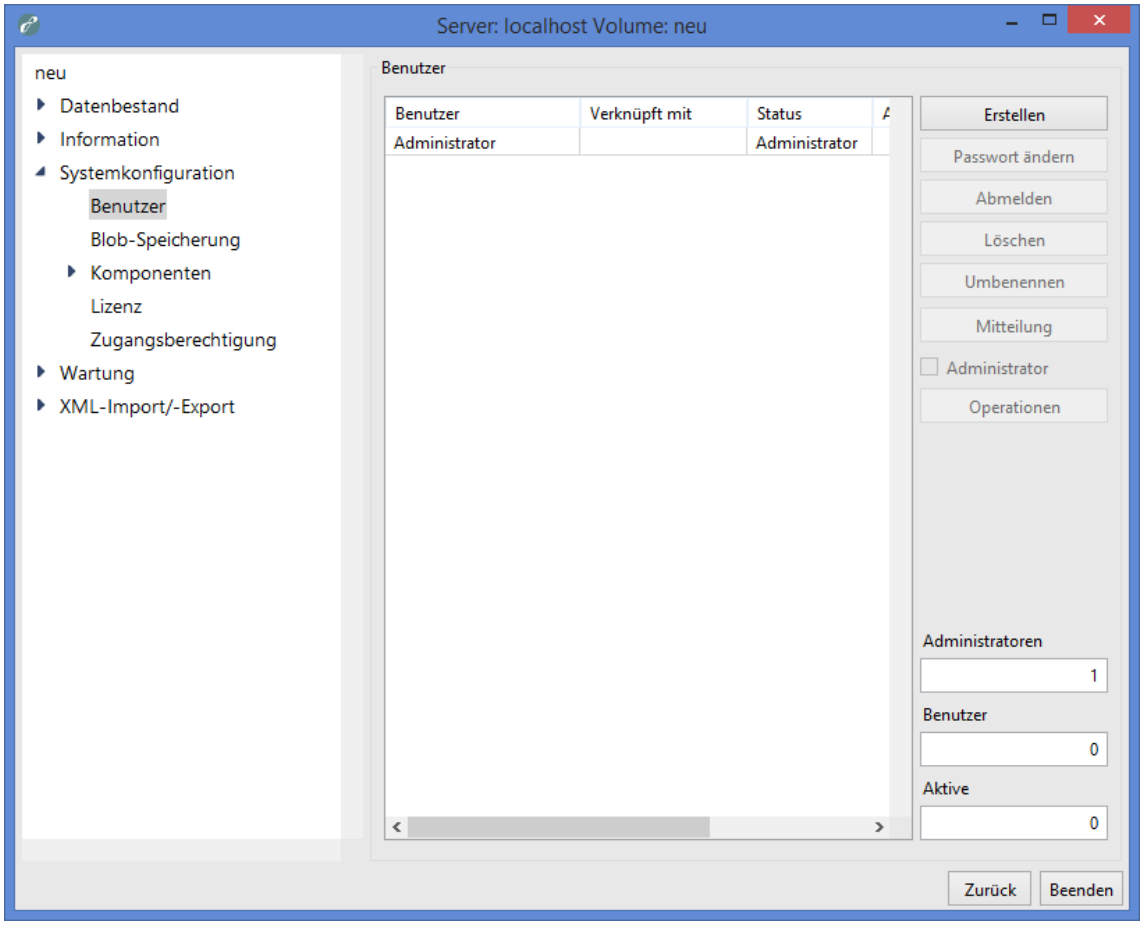
Über die Schaltfläche **Kopieren** werden alle Informationen in die Zwischenablage des Betriebssystems kopiert.

Die Schaltfläche **RSA-Key kopieren** kopiert den für jedes kompilierte Admin-Tool einzigartigen Schlüssel in die Zwischenablage des Betriebssystems. Dieser Schlüssel kann in die Initialisierungsdatei eines Mediators (Standarddateiname: *mediator.ini*) eingetragen werden und beschränkt dadurch den Zugang zu diesem Mediator über ein Admin-Tool auf Admin-Tools mit genau diesem Schlüssel.

2.4.2.3 Systemkonfiguration

2.4.2.3.1 Benutzer

Die Benutzerverwaltung gleicht derjenigen im Knowledge-Builder mit der Ausnahme, dass keine Verknüpfungen zwischen Nutzern und Objekten des nutzergenerierten Teilnetzes bearbeitet werden können.



Die tabellarische **Nutzerübersicht** zeigt für jeden im Wissensnetz registrierten Nutzer

- seinen Benutzernamen (*Benutzer*),
- mit welchem Objekt des nutzergenerierten Teilnetzes er verknüpft ist (*Verknüpft mit*),
- welchen Status er momentan besitzt (*Status*),
- an welchem Datum und zu welcher Uhrzeit er sich über den Knowledge-Builder im Wissensnetz angemeldet hat(*Anmeldedatum*), sofern er noch angemeldet ist, und
- mit welchem Verfahren das Passwort verschlüsselt ist (*Passworttyp*).

Die einzelnen Spalten der Tabelle sind über einen Klick auf den Spaltenkopf sortierbar.

Der *Status* gibt Auskunft darüber, ob ein Nutzer Administratorrechte besitzt, ob ein Nutzer mit Administratorrechten kein Passwort besitzt und ob ein Nutzer über den Knowledge-Builder im Wissensnetz angemeldet ist. Namen von Nutzern mit Administratorrechten ohne Passwort sind rot markiert.

Erstellen legt einen neuen Nutzer an. Benutzername (verpflichtend) und Passwort (optional) werden in einem eigenen Fenster festgelegt.Die Art und Menge der dafür erlaubten Zeichen ist nicht beschränkt.

Passwort ändern ändert das Passwort des in der **Nutzerübersicht** ausgewählten Nutzers. In zwei aufeinanderfolgenden Fenstern wird zweimal das neue Passwort eingegeben.

Abmelden meldet den in der **Nutzerübersicht** ausgewählten Nutzer nach einer Sicherheitsbestätigung aus dem Wissensnetz ab. Damit diese Operation eine Wirkung entfaltet, muss

dieser Nutzer aktuell über den Knowledge-Builder im Wissensnetz angemeldet sein.

Löschen löscht den in der **Nutzerübersicht** ausgewählten Nutzer nach einer Sicherheitsbestätigung. Mindestens ein Nutzer mit Administratorrechten muss verbleiben.

Umbenennen erlaubt über ein Freitextfeld in einem separaten Fenster die Vergabe eines neuen Benutzernamens für den in der **Nutzerübersicht** ausgewählten Nutzer. Bleibt das Freitextfeld leer, erfolgt keine Umbenennung.

Mitteilung sendet über ein Freitextfeld in einem separaten Fenster eine Nachricht an den in der **Nutzerübersicht** ausgewählten Nutzer. Die Nachricht wird im Wissensnetz zwischengespeichert und erscheint dem adressierten Nutzer in einem separaten Fenster im Knowledge-Builder, sobald er sich damit am Wissensnetz anmeldet. Der Nutzer kann auf diese Nachricht nicht antworten.

Administrator verleiht oder nimmt dem in der **Nutzerübersicht** ausgewählten Nutzer Administratorrechte. Damit ein Nutzer Administratorrechte erhalten kann, muss er ein Passwort besitzen. Nachdem er Administratorrechte besitzt, ist eine Löschung des Passworts indes möglich. Mindestens ein Nutzer muss Administratorrechte besitzen.

Operationen öffnet ein neues Fenster, in dem für den in der **Nutzerübersicht** ausgewählten Nutzer aus einer Liste von Operationen, namentlich

- Backup erstellen,
- Backup löschen,
- Backup wiederherstellen,
- Garage Collection,
- Kopieren,
- Log herunterladen,
- Volume herunterladen,
- Volume hochladen,
- Volume löschen,

diejenigen gewählt werden können, die dieser Nutzer im Rahmen der Einzelnetzverwaltung in Zukunft ohne Eingabe des Mediator-Passworts ausführen darf. Zur Bestätigung der Auswahl muss in das Freitextfeld **Server-Passwort für Operationen** das korrekte Mediator-Passwort eingegeben werden.

Die Operation **Operationen** ist nur wählbar für einen Nutzer mit Administratorrechten. Ihre Verwendung setzt außerdem voraus, dass ein Mediator-Passwort gesetzt wurde.

Das Feld **Administratoren** gibt die Anzahl aller im Wissensnetz registrierten Nutzer mit Administratorrechten an.

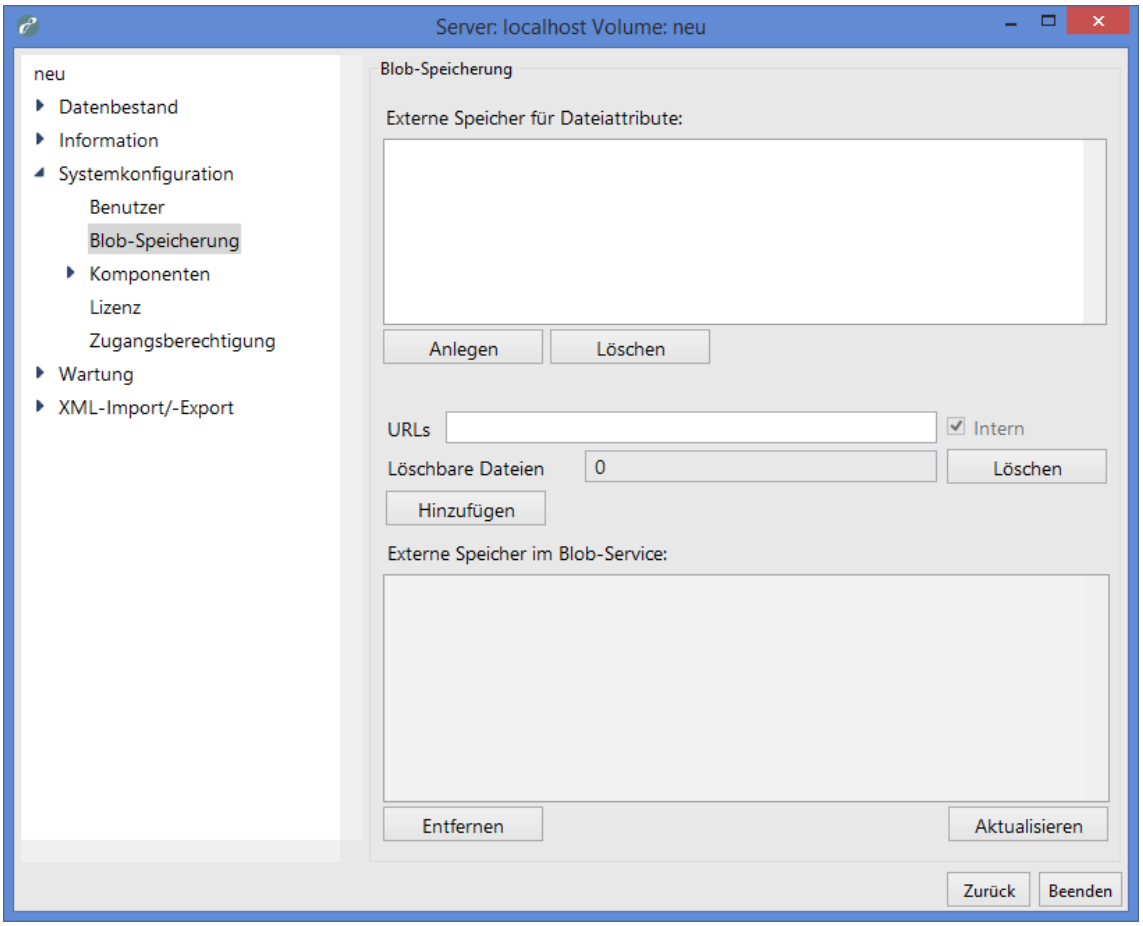
Das Feld **Benutzer** gibt die Anzahl aller im Wissensnetz registrierten Nutzer ohne Administratorrechte an.

Das Feld **Aktive** gibt die Anzahl aller gegenwärtig über den Knowledge-Builder im Wissensnetz angemeldeten Nutzer an.

2.4.2.3.2 Blob-Speicherung

Attributwerte von Attributen mit dem Attributdatentyp *Datei* (sogenannte Blobs) können auch wissensnetzextern in einem Blob-Speicher gespeichert werden. Dies hat den Vorteil, dass sie unabhängig vom Wissensnetz und damit bei Bedarf in einer anderen Systemumge-

bung verwaltet werden können. Um Blobs in einem Blob-Speicher zu sichern, muss der Blob-Speicher eingerichtet und mit einem konfigurierten Blob-Service (einem Software-Dienst) verbunden werden.



Anlegen erzeugt einen neuen Blob-Speicher. Er erscheint unter Verwendung des Namensformats *[Wissensnetz-ID]+[Blob-Speicher-ID]* in dem darüber liegenden Textfeld, der **Blob-Speicher-Gesamtübersicht**.

Löschen löscht den in der **Blob-Speicher-Gesamtübersicht** ausgewählten Blob-Speicher.

Das Zahlenfeld **Löschbare Dateien** zeigt die Anzahl der nicht mehr gebrauchten Blobs in dem in der **Blob-Speicher-Gesamtübersicht** ausgewählten Blob-Speicher. Blobs werden nicht mehr gebraucht, wenn ihre jeweiligen Attribute im Wissensnetz gelöscht wurden oder wenn die Verbindung zwischen Blob-Service und Blob-Speicher per Admin-Tool aufgehoben wurde.

Löschen löscht alle nicht mehr gebrauchten Blobs in dem in der **Blob-Speicher-Gesamtübersicht** ausgewählten Blob-Speicher.

Im Freitextfeld **URLs** kann ein Blob-Service identifiziert werden. Dies geschieht über die Eingabe der in der Initialisierungsdatei des zugehörigen Blob-Services (Standarddateiname: *blobservice.ini*) unter dem Schlüssel *interfaces* hinterlegten Netzadresse inklusive des Präfixes *http*. Soll der Blob-Service über mehrere Netzadressen angesprochen werden, können diese hintereinander mit Komma getrennt eingegeben werden.

Alternativ kann auch der im Mediator integrierte Blob-Service angesprochen werden. Dazu müssen in der Initialisierungsdatei des Mediators (Standarddateiname: *mediator.ini*) unter dem Schlüssel *startBlobService* der Wert *true* gesetzt und das Freitextfeld **URLs** leer gelassen

werden. Das rechts neben dem Freitextfeld **URLs** positionierte Kontrollkästchen **Intern** indiziert, ob der integrierte Blob-Service oder ein externer Blob-Service angesprochen wird. Die Konfiguration des im Mediator integrierten Blob-Services erfolgt nicht über die Mediator-Initialisierungsdatei, sondern über eine separate Initialisierungsdatei (Standarddateiname: *blobservice.ini*).

Hinzufügen verbindet den in der **Blob-Speicher-Gesamtübersicht** ausgewählten Blob-Speicher mit dem über das Freitextfeld **URLs** identifizierten Blob-Service. Dafür muss der Blob-Service aktiviert sein. Gelingt die Verknüpfung, erscheint der Blob-Speicher unter Verwendung des Namensformats *[Wissensnetz-ID]+[Blob-Speicher-ID]* in dem darunter liegenden Textfeld, der **Übersicht angemeldeter Blob-Speicher**.

Aktualisieren aktualisiert die **Übersicht angemeldeter Blob-Speicher**. Dazu muss ein Blob-Speicher in der **Blob-Speicher-Gesamtübersicht** ausgewählt sein.

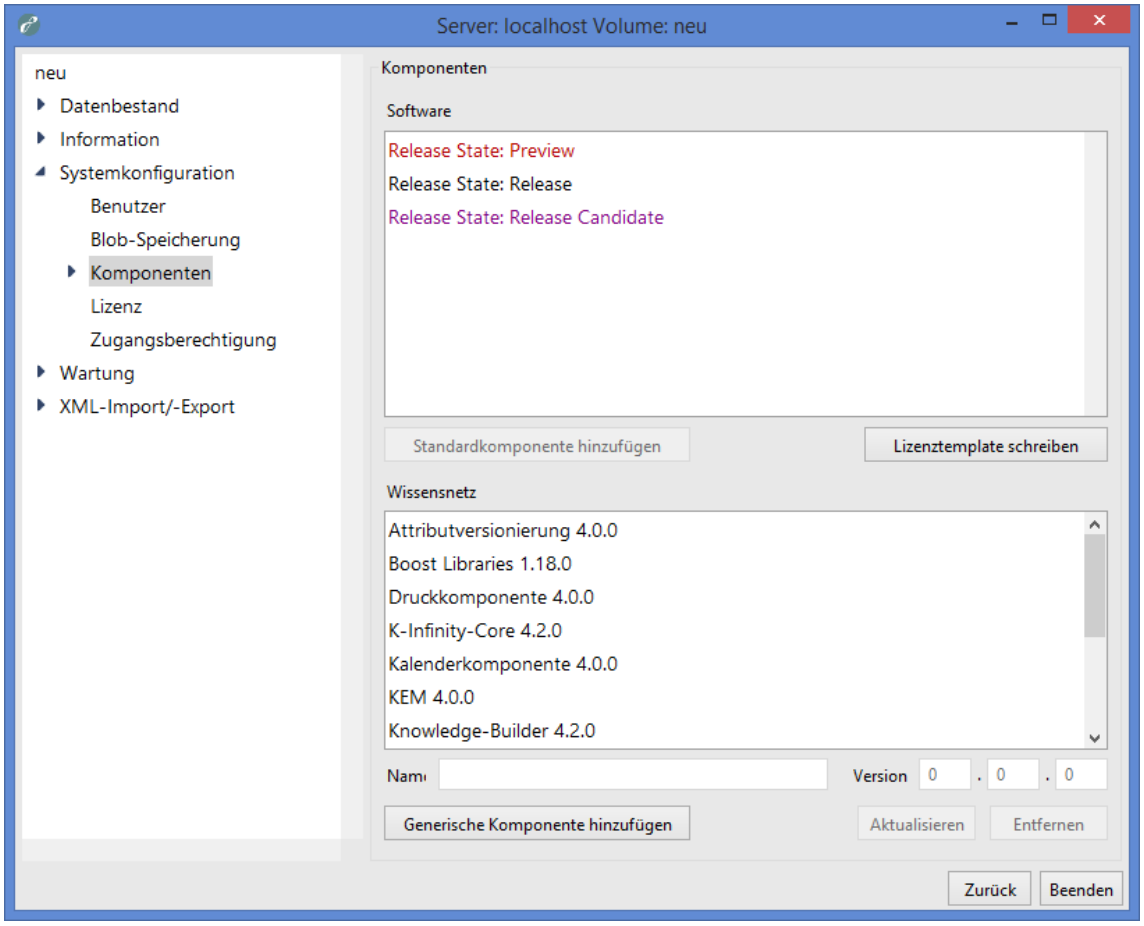
Entfernen unterbricht die Verbindung des in der **Übersicht angemeldeter Blob-Speicher** ausgewählten Blob-Speichers mit dem Blob-Service und entfernt den Blob-Speicher aus der Übersicht. Dabei verlieren alle im Blob-Speicher hinterlegten Blobs unwiderruflich ihre internen Verweise zu den jeweiligen Attributen im Wissensnetz und können im Wissensnetz nicht mehr abgerufen werden. Damit die Entfernung gelingt, muss der in der **Übersicht angemeldeter Blob-Speicher** ausgewählte Blob-Speicher auch in der **Blob-Speicher-Gesamtübersicht** ausgewählt sein.

Alle über einen Blob-Service gesicherten Blobs werden in einem relativ zur Position des Blob-Services liegenden Unterordner *blobs* abgelegt. Die interne Zuordnung jedes Blobs zu seinem Blob-Speicher und seinem Wissensnetz erfolgt über eine SQLite-Datenbank.

2.4.2.3.3 Komponenten

Wissensnetze bestehen aus Wissensnetzkomponenten. Neben den Basisfunktionalitäten gewähren sie den Wissensnetzen im Wesentlichen zusätzliche Schnittstellen und im Browser darstellbare Bedienoberflächen für die Nutzdaten (Webfrontends).

Eine besondere Untergruppe von Wissensnetzkomponenten sind die Veröffentlichungsstatuskomponenten (*Release States*), die in drei Varianten (*Preview*, *Release Candidate*, *Release*) existieren. Wird eine derartige Komponente im Wissensnetz installiert, können ausschließlich Software-Komponenten mit dem passenden Veröffentlichungsstatus auf das Wissensnetz zugreifen.



In der **Software-Liste** sind alle mit dem Admin-Tool mitgelieferten Wissensnetzkomponenten mit ihren jeweiligen Versionsnummern alphabetisch gelistet. Bedürfen sie einer eigenen Lizenz, ist außerdem vermerkt, ob die aktuelle Lizenz des Wissensnetzes sie umfasst oder nicht. Veröffentlichungsstatuskomponenten verfügen über keine Versionsnummer.

Wird eine Wissensnetzkomponente mit der rechten Maustaste angeklickt, erscheint ein Kontextmenü. Der dort verfügbare Menüpunkt **Standardkomponente hinzufügen** verfügt über die gleiche Funktionalität wie die gleichnamige Schaltfläche.

Standardkomponente hinzufügen installiert die in der **Software-Liste** ausgewählte Wissensnetzkomponente im Wissensnetz. Ein separates Fenster informiert über den Installationsstatus. Manche Wissensnetzkomponenten setzen für ihre Installation die Installation anderer Wissensnetzkomponenten im Wissensnetz voraus. Die meisten installierten Wissensnetzkomponenten (außer Veröffentlichungsstatuskomponenten) tauchen im Knowledge-Builder als eigene Einträge in der Rubrik *Technik* auf. Es kann immer nur eine Veröffentlichungsstatuskomponente zur gleichen Zeit installiert sein.

Lizenztemplate schreiben erzeugt eine inhaltlich zu vervollständigende Vorlage der für die Lizenzschlüsselgenerierung verwendeten Komponentenlizenzkonfigurationsdatei und speichert sie über einen Speicherdialog an einer frei wählbaren Stelle ab (Standarddateiname: *[Wissensnetz].componentLicenseTemplate.ini*). Unabhängig von der Konfiguration des gerade administrierten Wissensnetzes werden Konfigurationsplatzhalter für die Komponenten *KEM*, *i-views-Core* und *Knowledge-Builder* vorgegeben. In jedem Konfigurationsplatzhalter wird die im Admin-Tool mitgelieferte Versionsnummer der jeweiligen Wissensnetzkomponente voreingetragen.

In der **Wissensnetzliste** sind alle im Wissensnetz installierten Wissensnetzkomponenten mit

ihren jeweiligen Versionsnummern alphabetisch gelistet. Eine installierte Wissensnetzkomponente, für die im Admin-Tool eine neuere Version mitgeliefert ist, ist rot markiert. Die optionale Komponente *Knowledge-Builder* ist bei einem neuen Wissensnetz standardmäßig vorinstalliert.

Die Textfelder **Name** und **Version** zeigen den Namen respektive die dreistellige Versionsnummer der in der **Wissensnetzliste** ausgewählten installierten Wissensnetzkomponente.

Generische Komponente hinzufügen fügt der **Wissensnetzliste** eine generische Modellkomponente oder eine generische Softwarekomponente hinzu. Die Auswahl des Komponententyps erfolgt in einem separaten Fenster. Generische Komponenten erlauben die Bündelung projektspezifisch angefertigter Wissensnetzerweiterungen und vereinfachen deren (De-)Installation und Versionskontrolle über das Admin-Tool. Name und Versionsnummer einer im Wissensnetz installierten generischen Wissensnetzkomponente können in den entsprechend benannten Textfeldern frei vergeben werden.

Aktualisieren (die Bezeichnung wechselt zu **Erneuern**, wenn sie aktiviert werden kann) aktualisiert die in der **Wissensnetzliste** ausgewählte installierte Wissensnetzkomponente auf die im Admin-Tool mitgelieferte Version. Weicht die Sprache des aktuell laufenden Admin-Tools von der Sprache des Admin-Tools ab, mit dem die Wissensnetzkomponente ursprünglich im Wissensnetz installiert wurde, werden außerdem die Bezeichner aller Elemente und Elementtypen dieser Wissensnetzkomponente aktualisiert. Je nach Wissensnetzkomponente fügt die Aktualisierung den alten Bezeichnern neue Bezeichner in der Sprache des aktuell laufenden Admin-Tools hinzu (in Abhängigkeit von der Spracheinstellung des Knowledge-Builders wird dann die jeweils zutreffende Sprachversion dargestellt) oder ersetzt die alten Bezeichner mit den neuen Bezeichnern.

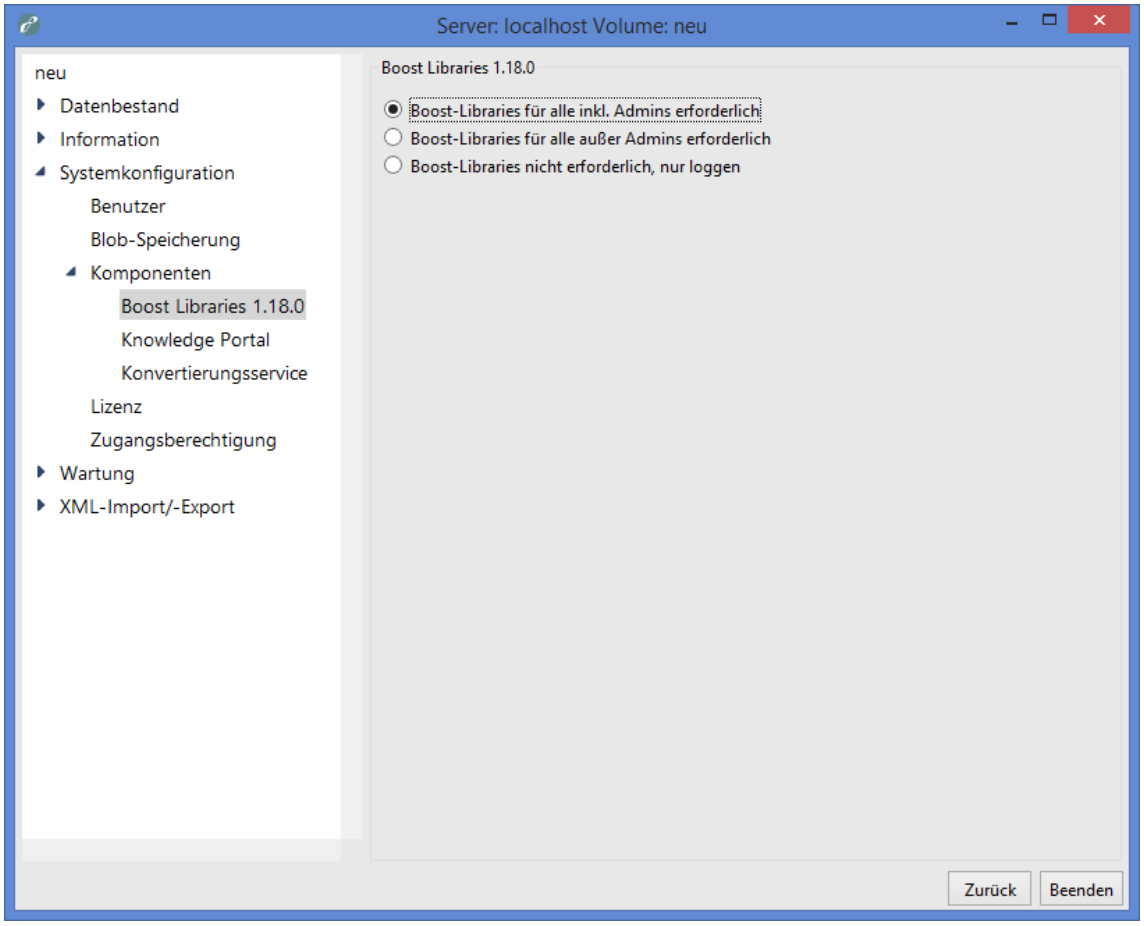
Entfernen deinstalliert die in der **Wissensnetzliste** ausgewählte installierte Wissensnetzkomponente. Sofern Wissensnetzkomponenten im installierten Zustand im Knowledge-Builder über einen eigenen Eintrag in der Rubrik *Technik* verfügen, hinterlassen sie dort nach ihrer Deinstallation ein eigenes Teilnetz, das manuell entfernt werden muss. Wissensnetzkomponenten lassen sich nur entfernen, wenn keine weiteren Wissensnetzkomponenten installiert sind, die von den zu deinstallierenden Wissensnetzkomponenten abhängig sind. Die beiden Wissensnetzkomponenten *i-views-Core* und *View-Konfiguration* bieten Basisfunktionalitäten und lassen sich nicht entfernen.

Boost Libraries 1.18.0

Dieses Konfigurationsmenü erscheint nur, wenn die Wissensnetzkomponente *Boost Libraries* installiert ist.

Mit Ausnahme des Blob-Services und des Mediators können alle Software-Komponenten JavaScript interpretieren. Um den Interpretationsumfang und die Interpretationsgeschwindigkeit von in JavaScript eingebetteten regulären Ausdrücken zu verbessern, kann deren Interpretation an die Bibliothek Boost.Regex übergeben werden. Unter Windows und Linux muss sich dazu die Bibliothek (Dateiname in Windows: *boost_regex.dll*, Dateiname in Linux: *libboost_regex.so*) im gleichen Verzeichnis befinden wie die übergebende Software-Komponente. In Mac OS ist die Bibliothek in die Datei der übergebenden Software-Komponente integriert.

Die Wissensnetzkomponente *Boost Libraries* ermöglicht die Sicherstellung, dass auf die Boost.Regex-Bibliothek zugegriffen werden kann.



Ist die Option **Boost-Libraries für alle inkl. Admins erforderlich** ausgewählt, können alle Software-Komponenten außer dem Admin-Tool auf das Wissensnetz nur zugreifen, wenn sie auf die Bibliothek Boost.Regex zugreifen können.

Ist die Option **Boost-Libraries für alle außer Admins erforderlich** ausgewählt, können alle Software- Komponenten außer dem Admin-Tool auf das Wissensnetz nur zugreifen, wenn sie auf die Bibliothek Boost.Regex zugreifen können. Ausgenommen von dieser Zugriffsperre sind Nutzer mit Administratorrechten, die das Wissensnetz über den Knowledge-Builder betreten.

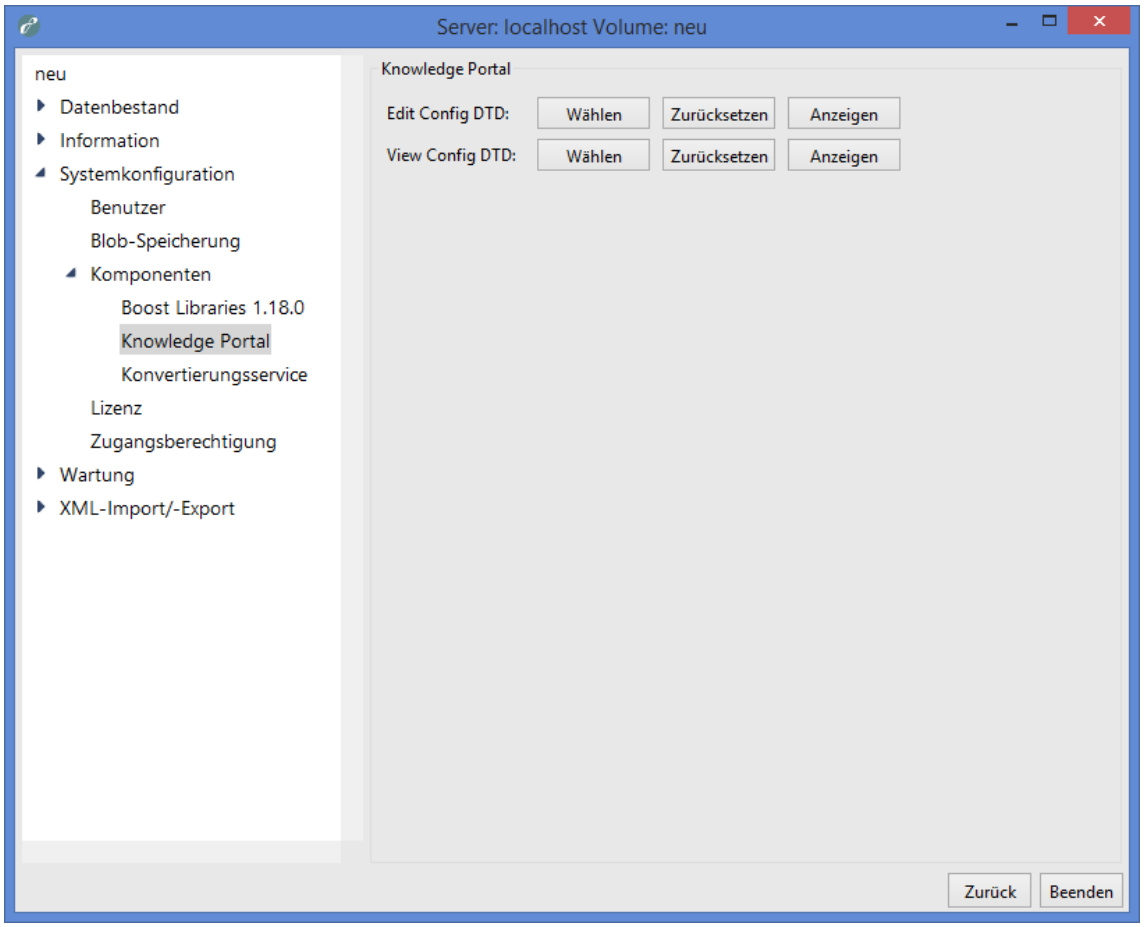
Ist die Option **Boost-Libraries nicht erforderlich, nur loggen** ausgewählt, hinterlegt jede Software- Komponente in seiner jeweiligen Protokolldatei, sofern vorhanden, eine entsprechende Warnung, wenn es beim Start nicht auf die Bibliothek Boost.Regex zugreifen kann. Ein Zugriff auf das Wissensnetz bleibt unabhängig davon möglich.

Knowledge Portal

Dieses Konfigurationsmenü erscheint nur, wenn die Wissensnetzkomponente *Knowledge-Portal* installiert ist.

Die Wissensnetzkomponente *Knowledge-Portal* ermöglicht einem Wissensnetz den Betrieb eines Knowledge-Portals (eines über einen Browser darstellbaren Frontends). Die Konfiguration der Darstellungs- und Bedienelemente dieses Frontends erfolgt im Knowledge-Builder an den entsprechenden Elementtypen über einen von der Wissensnetzkomponente speziell dafür bereitgestellten Editor mit Hilfe der Auszeichnungssprache XML. Zur einfacheren Wartung und logischen Reglementierung der XML-Dokumente lassen sich Schemata im Format DTD installieren, anhand derer die XML-Dokumente validiert werden können.

Im Frontend wird eine Bearbeitungssicht und eine Präsentationssicht mit jeweils exklusiven Darstellungs- und Bedienelementen unterschieden. Für beide Sichten werden separate DTD-Schemata geführt. Die nachfolgend erläuterten Bedienelemente existieren jeweils für jede Sicht.



Über die Schaltfläche **Wählen** kann auf das Dateisystem des Betriebssystems zugegriffen werden, um eine DTD-Schema-Datei für die jeweilige Sicht zu laden und im Wissensnetz zu installieren. Der Standarddateiname für Bearbeitungssicht-DTDs lautet *editConfig.dtd*, der Standarddateiname für Präsentationssicht-DTDs lautet *viewConfig.dtd*.

Zurücksetzen löscht das für die jeweilige Sicht installierte DTD-Schema aus dem Wissensnetz.

Anzeigen stellt das für die jeweilige Sicht installierte DTD-Schema in einem eigenen Fenster dar. Dort kann es in die Zwischenablage des Betriebssystems kopiert (Schaltfläche **In Zwischenablage kopieren**) oder über einen Speicherdialog als frei benennbare Textdatei an eine beliebige Stelle exportiert werden (Schaltfläche **Speichern**). Das Fenster verfügt außerdem über ein eigenes Kontextmenü, welches mit einem rechten Mausklick geöffnet werden kann:

- **Suche** erlaubt die Eingabe einer zu suchenden Zeichenkette in einem separaten Fenster und findet ihr nächstes Auftreten gemäß der Leserichtung relativ zur Position der per Mausklick setzbaren Schreibmarke. Bei der Suche wird Groß- und Kleinschreibung unterschieden.
- **Alles markieren** markiert den gesamten Text. Alternativ kann mit dem Mauszeiger ein beliebiger Textausschnitt markiert werden.
- **Kopieren** kopiert den gewählten Textbereich in die Zwischenablage des Betriebssys-

tems.

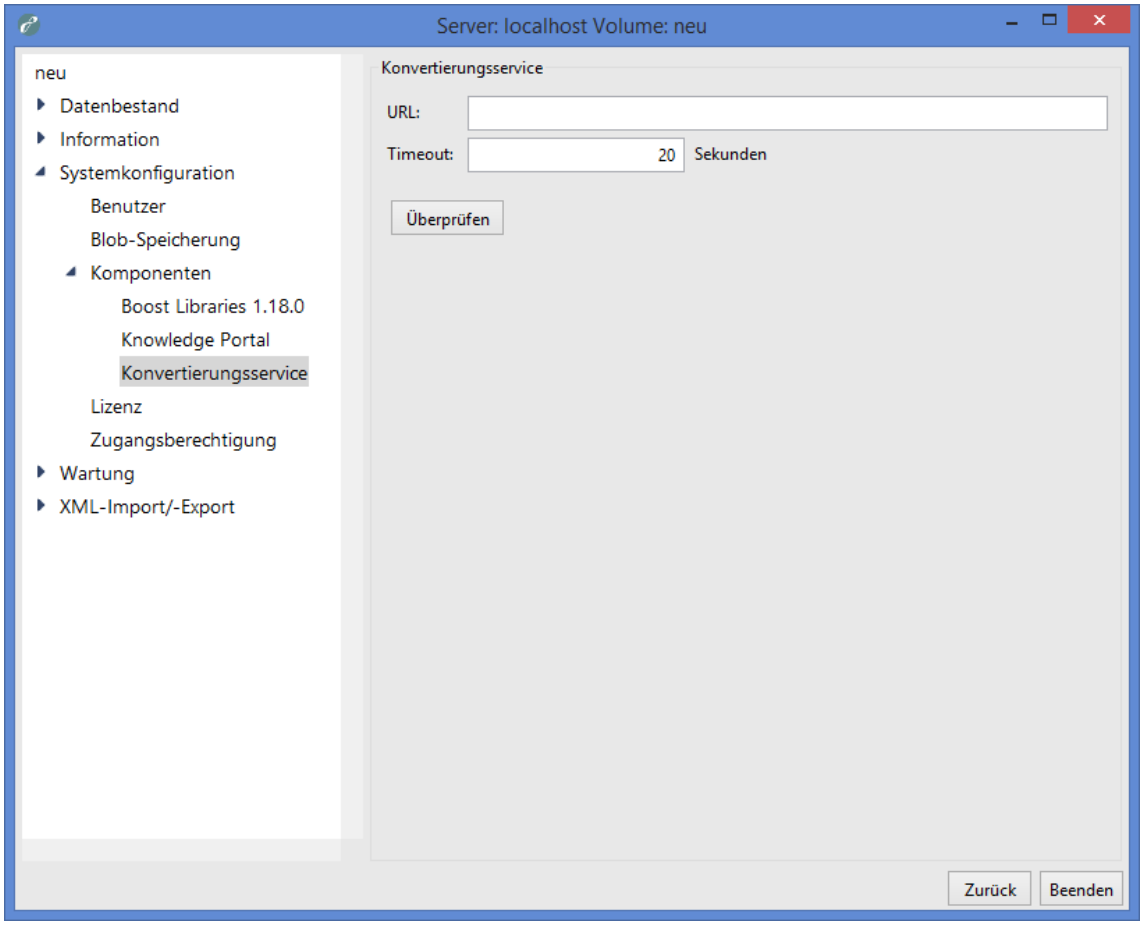
Konvertierungsservice

Dieses Konfigurationsmenü erscheint nur, wenn die Wissensnetzkomponente *Druckkomponente* installiert ist.

Die *Druckkomponente* ermöglicht die Integration ausgewählter Wissensnetzelemente in ein speicherbares elektronisches Dokument. Dazu muss eine Dokumentvorlage in den Formaten ODT, DOCX oder RTF über den Knowledge-Builder in das Wissensnetz importiert und mit den in ein Dokument zu integrierenden Wissensnetzelementen verknüpft werden. Die Gestaltung dieser Dokumentvorlage erfolgt in einem externen Office-Programm, von Elementen des Wissensnetzes auszufüllende Platzhalter lassen sich mit Hilfe von KScript und KPath definieren.

Zum Funktionsumfang der Druckkomponente gehört der Konvertierungsservice. Soll im Knowledge-Builder über den Kontextmenüpunt **Drucken** ein Dokument erzeugt werden, lassen sich neben dem Ursprungsformat der importierten Dokumentvorlage diverse andere Ausgabeformate wählen, in die die Dokumentvorlage konvertiert werden kann. Damit diese Konvertierung funktioniert, müssen eine passend konfigurierte Bridge (ein Software-Dienst) gestartet und mit der Druckkomponente verknüpft sowie eine Version von LibreOffice oder OpenOffice installiert sein.

Passend konfiguriert wird die Bridge über ihre Initialisierungsdatei (Standarddateiname: *bridge.ini*). Dort muss in der Sektion *[KHTTPRestBridge]* unter dem Schlüssel *services* der Wert *jodService* ergänzt werden. Außerdem ist eine neue Sektion *[file-format-conversion]* anzulegen und dort über das Schlüsselwertpaar *sofficePath="[Dateipfad]/soffice.exe"* mit einer korrekten Pfadangabe der Ort der Startdatei von LibreOffice beziehungsweise OpenOffice zu hinterlegen.



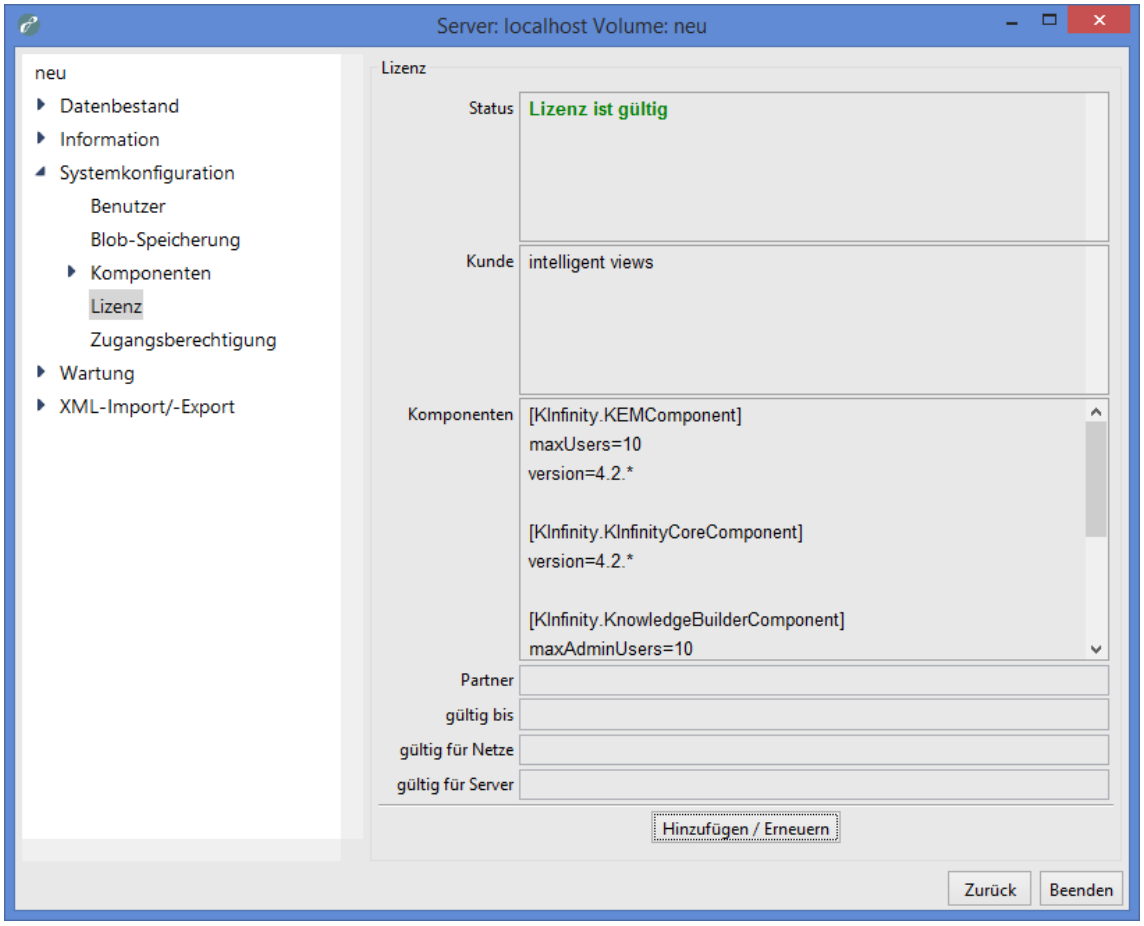
Die Verknüpfung der Bridge mit der Druckkomponente erfolgt über das Freitextfeld **URL**. Dort wird die Netzadresse der Bridge im Format *http:///[Bridge-IP-Nummer]:[Bridge-Port]/jodService/jodconverter/se* eingetragen. Der Pfadabschnitt */jodService/jodconverter/service* ist historisch bedingt und aktiviert den vordefinierten jodService.

Überprüfen startet einen Testprozess. Der Testprozess schickt über REST ein Testdokument an die über die Netzadresse festgelegte Bridge und erwartet, dass ein ordnungsgemäß konvertiertes Testdokument zurückgeschickt wird. Das Testergebnis wird in einem separaten Fenster ausgegeben.

Im Freitextfeld **Timeout** wird festgelegt, wie viele Sekunden lang auf die Rücksendung des konvertierten Testdokuments gewartet wird, bevor eine Fehlermeldung generiert wird. Die Voreinstellung liegt bei 20 Sekunden.

2.4.2.3.4 Lizenz

Ein Wissensnetz muss eine gültige Lizenz besitzen, damit der Knowledge-Builder und andere Software- Komponenten (mit Ausnahme des Admin-Tools) damit arbeiten können.



Das Feld **Status** gibt an, ob die Lizenz gegenwärtig gültig oder ungültig ist. Falls sie ungültig ist, wird außerdem ein Grund genannt. Gründe für eine ungültige Lizenz können die Überschreitung des Gültigkeitsdatums oder der maximalen Anzahl erlaubter registrierter Nutzer sein.

Das Feld **Kunde** beschreibt den Kunden, für den die Lizenz ausgestellt wurde. Neben dem Namen können auch die Adresse und die Abteilung genannt sein.

Das Feld **Komponenten** stellt den Inhalt der für die Lizenzschlüsselgenerierung verwendeten Komponentenlizenzkonfigurationsdatei *[Wissensnetz].componentLicenseTemplate.ini* dar. Dort werden Festlegungen über

- die lizensierten Versionen einzelner Komponenten (*version*),
- die maximale Anzahl registrierter Nutzer mit Administratorrechten (*maxAdminUsers*) und
- die maximale Anzahl registrierter Nutzer ohne Administratorrechte (*maxUsers*) getroffen.

Das Feld **Partner** enthält den Namen des Partners, über den die Lizenz weitergegeben wird.

Das Feld **gültig bis** enthält das Datum, nach dessen Ablauf die Lizenz erlischt.

Das Feld **gültig für Netze** enthält eine Liste der Namen aller Netze, auf die die Lizenz beschränkt ist. Möglich ist eine Erfassung über einen regulären Ausdruck.

Das Feld **gültig für Server** enthält eine Liste aller IP-Adressen und Port-Nummern, über die

ein an das Wissensnetz angeschlossener Mediator erreicht werden darf.

Die Felder **Partner**, **gültig bis**, **gültig für Netze** und **gültig für Server** können leer sein.

Alle Felder verfügen über ein Kontextmenü, das per Rechtsklick aktiviert werden kann:

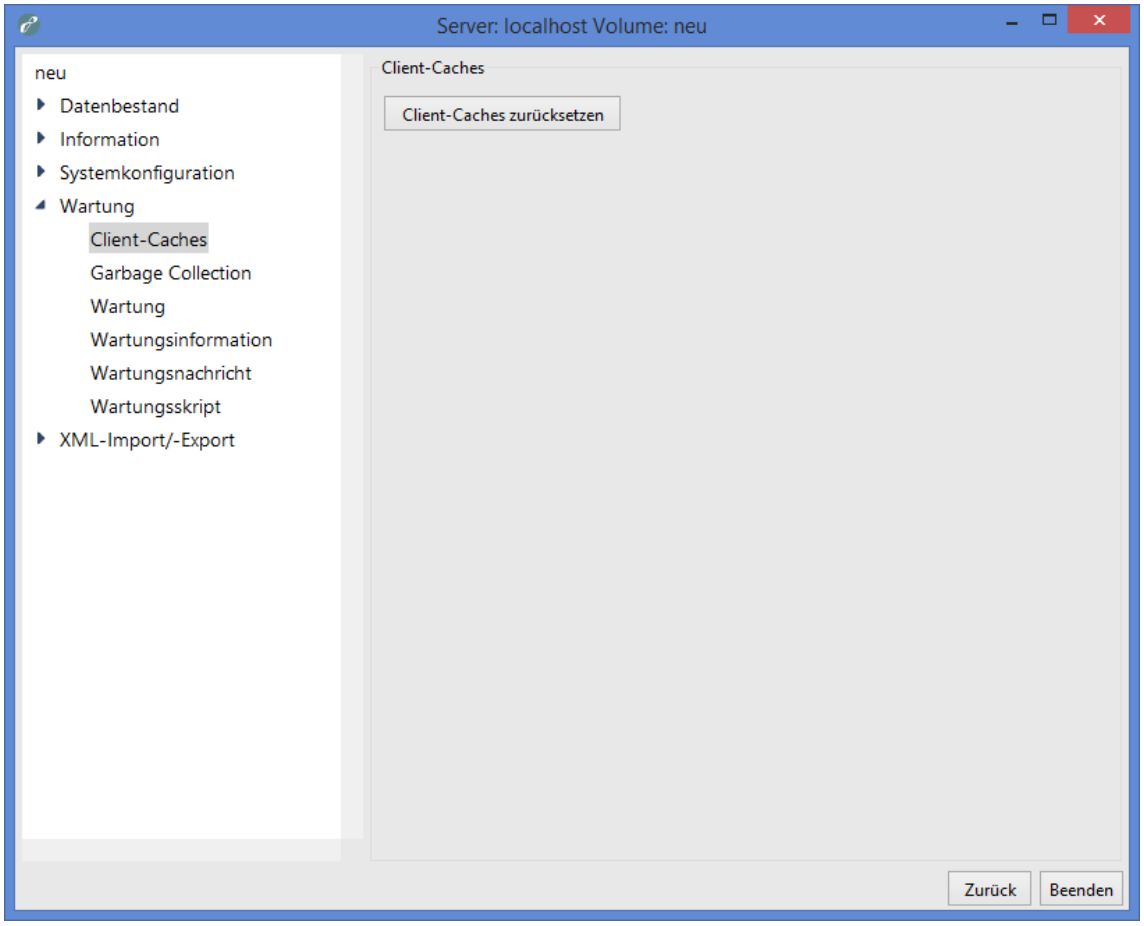
- **Select All** markiert den gesamten Text. Alternativ kann mit dem Mauszeiger ein beliebiger Textausschnitt markiert werden.
- **Copy** kopiert den gewählten Textbereich in die Zwischenablage des Betriebssystems.
- **Find Again** sucht nach dem gewählten Textbereich und findet sein nächstes Auftreten gemäß der Leserichtung.
- **Find** erlaubt die Eingabe einer zu suchenden Zeichenkette in einem separaten Fenster und findet ihr nächstes Auftreten gemäß der Leserichtung relativ zur Position der per Mausklick setzbaren Schreibmarke. Bei der Suche wird Groß- und Kleinschreibung unterschieden.

Hinzufügen / Erneuern erlaubt das Laden eines neuen Lizenzschlüssels (Dateiname: *[Lizenzname].key*) über das Dateisystem des Betriebssystems.

2.4.2.4 Wartung

2.4.2.4.1 Client-Caches

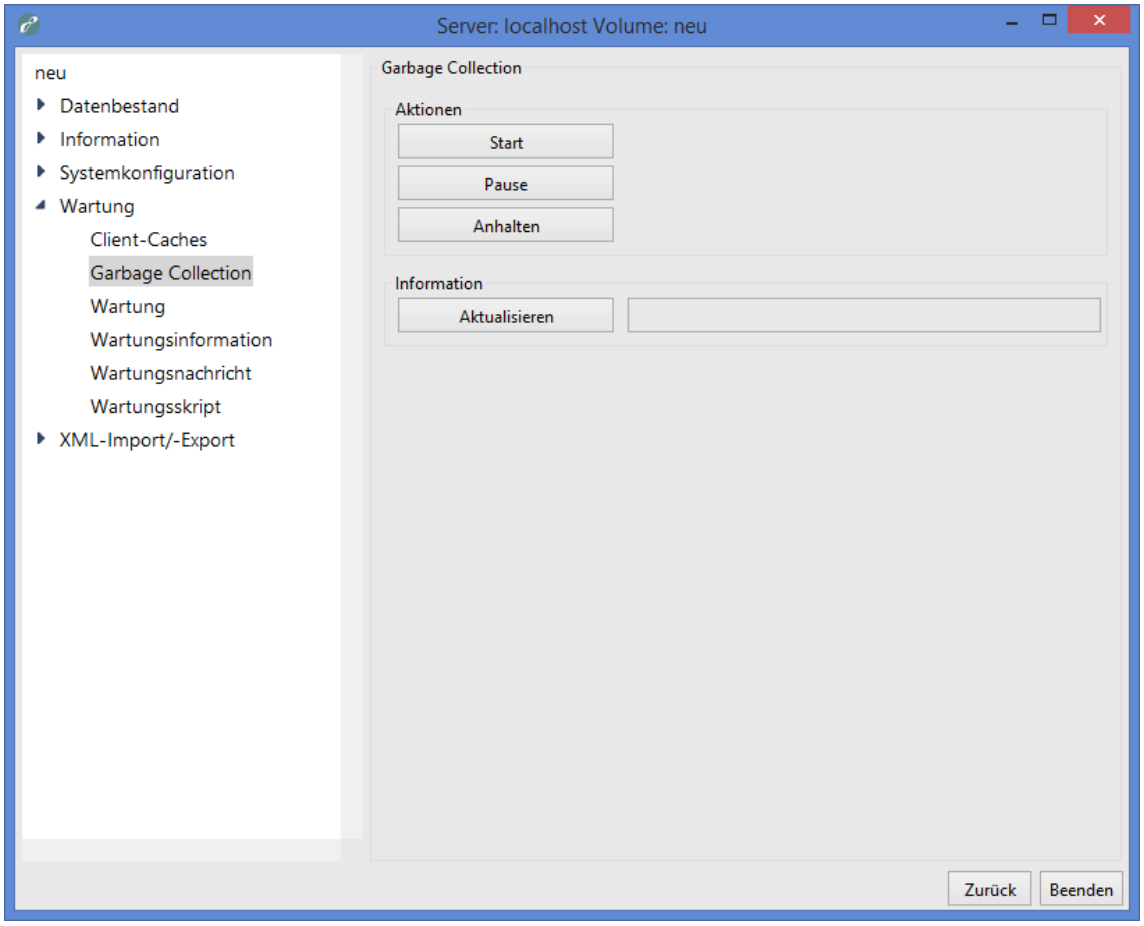
Zur Performanzverbesserung verwenden auf das Wissensnetz zugreifende Software-Komponenten häufig einen eigenen Pufferspeicher (Cache). Darin werden Schema- und Konfigurationsdaten des Wissensnetzes zwischengespeichert, um im Falle einer späteren Verwendung schneller auf sie zugreifen zu können.



Client-Caches zurücksetzen löscht diese zwischengespeicherten Daten. Dies ist sinnvoll, wenn sie aufgrund von Änderungen am Schema oder an der Konfiguration veraltet sind. Diese Operation setzt voraus, dass das Wissensnetz über einen Mediator angesteuert wird.

2.4.2.4.2 Garbage Collection

Die Garbage-Collection ist ein Verfahren, das nicht mehr referenzierte Objekte (nach programmierterminologischer Lesart) in einem Wissensnetz löscht und damit den Speicherverbrauch des Wissensnetzes minimiert. Die Nutzung der Garbage-Collection setzt voraus, dass das zu bereinigende Wissensnetz über einen Mediator angesteuert wird.



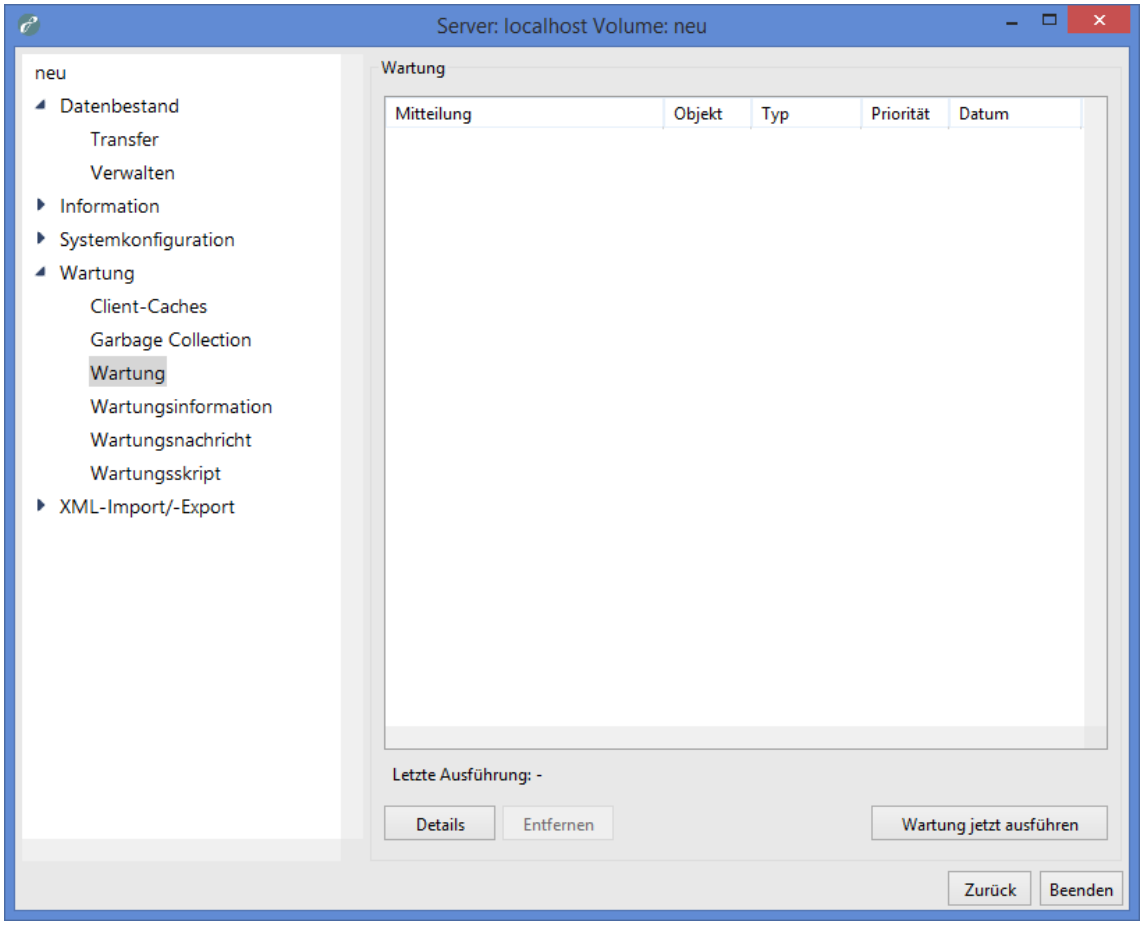
Start startet eine neue Speicherbereinigung für das Wissensnetz oder setzt eine pausierte Speicherbereinigung fort. Es erfolgt keine Rückmeldung, wann der Vorgang abgeschlossen ist. Der Stand des Fortschritts kann über den Menüpunkt **Aktualisieren** in Erfahrung gebracht werden.

Pause unterbricht die Durchführung der aktiven Speicherbereinigung für das Wissensnetz.

Anhalten bricht die Durchführung der aktiven Speicherbereinigung für das Wissensnetz ab.

Aktualisieren schreibt den aktuellen Zustand der Speicherbereinigung für das Wissensnetz in das nebenstehende Textfeld. Ist eine Speicherbereinigung aktiv, erfolgt zusätzlich eine Rückmeldung über den Stand des Fortschritts in Form einer Prozentangabe.

2.4.2.4.3 Wartung



Wartung jetzt ausführen überprüft

- die Lizenz (*Lizenz*),
- Indizes (*Indizes*),
- registrierte Objekte (*die Registratur*),
- Rechte (*Zugriffsrechte*),
- Trigger (*Trigger*) und
- installierte Wissensnetzkomponenten (*aktive Komponenten*)

auf Mängel. Im Zuge der Prüfung wird auch die über den Knowledge-Builder einsehbare Statistik von Eigenschaftshäufungen pro Objekt (Metriken) aktualisiert.

Gefundene Mängel werden in einer tabellarischen **Mangelübersicht** gesammelt. Für jeden Mangel wird dort

- eine kurze Beschreibung, falls einschlägig inklusive der Cluster-ID und der Frame-ID (Format *Cluster-ID/Frame-ID*) des mangelhaften Objekts (in programmierterminologischer Lesart) (*Mitteilung*),
- das vom Mangel betroffene übergeordnete Wissensnetzelement (*Objekt*),
- dessen Typ (*Typ*),

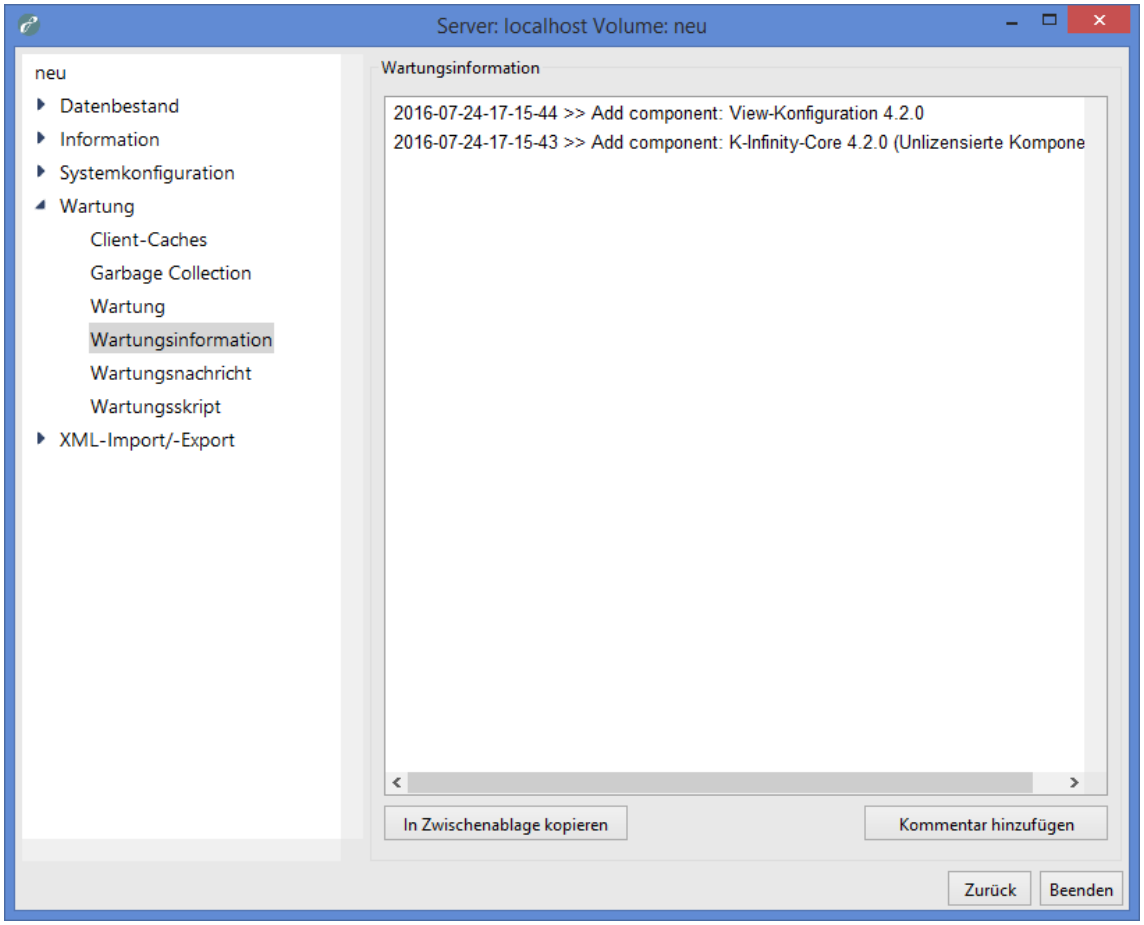
- die Schwere des Mangels (*Priorität*) und
- dessen ersten Feststellungszeitpunkt in Form eines Datums (*Datum*)

ausgegeben. Die einzelnen Spalten der Tabelle sind über einen Klick auf den Spaltenkopf sortierbar.

Details stellt alle in der **Mangelübersicht** des ausgewählten Mangels gelisteten Daten in einem neuen Fenster dar. Ergänzt werden die Uhrzeit des ersten Feststellungszeitpunkts sowie Datum und Uhrzeit des letzten Feststellungszeitpunkts. Die Daten können dort in die Zwischenablage des Betriebssystems kopiert (Schaltfläche **In Zwischenablage kopieren**) oder über einen Speicherdialog als frei benennbare Textdatei an eine beliebige Stelle exportiert werden (Schaltfläche **Speichern**). Die über die Schaltfläche **Details** ausgelöste Operation kann alternativ über einen Doppelklick auf einen Mangel in der Mangelübersicht erwirkt werden.

Entfernen löscht den in der **Mangelübersicht** ausgewählten Mangel. Dies hat keine Auswirkungen auf den ersten Feststellungszeitpunkt des Mangels.

2.4.2.4.4 Wartungsinformation



Über diesen Menüpunkt lässt sich eine chronologisch sortierte **Wartungshistorie** aller wesentlichen Administrationsvorgänge im Wissensnetz seit seiner Entstehung abrufen. Erfasst werden Backup- und Transfervorgänge, Komponenteninstallationen und -aktualisierungen sowie Ausführungen von Wartungsskripten und der Garbage-Collection, jeweils mit Datum und Uhrzeit.

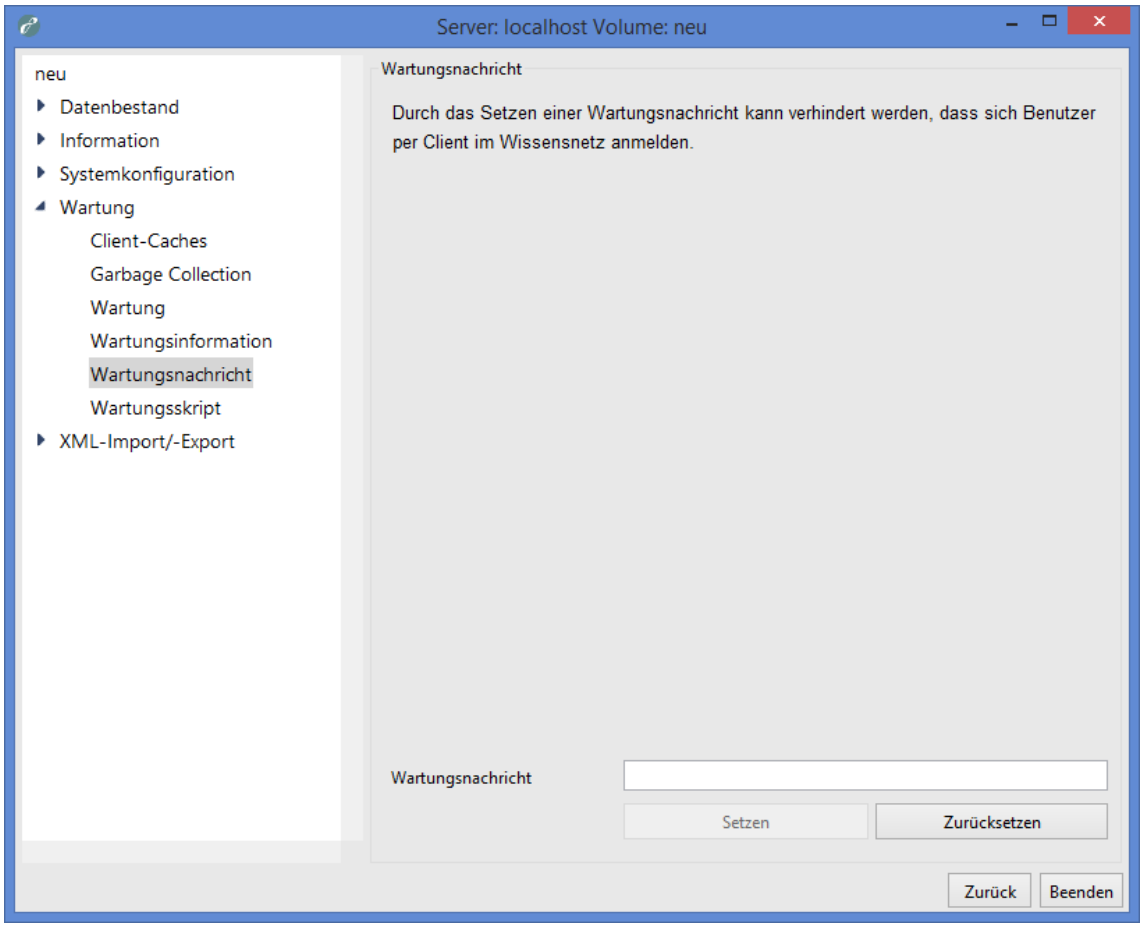
Die **Wartungshistorie** verfügt über ein Kontextmenü, das per Rechtsklick aktiviert werden kann:

- **Select All** markiert den gesamten Text. Alternativ kann mit dem Mauszeiger ein beliebiger Textausschnitt markiert werden.
- **Copy** kopiert den gewählten Textbereich in die Zwischenablage des Betriebssystems.
- **Find Again** sucht nach dem gewählten Textbereich und findet sein nächstes Auftreten gemäß der Leserichtung.
- **Find** erlaubt die Eingabe einer zu suchenden Zeichenkette in einem separaten Fenster und findet ihr nächstes Auftreten gemäß der Leserichtung relativ zur Position der per Mausklick setzbaren Schreibmarke. Bei der Suche wird Groß- und Kleinschreibung unterschieden.

In Zwischenablage kopieren kopiert die gesamte **Wartungshistorie** in die Zwischenablage des Betriebssystems.

Kommentar hinzufügen erlaubt die Eingabe einer Anmerkung über ein Freitextfeld in einem separaten Fenster. Sie wird mit einem Zeitstempel versehen und in die **Wartungshistorie** aufgenommen. In die **Wartungshistorie** aufgenommene Anmerkungen lassen sich nicht löschen.

2.4.2.4.5 Wartungsnachricht



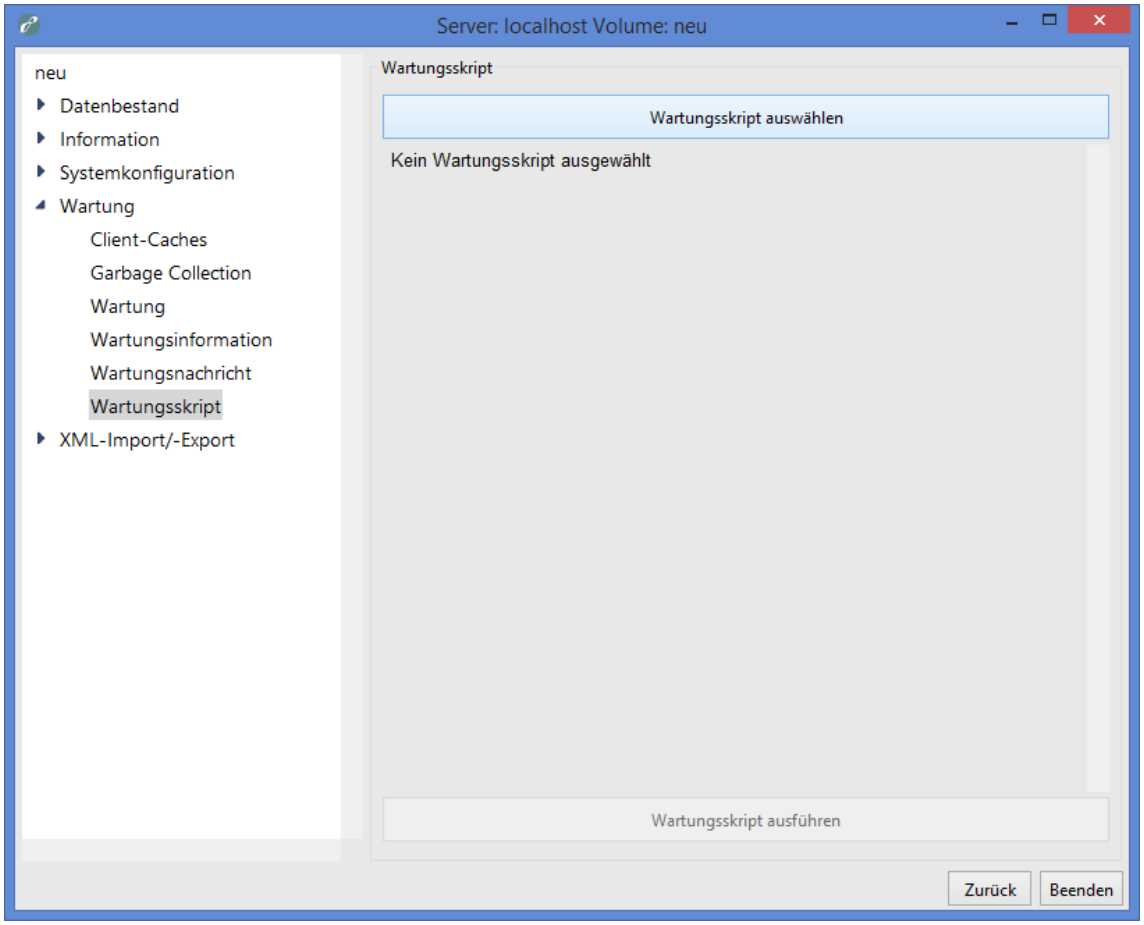
Die Schaltfläche **Setzen** aktiviert eine Wartungssperre, die jedem Nutzer den Zutritt zum

Wissensnetz über den Knowledge-Builder verwehrt. Dafür muss eine Wartungsnachricht formuliert werden.

Die Wartungsnachricht wird im Freitextfeld **Wartungsnachricht** formuliert. Sie wird jedem Nutzer, der das Wissensnetz bei aktivierter Wartungssperre über den Knowledge-Builder betreten will, in Form einer Fehlermeldung angezeigt.

Die Schaltfläche **Zurücksetzen** hebt die zuvor gesetzte Wartungssperre auf und löscht die Wartungsnachricht.

2.4.2.4.6 Wartungsskript



Über **Wartungsskript auswählen** kann auf das Dateisystem des Betriebssystems zugegriffen werden, um ein Wartungsskript (Dateiname: *[Wartungsskript].kss*) zu laden. Wartungsskripte werden fallspezifisch in der Programmiersprache Smalltalk angefertigt und erlauben Operationen, die sich nicht über die vordefinierten Funktionen des Admin-Tools oder über die KEM- oder JS-Schnittstellen realisieren lassen.

Verfügt das Wartungsskript über eine Beschreibung, wird diese nach dem Laden des Wartungsskripts in einem unsichtbaren Textfeld unterhalb der Schaltfläche **Wartungsskript auswählen** ausgegeben. Dieses Textfeld verfügt über ein Kontextmenü, das per Rechtsklick aktiviert werden kann:

- **Select All** markiert den gesamten Text. Alternativ kann mit dem Mauszeiger ein beliebiger Textausschnitt markiert werden.

- **Copy** kopiert den gewählten Textbereich in die Zwischenablage des Betriebssystems.
- **Find Again** sucht nach dem gewählten Textbereich und findet sein nächstes Auftreten gemäß der Leserichtung.
- **Find** erlaubt die Eingabe einer zu suchenden Zeichenkette in einem separaten Fenster und findet ihr nächstes Auftreten gemäß der Leserichtung relativ zur Position der per Mausklick setzbaren Schreibmarke. Bei der Suche wird Groß- und Kleinschreibung unterschieden.

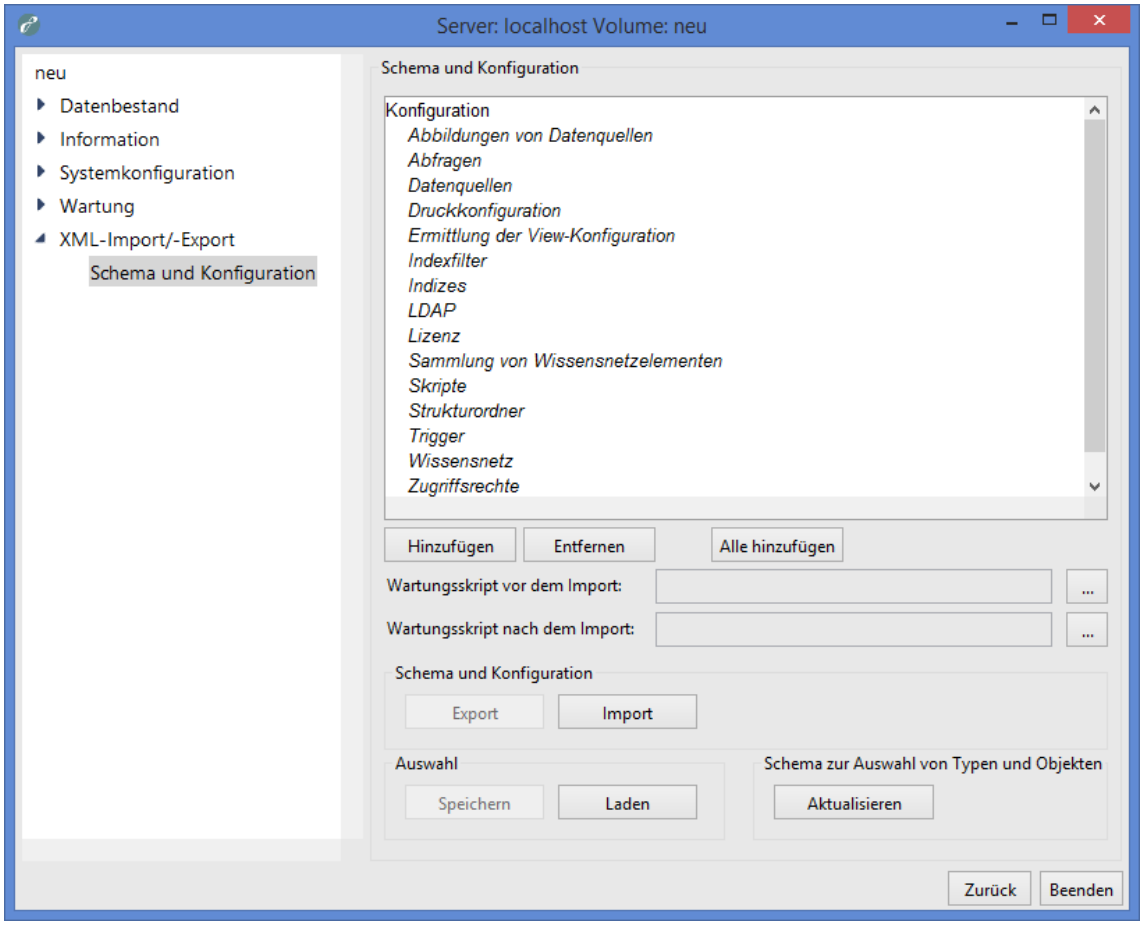
Wartungsskript ausführen startet das Wartungsskript. Ein separat erscheinendes Fenster gibt Auskunft, wenn das Wartungsskript vollzogen wurde, und bietet je nach Skript zusätzliche Ausführungsinformationen oder erlaubt skriptspezifische Ausführungsoptionen.

2.4.2.5 XML-Import/-Export

2.4.2.5.1 Schema und Konfiguration

Ein Wissensnetz im weiteren Sinne besteht neben den nutzergenerierten und über Komponenten eingebrachten Teilnetzen (Schemata mit Nutzdaten) noch aus diversen weiteren Bausteinen (Konfigurationen), die dieses Teilnetz funktional erweitern, konfigurieren oder damit arbeiten. Im Rahmen dieses Menüpunkts werden Schemata und Konfigurationen zusammenfassend als Konfigurationen bezeichnet.

Zahlreiche Konfigurationen eines Wissensnetzes lassen sich gezielt exportieren und importieren.



Die **Konfigurationsübersicht** bietet einen listenartigen Überblick über alle mittels der im Folgenden beschriebenen Operationen prinzipiell transferierbaren Konfigurationstypen eines Wissensnetzes. Prinzipiell transferierbar sind

- einzelne registrierte Abbildungen von Datenquellen (*Abbildungen von Datenquellen*),
- einzelne von Administratoren konfigurierte und benutzerdefinierte Suchfelder (*Abfragen*),
- einzelne Datenquellenzugriffseinstellungen zur Nutzung für Abbildungen von Datenquellen (*Datenquellen*),
- die Druckkonfiguration (*Druckkonfiguration*),
- die Menge aller innerhalb der Rubrik *Ermittlung* der View-Konfiguration definierten Bausteine (*Ermittlung der View-Konfiguration*),
- einzelne Indexfilter (*Indexfilter*),
- einzelne Indexerkonfigurationen (*Indizes*),
- die LDAP-Authentifizierung (*LDAP*),
- die Lizenz des Wissensnetzes (*Lizenz*),
- einzelne registrierte Sammlungen semantischer Objekte (*Sammlung von Wissensnetzelementen*),
- einzelne registrierte Skripte (*Skripte*),
- den Arbeitsordner (*Strukturordner*),
- die Menge aller innerhalb der Rubrik Trigger definierten Bausteine (*Trigger*),
- einzelne Teilnetze (*Wissensnetz*) und
- die Menge aller innerhalb der Rubrik Rechte definierten Bausteine (*Zugriffsrechte*).

Die **Konfigurationsübersicht** verwaltet überdies alle konkret zum Export bestimmten Konfigurationen. Zum Export bestimmte Konfigurationen erscheinen als ausklappbare Listenunterpunkte ihrer jeweiligen Konfigurationstypen. Benötigen diese Konfigurationen für ihren erfolgreichen Export andere Konfigurationen, sind diese anderen Konfigurationen wiederum in Form ausklappbarer Listenunterpunkte der jeweiligen Konfigurationen aufgeführt. Konfigurationstypen ohne eigene Konfigurationen sind kursiv gekennzeichnet, Konfigurationstypen mit eigenen Konfigurationen sind fett gekennzeichnet und stellen die Anzahl ihrer zugeordneten Konfigurationen in Klammern dar. Konfigurationstypen und Konfigurationen jedes Konfigurationstyps sind jeweils in alphabetischer Reihenfolge sortiert.

Das Ein- und Ausklappen von Listenunterpunkten in der **Konfigurationsübersicht** geschieht per Klick auf die sich links neben den Listenpunkten befindlichen Dreiecksymbole. Alternativ lässt sich dies über ein Kontextmenü realisieren, das über einen Klick mit der rechten Maustaste auf einen Listenpunkt aufgerufen werden kann:

- **Expand** klappt alle direkten Listenunterpunkte des gewählten Listenpunkts aus.
- **Expand fully** klappt alle direkten und alle indirekten Listenunterpunkte des gewählten Listenpunkts aus.
- **Contract fully** klappt alle Listenunterpunkte des gewählten Listenpunkts wieder ein.

Hinzufügen fügt der **Konfigurationsübersicht** eine Konfiguration des dort ausgewählten Konfigurationstyps hinzu. Existiert mehr als eine Konfiguration für den ausgewählten Konfigurationstyp im Wissensnetz, schließt sich eine Auswahlmöglichkeit in einem separaten Fenster an. Die Auswahl erfolgt dort entweder einzeln per Klick auf die jeweiligen Konfig-

urationen in einer Liste oder pauschal über die Schaltfläche **Alles aus-/abwählen**.

Entfernen löscht entweder alle Konfigurationen des in der **Konfigurationsübersicht** ausgewählten Konfigurationstyps oder die in der **Konfigurationsübersicht** ausgewählte Konfiguration.

Alle hinzufügen fügt der **Konfigurationsübersicht** alle im Wissensnetz existierenden Konfigurationen hinzu und verteilt diese auf die jeweils passenden Konfigurationstypen.

Über die Schaltflächen **...** kann auf das Dateisystem des Betriebssystems zugegriffen werden, um ein Wartungsskript (Dateiname: *[Wartungsskript].kss*) zu laden. Wartungsskripte werden fallspezifisch in der Programmiersprache Smalltalk angefertigt und erlauben Operationen, die sich nicht über die vordefinierten Funktionen des Admin-Tools oder über die KEM- oder JS-Schnittstellen realisieren lassen.

Wird ein Wartungsskript geladen, erscheint der Dateiname des gewählten Wartungsskripts im links von der jeweiligen Schaltfläche positionierten Textfeld. Werden im Anschluss Konfigurationen importiert, wird das Wartungsskript ausgeführt. Werden im Anschluss Konfigurationen exportiert, wird das Wartungsskript ebenfalls exportiert und erst beim Import dieser Konfigurationen ausgeführt. Der genaue Ausführungszeitpunkt des Wartungsskripts in Relation zum Importprozess hängt davon ab, über welche der beiden **...**-Schaltflächen es geladen wurde. Er befindet sich entweder vor dem Start des Importprozesses oder nach dem Ende des Importprozesses.

Export exportiert die in der **Konfigurationsübersicht** ausgewählten Konfigurationen. Zur Auswahl stehen der Export in eine einzige Archivdatei im Archivformat *tar* oder in einzelne Dateien in einem Ordner. Die Auswahl der Exportmethode vollzieht sich in einem separaten Fenster:

- In den Freitextfeldern **Datei** oder **Verzeichnis** kann der Name der Archivdatei (Dateiname: *[Wissensnetz].tar*) respektive des Ordners (kein Standardname) angegeben werden. Die Archivdatei respektive der Ordner wird im gleichen Ordner wie das Admin-Tool angelegt. Alternativ kann über **Wählen** ein Speicherdialog aufgerufen werden, um Name und Speicherort der Archivdatei respektive des Ordners frei festzulegen.

Import importiert nach einer Bestätigungsfrage Konfigurationen in das Wissensnetz. Zur Auswahl stehen der Import aus einer einzigen Archivdatei im Archivformat *tar* oder aus einzelnen Dateien in einem Ordner. Die Auswahl der Importmethode vollzieht sich in einem separaten Fenster:

- In den Freitextfeldern **Datei** oder **Verzeichnis** kann der Name der Archivdatei (Dateiname: *[Wissensnetz].tar*) respektive des Ordners (kein Standardname) angegeben werden. Die Archivdatei respektive der Ordner wird im gleichen Ordner wie das Admin-Tool gesucht. Alternativ kann über **Wählen** auf das Dateisystem des Betriebssystems zugegriffen werden, um eine Archivdatei respektive einen Ordner an einer beliebigen Stelle auszuwählen.
- Ist die zu importierende Archivdatei respektive der zu importierende Ordner gewählt, erscheint in einem weiteren Fenster eine Übersicht über alle darin enthaltenen Konfigurationen. Diese Übersicht kann dort in die Zwischenablage des Betriebssystems kopiert (Schaltfläche **In Zwischenablage kopieren**) oder über einen Speicherdialog als frei benennbare Textdatei an eine beliebige Stelle exportiert werden (Schaltfläche **Speichern**). Die Schaltfläche **Import** startet den Importprozess. Das Fenster verfügt außerdem über ein eigenes Kontextmenü, welches mit einem rechten Mausklick geöffnet werden kann:
  - **Suche** erlaubt die Eingabe einer zu suchenden Zeichenkette in einem separaten Fenster und findet ihr nächstes Auftreten gemäß der Leserichtung relativ zur Position der per Mausklick setzbaren Schreibmarke. Bei der Suche wird Groß- und Kleinschreibung unterschieden.

- **Alles markieren** markiert den gesamten Text. Alternativ kann mit dem Mauszeiger ein beliebiger Textausschnitt markiert werden.
- **Kopieren** kopiert den gewählten Textbereich in die Zwischenablage des Betriebssystems.

Speichern speichert die in der Konfigurationsübersicht für dieses Wissensnetz aktuell getroffene Auswahl an Konfigurationen als XML-Datei. Über einen Speicherdialog werden Name und Ort der XML-Datei (Standarddateiname: *instruction.xml*) festgelegt.

Laden greift auf das Dateisystem des Betriebssystems zu, um eine zuvor gespeicherte Auswahl an Konfigurationen für dieses Wissensnetz aus einer XML-Datei (Standarddateiname: *instruction.xml*) zu laden.

Aktualisieren fügt dem Wissensnetz die mit dem Attributdatentyp Boolesch ausgestatteten Attributtypen

- *XML: Alle Objekte exportieren*,
- *XML: Direkte Objekte exportieren*,
- *XML: Typ und alle Untertypen nicht exportieren* und
- *XML: Untertypen nicht exportieren*

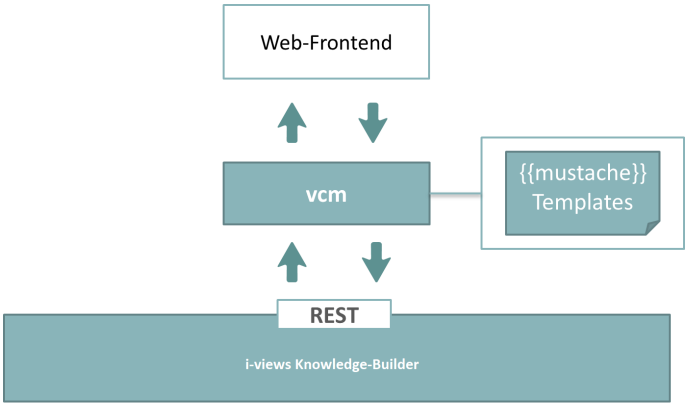
hinzu, falls sie darin noch nicht existieren. Diese Attributtypen werden benötigt, um bei einem Export einer Konfiguration des Konfigurationstyps Wissensnetz auszuwählen, welche in dieser Konfiguration befindlichen Elemente und Elementtypen jeweils exportiert und nicht exportiert werden sollen. Dazu werden diese Attributtypen über den Knowledge-Builder an passende Objekttypen gehängt und mit passenden Attributwerten versehen.

Soweit nicht anders über diese Attributwerte konfiguriert, gilt für jeden Objekttyp, dass er selbst exportiert wird, nicht aber seine Objekte. Wird ein Objekt oder Objekttyp exportiert, werden alle direkt mit ihm verbundenen Attribute und Relationen sowie deren Attribut- respektive Relationstypen ebenfalls exportiert.

3 ViewConfig-Mapper

3.1 Introduction

Mit dem Viewconfig-Mapper (kurz VCM) können über einen einfachen Weg View-Konfigurationen in ein Web-Frontend transportiert und dort dargestellt werden. Dazu wird das in der View-Konfiguration generierte JSON über die REST-Schnittstelle von i-views in das Frontend transportiert und dort mithilfe von Mustache-Templates in HTML übersetzt.



Außerdem werden gängige Interaktionen wie z.B. die Pflege der Inhalte direkt unterstützt und die Möglichkeit geboten, benutzerspezifische Aktionen, die in der View-Konfiguration definiert wurden, über vcm im Frontend auszuführen.

Der Viewconfig Mapper ist eine Single-Page-Applikation, die client-seitig im Web-Browser läuft. Sie verwendet ractive (ractive.js.org) für eine interaktive und reaktive Anwendung, die auf mustache-Templates (mustache.github.io/) basiert.

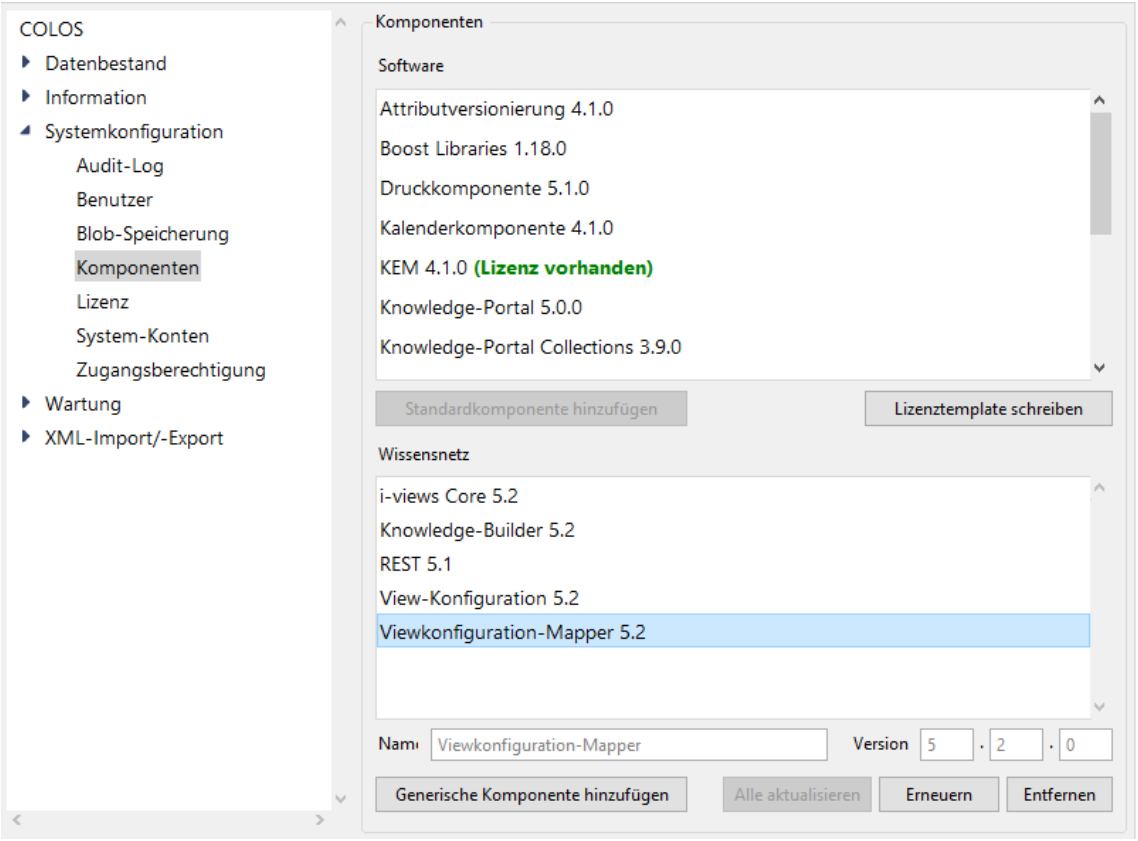
3.2 Configuration

Das übliche Vorgehen sieht die Aktivierung der ViewconfigMapper-Komponente im Wissensnetz und das Anlegen eines Anpassungsprojekts vor, in das vcm eingebunden wird. Um das Look&Feel anzupassen, kann es schon ausreichen, Änderungen allein im CSS vorzunehmen. vcm unterstützt LESS (lesscss.org/). Für aufwendigere Anpassungen können auch die Templates geändert oder erweitert werden.

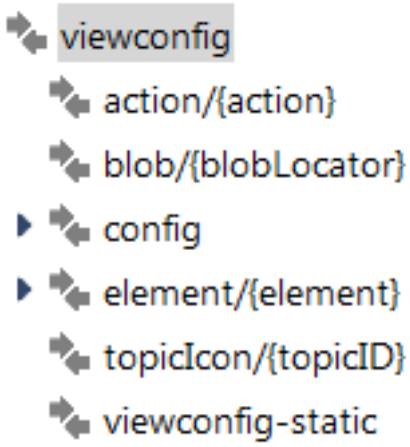
Grunt (gruntjs.com/) wird als TaskRunner verwendet, und als Package Manager Bower (bower.io/). Genaueres dazu sowie eine Auflistung der Grunt-Tasks ist in der README.md im Projekt verfügbar.

3.2.1 Die ViewconfigMapper-Komponente

Um den Viewconfig Mapper zu verwenden, ist zunächst das Aktivieren der entsprechenden Komponente im Admin-Tool Voraussetzung.



Die Komponente sorgt für das Anlegen der speziell benötigten Eigenschaften in der View-Konfiguration und legt außerdem alle REST-Services an, die vcm benötigt. (Achtung: Alle Requests sind so vorkonfiguriert, dass sie eine Authentifizierung erwarten. Für eine Authentifizierung wird am Objekt des Nutzers das Attribut password und login benötigt, dessen Schema durch die Komponente erzeugt wird. Die Verknüpfung des Benutzers in den Einstellungen des Knowledge-Builders ist hierfür nicht nötig.)



Diese sind im einzelnen:

- action
- blob

- config
- element
- topicIcon
- viewconfig-static

"action" und "element" übernehmen die gesamte Kommunikation zwischen dem Viewconfig Mapper und i-views. "blob" und "topicIcon" sind für das Ausliefern der Mediendaten innerhalb eines Netzes zuständig. "viewconfig-static" definiert den Bereich der REST-Bridge, indem sich die VCM-Frontend-Dateien (Skripte, Templates, etc.) befinden. "config" wird bei der Initialisierung von vcm aufgerufen, um grundsätzliche Konfigurationen (wie Sprache und Start-Topic) zu konfigurieren. Sämtliche REST-Services sind vorkonfiguriert, so dass ein Anpassen nicht unbedingt erforderlich ist. Jedoch ist es empfehlenswert, den "config"-Request anzupassen:

```
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"));

}
```

Anzupassende Werte sind

- **application**: Die in der View-Konfiguration konfigurierte Anwendung für den Viewconfig Mapper. Dies ist standardmäßig "viewConfigMapper" und muss daher nicht angepasst werden.
- **user**: Nutzerkonfiguration. Die derzeitige vcm-Version liest lediglich den konfigurierten Namen des Nutzers zur Anzeige im Frontend aus.
- **startElement**: ID oder interner Name des Topics, das beim Aufrufen der Startseite initial angezeigt werden soll. Vorkonfiguriert ist der Wurzeltyp des Wissensnetzes. Dies sollte angepasst werden.
- **language**: Vorkonfiguriert ist die Sprache des anfragenden Browsers. Für spezifische

Spracheinstellungen sollte dieses Attribut konfiguriert werden. Jegliche I18N-Einstellungen sind in den Frontend-Templates vorgesehen und können außerdem im Attribut "translations" erweitert werden. Anpassungen daran sollten in diesen Templates vorgenommen werden. An dieser Stelle geht es lediglich um die Festlegung der Sprache.

- **translations**: I18N-Templates befinden sich im Frontend und sollten dort angepasst werden. An dieser Stelle können sie funktional erweitert werden.

3.2.2 Anlegen eines Projekts mit dem Viewconfig Mapper

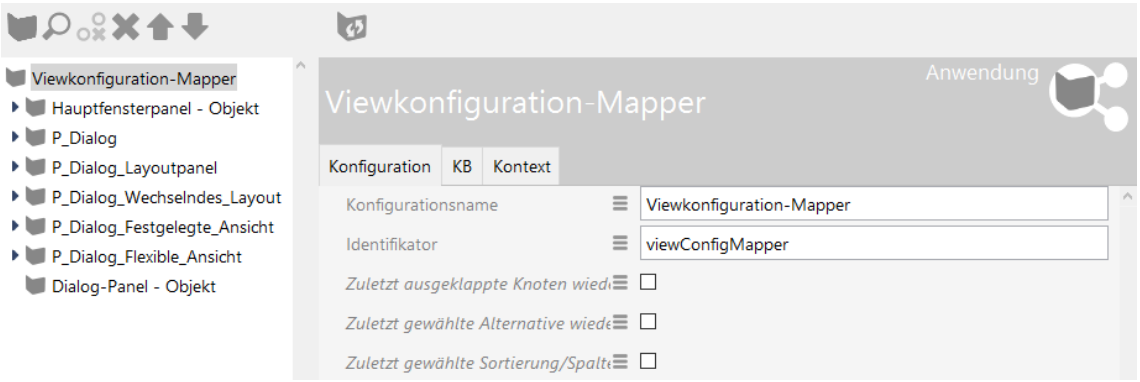
Zum einfachen Anlegen eines Anpassungsprojekts existiert im Git ein Projekt-Template unter gitlab.ivda.i-views.de/product/viewconfigmapper/grunt-init-viewconfigmapper.git. In der README.md des Projekts sind alle weiteren Schritte erklärt. Beim Initialisieren werden gewisse Parameter benötigt, so wird z.B. nach dem Basispfad für Requests und nach dem Namen der Applikation gefragt. Diese Daten sollten beim ersten Aufruf bereit liegen.

3.2.3 View-Konfigurationen für den Viewconfig Mapper

Der Viewconfig-Mapper interpretiert alle View-Konfigurationen, die in i-views erstellt wurden. Dabei gibt es jedoch ein paar Unterschiede zwischen der Verarbeitung im Knowledge-Builder und im Viewconfig-Mapper, auf die in diesem Kapitel eingegangen wird.

3.2.3.1 Panel configuration

Falls die Web-Anwendung auf einer Panel-Konfiguration basieren soll, muss die Anwendung mit der Panel-Konfiguration verknüpft werden.



Dazu wird an der Anwendung ein Objekt eines Hauptfensterpanels angehängt, an das jegliche weitere Panel-Konfigurationen angehängt werden können. Weitere Panels (wie z.B. Dialog-Panels) sind optional. Falls sie jedoch im Web-Frontend verwendet werden sollen, müssen sie über diesen Weg mit der Anwendung verbunden sein. Es reicht nicht, sie z.B. nur als Zielfenster einer Aktion zu definieren, da sie sonst für die Anzeige der Anwendung nicht berücksichtigt würden.

3.2.3.2 anwenden in

Um eine geeignete View-Konfiguration für ein semantisches Element zu bestimmen, wird einerseits nach dem Typ des Elements geschaut sowie nach dem Kontext, in dem eine View-Konfiguration verwendet werden soll. Dieser Kontext wird über die Relation "anwenden in" bestimmt. Für die Verwendung einer View-Konfiguration in der Anwendung

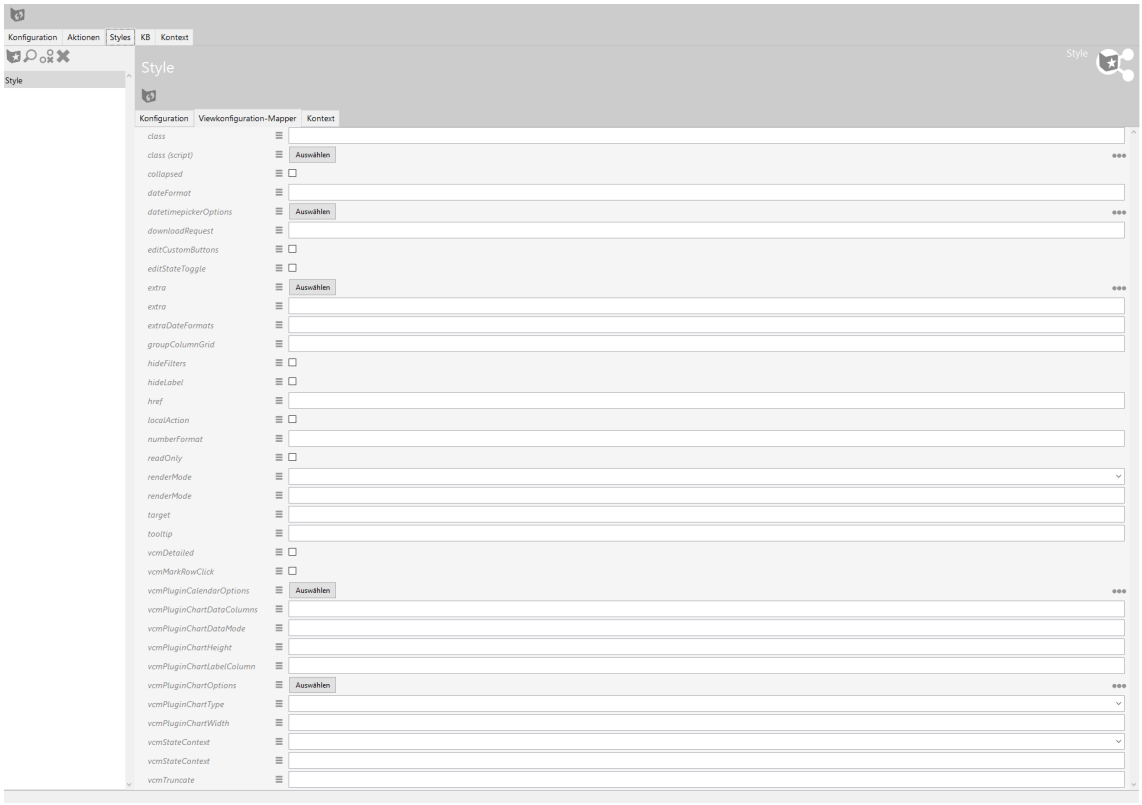
"Viewconfiguration-Mapper" sollte daher darauf geachtet werden, dass die Relation entsprechend gezogen wurde.In folgendem Beispiel wurde die Relation zu einem Panel der Anwendung "Viewconfiguration-Mapper" gezogen.

3.2.3.3 Style

Grundsätzlich wird die Darstellung in HTML über die Templates bestimmt, so dass dies nicht Teil der View-Konfiguration ist. Um aber dem Ersteller der View-Konfigurationen die Möglichkeit zu geben, in einigen Aspekten doch Einfluss auf die Darstellung zu haben, gibt es sogenannte "Styles".

Die Einstellungen zu den Styles für die Darstellung im Web-Frontend durch den Viewkonfiguration-Mapper sind unter dem Reiter "Viewkonfiguration-Mapper" verfügbar. Voraussetzung hierfür ist eine installierte Viewkonfiguration-Mapper-Komponente im KB.

Insgesamt sind mehrere Einstellungsmöglichkeiten zu den Styles vorhanden (siehe Abbildung):



Es gibt eine Reihe von Style-Elementen, die bereits in i-views definiert sind. Um welche Elemente es sich handelt und wie diese Style-Elemente im Knowledge-Builder angelegt werden, sodass sie dann mit einzelnen Elementen der View-Konfiguration einer Anwendung verknüpft werden können, wird im Folgenden erläutert.

Zunächst muss das Element der View-Konfiguration ausgewählt werden, mit dem wir ein oder mehrere Style-Elemente verknüpfen wollen. Je nach Typ des View-Konfiguration-Elements stehen verschiedene Reiter zur Konfiguration der Styles zur Verfügung ("Aktionen und Styles" -> "Styles" oder direkt "Styles"). Diesen Reiter wählen wir aus und können dann entweder ein neues Style-Element definieren oder ein bereits vorhandenes Style-Element verknüpfen . Wenn wir ein neues Style-Element definieren, müssen wir diesem zuerst einen Konfigurationsnamen geben. Auf der rechten Seite des Editors kann daraufhin die Konfiguration vorgenommen werden.

Im Folgenden werden die einzelnen Konfigurationsmöglichkeiten für ein Style-Element erläutert:

Name	Attributtyp	Konfigurationstyp	Beschreibung

class	Zeichenkette	CSS-Klasse	Styling durch Angabe einer vordefinierten CSS-Klasse im CSS des Viewconfiguration-Mappers oder im Skript "viewconfigmapper.config.GET". Siehe auch "Verwendung von CSS".
class (skript)	Verweis auf Skript	CSS per JavaScript	Styling durch Angabe von CSS-Klassen mithilfe eines JavaScript
collapsed	Boolean	Gruppe	Wird verwendet für renderMode "Panel"
dateFormat	Zeichenkette		
datetimepickerOptions	Verweis auf Skript		
downloadRequest	Zeichenkette		
extraDateFormats	Zeichenkette		
groupColumnGrid	Zeichenkette	Gruppe	Als Eingabe wird ein String mit Zahlen erwartet, die durch ein Leerzeichen oder ein Komma getrennt werden. Jede Zahl definiert die Anzahl der Spalten, wenn das Maximum 12 Spalten beträgt.
hideFilters	Boolean		Blendet die Tabellen-Suchfilter des Tabellen-Kopfes aus
hideLabel	Boolean		Blendet die Beschriftung eines View-Konfiguration-Elements aus (Beschriftung des Reiters einer Alternative bleibt bestehen)
href	Zeichenkette	Hyperlink	Link zu einer Webseite oder einem Ordnerpfad nach dem HTML-Standard
localAction	Boolean		Beschränkt die Wirkung einer Aktion auf das aktuelle Panel

numberFormat	Zeichenkette		Bestimmt die Formatierung von Werten des Wertetyps "Fließkommazahl" (Float) oder "Ganzzahl" (Integer)
readOnly	Boolean	Eigenschaften	Die Eigenschaften des View-Konfiguration-Elements können in der Anwendung nur gelesen und nicht bearbeitet werden; der "Bearbeiten-Button" ist nicht sichtbar
renderMode	Auswahl	Eigenschaft	Siehe Unterkapitel "RenderModes"
renderMode	Zeichenkette	Eigenschaft	Siehe Unterkapitel "RenderModes"
target	Zeichenkette		
tooltip	Zeichenkette	Kontexthilfe	Hinweis, der bei Mouse-Over eingeblendet wird
vcmDetailed	Boolean		
vcmMarkRowClick	Boolean	Tabelle	Markiert eine angeklickte Tabellenzeile
vcmPluginCalendarOptions	Verweis auf Skript	VCM-Plugin	Default-Werte, die sich per Skript festlegen lassen, bspw. Startdatum beim Aufruf der Kalender-Ansicht
vcmPluginChartDataColumns	Zeichenkette	VCM-Plugin	Zeichenkette mit Angabe zu den Spalten, deren Inhalt visualisiert werden soll; Standard-Wert: "1-n"
vcmPluginChartDataMode	Zeichenkette	VCM-Plugin	Wird verwendet, wenn die Daten der zugrundeliegenden Tablle entweder zeilenweise ("rows") oder spaltenweise ("columns") für das darzustellende Diagramm ausgelesen werden soll; bei fehlender Angabe gilt per Default der DataMode "rows"

vcmPlugin-ChartHeight	Zeichenkette	VCM-Plugin	Absolute Höhe eines Diagramms in Pixel (Bsp.: "300px")
vcmPluginChartLabelColumn	Zeichenkette	VCM-Plugin	Nummer der Spalte, deren Inhalt für die Beschriftung verwendet werden soll; Standard-Wert: 0
vcmPluginChartOptions	Verweis auf Skript	VCM-Plugin	Skript, mit dem sich die Darstellung von Bestandteilen des Diagramms steuern lässt: Darstellung von Legenden, Skalierung von Achsen etc.
vcmPluginChart-Type	Auswahl	VCM-Plugin	Auswahlmöglichkeiten für den RenderMode "chart" (anwendbar für Tabellen): • bar • doughnut • line • pie • pole • radar Standard-Wert: "line"
vcmPlugin-ChartWidth	Zeichenkette	VCM-Plugin	Absolute Breite eines Diagramms in Pixel (bsp.: "380px")
vcmStateContext	Auswahl		Auswahlmöglichkeiten: • global • page • none

3.2.3.3.1 Definition eigener Style-Attribute

Abgesehen von der Anwendung vordefinierter Style-Attribute ist es möglich, eigene Style-Attribute zu definieren.

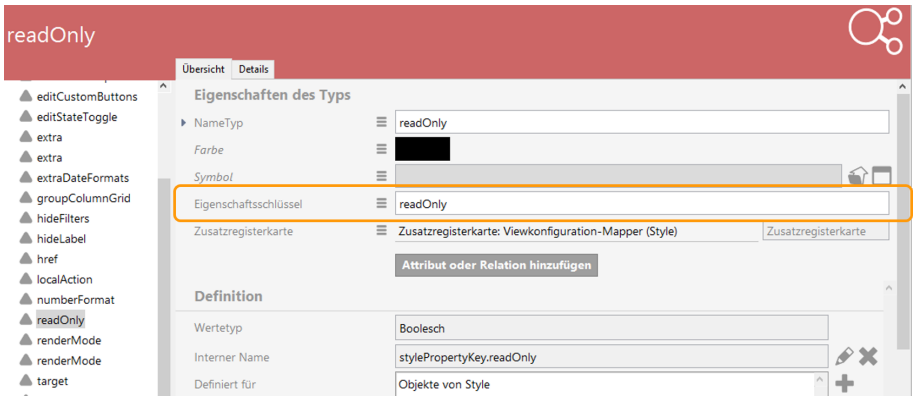
Unter View-Konfiguration > Attributtypen können die Attribute der Styles angelegt werden.

Damit das Style-Attribut auch in die JSON-Ausgabe geschrieben wird, muss bei dem Attribut

im Schema noch eine Ergänzung vorgenommen werden. Zum Schema gelangt man, indem man im Menü ☰ des Attributs auf "Schema" klickt. Im Schema muss man dann das Attribut "Eigenschaftsschlüssel" hinzufügen und dort den Namen des Attributs eingeben.

Im Feld "definiert für" muss der Eintrag "Objekte von Style" stehen. Das Hinzufügen des Eintrags erfolgt durch einen Klick auf das Plus-Symbol des "Hinzufügen"-Buttons. Nach der Eingabe des Suchebegriffs "Style" erfolgt eine Anzeige von Suchergebnissen, aus welchen der Eintrag "Style (View-Konfiguration)" auszuwählen ist.

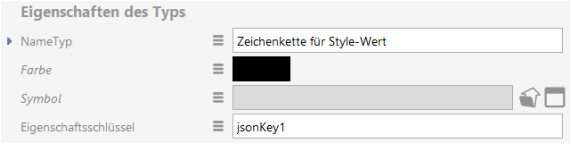
Schließlich muss noch die Zusatzregisterkarte ausgewählt werden, in der das neue Style-Element angezeigt werden soll.



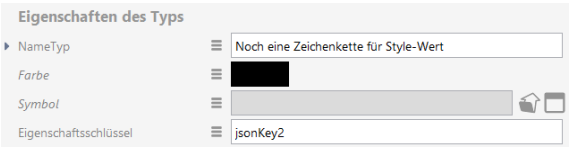
In der JSON-Ausgabe werden dann die Schlüssel-Wert-Paare (*StylePropertyKey* -> Style-Eigenschaft) als Array unter dem Schlüssel *additionalConfig* rausgeschrieben.

Beispiel

Konfiguration des Typs *Zeichenkette für Style-Wert*

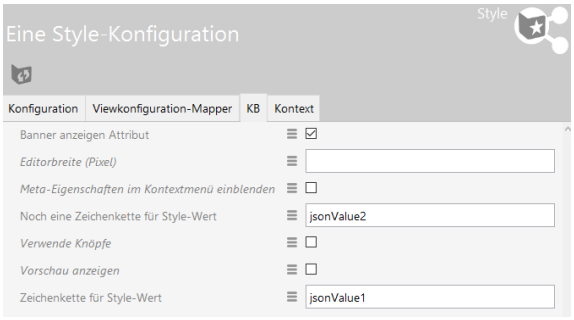


Konfiguration des Typs *Noch eine Zeichenkette für Style-Wert*



Konfiguration des Typs *Banner anzeigen Attribut*

Konfiguration des Objektes *Eine Style-Konfiguration* vom Typ *Style*



JSON-Ausgabe:

```
"properties": [{
    "values": [{ ... }],
    "label": "Vorname",
    "additionalConfig": {
        "jsonKey1": ["jsonValue1"],
        "jsonKey2": ["jsonValue2"],
        "Banner anzeigen": ["true"]
    },
    "viewId": "ID34304_461524079",
    "schema": { ... }
}
```

3.2.3.3.2 RenderModes

Mithilfe von renderModes können zusätzliche, vordefinierte Style-Eigenschaften angewendet werden.

RenderModes sind in der View-Konfiguration in den Styles unter dem Reiter "Viewkonfiguration-Mapper" verfügbar, einmal per Dropdown-Menü und zusätzlich per Eingabezeile. Hierbei hat der frei wählbare Wert per Eingabezeile eine höhere Präzedenz, überschreibt also einen Wert, der per Dropdown gewählt wurde.

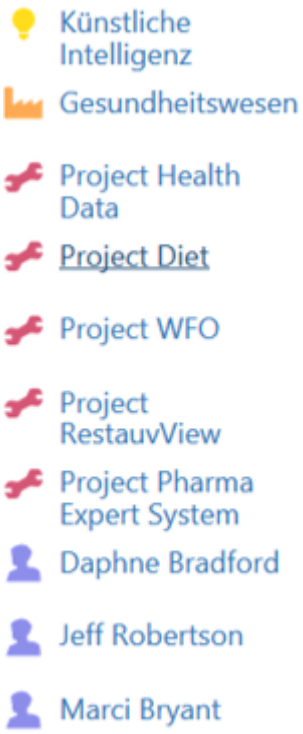
Die im Dropdown-Menü verfügbaren renderModes sind wie folgt:

render-Mode	Erläuterung	Anwend-barkeit
bread-crumb	Zeigt die Hierarchie mit Pfadnavigation an	Hierarchie
calendar	Darstellung von Datumsangaben in einer Kalender-Ansicht; Grundlage hierfür ist eine Tabelle, die Attribute des Wertetyps *Zeit*, *Datum*, *Datum und Uhrzeit*, *Flexible Zeit* oder *Intervall* mit Typ Datum und Uhrzeit beinhaltet.	Tabelle

chart	Darstellung der Daten einer Tabelle in einem Diagramm. Unter *vcmPluginChartType* kann die Art des Diagramms ausgewählt werden. Unter *vcmPluginChartOptions* kann mittels Skript eine genauere Formatierung des Diagramms vorgenommen werden, bspw. Achsen-Skalierung, Anzeige von Legenden etc.	Tabelle
download	Link auf Dateidownload	Aktion
edit	Untergeordnete Eigenschaften werden editierbar	Gruppe

external	Erzeugt in Verbindung mit href einen externen Link; lässt sich bspw. in Kombination mit *Symbol* und *Tooltip* verwenden.
	Für dynamische Links kann im href-Attribut ein Bezeichner in geschweiften Klammern verwendet werden. Wenn das *extra*-Skript ein JavaScript-Objekt mit einem Wert für den Bezeichner liefert, wird dieser automatisch eingefügt.
	Zum Beispiel kann auf folgende Weise eine Google-Suche nach dem Namen des aktuellen Objekts ausgelöst werden:
	href: https://www.google.com/search?q={search}
	extra-Skript:
	
	`function additionalPropertyValue(element, context) {`
	`  return { search: element.name() }`
	`}`
		Aktion

grid	In Kombination mit der Eigenschaft *groupColumnGrid* kann eine Aufteilung des Layouts nach einem vorkonfigurierten Raster mit 12 Einheiten vorgenommen werden. Je nach Anzahl der Elemente kann die relative Verteilung mit Angabe der zur Verfügung stehenden Einheiten per Element definiert werden. Bsp.: 5 3 4	Gruppe
html	Zeigt die Zeichenkette ohne Maskierung	Zeichenketten-Eigenschaft
markdown	Wandelt mit Auszeichnungen versehene Textteile in Text mit Hervorhebungen durch Inline-Formatierung um	Text oder Zeichenketten-Attribut

medialist	Dastellung der Tabelleneinträge als HTML-Textlink; Anzeige der Elemente mitsamt Symbol 	Tabelle
multiline	Wird benötigt, wenn in einer Edit-View das Eingabefeld zu einer Zeichenkette mehrzeilig dargestellt werden soll	Eigen-schaft
nolink	Das Relationsziel wird nicht verlinkt, sondern nur textuell angezeigt.	Relations-Eigenschaft
panel	Bewirkt die Darstellung als aufklappbare Gruppe	Gruppe
pre	Zeigt die Zeichenkette als preformatierten und scrollbaren Text an	Zeichenketten-Eigenschaft
table	Tabellarische Darstellung	Gruppe
timeline	Darstellung eines Datensatzes in Form einer Timeline; Anord-nung kann senkrecht oder waagrecht erfolgen	Skript-generierte View in Gruppe
transla-tions	Zeigt Sprachvarianten an (beim Zeichenketten-Attribut mit den jeweiligen Flaggen-Icons)	Eigen-schaft

Die in der Eingabezeile verfügbaren renderModes stehen im Zusammenhang mit Bootstrap. Dazu zählen beispielsweise folgende renderModes:

render-Mode	Erläuterung	Anwend-barkeit
email	Erzeugt einen Link auf die Email-Adresse	Zeichenketten-Eigenschaft
image	Zeigt ein Icon an der Aktion an	Aktion
jumbotron	Hervorgehobene Darstellung. Siehe getbootstrap.com/docs/4.1/components/jumbotron/	Gruppe
well	Erzeugt eine Box mit eingedrücktem Effekt. Siehe getbootstrap.com/docs/3.3/components/#wells	Gruppe

3.2.3.4 Globaler Kontext

Der Viewconfig-Mapper verwendet die Session Storage, um die Menüführung für den Nutzer verständlich zu gestalten. Damit dies auch funktioniert, wenn aktualisierte Daten aus dem Backend geladen werden, wird bei allen Aktionsrequests ein für das Backend verständlicher Teil aus der Session Storage mitgeschickt, der sogenannte globale Kontext. Dieser hat z.B. Einfluss auf das Auswählen des aktiven Reiters einer Alternative oder die Sortierung einer Tabelle.

3.2.4 Login-Configuration

3.2.4.1 JWT Authentication

3.2.4.1.1 Configuring the login form

Das Loginformular kann über folgende Übersetzungsschlüssel angepasst werden:

Schlüssel	**Beschreibung**
login.form.title	Überschrift des Formulars
login.form.message	Beschreibungs-/Willkommenstext
login.form.username.label	Label des Benutzernamenfeldes
login.form.username.placeholder	Placeholder des Benutzernamenfeldes
login.form.password.label	Label des Passwortfeldes
login.form.password.placeholder	Placeholder des Passwortfeldes

3.2.5 Anpassen der Templates

Das Projekt-Template enthält die Verzeichnisse components/ und partials/ unter dem Verzeichnis webroot/. In beiden Verzeichnissen finden sich Beispiele für ViewconfigMapper-Komponenten und -Partials. An diesen Stellen können neue Templates hinzugefügt werden. Die Basis-Templates des ViewconfigMappers stehen weiterhin zur Verfügung, so dass nur für spezielle Anpassungen Templates erstellt werden müssen.

Im Verzeichnis js/ befindet sich eine JavaScript-Datei, in der der ViewconfigMapper initialisiert wird.

```
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);
```

Der ViewconfigMapper bekommt die Konfigurationseinstellungen, Partials, Komponenten und Übersetzungen übergeben. Außerdem wird festgelegt, an welche Stelle der Inhalt gerendert werden soll (Im Beispiel an <div id=viewconfigmapper"/>). Für Partials und Komponenten ist es nur wichtig, dass sie sich in den entsprechenden Verzeichnissen befinden, da Grunt-Tasks existieren, die die Dateien extrahieren und in eigene JavaScript-Files auslagern.

Werte für application, ajaxBasePath und instanceId würden beim Initialisierungsaufruf des Projekt-Templates gesetzt.

3.2.6 Betreiben des Frontends

Das Frontend kann über grunt gebaut werden. Die zum Betrieb benötigten Dateien befinden sich nach der Generierung im Verzeichnis /webroot. Der Einstieg erfolgt, solange es nicht anders konfiguriert wurde, über die Startseite index.html.

Im einfachsten Fall können die Dateien lokal liegen und können dann lediglich client-seitig genutzt werden.

Um das Frontend zugänglich zu machen, gibt es mehrere Möglichkeiten. Die Komponente ViewconfigMapper generiert automatisch einen REST-Service, der statische Dateien ausliefern kann. Dies kann man nutzen, indem man die Dateien des Verzeichnis webroot in das entsprechende Verzeichnis in der genutzten REST-Bridge legt (Default ist viewconfig-static). Danach lässt sich das Frontend in der Defaultkonfiguration über HOST:PORT/viewconfig/viewconfig-static/index.html ansprechen. Zusätzlich ist es aber auch möglich, die Dateien über einen entsprechenden Server auszuliefern.

3.3 Actions

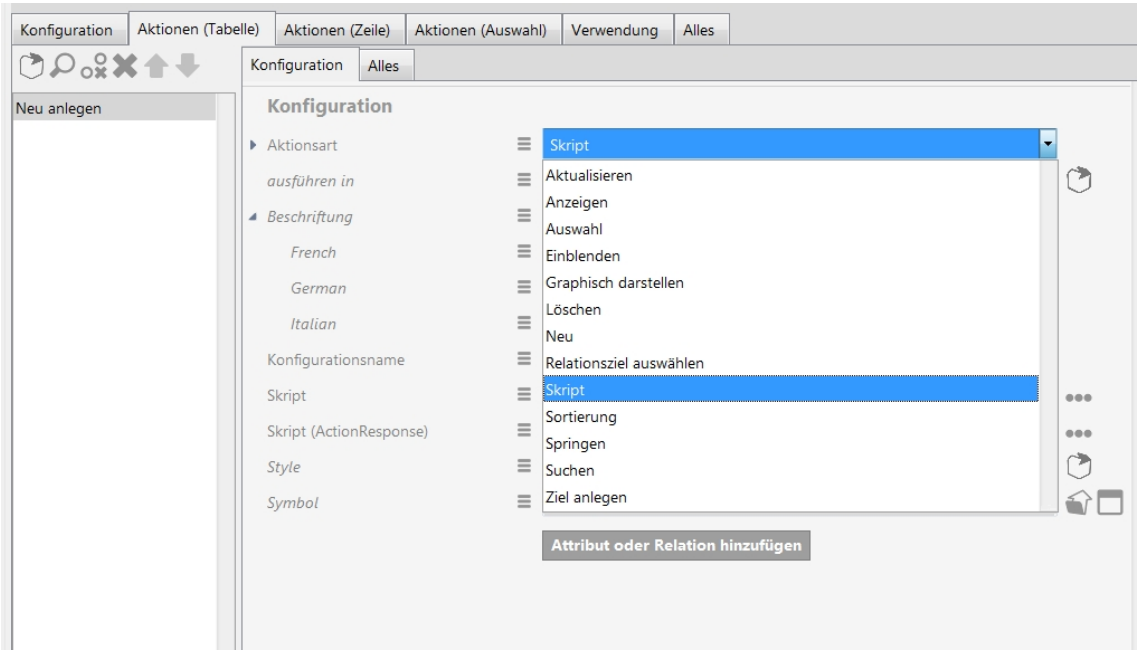
In diesem Kapitel werden alle VCM-spezifischen Aktionseinstellungsmöglichkeiten und Skript-Parameter (context, actionResponse) beschrieben.

vcm unterstützt Standard-Interaktionen wie das Editieren von Inhalten, ohne dass dies extra konfiguriert werden muss. Es ist aber möglich, in der View-Konfiguration benutzerdefinierte Aktionen zu definieren, die dann auch in vcm ausgeführt werden können.

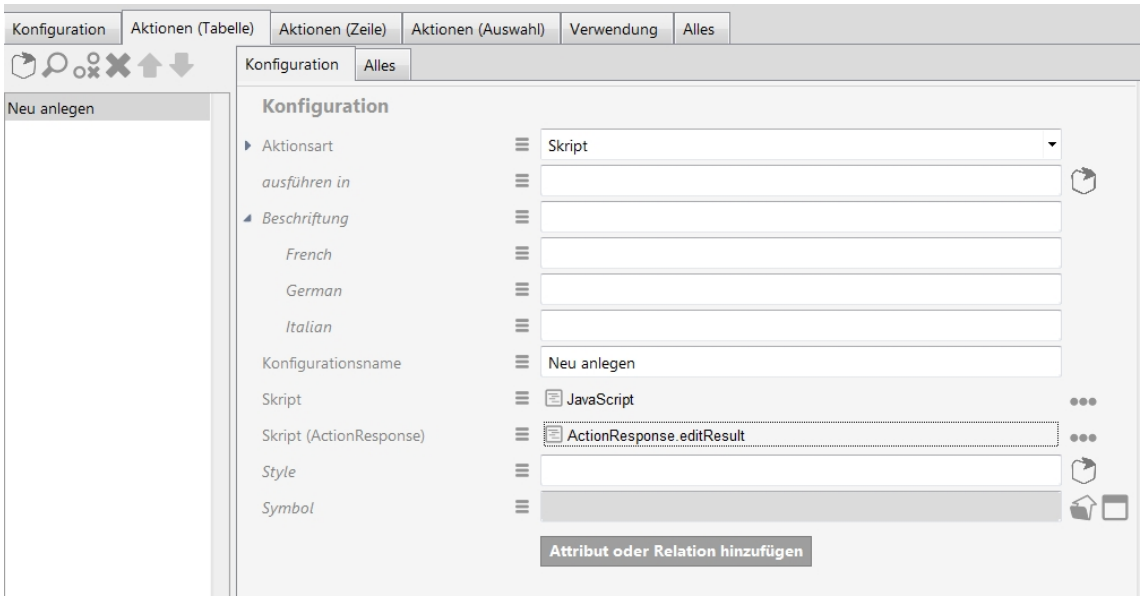
Es gibt zwei Sorten von benutzerdefinierten Aktionen:

- Standard-Aktion mit benutzerdefiniertem Rückgabewert
- Skript-Aktion

Die Auswahl geschieht über ein Dropdown-Menü.



Für eine Skript-Aktion muss in diesem Menü "Skript" ausgewählt werden und in der Liste unter dem Eintrag "Skript" ein Skript angelegt werden.



Um den Rückgabewert einer Aktion zu bestimmen, kann für beide Aktionsarten unter dem Eintrag "Skript (Action Response)" ein Skript angelegt werden.

Ein typisches Rückgabeskript:

```
function actionResponse(element, context, actionResult){
 var actionResponse = new $k.ActionResponse();
 actionResponse.setFollowup("show");
 actionResponse.setData({
                elementId: actionResult.idString(),
                viewMode: "edit"
            });
 return actionResponse;
}
```

Mögliche Follow-Ups:

- show - Die übergebenen Inhalte werden geladen und auf der ganzen Seite dargestellt.
- refresh - Alle Komponenten aktualisieren ihre Inhalte.
- reload - Die Seite wird neu geladen.
- update - Die übergebenen Inhalte werden geladen und in der View dargestellt, die in "ausführen in" bestimmt wurde. Es kommt zu einem Fehler, wenn "ausführen in" nicht gesetzt ist.

Außerdem ist es möglich, eigene Follow-Ups zu definieren. Die Interaktion mit dem Frontend muss dann aber darauf angepasst werden, da vcm dafür standardmäßig nicht ausgelegt ist. Auf der anderen Seite ist es aber auch möglich, die derzeitigen Reaktionen auf einen Follow-Up im eigenen Abpassungsprojekt zu überschreiben.

Weitere mögliche Rückgabewerte:

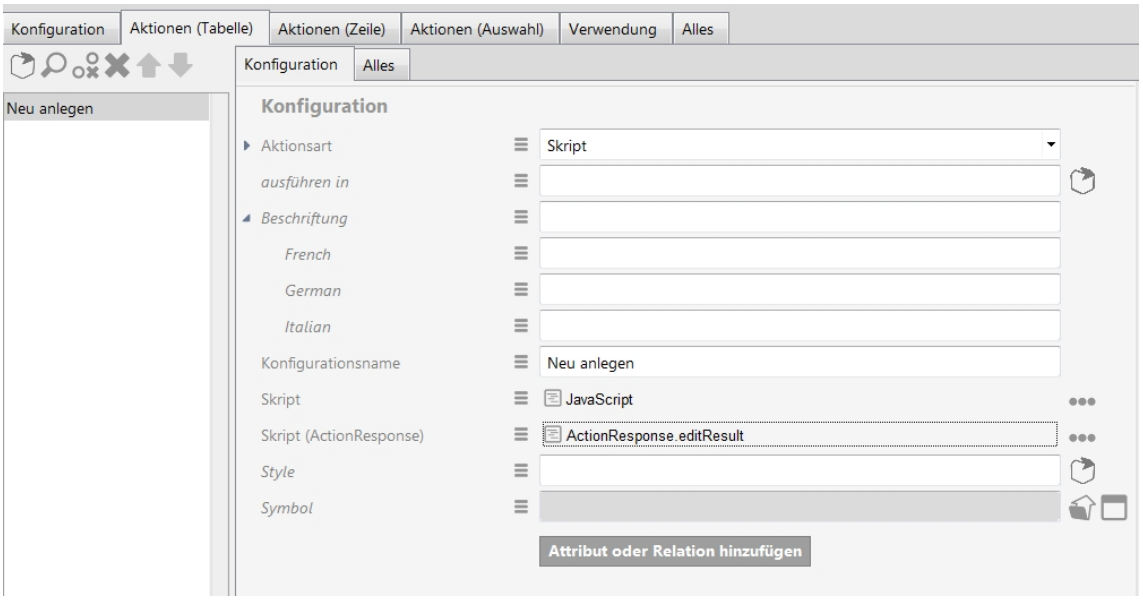
- elementId - ID des anzuzeigenden Elements
- viewMode - Derzeit gibt es nur die Unterscheidung zwischen Lese- und Schreibmodus, wobei standardmäßig der Lesemodus angenommen wird und nur bei viewMode: "edit"

der Schreibmodus angezeigt wird.

Auch hier ist es möglich, eigene Werte zu definieren und sein Frontend darauf anzupassen.

3.3.1 ausführen in

Beim Anlegen einer benutzerdefinierten Aktion kann auch die Relation "ausführen in" gezogen werden. Dies bewirkt, dass die zurückgegebenen Daten nicht auf alle vcm-Inhalte angewendet werden, sondern sich die Änderung nur auf eine bestimmte View bezieht. Diese View muss als Relationsziel von "ausführen in" gesetzt werden.

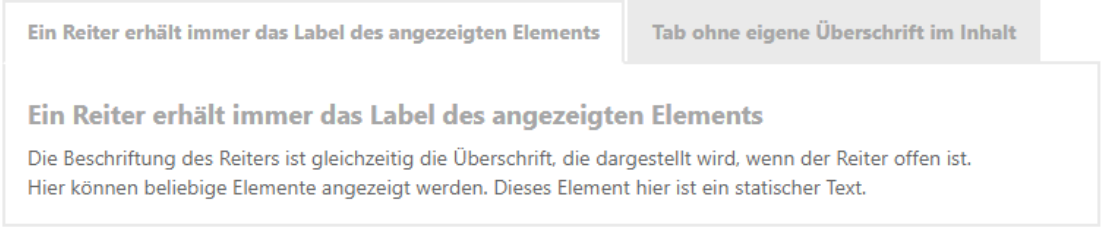


3.4 View-Konfigurationselemente

3.4.1 General parameters

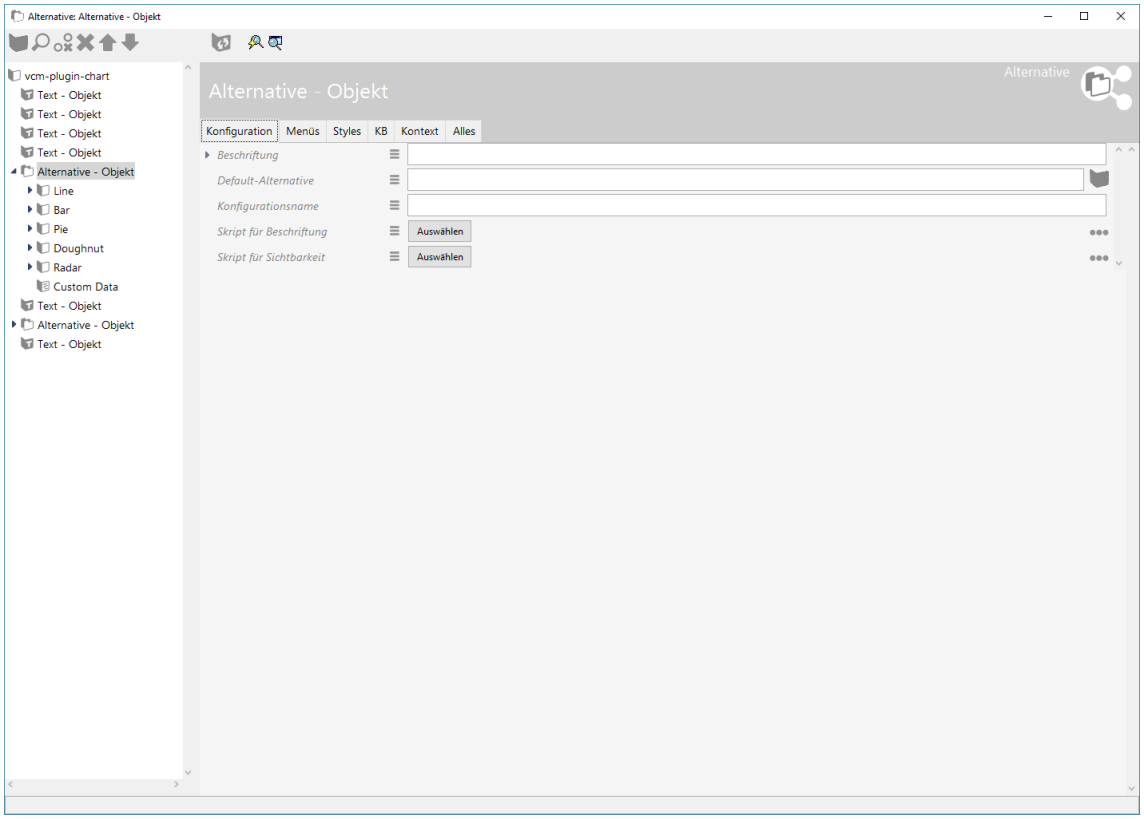
3.4.2 Alternative

Bei einer Alternative-View handelt es sich um eine Sammel-View für andere Views, d.h. mithilfe dieses Viewtyps lassen sich Views zusammenfassen, die Daten zu einem gemeinsamen Objekt anzeigen (z.B. eine Eigenschaften-View mit den Lebensdaten eines Künstler und eine Tabelle-View, in der die Werke des Künstlers aufgelistet werden). Im Gegensatz zur Gruppe-View werden die zusammengefassten Views aber nicht gleichzeitig, sondern abwechselnd (z.B. über Reiter) angezeigt.



Um die Views zusammenzufassen, werden die entsprechenden Views als Unterviews an

die Alternative-View angehängt. Ihre Position entscheidet dabei über die Reihenfolge der Anzeige. Deswegen kann die Position über die Pfeiltasten geändert werden.



Im Reiter "Konfiguration" gibt es Möglichkeiten zur Bestimmung der allgemeinen Darstellung der Liste:

Beschriftung	Der hier eingegebene Wert erscheint als Überschrift der Alternative
Default-Alternative	Per Default wird die erste angehängt View angezeigt. Falls man z.B. möchte, dass die View auf dem dritten Reiter zuerst angezeigt wird, kann man diese View hier angeben. Im Frontend wird sich die zuletzt angezeigte View innerhalb einer Session gemerkt, so dass der Nutzer, wenn er innerhalb einer Session eine Alternative-View mehrfach betrachtet, auf den zuletzt von ihm betrachteten Reiter landet.
Konfigurationsname	Über den Konfigurationsnamen können Views und Panels identifiziert werden.
Skript für Beschriftung	Alternativ zu "Beschriftung" kann der Titel der Alternative in einem Skript bestimmt werden.

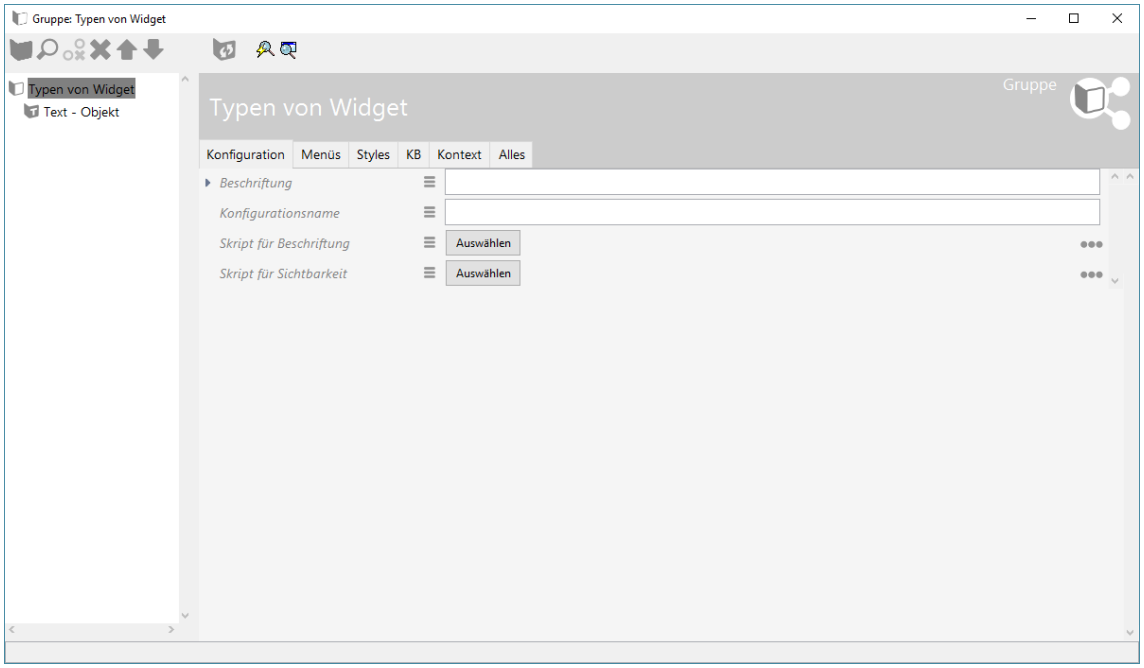
Skript für Sicht-barkeit	Über dieses Skript kann festgelegt werden, ob und unter welchen Bedingungen die Alternative angezeigt werden soll.

Im Reiter"Menüs" lassen sich Aktionen zu der Alternative konfigurieren, und im Reiter "Styles" können bestimmte Darstellungsoptionen ausgewählt werden. Der "KB"-Reiter enthält Optionen, die nur für den Knowledge-Builder gelten und im Web-Frontend keine Anwendung finden. Über den "Kontext"-Reiter lässt sich konfigurieren, für welche Objekttypen die Alternative-View verwendet werden soll und in welchen Anwendungskontexten.

Eine Alternative-View sollte dann verwendet werden, wenn mehrere Views auf den Daten eines Objekts oder Typs basieren, aber nicht gleichzeitig sondern alternativ angezeigt werden sollen.

3.4.3 Group

Bei einer Gruppen-View handelt es sich um eine Sammel-View für andere Views, d.h. mithilfe dieses Viewtyps lassen sich Views gruppieren, die Daten zu einem gemeinsamen Objekt anzeigen (z.B. eine Eigenschaften-View mit den Lebensdaten eines Künstler und eine Tabelle-View, in der die Werke des Künstlers aufgelistet werden). Um die Views zu gruppieren, werden die entsprechenden Views als Unterviews an die Gruppe-View angehängt. Ihre Position entscheidet dabei über die Reihenfolge der Anzeige. Deswegen kann die Position über die Pfeilbuttons geändert werden.



Im Reiter "Konfiguration" gibt es Möglichkeiten zur Bestimmung der allgemeinen Darstellung der Liste:

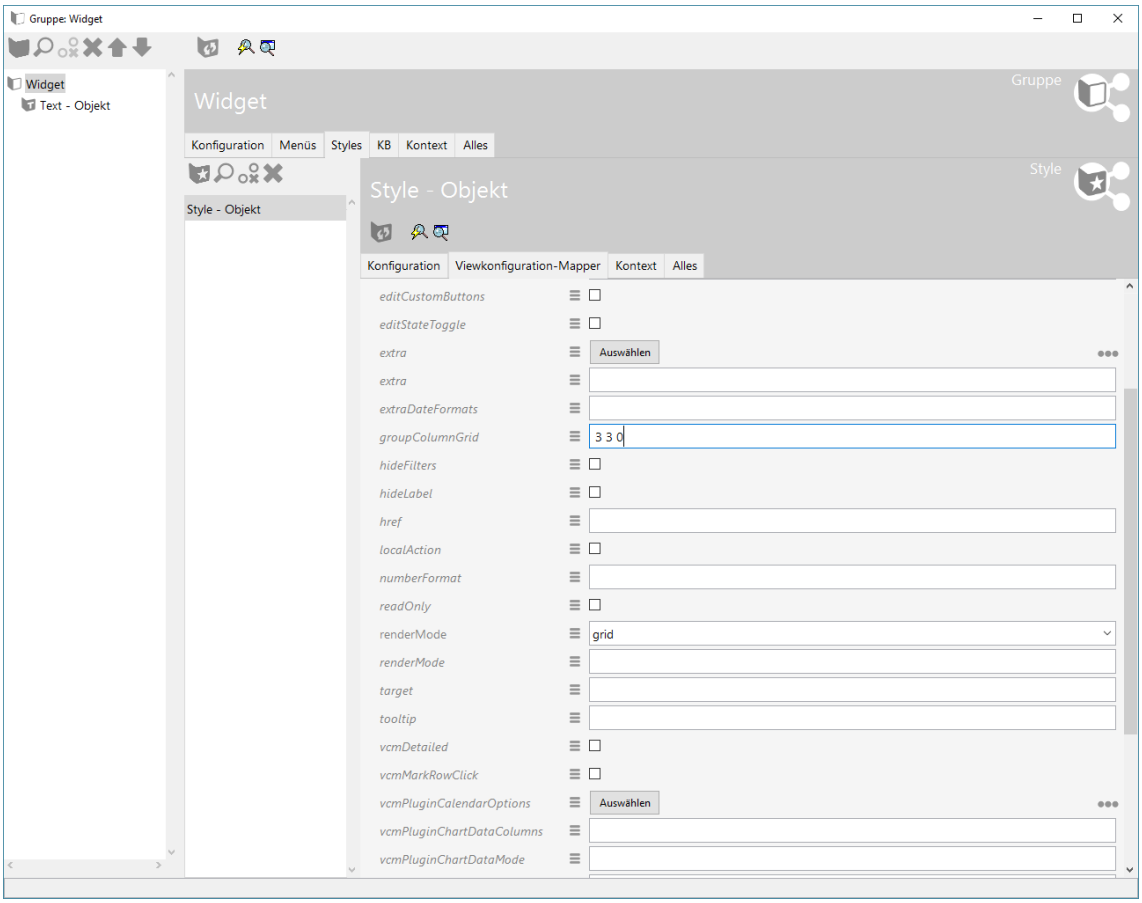
Beschriftung	Der hier eingegebene Wert erscheint als Überschrift der Gruppe

Konfigurationsname	Über den Konfigurationsnamen können Views und Panels identifiziert werden.
Skript für Beschriftung	Alternativ zu "Beschriftung" kann der Titel der Gruppe in einem Skript bestimmt werden.
Skript für Sichtbarkeit	Über dieses Skript kann festgelegt werden, ob die Gruppe angezeigt werden soll.

Im Reiter"Menüs" lassen sich Aktionen zu der Gruppe konfigurieren, und im Reiter "Styles" können bestimmte Darstellungsoptionen ausgewählt werden. Der "KB"-Reiter enthält Optionen, die nur für den Knowledge-Builder gelten und im Web-Frontend keine Anwendung finden. Über den "Kontext"-Reiter lässt sich konfigurieren, für welche Objekttypen die Gruppen-View verwendet werden soll und in welchen Anwendungskontexten.

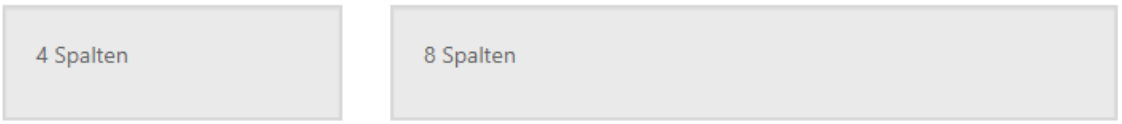
Eine Gruppen-View sollte dann verwendet werden, wenn mehrere Views gleichzeitig gruppiert angezeigt werden sollen, die auf den Daten eines Objekts oder Typs basieren. Im Gegensatz dazu steht die Alternative, die die enthaltenen Views zu einem Objekt alternierend (z.B. als reiter) anzeigt.

Im Web-Frontend gibt es unterschiedliche Möglichkeiten, die gruppierten Views anzuzeigen. Falls nicht anders konfiguriert, werden die Views untereinander angeordnet. Mit einem Style kann man eine horizontale oder Grid-Anordnung ermöglichen:



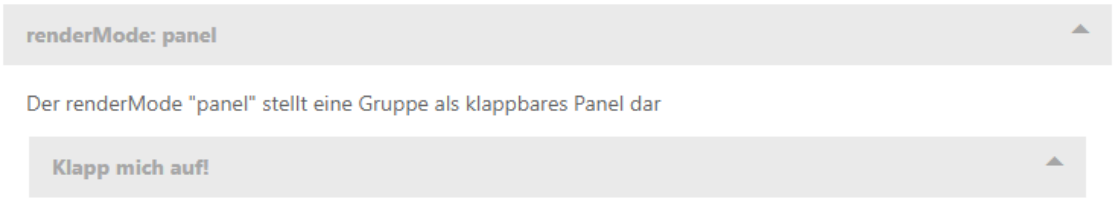
Dazu wird ein Style-Objekt an der Gruppe-View angelegt und im Reiter Viewkonfiguration-

Mapper wird als "renderMode" der Wert "grid" ausgewählt und bei "groupColumnGrid" die gewünschte Grid-Konfiguration eingetragen.



Die Beispiel-View hat das Grid "4 8 0". Die Summe der Summanden muss immer zwölf ergeben.

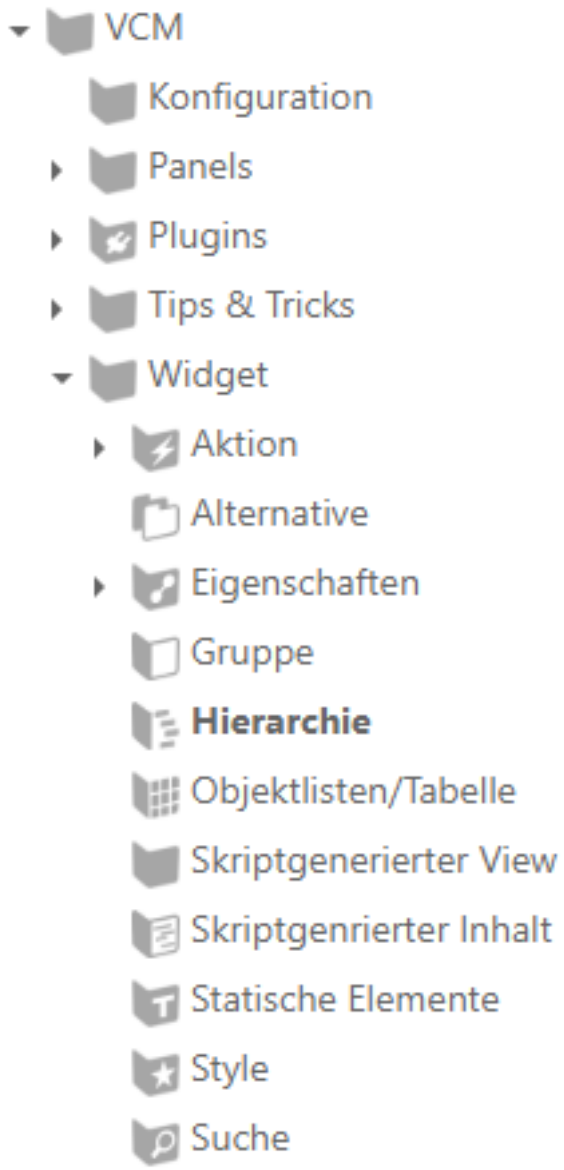
Wählt man den "renderMode" "panel" aus, erhält man eine aufklappbare Gruppe.



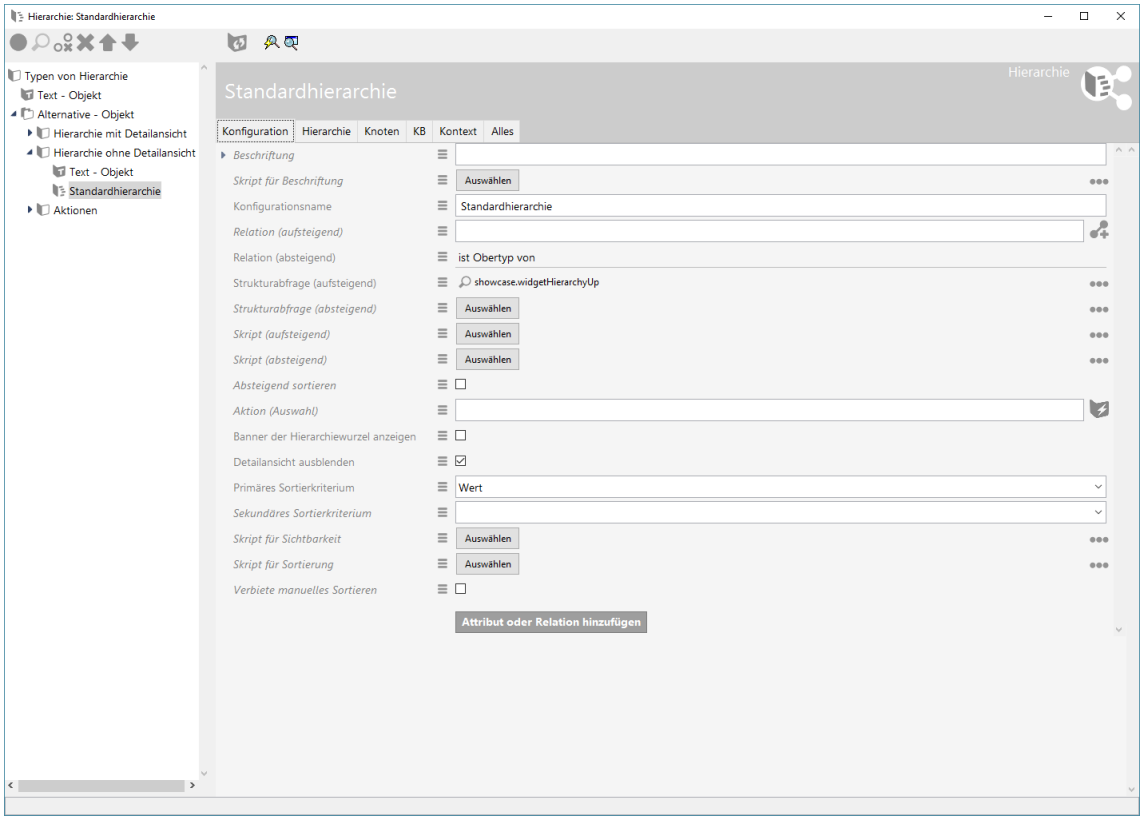
Auch die aus Bootstrap bekannten "renderMode"-Werte "jumbotron" und "well" werden bei der Gruppe unterstützt.

3.4.4 Hierarchy

Bei einer Hierarchie-View handelt es sich um eine hierarchische Ansicht zu konfigurierbaren Aspekten eines Objekts.



Die Konfiguration im Knowledge-Builder erfolgt über das Anlegen einer Hierarchie-View.



Im Reiter "Konfiguration" gibt es Möglichkeiten zur Bestimmung der allgemeinen Darstellung der Hierarchie:

Beschriftung	Der hier eingegebene Wert erscheint als Überschrift der Hierarchie
Skript für Beschriftung	Alternativ zu "Beschriftung" kann der Titel der Hierarchie in einem Skript bestimmt werden.
Konfigurationsname	Über den Konfigurationsnamen können Views und Panels identifiziert werden.

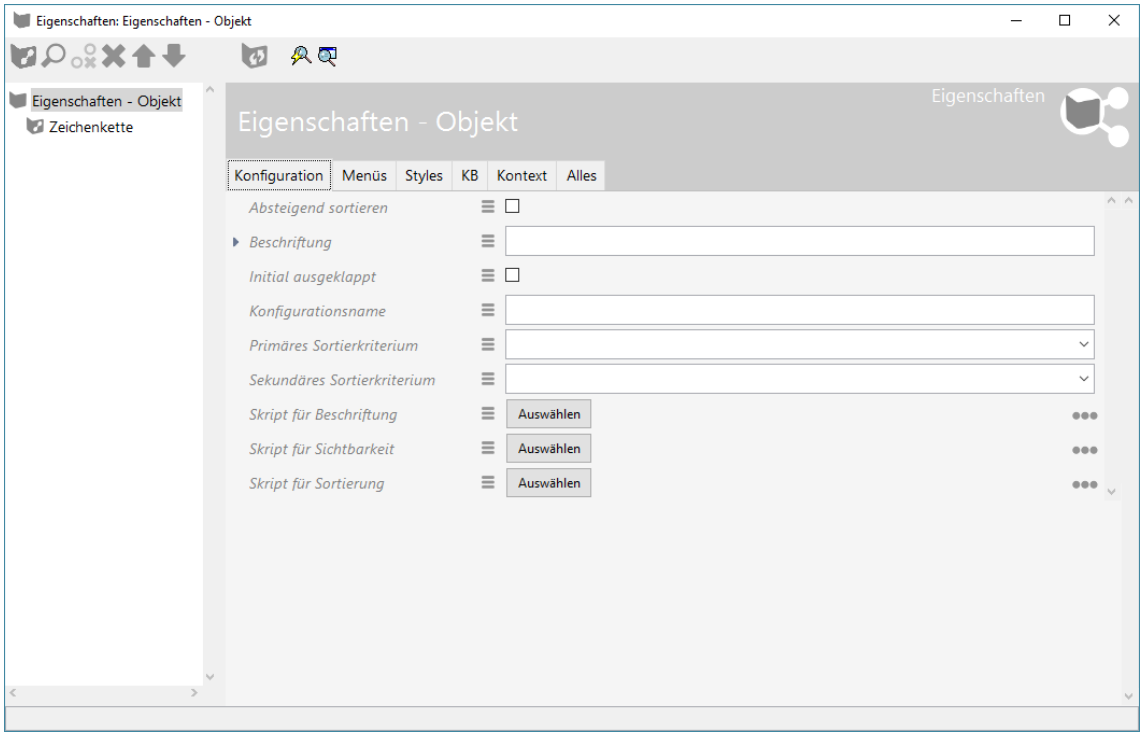
Relation (aufsteigend) Relation (absteigend) Strukturabfrage (aufsteigend) Strukturabfrage (absteigend) Skript (aufsteigend) Skript (absteigend)	Die Hierarchie-View geht von einem Objekt als Basis aus. Welche Knoten und Äste für dieses Objekt angezeigt werden sollen, ist sowohl in aufsteigender als auch in absteigender Richtung konfigurabel. Dabei kann eine im Netz definierte Relation als Verbindung zwischen den Knoten ausgewählt werden, aber auch eine Strukturabfrage oder sogar ein Skript. Diese drei Arten können vermischt werden, d.h. es ist möglich z.B. in absteigende Richtung eine Relation anzugeben und in aufsteigende Richtung eine Strukturabfrage. Die Angabe beider Richtungen ist optional, es ist auch möglich nur die aufsteigende oder nur die absteigende Richtung zu konfigurieren. Im ersten Fall wäre dann das Objekt, auf dem die Hierarchie basiert, der unterste Knoten. Und im zweiten Fall wäre das Basisobjekt der Hierarchie dann der Wurzelknoten der Hierarchie.
Absteigend sortieren	Per default wird die Hierarchie aufsteigend sortiert. Durch Aktivieren der Checkbox wird diese Sortierreihenfolge umgekehrt.
Aktion (Auswahl)	An dieser Stelle kann die Aktion konfiguriert werden, die erfolgen soll, wenn ein Nutzer ein Element in der Hierarchie selektiert.
Banner der Hierarchiewurzel anzeigen	Über diese Checkbox kann festgelegt werden, ob das Banner des Wurzelelements in der Hierarchie-View angezeigt werden soll. Diese Konfiguration wird nur im Knowledge-Builder berücksichtigt.
Detailansicht ausblenden	Standardmäßig wird zu einem selektierten Knoten (siehe auch Aktion (Auswahl) eine Detailansicht angezeigt, dies kann über diese Option deaktiviert werden.
Primäres Sortierkriterium	Über das Sortierkriterium wird bestimmt, nach welchem Aspekt die Sortierung der Hierarchieelemente auf eine Ebene erfolgen soll.
Sekundäres Sortierkriterium	Wie "Primäres Sortierkriterium", wird aber nur genutzt, falls die errechnete Position aus "Primäres Sortierkriterium" bei zwei oder mehr Attributen gleich ist.
Skript für Sichtbarkeit	Über dieses Skript kann festgelegt werden, ob die Liste angezeigt werden soll.
Skript für Sortierung	Dieses Skript wird genutzt, wenn als primäres oder sekundäres Sortierkriterium "Skript für Sortierung" ausgewählt wurde.

Verbiete manuelles Sortieren	Über diese Option wird die Möglichkeit eine Hierarchie durch den Nutzer umsortieren zu lassen unterbunden. Diese Option wird nur im Knowledge-Builder angewendet.

Es ist möglich, an der ganzen Hierarchie Aktionen und Styles zu konfigurieren oder diese nur auf Knotenebene anzubringen. Daher gibt es einen Reiter "Hierarchie" mit den Unterpunkten "Menüs" und "Styles" und einen Reiter "Knoten" mit den gleichen Unterpunkten. Im Reiter"Menüs" lassen sich Aktionen zu der Liste konfigurieren, und im Reiter "Styles" können bestimmte Darstellungsoptionen ausgewählt werden. Der "KB"-Reiter enthält Optionen, die nur für den Knowledge-Builder gelten und im Web-Frontend keine Anwendung finden. Über den "Kontext"-Reiter lässt sich konfigurieren, für welche Objekttypen die Hierarchie-View verwendet werden soll und in welchen Anwendungskontexten.

3.4.5 Eigenschaften

Bei einer Eigenschaften-View handelt es sich um eine Liste von Attributen und Relationen zu einem Objekt.



Im Reiter "Konfiguration" gibt es Möglichkeiten zur Bestimmung der allgemeinen Darstellung der Liste:

Absteigend sortieren	Grundsätzlich werden die enthaltenen Attribute/Relationen in der Reihenfolge angezeigt, wie es die Reihenfolge der eingehängten Eigenschaft-Views vorgibt. Da es bei diesen aber möglich ist, übergeordnete Typen (z.B. "Benutzerrelation") anzugeben, werden die so zusammengefassten Eigenschaften nach ihrem Namen aufsteigend sortiert. Durch Aktivieren der Checkbox "Absteigend sortieren" kann diese Reihenfolge geändert werden.
Beschriftung	Der hier eingegebene Wert erscheint als Überschrift der Liste
Initial ausgeklappt	Im Knowledge-Builder werden ab einer großen Anzahl Eigenschaften nicht direkt angezeigt, sondern ausklappbar. Durch das Aktivieren dieser Option werden sie direkt ausgeklappt.
Konfigurationsname	Über den Konfigurationsnamen können Views und Panels identifiziert werden.
Primäres Sortierkriterium	Grundsätzlich werden die enthaltenen Attribute/Relationen in der Reihenfolge angezeigt, wie es die Reihenfolge der eingehängten Eigenschaft-Views vorgibt. Über diesen Punkt kann dieses Verhalten geändert werden. Mögliche Werte sind "Position", "Skript für Sortierung" und "Wert". Bei "Wert" wird nach dem Attributwert sortiert und nicht nach dem Namen des Attributs.
Sekundäres Sortierkriterium	Wie "Primäres Sortierkriterium", wird aber nur genutzt, falls die errechnete Position aus "Primäres Sortierkriterium" bei zwei oder mehr Attributen gleich ist.
Skript für Beschriftung	Alternativ zu "Beschriftung" kann der Titel der Liste in einem Skript bestimmt werden.
Skript für Sichtbarkeit	Über dieses Skript kann festgelegt werden, ob die Liste angezeigt werden soll.
Skript für Sortierung	Dieses Skript wird genutzt, wenn als primäres oder sekundäres Sortierkriterium "Skript für Sortierung" ausgewählt wurde.

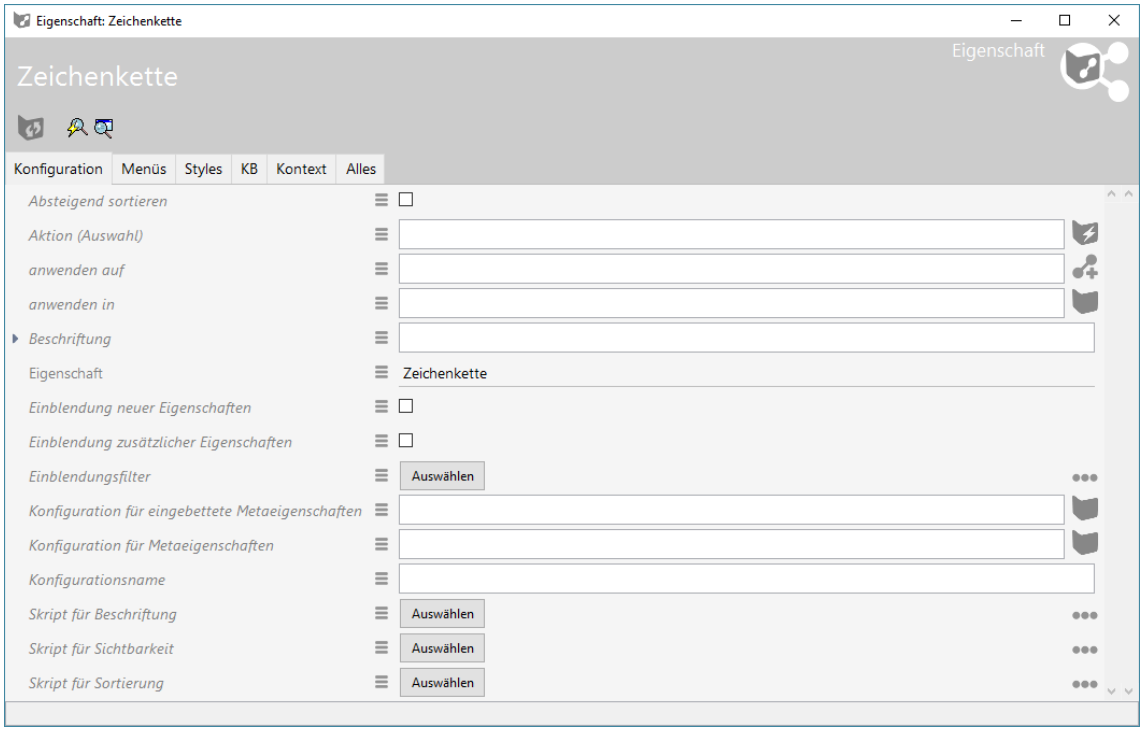
Im Reiter"Menüs" lassen sich Aktionen zu der Liste konfigurieren, und im Reiter "Styles" können bestimmte Darstellungsoptionen ausgewählt werden. Der "KB"-Reiter enthält Optionen, die nur für den Knowledge-Builder gelten und im Web-Frontend keine Anwendung finden. Über den "Kontext"-Reiter lässt sich konfigurieren, für welche Objekttypen die Eigenschaften-View verwendet werden soll und in welchen Anwendungskontexten.

Für die Lese-Ansicht kann die Eigenschaften-View alleinstehend verwendet werden, wird aber oft auch in Gruppe- oder Alternative-Views verwendet. Um die Bearbeitung der Objekteigenschaften zu ermöglichen, muss eine Eigenschaften-View in eine Edit-View eingehängt werden.

Die Attribute und Relationen, die zu einem Objekt angezeigt werden sollen, sind konfigurabel. Dazu werden einer Eigenschaften-View Eigenschaft-Views hinzugefügt, an denen das/die entsprechende Attribut/Relation ausgewählt werden und bestimmt wird, wie diese im einzelnen dargestellt werden sollen.

3.4.6 Property

Bei einer Eigenschaft-View handelt es sich um die Darstellunskonfiguration eines Attributs oder einer Relation zu einem Objekt. Eine Eigenschaft-View kann nur innerhalb einer Eigenschaften-View verwendet werden.

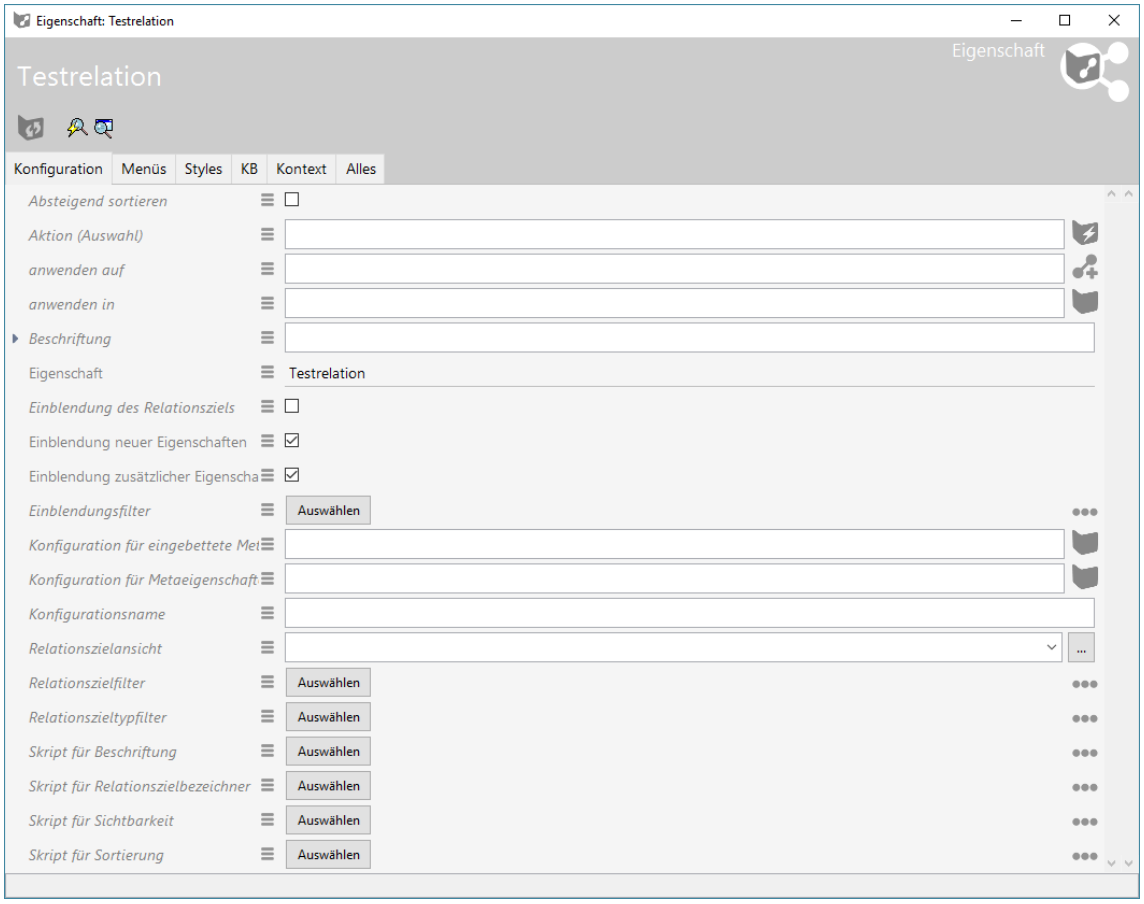


Im Reiter "Konfiguration" gibt es Möglichkeiten zur Bestimmung der allgemeinen Darstellung der Attrubute/Relationen:

Absteigend sortieren	Da zu einem Objekt mehrere Attribute/Relationen eines Typs existieren können, werden diese bei Mehrfachvorkommen nach dem Eigenschaftswert aufsteigend sortiert. Diese Option dreht die Sortierreihenfolge. Es ist auch möglich bei "Eigenschaft" einen übergeordneten Attributtyp (z.B. Benutzerrelation) auszuwählen. Die Sortierreihenfolge der untergeordneten Attributtypen wird aber über die Sortierungskonfiguration der dazugehörigen Eigenschaften-View bestimmt. Auf Basis der Eigenschaft-View kann nur die Sortierung innerhalb des gleichen Attributs Einfluss genommen werden.
Aktion (Auswahl)	Die hier konfigurierte Aktion wird ausgeführt, wenn der Eigenschaftswert im Frontend ausgewählt wird (z.B. per Klick).
anwenden auf/anwenden in	An dieser Stelle kann der Kontext genauer bestimmt werden, so dass unterschiedliche Konfigurationen einer Eigenschaft in einer Eigenschaften-View genutzt werden können.

Beschriftung	Eine Eigenschaft wird als Name-Wert-Paar in den Frontends dargestellt. Als default wird der Name der konfigurierten Eigenschaft genommen. Über "Beschriftung" kann ein anderer Wert als Name angezeigt werden.
Eigenschaft	An dieser Stelle wird der Attribut-/Relationstyp des dargestellten Objekts bestimmt. Es ist möglich, einen übergeordneten Attributtyp anzugeben. Wenn z.B. "Benutzerrelation" ausgewählt wird, werden in der dargestellten Liste alle Relationsausprägungen, die der Benutzerrelation untergeordnet sind, zu diesem Objekt angezeigt. Die Sortierreihenfolge dieser Eigenschaften kann in der Eigenschaften-View konfiguriert werden.
Einblendung neuer Eigenschaften	Falls noch kein Attribut dieses Typs am dargestellten Objekt existiert, würde in der Default-Ansicht das Attribut nicht aufgeführt. Über diese Option kann man konfigurieren, dass stattdessen ein Platzhalter mit einem Eingabefeld für den Wert des neuen Attributs angezeigt wird, falls noch kein Attributwert existiert.
Einblendung zusätzlicher Eigenschaften	Falls das konfigurierte Attribut mehrfach vorkommen darf, kann man über diese Option festlegen, dass automatisch ein neuer Platzhalter für einen weiteren Attributwert angezeigt wird.
Einblendungsfilter	Über dieses Skript können die vorhandenen Attribute dieses Typs gefiltert werden, sodass in der Darstellung nur bestimmte von ihnen berücksichtigt werden.
Konfiguration für eingebettete Metaeigenschaften/Konfiguration für Metaeigenschaften	Falls das Attribut Metaeigenschaften enthält, kann an dieser Stelle eine View angebracht werden, um die Darstellung dieser Eigenschaften zu bestimmen. Der einzige Unterschied bei den eingebetteten Metaeigenschaften ist, dass sie eingerückt unter dem Attribut erscheinen.
Konfigurationsname	Über den Konfigurationsnamen können Views und Panels identifiziert werden.
Skript für Beschriftung	Alternativ zu "Beschriftung" kann der dargestellt Attributsname in einem Skript bestimmt werden.
Skript für Sichtbarkeit	Über dieses Skript kann festgelegt werden, ob diese Eigenschaft angezeigt werden soll.
Skript für Sortierung	Dieses Skript kann genutzt werden, wenn mehrere Ausprägungen einer Eigenschaft zu einem Objekt existieren.

Für Relationen gibt es noch weitere Optionen:



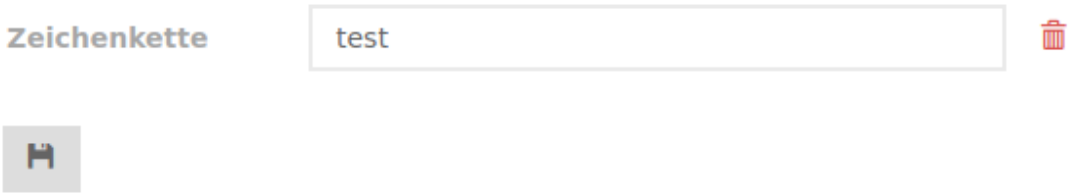
Einblendung des Relationsziels	Per Default wird als Attributwert der Name des Relationszielsobjekts als Link angezeigt. Über diese Option werden konfigurierte Eigenschaften des Relationsziels angezeigt. Die dafür verwendete View kann über ihren Kontext bestimmt werden, d.h. bei "anwenden in" der entsprechenden View sollte die Eigenschaft-View angegeben werden.
Relationszielansicht	Defaultmäßig wird ein Link bzw.ein Relationszieleditor im Bearbeitungsmodus angezeigt. Es kann aber sinnvoll sein, anstelle dessen z.B. eine Drop-Down-Liste mit schon vorgefilterten Relationszielen anzuzeigen. Diese alternativen Ansichten sind an dieser Stelle konfigurierbar.
Relationszielfilter	Um den Benutzer in seiner Auswahl eines geeigneten Relationsziels zu unterstützen, kann hier eine Filterungsabfrage angebracht werden.
Relationszieltypfilter	Falls mehrere Objekttypen als Ziel einer Relation definiert wurden, kann an dieser Stelle ein Filter auf die angezeigten Typen konfiguriert werden.
Skript für Relationszielbezeichner	Per Default wird der Name des Relationszielobjekts angezeigt. Dies kann per Skript an dieser Stelle angepasst werden.

Im Reiter "Menüs" lassen sich weitere Aktionen zu der Eigenschaft konfigurieren, und im Reiter "Styles" können bestimmte Darstellungsoptionen ausgewählt werden. Der "KB"-Reiter enthält Optionen, die nur für den Knowledge-Builder gelten und im Web-Frontend keine Anwendung finden. Über den "Kontext"-Reiter lässt sich nachvollziehen, in welchen Views die Eigenschaft-View angewendet wird.

3.4.7 Edit

Eine Edit-View dient zur Pflege von Attributen oder Relationen.

Dabei werden alle Kind-Konfigurationen vom Typ Eigenschaften als Formularfelder angezeigt. Eine Edit-View darf genau eine Kind-View enthalten. Dies sollte eine Eigenschaften Konfiguration sein. Über eine Gruppen-View ist es aber auch möglich, mehrere Eigenschaften-Views zu verwenden. Änderungen können über einen Speicherbutton mit dem Wissensnetz synchronisiert werden.



Im Reiter "Konfiguration" gibt es Möglichkeiten zur Bestimmung der allgemeinen Darstellung der Edit-View:

Editiermodus umschaltbar	Über dieses Option ist der Formularmodus "Umschaltbar". D.h. Eigenschaften werden zuerst im nur Lesemodus angezeigt. Über einen Umschaltbutton kann dann in den Pflegemodus geschaltet werden.
Nur benutzerdefinierte Schaltflächen	Über diese Option können eigene Buttons zum Umschalten und Speichern konfiguriert werden. Hierfür muss ein Menü mit Aktionen konfiguriert werden.

Benutzerdefinierte Schaltflächen

Ist die Option *Nur benutzerdefinierte Schaltflächen* aktiv, können eigene Buttons zum Umschalten und Speichern konfiguriert werden. Dabei muss an der jeweiligen Aktion eine eigene ActionResponse mit definierten Followups konfiguriert werden:

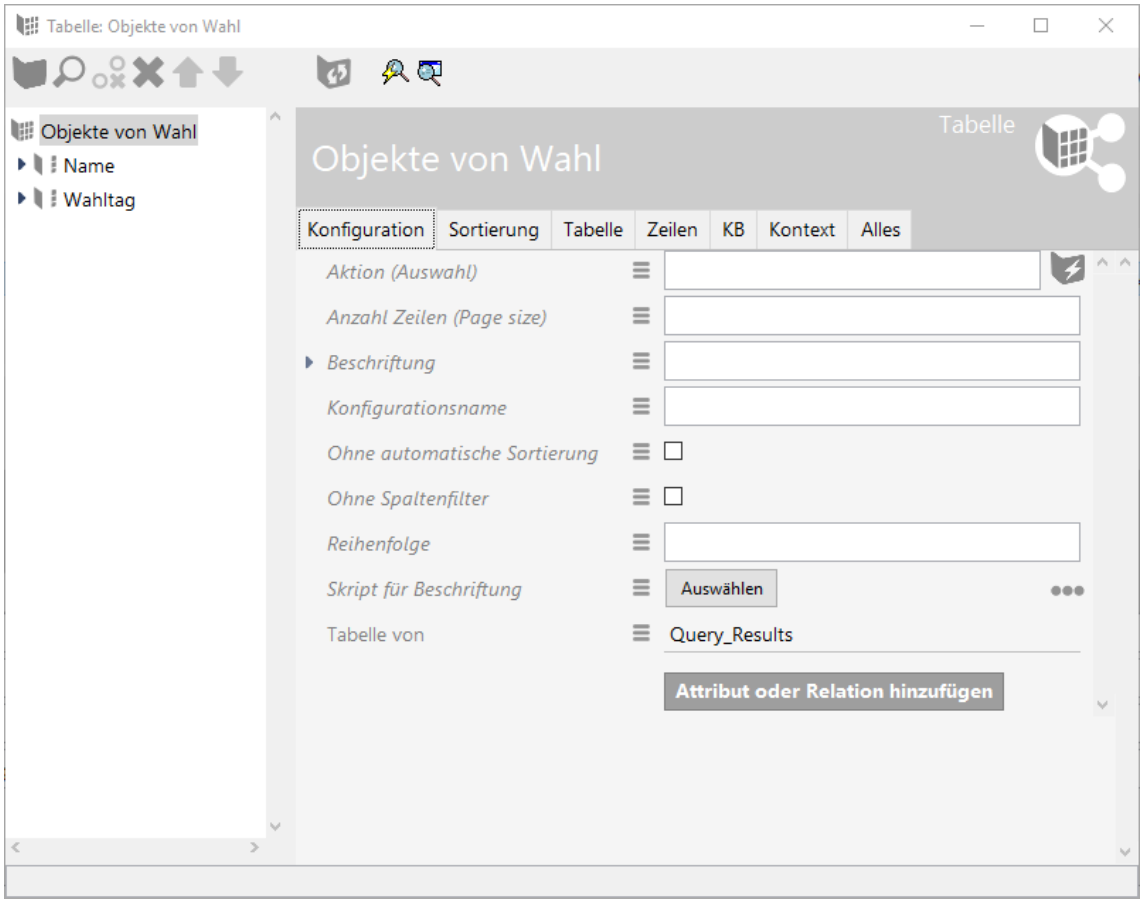
Aktion	**Followup**
Speichen	edit-save
Umschalten Lesemodus	edit-state-read
Umschalten Pflegemodus	edit-state-edit

Beispiel für einen eigenen Speichern Button:

```
function actionResponse(element, context, actionResult) {
    var actionResponse = new $k.ActionResponse();
    actionResponse.setFollowup("edit-save");
    return actionResponse;
}
```

3.4.8 Table

Bei einer Tabellen-View handelt es sich um die Darstellungskonfiguration einer Liste von Objekten. Eine Tabellen-View kann unabhängig an verschiedenen Stellen verwendet werden und ihr Inhalt ist vom Kontext abhängig.



Im Reiter "Konfiguration" gibt es Möglichkeiten zur Bestimmung der allgemeinen Darstellung und des Verhaltens.

Aktion (Auswahl)	Die hier konfigurierte Aktion wird ausgeführt, wenn eine Zeile im Frontend ausgewählt wird (z.B. per Klick).
Anzahl Zeilen (Page size)	Hier wird die maximale Anzahl Zeilen angegeben, die auf einer Seite angezeigt werden.
Automatische Suche	Optionen: • Automatische Suche • Automatische Suche bis Grenzwert • Keine automatische Suche
Beschriftung	Eine Tabelle wird in den Frontends mit Überschrift dargestellt. Als default wird der Name aus dem Kontext erzeugt. Über "Beschriftung" kann ein anderer Wert als Name angezeigt werden.

Konfigurationsname	Über den Konfigurationsnamen können Views und Panels identifiziert werden.
Ohne Spaltenfilter	Hier kann bestimmt werden, ob zwischen Tabellen-Kopfleiste und Tabelleninhalt ein Spaltenfilter dargestellt werden soll. Anhand des Spaltenfilters kann für die jeweilige Spalte das Suchergebnis durch Eingabe eines Begriffs gefiltert werden.
Skript für Beschriftung	Alternativ zu "Beschriftung" kann der dargestellt Attributsname in einem Skript bestimmt werden.
Tabelle von	Hier wird die Ansicht referenziert, deren Ergebnisse in der vorliegenden Tabelle angezeigt werden. Dies kann eine Suche, eine Suchergebnis-Ansicht oder wieder eine Tabelle sein.

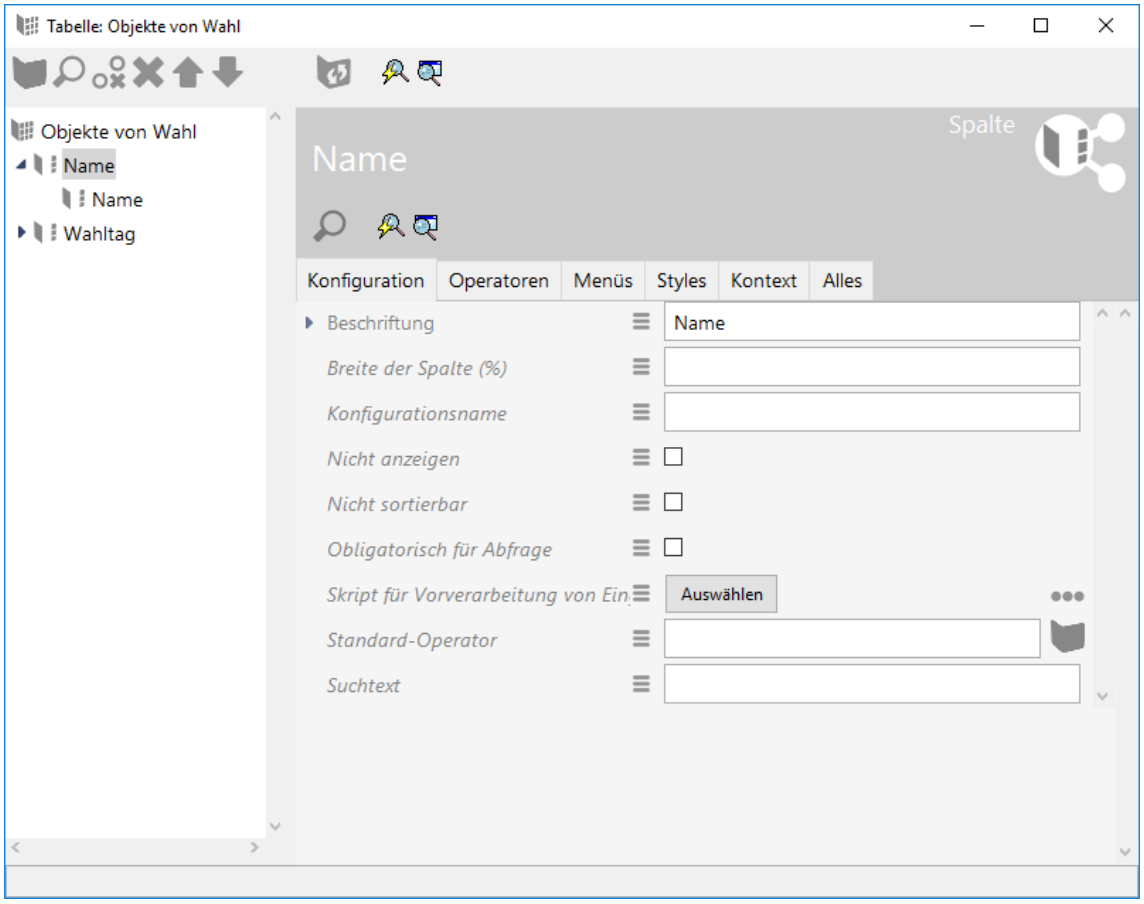
Im Reiter "Sortierung" kann das Sortierverhalten anhand der Spalten konfiguriert werden.

Der Reiter "Tabelle" enthält zwei Unterpunkte: "Menüs" und "Styles". Im Reiter "Menüs" lassen sich weitere Aktionen zu der Tabelle konfigurieren, und im Reiter "Styles" können bestimmte Darstellungsoptionen, die sich auf die gesamte Tabelle auswirken ausgewählt werden. Im nächsten Reiter, "Spalten" $>$ "Styles" werden dann entsprechend die Darstellungsoptionen für Spalten ausgewählt.

Die Spalten der Tabelle werden durch Subkonfigurationen definiert, siehe nächster Abschnitt. Die Reihenfolge der Spalten kann in der Baumansicht auf der linken Seite mittels der Pfeil-Buttons darüber verändert werden.

Die Spalten-View repräsentiert die Konfiguration einer gesamten Spalte. Hier lässt sich Einfluss auf die Darstellung und das Verhalten (z.B. Filterung) nehmen.

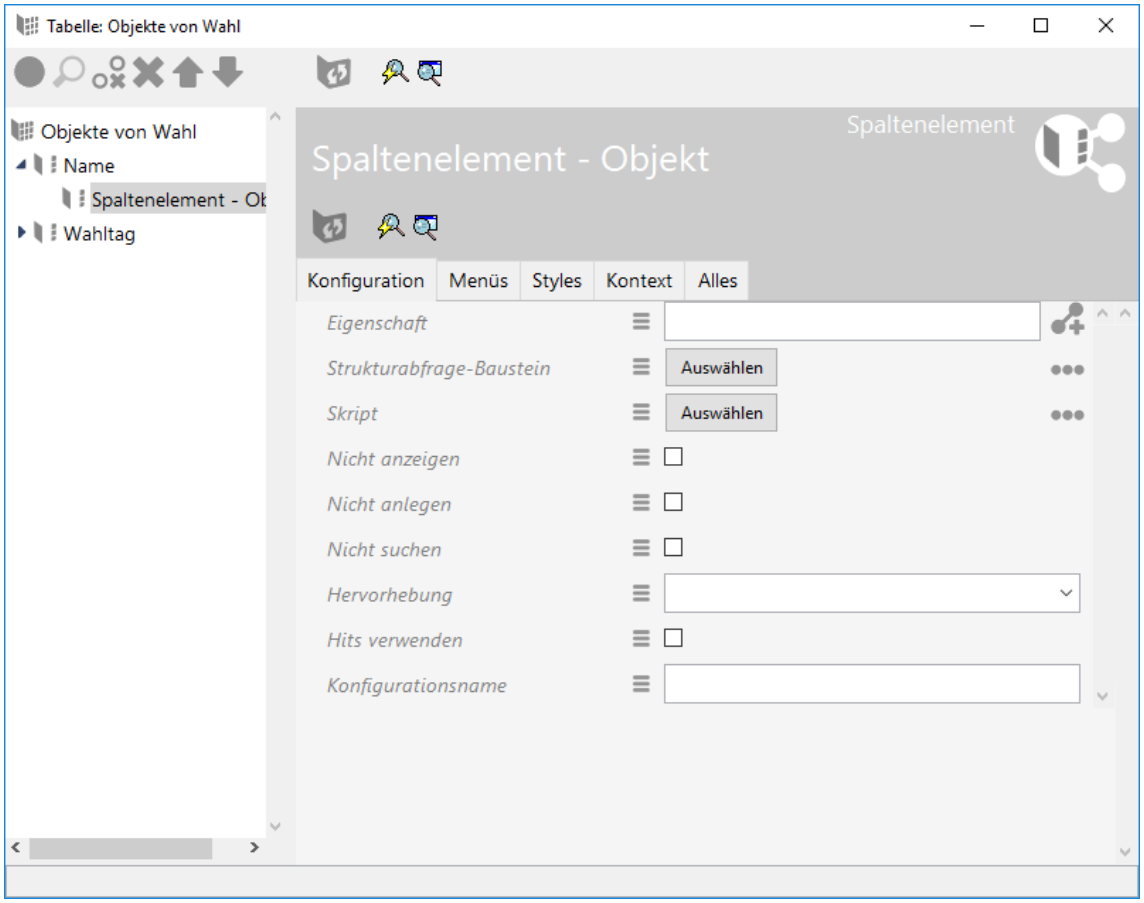
Der Inhalt der Zellen ("Spalten-Element") wird wiederum durch eine Subkonfiguration festgelegt, wie im folgenden Abschnitt beschrieben.



Beschriftung	Angezeigter Name der Spalte
Breite der Spalte (%)	Breite der Spalte in Prozent, bezogen auf die Breite der Tabelle
Konfigura-tionsname	Über den Konfigurationsnamen können Views und Panels identifiziert werden.
Nicht anzeigen	Hiermit wird eine Spalte ausgeblendet. Sie wird trotzdem im Hintergrund berechnet und kann z.B. für die Sortierung verwendet werden.
Nicht sortierbar	In der Standardeinstellung lassen sich Spalten mit Klick auf den Header sortieren. Hier lässt sich diese Funktionalität deaktivieren.
Skript für Beschriftung	Alternativ zu "Beschriftung" kann der dargestellt Attributsname in einem Skript bestimmt werden.
Skript für Vorverarbeitung von Eingabefeldern	Der Text, der im Spaltenfilter angegeben wurde, lässt sich hier mittels Skript beeinflussen.
Standard-Operator	Hier kann aus den möglichen Filter-Operatoren der Default ausgewählt. Wenn hier nichts konfiguriert ist, wird der erste aus der Liste gewählt.

Suchtext	Hier lässt sich der Text für den Spaltenfilter im Voraus festlegen.

Die Spaltenelement Sub-Konfiguration bestimmt den Inhalt der Spalte. Der Inhalt wird typischerweise von den Elementen abgeleitet, auf die sich die Tabelle bezieht.

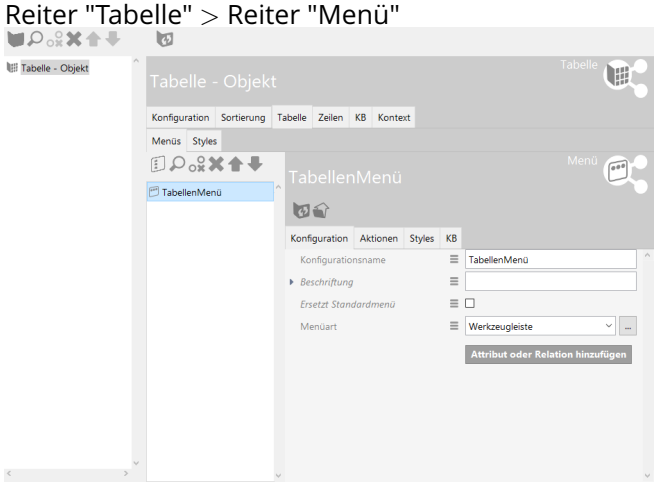
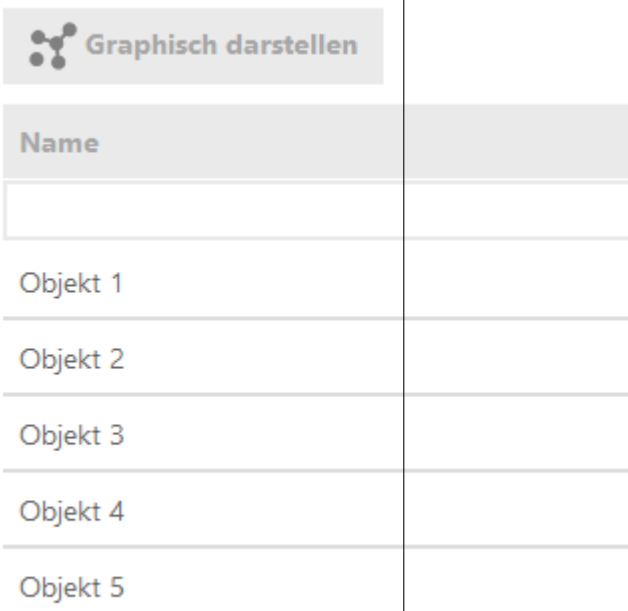


Eigenschaft	Die in dieser Spalte anzuzeigende Eigenschaft des Elements
Strukturabfrage-Baustein	Alternativ zu "Eigenschaft" kann der anzuzeigende Inhalt auch über eine Strukturabfrage bestimmt werden
Skript	Alternativ zu den ersten beiden Möglichkeiten, kann der anzuzeigende Inhalt auch über ein Skript vom Element abgeleitet werden
Nicht anzeigen	Hiermit wird das Spaltenelement ausgeblendet. Es wird trotzdem im Hintergrund berechnet und kann z.B. für die Sortierung oder Filterung verwendet werden.
Hervorhebung	Hier kann ausgewählt werden, ob der Inhalt des Spaltenelements durch Unterstreichung hervorgehoben werden soll.

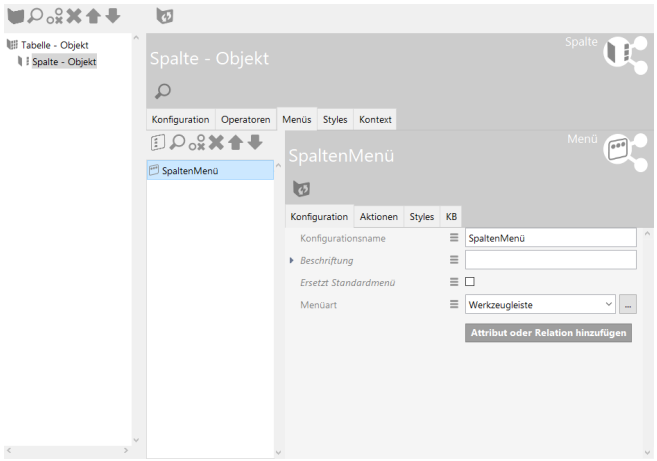
Hits verwenden	Erlaubt die Verwendung aller Metaeigenschaften eines Suchergebnisses ("Hit") wie bspw. Qualität, Ursache etc. Im Fall einer Weiterverarbeitung der Suchergebnisse per Skript wird entweder das JavaScript-Objekt $k.SemanticElement oder $k.Hit weitergereicht.
Konfigurationsname	Über den Konfigurationsnamen können Views und Panels identifiziert werden.

3.4.8.1 Menüs in Tabellen

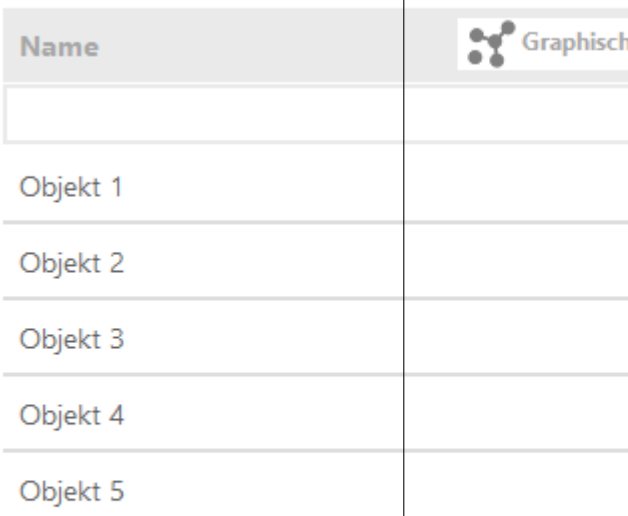
Menüs können an unterschiedlichen Stellen einer Tabelle konfiguriert werden. Die Wahl des Konfigurationsortes bestimmt, ob ein Menü für die gesamte Tabelle, für die Spalte der Tabelle oder für jedes Spaltenelement verfügbar ist:

Konfigurationsort	**Menü mit Aktionen für Element**
Tabelle: Reiter "Tabelle" > Reiter "Menü" 	Aktionen für die Tabelle gesamt: 

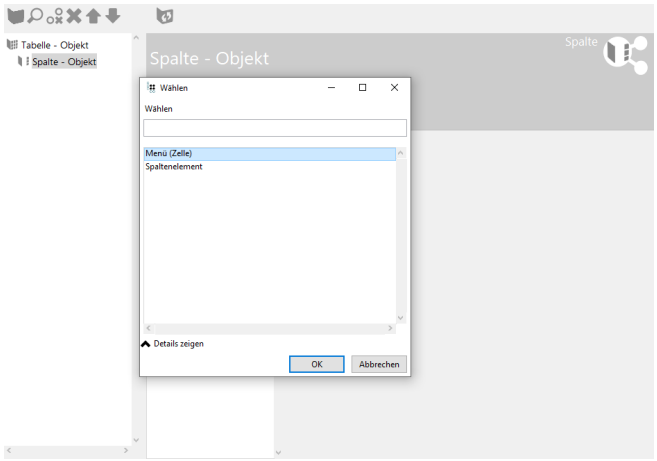
Spalte:
Reiter "Menüs"



Aktionen werden in der *Spaltenbeschreibung* einer Tabelle angezeigt:

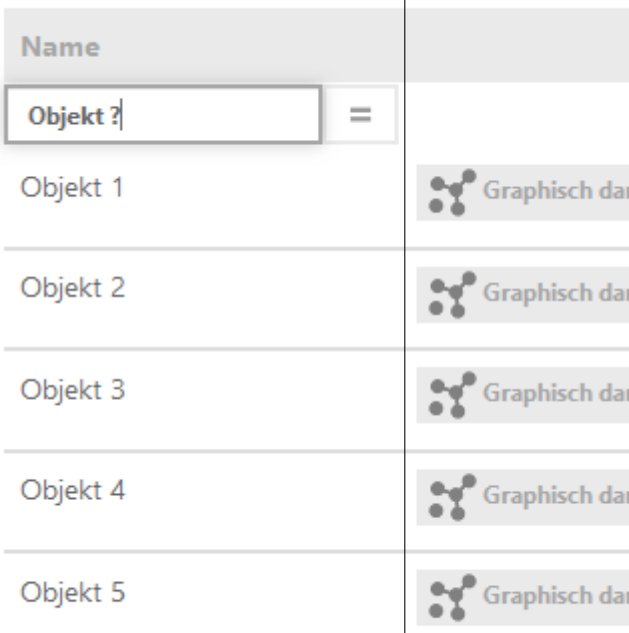


Spalte:
Menü als Unterelement der Spalte

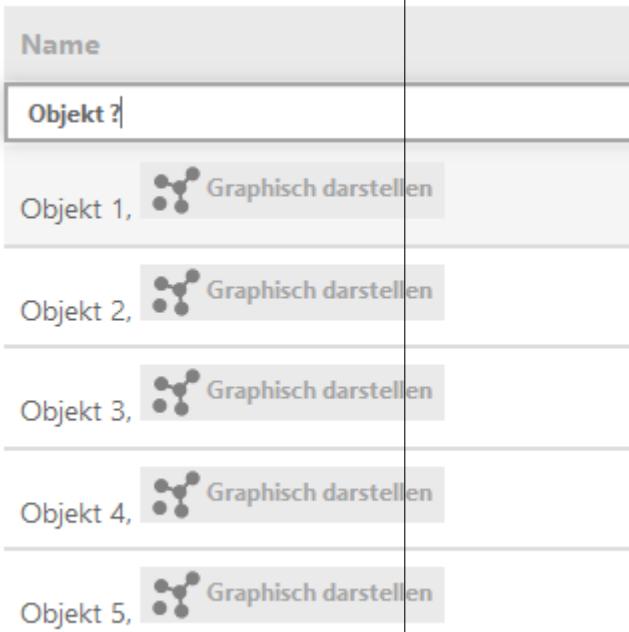


Aktionen werden in einer Spalte *in jeder Zeile* ausgegeben:

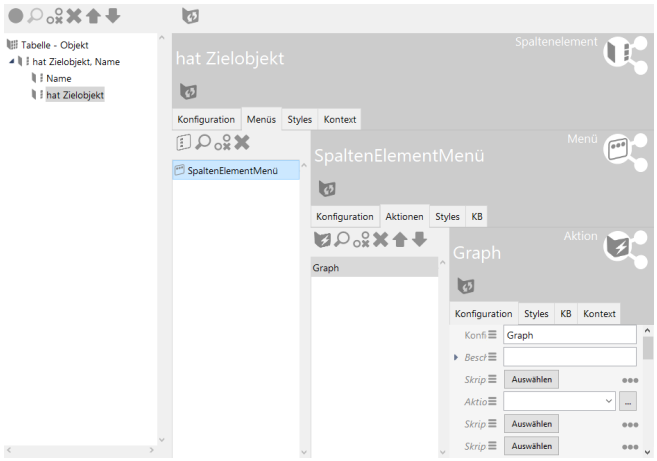
Menü in separater Spalte:



Menüelement in gleicher Spalte wie das anzuzeigende Spaltenelement:

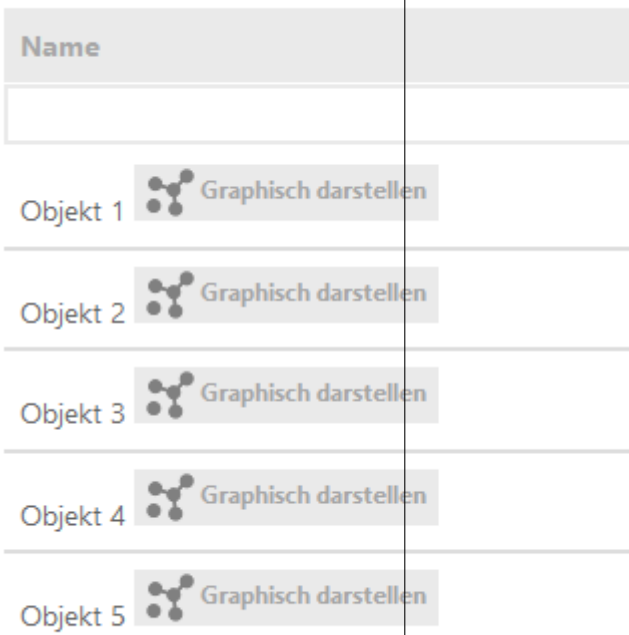


Spaltenelement:
Reiter "Menüs"



Hinter jedem Wert wird die Aktion ausgegeben:

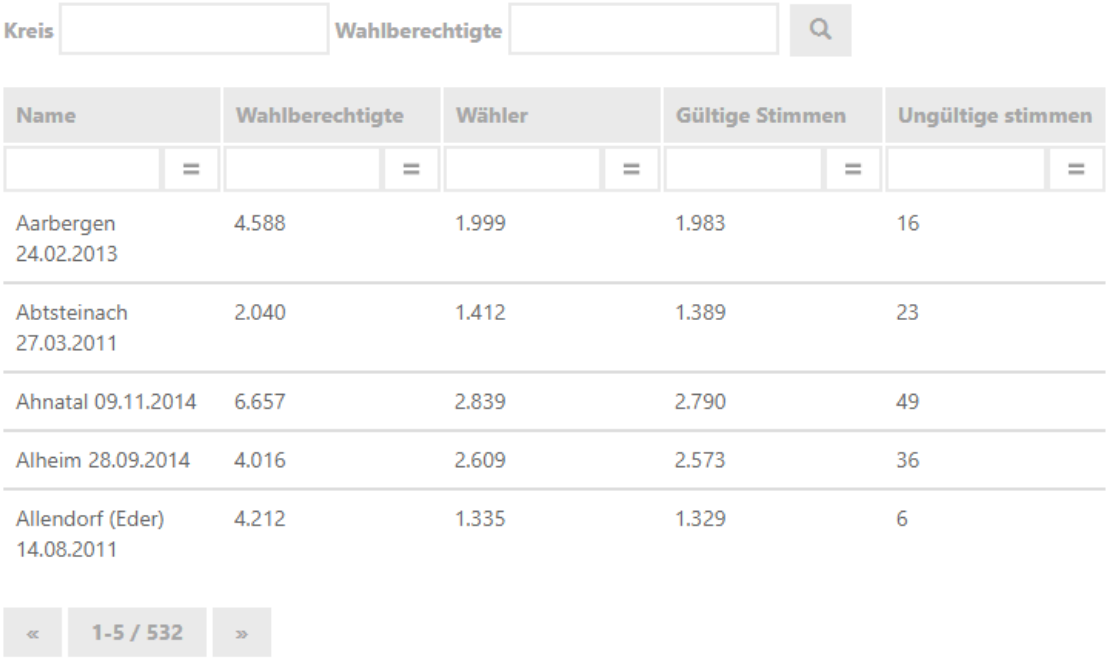
Ausgabe bei einem Objekt pro Spaltenlement:



Ausgabe bei mehreren Objekten pro Spaltenelement, bspw. bei der Darstellung von Zielobjekten einer Relation. Die Zielobjekte werden durch Kommata getrennt dargestellt (Konfiguration wie links dargestellt). In diesem Fall ist aus Platzspargründen die Verwendung von Icons vorzuziehen; die Beschriftung kann alternativ durch einen Tooltip ersetzt werden (Anzeige durch Mouse-Over):

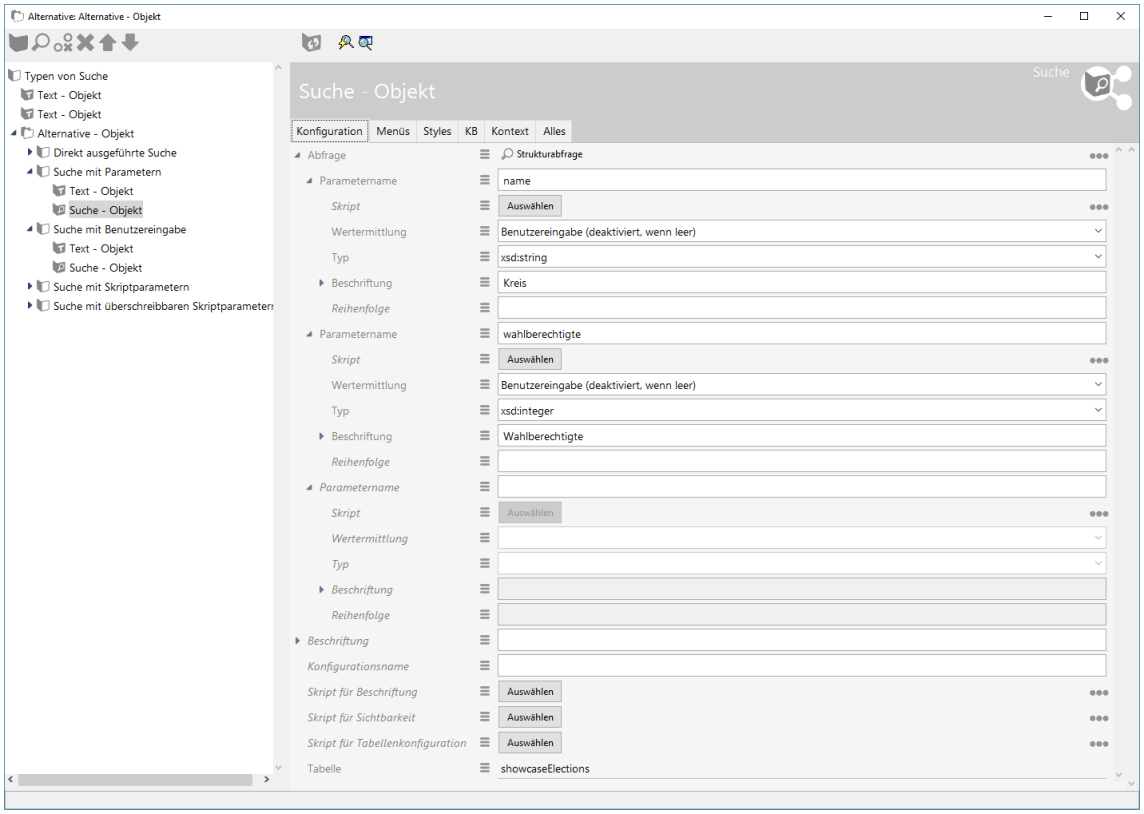
3.4.9 Search

Mit einer Suche-View lassen sich Suchseiten erstellen, auf denen die Suchabfrage und die -ergebnisse gleichzeitig angezeigt werden. Sollte die Suche keine oder nur optionale Parameter haben, wird die Suche sofort ausgeführt und die Ergebnisse direkt angezeigt. Gibt es obligatorische Parameter, wird die Suche erst nach einer Benutzereingabe ausgeführt.



Soll z.B. nur ein Suchschlitz im Titel einer Seite angezeigt werden und die Ergebnisse dieser Suche dann weiter unten auf der Seite, ist dies über die Konfiguration von Suchfeldansicht und Suchergebnisansicht möglich. Außerdem kann man auch Suchfacetten anlegen. Diese drei Konfigurationen werden in den Unterabschnitten dieses Kapitels näher beschrieben.

Für eine einfache Suchseite legt man im Knowledge-Builder eine Suche-View an.



Im Reiter "Konfiguration" gibt es Möglichkeiten zur Bestimmung der allgemeinen Darstellung der Suche:

Abfrage	Hier wird die Suche konfiguriert, die beim Ausführen der Suche ausgeführt werden soll.
Parameter-name	Der Name eines Suchparameters. Alle Parameter, die in der Suche konfiguriert wurden, müssen an dieser Stelle auch konfiguriert werden, damit es nicht zu Fehlern in der Suche kommt.
Skript	Falls der Parameterwert über ein Skript ermittelt werden soll, muss dieses hier konfiguriert werden.

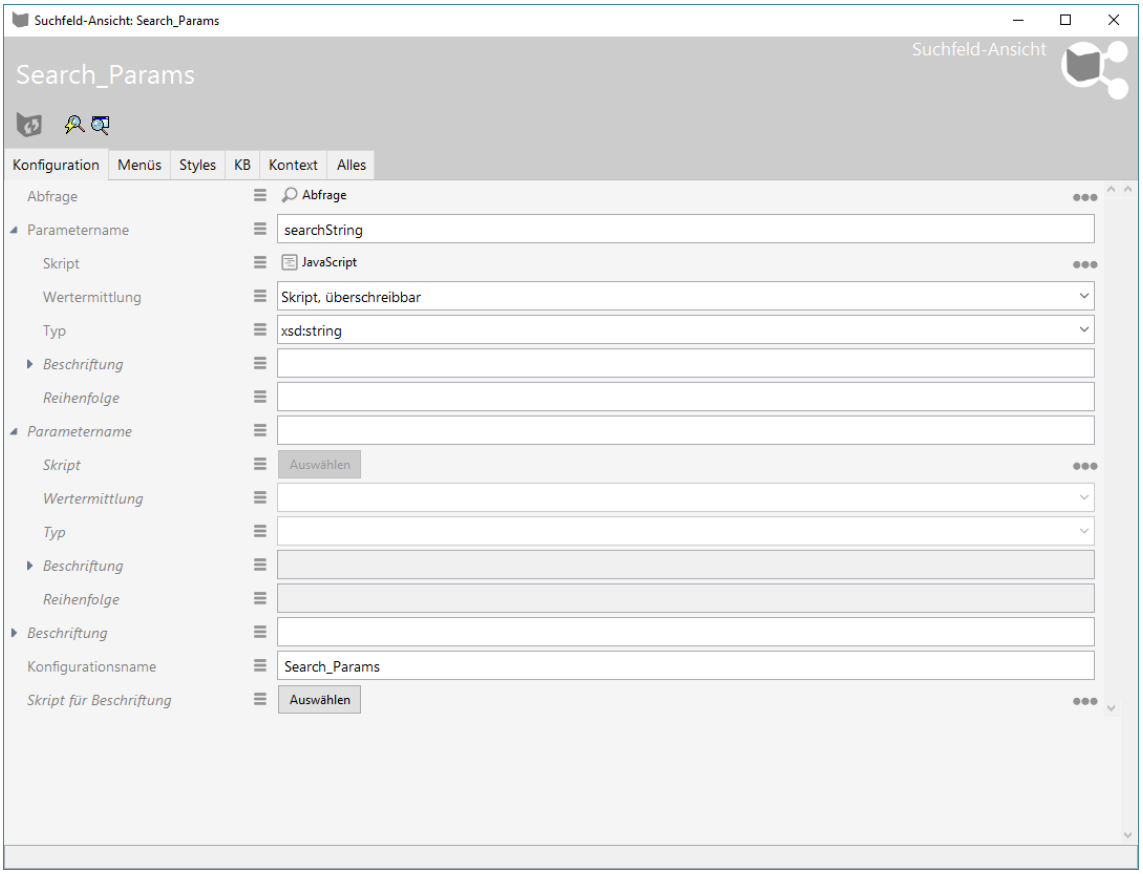
Wertermittlung	Hier wird angegeben, wie der Parameterwert bestimmt werden soll. • "Skript" (Wertermittlung per Skript) • "Skript, überschreibbar" (Wertermittlung per Skript, wird aber über Benutzereingaben im Frontend überschrieben • "Benutzereingabe (optional)" (Der Parameterwert wird aus der Benutzereingabe übernommen, falls er gesetzt wird. Er wird dem Nutzer als optional im Frontend angezeigt. Bitte beachten, dass die Suche dann auch so konfiguriert ist, dass dieser Parameter nicht gesetzt sein muss) • "Benutzereingabe (obligatorisch)" (Der Nutzer muss im Frontend einen Wert eingeben, ansonsten wird die Suche nicht ausgeführt) • "Benutzereingabe (deaktiviert, wenn leer)" (Der Parameter wird für die Suche gesetzt, wenn es eine Benutzereingabe gab. Ansonsten wird der Parameter beim Ausführen der Suche deaktiviert)
Typ	Datentyp des Parameters
Beschriftung	Die Bezeichnung des Parameters im Frontend
Reihenfolge	Die Reihenfolge der Parameter, wie sie im Frontend angezeigt werden
Beschriftung	Der hier eingegebene Wert erscheint als Überschrift der Suche
Konfigurationsname	Über den Konfigurationsnamen können Views und Panels identifiziert werden.
Skript für Beschriftung	Alternativ zu "Beschriftung" kann der Titel der Gruppe in einem Skript bestimmt werden.
Skript für Sichtbarkeit	Über dieses Skript kann festgelegt werden, ob die Gruppe angezeigt werden soll.
Skript für Tabellenkonfiguration	Alternativ zu "Tabelle" kann an dieser Stelle über ein Skript die dargestellte Tabelle bestimmt werden.

Tabelle	Alle Suchergebnisse werden im Frontend in der Tabellenkonfiguration angezeigt, die hier konfiguriert wird.

Im Reiter"Menüs" lassen sich Aktionen zu der Suche konfigurieren, und im Reiter "Styles" können bestimmte Darstellungsoptionen ausgewählt werden. Der "KB"-Reiter enthält Optionen, die nur für den Knowledge-Builder gelten und im Web-Frontend keine Anwendung finden. Über den "Kontext"-Reiter lässt sich konfigurieren, für welche Objekttypen die Suche-View verwendet werden soll und in welchen Anwendungskontexten.

3.4.9.1 Search field view

Eine Suchfeld-Ansicht wird verwendet, wenn z.B. nur ein Suchschlitz an einer Stelle angezeigt werden soll, nicht aber die Suchergebnisse. Die Konfiguration erfolgt wie bei der Suche-View, jedoch ohne die Konfiguration zur Ergebnisdarstellung.



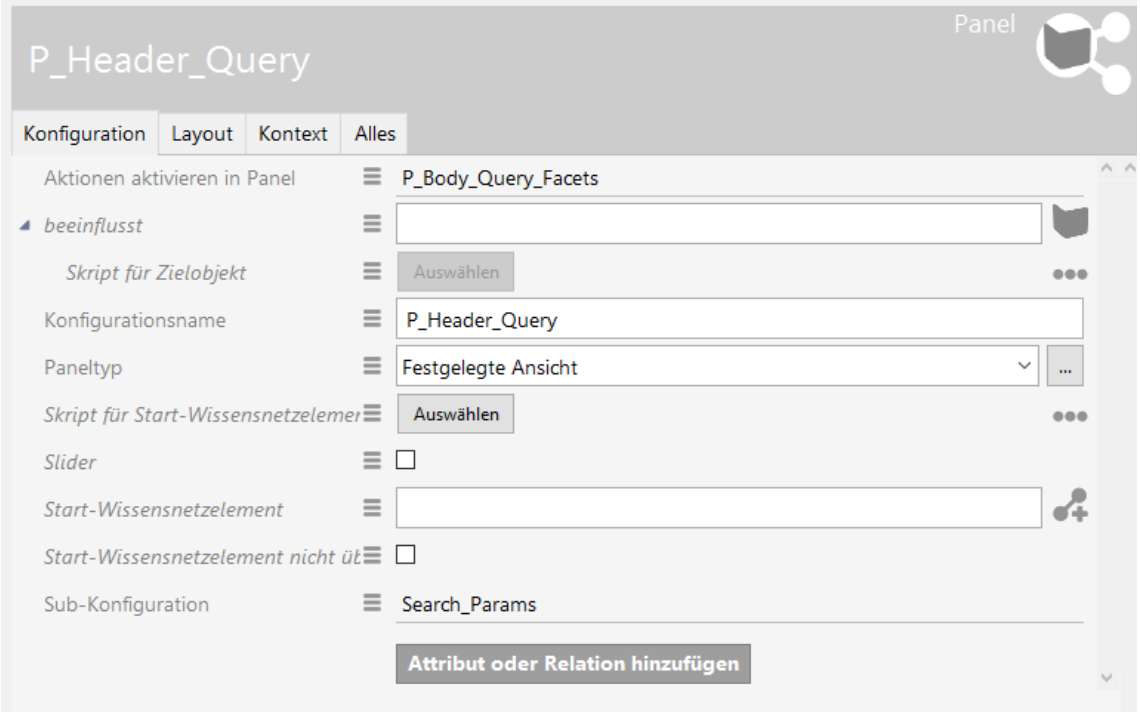
Im Reiter "Konfiguration" gibt es Möglichkeiten zur Bestimmung der allgemeinen Darstellung des Suchfeldes:

Abfrage	Hier wird die Suche konfiguriert, die beim Ausführen der Suche ausgeführt werden soll.

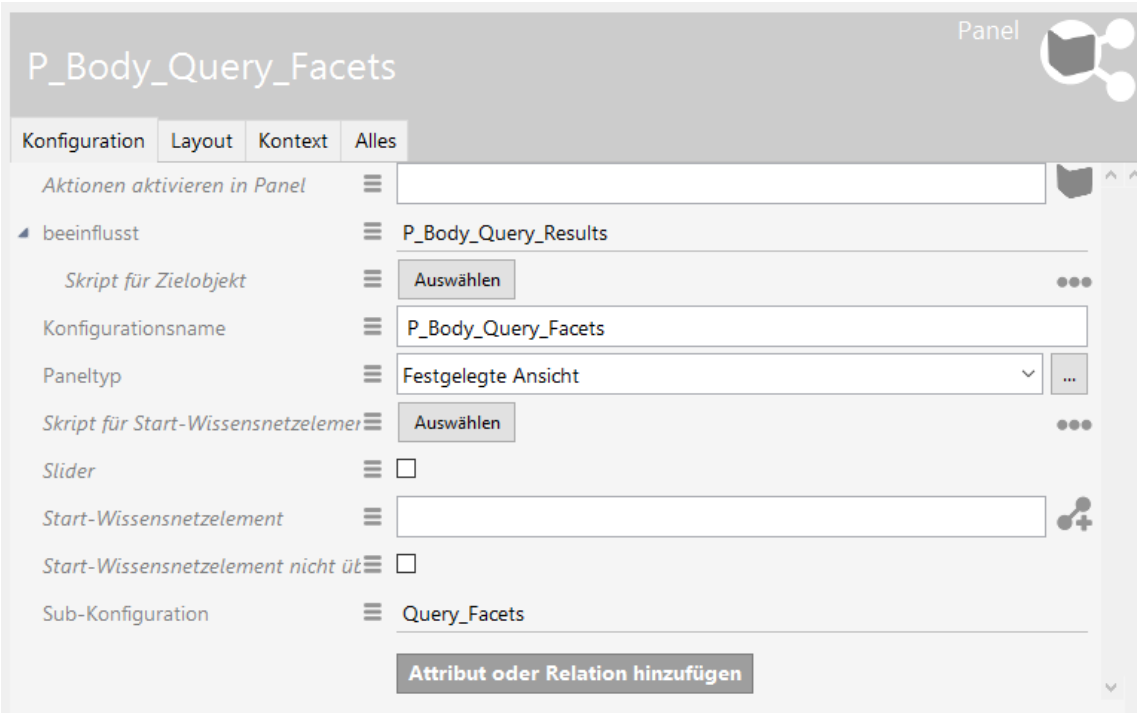
Parameter-name	Der Name eines Suchparameters. Alle Parameter, die in der Suche konfiguriert wurden, müssen an dieser Stelle auch konfiguriert werden, damit es nicht zu Fehlern in der Suche kommt.
Skript	Falls der Parameterwert über ein Skript ermittelt werden soll, muss dieses hier konfiguriert werden.
Wertermittlung	Hier wird angegeben, wie der Parameterwert bestimmt werden soll. • "Skript" (Wertermittlung per Skript) • "Skript, überschreibbar" (Wertermittlung per Skript, wird aber über Benutzereingaben im Frontend überschrieben • "Benutzereingabe (optional)" (Der Parameterwert wird aus der Benutzereingabe übernommen, falls er gesetzt wird. Er wird dem Nutzer als optional im Frontend angezeigt. Bitte beachten, dass die Suche dann auch so konfiguriert ist, dass dieser Parameter nicht gesetzt sein muss) • "Benutzereingabe (obligatorisch)" (Der Nutzer muss im Frontend einen Wert eingeben, ansonsten wird die Suche nicht ausgeführt) • "Benutzereingabe (deaktiviert, wenn leer)" (Der Parameter wird für die Suche gesetzt, wenn es eine Benutzereingabe gab. Ansonsten wird der Parameter beim Ausführen der Suche deaktiviert)
Typ	Datentyp des Parameters
Beschriftung	Die Bezeichnung des Parameters im Frontend
Reihenfolge	Die Reihenfolge der Parameter, wie sie im Frontend angezeigt werden
Beschriftung	Der hier eingegebene Wert erscheint als Überschrift der Suche
Konfigurationsname	Über den Konfigurationsnamen können Views und Panels identifiziert werden.
Skript für Beschriftung	Alternativ zu "Beschriftung" kann der Titel der Gruppe in einem Skript bestimmt werden.

Im Reiter"Menüs" lassen sich Aktionen zur Suchfeld-Ansicht konfigurieren, und im Reiter "Styles" können bestimmte Darstellungsoptionen ausgewählt werden. Der "KB"-Reiter enthält Optionen, die nur für den Knowledge-Builder gelten und im Web-Frontend keine Anwendung finden. Über den "Kontext"-Reiter lässt sich konfigurieren, für welche Objekttypen die Suchfeldansicht verwendet werden soll und in welchen Anwendungskontexten.

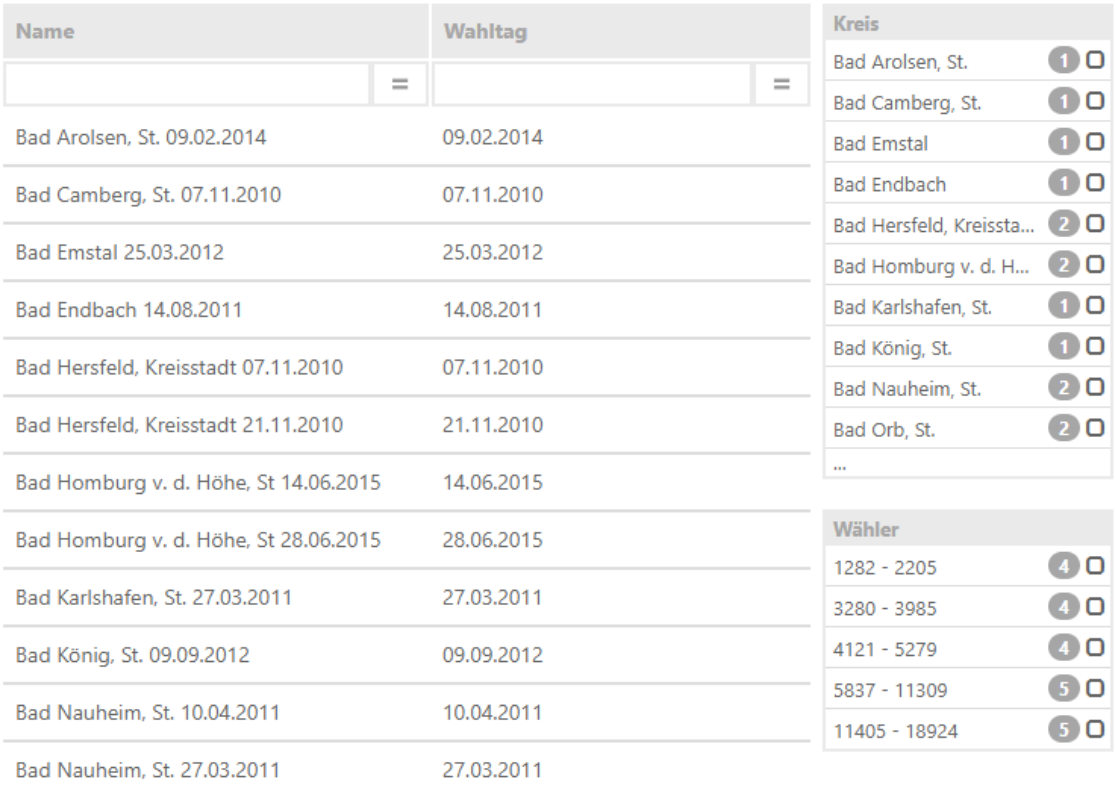
Suchfeld-Ansichten lassen sich mit Suchergebnis-Ansichten und Facetten-Ansichten kombinieren. Damit nach einer Suche aus einer Suchfeld-Ansicht die Ergebnisse in einer Suchergebnis- oder Facetten-Ansicht sichtbar werden, müssen die Aktionen entsprechend konfiguriert werden. Die einfachste Option ist, auf dem Panel, das die Suchfeld-Ansicht enthält, zu konfigurieren, dass die Aktionen in einem Panel ausgeführt werden sollen, das eine Facetten-Ansicht bzw. eine Suchergebnis-Ansicht enthält.



Möchte man alle drei Ansichten miteinander verbinden, aktiviert man die Aktionen einer Suchfeld-Ansicht wie oben beschrieben in einem Panel, das eine Suchergebnis- oder Facetten-Ansicht enthält und auf diesem Panel konfiguriert man, dass das andere Ergebnis-Ansichtspanel von diesem Panel beeinflusst wird.



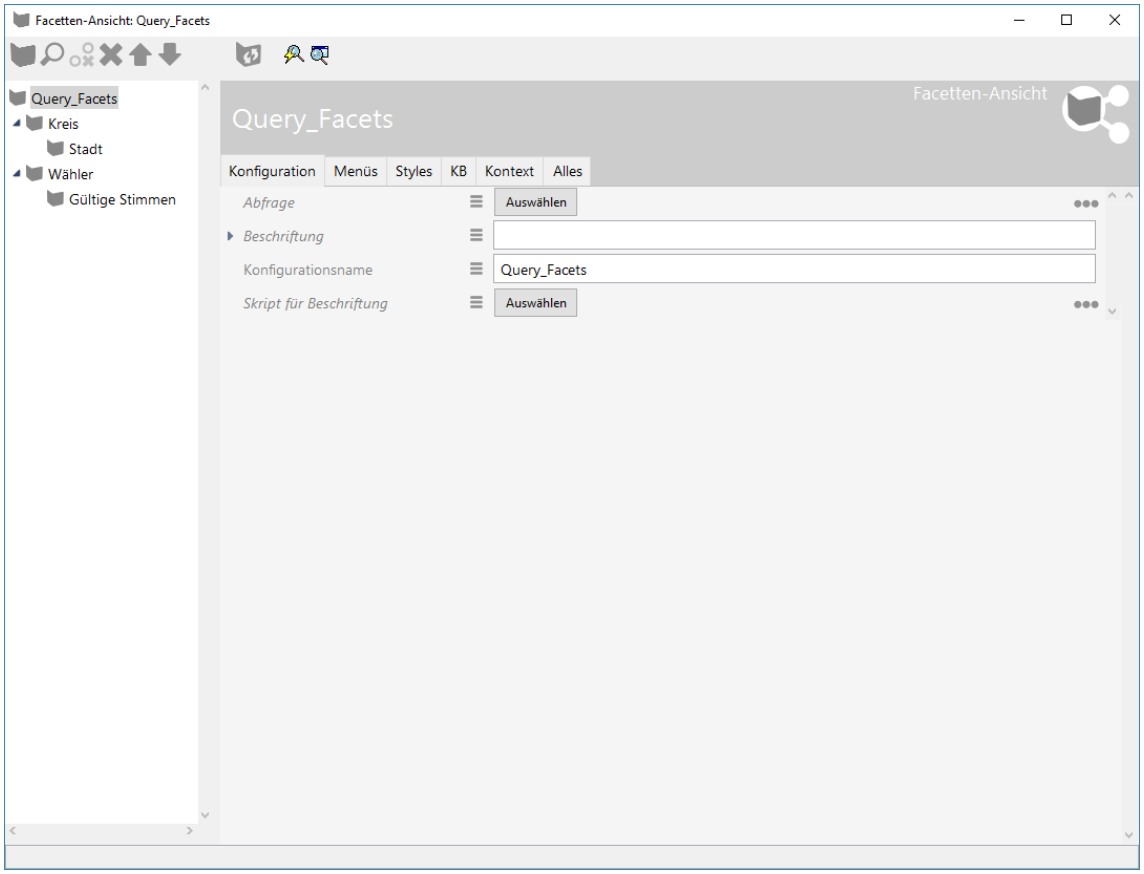
3.4.9.2 Facet view



Facetten sind für das Web-Frontend konfigurierbar, nicht für den Knowledge-Builder. Wenn eine Suche mit Facetten konfiguriert werden soll, dann ist bei der Panel-Beeinflussung die

Wirkungskette zu beachten:
Suchfeld-Ansicht Facette Suchergebnis.

Facetten-Ansicht

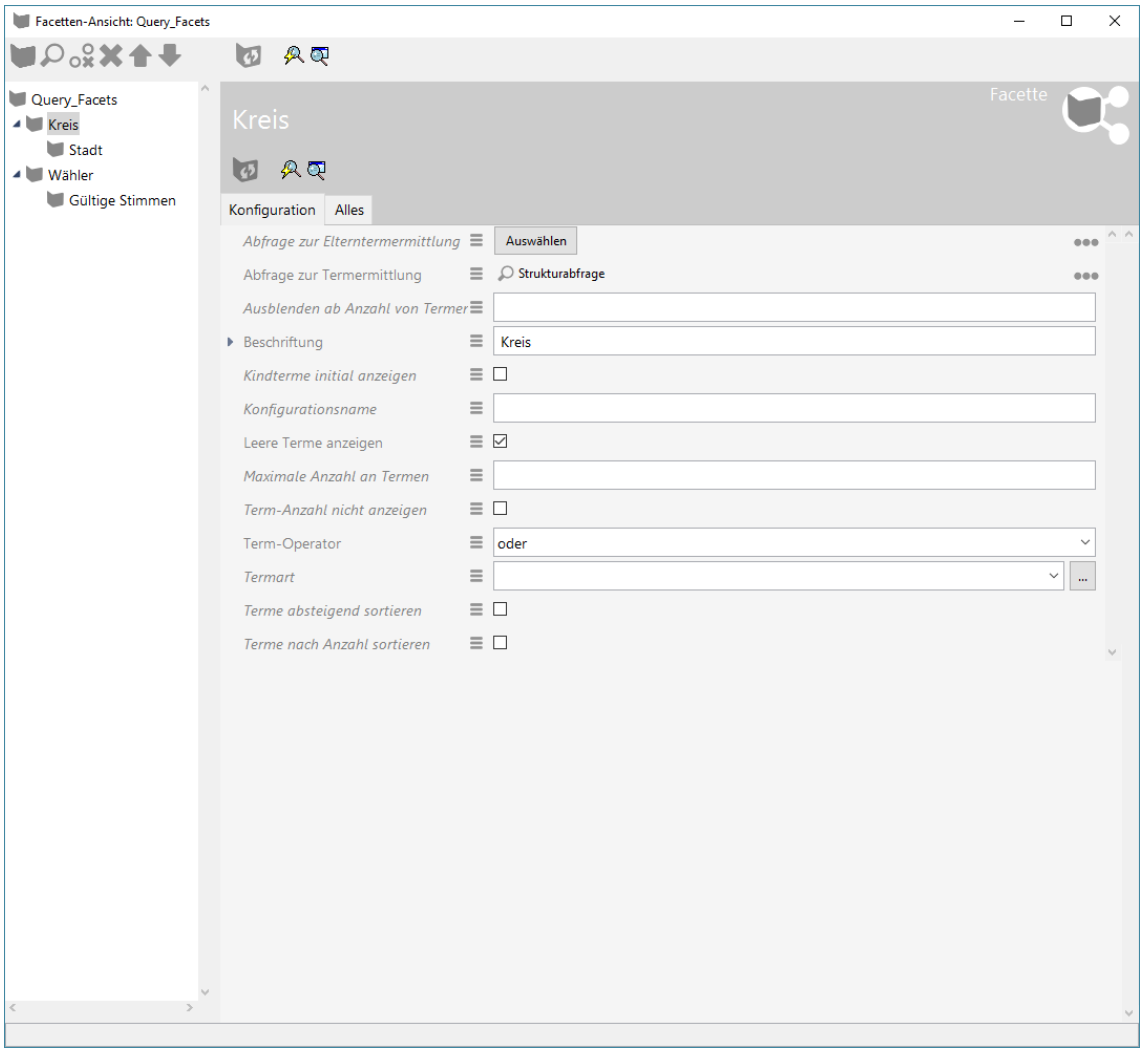


Zunächt ist eine Facetten-Ansicht zu erstellen, in der die einzelnen Facetten konfiguriert werden. Die Facetten-Ansicht kann folgende Eigenschaften haben:

Abfrage	Die Suche, mit der die Ansicht verknüpft ist. Wenn die Ansicht mit anderen verwandten Ansichten wie z.B. der Suchfeld-Ansicht verknüpft ist, muss hier keine Abfrage konfiguriert werden.
Beschriftung	Der Titel, der über der Facetten-Ansicht im Frontend erscheinen soll.
Konfigurationsname	Über Konfigurationsnamen lassen sich Ansichten und Panels identifizieren.
Skript für Beschriftung	Alternativ zu einer festen Beschriftung kann der Titel auch über ein Skript gesetzt werden.

Facette

Um Facetten zu konfigurieren, werden mit der Aktion Facette neu anlegen oder Facette hinzufügen Facetten-Views erstellt. Diese können mehrfach hierarchisch angeordnet werden.



An einer Facetten-Konfiguration stehen folgende Einstellungsmöglichkeiten zur Verfügung:

Abfrage zur Elterntermermittlung	Soll eine Term-Hierarchie gebildet werden, muss durch diese Abfrage definiert werden, wie sich Eltern-Kind-Elemente finden. Das Eingangsobjekt ist dabei das Kind-Element. Der Bezeichner *parentTerm* kennzeichnet das Eltern-Element. Anmerkung: Alle Terme, die eine Hierarchie bilden sollen, müssen durch die *Abfrage* an der Facetten-Konfiguration gefunden werden.

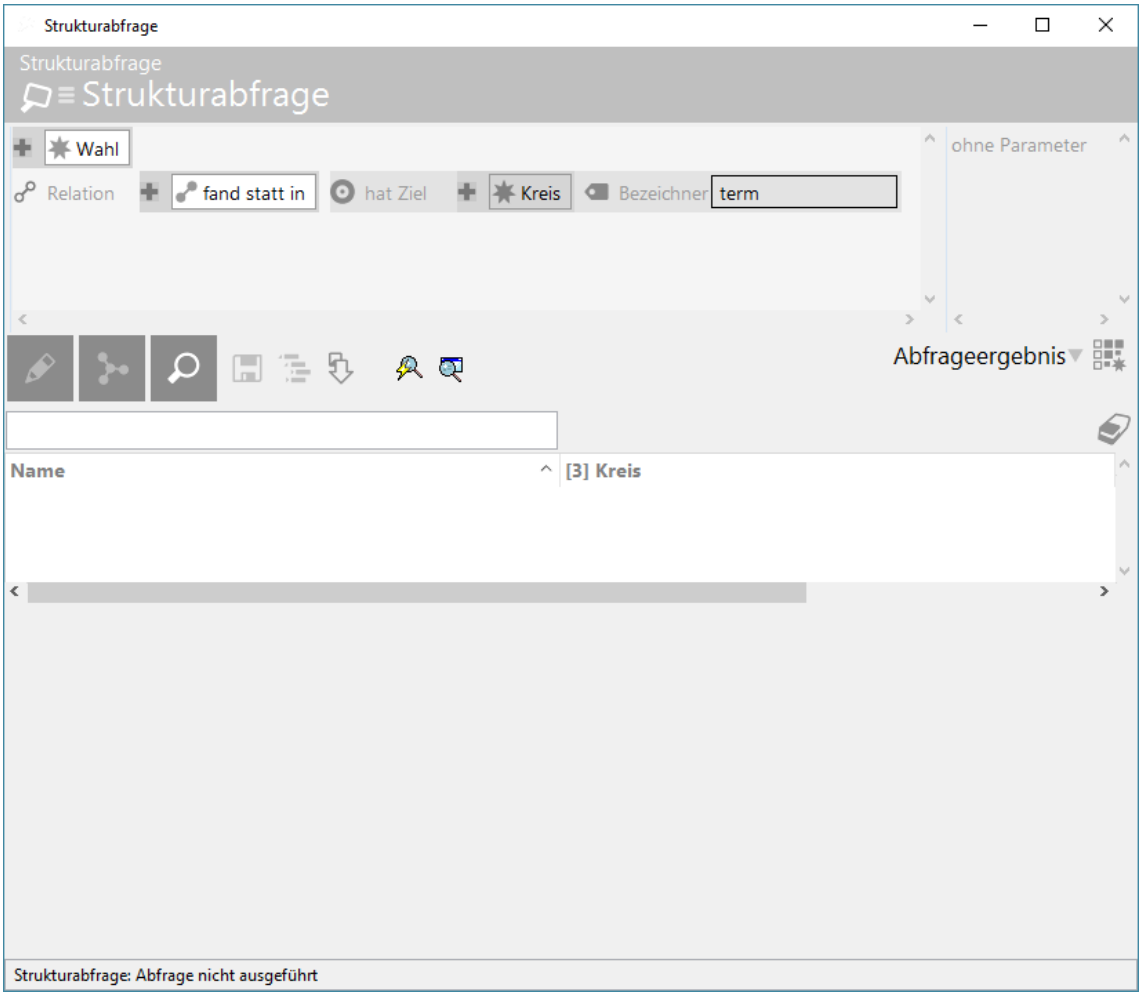
Abfrage zur Termermittlung	Diese Abfrage ist obligatorisch. Sie wird benötigt, um die Terme für eine Facette zu ermitteln. Das Eingangsobjekt ist vom Typ gleich den Suchergebnissen der Suche, die in der Suche-Konfiguration definiert ist. Die zu findenden Terme sind durch den Bezeichner *term* gekennzeichnet. Ein Beispiel hierzu befindet sich weiter unten.
Ausblenden ab Anzahl von Termen	Die Facette wird ausgeblendet, wenn die der Facette zugrundeliegenden Suchergebnisse diese Anzahl übersteigen.
Beschriftung	Hierüber ist die Beschriftung angegeben. Ist dieser Wert nicht gesetzt, wird der Name des Eingangsobjektes der *Abfrage* verwendet.
Kinder initial anzeigen	Falls die Facette hierarchisch aufgebaut ist, kann über diese Option definiert werden, ob die Unter-Facetten initial angezeigt werden sollen.
Konfigurationsname	Views und Panels können über einen Konfigurationsnamen identifiziert werden.
Leere Terme anzeigen	Standardmäßig werden Terme, die keine Anzahl haben, nicht eingeblendet. Durch setzen dieses Flags werden auch diese angezeigt.
Maximale Anzahl von Termen	Hier kann die maximale Anzahl der Terme festgelegt werden, die angezeigt werden sollen. Standardmäßig werden immer alle Terme dargestellt.
Termanzahl nicht anzeigen	Im Frontend wird hinter dem Facettentitel die Anzahl der gefundenen Terme angezeigt. Über diese Option kann die Anzeige der Anzahl deaktiviert werden. Vorteil ist eine mögliche Performance-Steigerung der Suchergebnis-Anzeige.
Termen-Operator	An dieser Stelle kann konfiguriert werden, wie die Terme miteinander verknüpft werden: • *"und"*: Suchergebnisse sollen auf alle Terme zutreffend sein • *"oder"*: Suchergebnisse sollen auf (mindestens) einen Term zutreffen

Termart	Hier stehen folgende Einstellungsmöglichkeiten zur Verfügung: • *Dynamisch:* Die Wertebereiche der Terme werden automatisch ermittelt. Die Werte, die zur Termbildung verwendet werden, müssen mit dem Parameter *termValue* versehen sein. Mit dem Parameter *Maximale Anzahl an Termen* lässt sich festlegen, wie viel Terme ausgebildet werden. • *Statisch:* Es müssen alle Terme, die angezeigt werden sollen, für sich konfiguriert werden. Es muss für jeden Term eine Suche angelegt werden, die die möglichen Treffer in der Hauptsuche beschreibt. Wird keine Termart ausgewählt (Standardverhalten), werden die Terme anhand der Abfrage an der Facetten-Konfiguration ermittelt. In der Abfrage können nur Relationsziele als mögliche Terme ausgebildet werden. Ein Beispiel findet sich unten.
Terme aufsteigend sortieren	Standardmäßig werden die Namen oder die Anzahlen aufsteigend sortiert. Mit diesem Flag lässt sich die Sortierung umdrehen.
Terme nach Anzahl sortieren	Standardmäßig werden die Terme alphabetisch nach dem Namen sortiert. Über diese Option werden sie nach der gefundenen Anzahl der Ergebnisse sortiert.

Beispiele

Facettierung nach Relationszielen

Im Prinzip ist alles möglich, was sonst auch in Strukturabfragen möglich ist. Zu beachten ist, dass die Ausprägungen einer Facette über den Bezeichner "term" ermittelt werden:

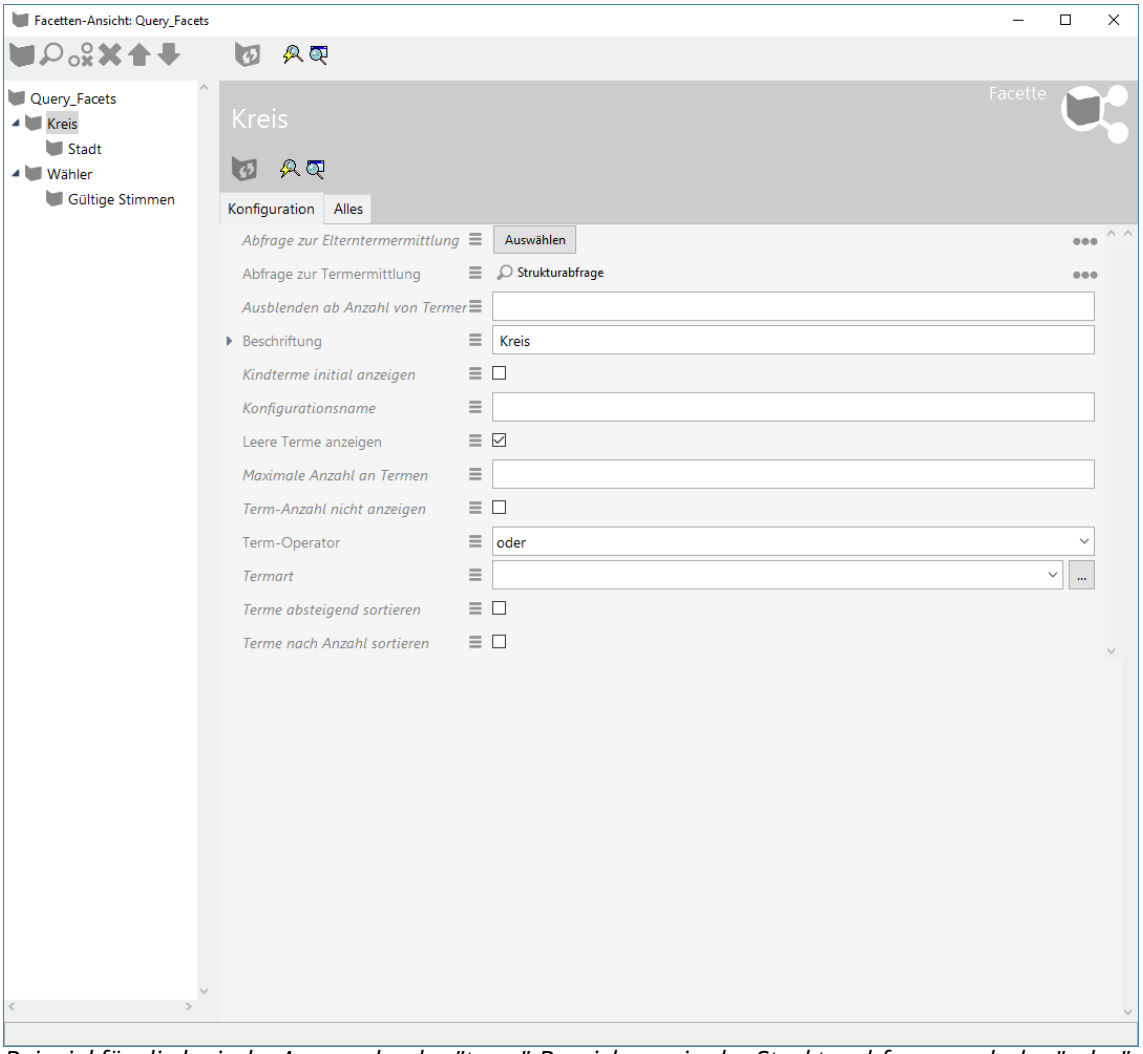


Beispiel einer Abfrage zur Termermittlung mit Relationszielen als Terme

Beispiel einer Abfrage zur Termermittlung mit Relationszielen als Terme

Logische Facettierung

Es ist auch möglich, dass der Bezeichner mehrfach in einer Strukturabfrage verwendet wird. Dann wird er über den bei "Term-Operator" definierten Logik (UND/ODER) verknüpft:

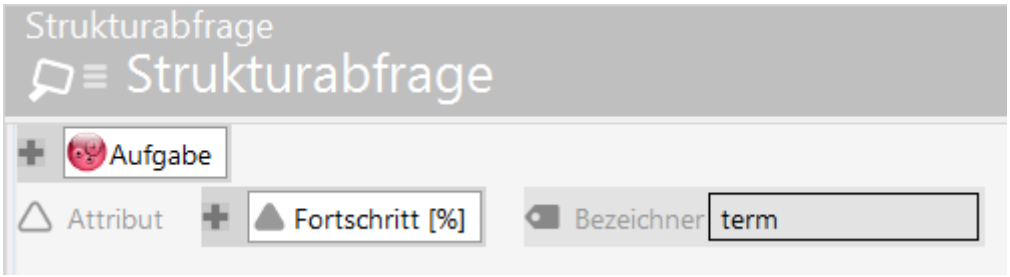


Beispiel für die logische Ansprache des "term"-Bezeichners in der Strukturabfrage nach der "oder"-Logik

Facettierung nach Attributwerten

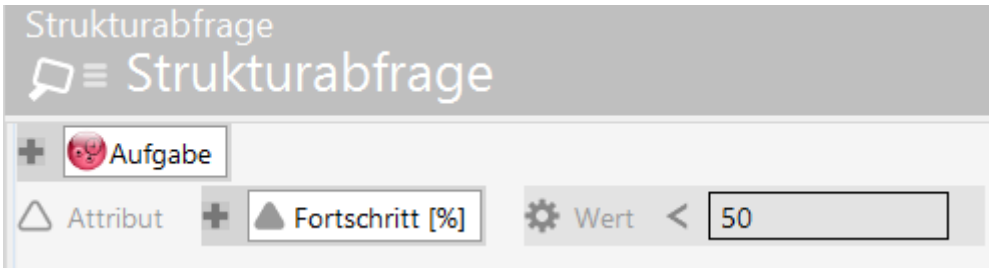
Suchergebnisse können anhand fest vorgegebener Attributwerte facettiert werden. Hierfür muss die Termart "statisch" gewählt werden. Wenn die Termart "statisch" gewählt wird, müssen die Terme eigenhändig über den Button "Neues verknüpfen" unter der jeweiligen Facette angelegt werden. Hierzu ist die Konfiguration wie folgt aufgebaut:

1. Die Strukturabfrage an der Facette enthält wie gewohnt die Elemente, die gefiltert werden sollen mit dem Bezeichner "term" an der Eigenschaft:



Beispiel einer Abfrage zur Termermittlung mit Attributwerten als Terme

2. Der eigentlichen Facette untergeordnet befindet eine weitere Facette mit einer Abfrage zur Eingrenzung der Terme. Die Strukturabfrage für die Terme enthält dann nur noch die Bedingungen der Eigenschaften an den Elementen:



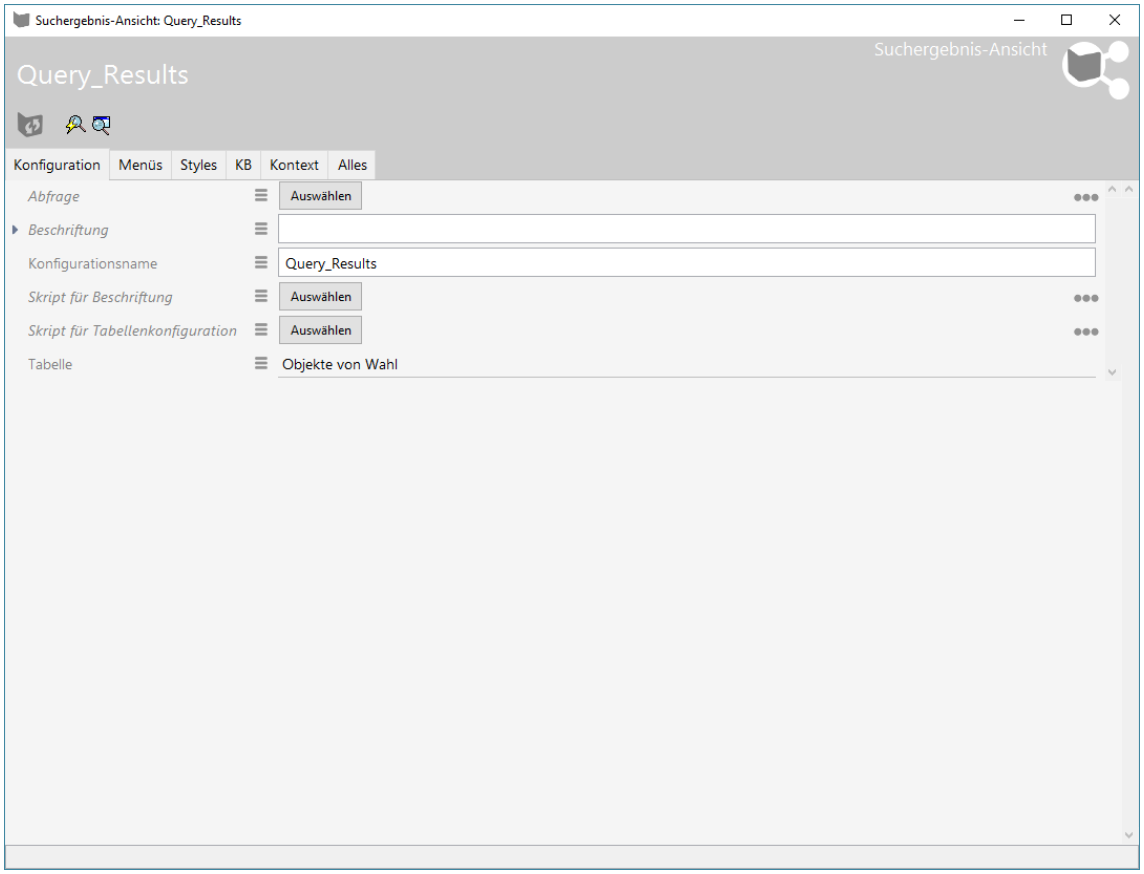
Beispiel einer Abfrage eines statischen Terms (vorgegebener Attributwert)

Eine Beschriftung am Term ist obligatorisch, da der Term sonst nicht angezeigt wird.

3.4.9.3 Search result view

Eine Suchergebnis-Ansicht wird dann verwendet, wenn in einer View nur die Ergebnisse einer Suche angezeigt werden sollen und nicht die Suchparameter. Falls die konfigurierte Suche parameterlos ist, reicht die Konfiguration einer Suchergebnis-Ansicht. Gibt es Parameter, sollte die Suchergebnis-Ansicht mit einer Suchfeld-Ansicht verknüpft werden.

Sie kann im Knowledge-Builder angelegt werden.



Im Reiter "Konfiguration" gibt es Möglichkeiten zur Bestimmung der allgemeinen Darstellung der Suche:

Abfrage	Hier wird die Suche konfiguriert, die beim Ausführen der Suche ausgeführt werden soll.
Beschriftung	Der hier eingegebene Wert erscheint als Überschrift der Suche
Konfigurationsname	Über den Konfigurationsnamen können Views und Panels identifiziert werden.
Skript für Beschriftung	Alternativ zu "Beschriftung" kann der Titel der Gruppe in einem Skript bestimmt werden.
Skript für Tabellenkonfiguration	Alternativ zu "Tabelle" kann an dieser Stelle über ein Skript die dargestellte Tabelle bestimmt werden.
Tabelle	Die Suchergebnisse werden im Frontend in der Tabellenkonfiguration angezeigt, die hier konfiguriert wird.

Im Reiter"Menüs" lassen sich Aktionen zu der Suche konfigurieren, und im Reiter "Styles" können bestimmte Darstellungsoptionen ausgewählt werden. Der "KB"-Reiter enthält Optionen, die nur für den Knowledge-Builder gelten und im Web-Frontend keine Anwendung finden. Über den "Kontext"-Reiter lässt sich konfigurieren, für welche Objekttypen die Suche-View verwendet werden soll und in welchen Anwendungskontexten.

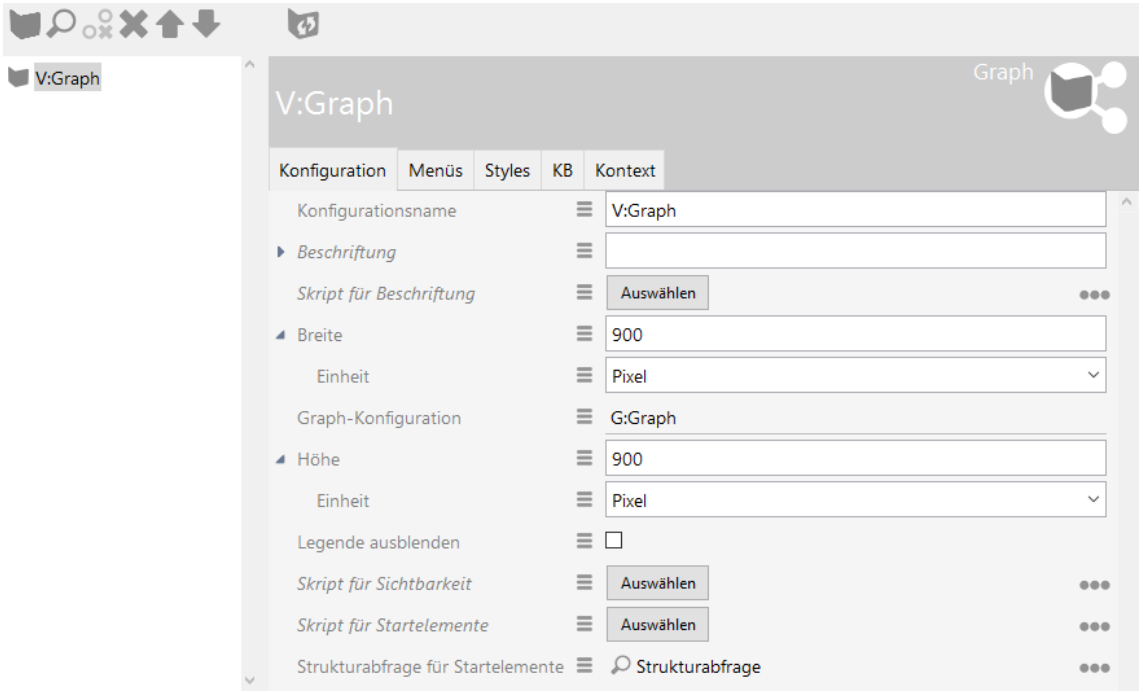
3.4.10 Graph configuration

Eine Graph-Konfiguration dient dazu, Objekte in einem Graphen darstellen zu können. Eine erste Einführung zur Nutzung des Graphen im Knowledge-Builder ist unter *Knowledge-Builder > Grundlagen > Graph-Editor* zu finden.

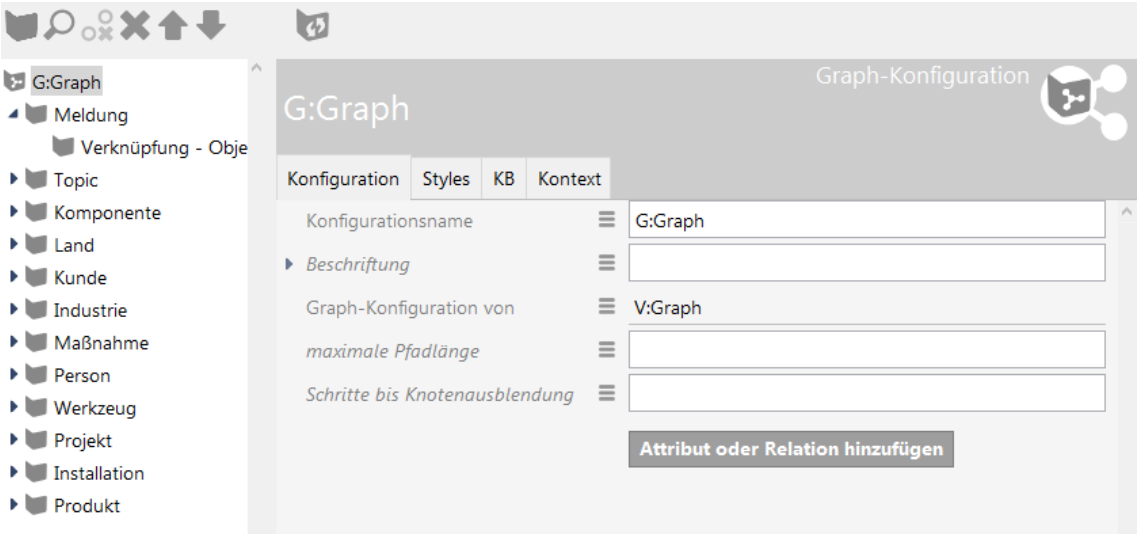
Details zu den Einstellungsmöglichkeiten der verschiedenen Views, die für das Einbinden eines Graphen in das Frontend benötigt werden, sind unter *Knowledge-Builder > View-Konfiguration > View-Konfigurationselemente > Graph* erläutert.

Es wird eine **Graph-View** sowie eine **Graph-Konfigurations-View** zur Darstellung benötigt. Das Panel, in dem der Graph angezeigt werden soll, erhält eine Graph-View ("V:Graph"). Bis zur Version 5.1 war das Kontextelement (genannt Start-Wissensnetzelement) optional und wurde beim Start der Applikation im Graphen angezeigt. Mit Version 5.2 ist die Vergabe eines Kontextelements obligatorisch, um keine Fehlermeldung hervorzurufen. Dabei spielt das Objekt selbst keine Rolle, es wird nicht Standardmäßig angezeigt.

Die Graph-View muss nur eine Verknüpfung zur Graph-Konfiguration enthalten. Optional, aber meist vorhanden, ist die Einstellung der Größe des Graph-Feldes über die Felder *Breite* und *Höhe*.



Die **Graph-View** sorgt dafür, dass der Graph insgesamt angezeigt wird. Um festzulegen, welche Knoten und Relationen angezeigt werden sollen, wird die **Graph-Konfiguration** verwendet.

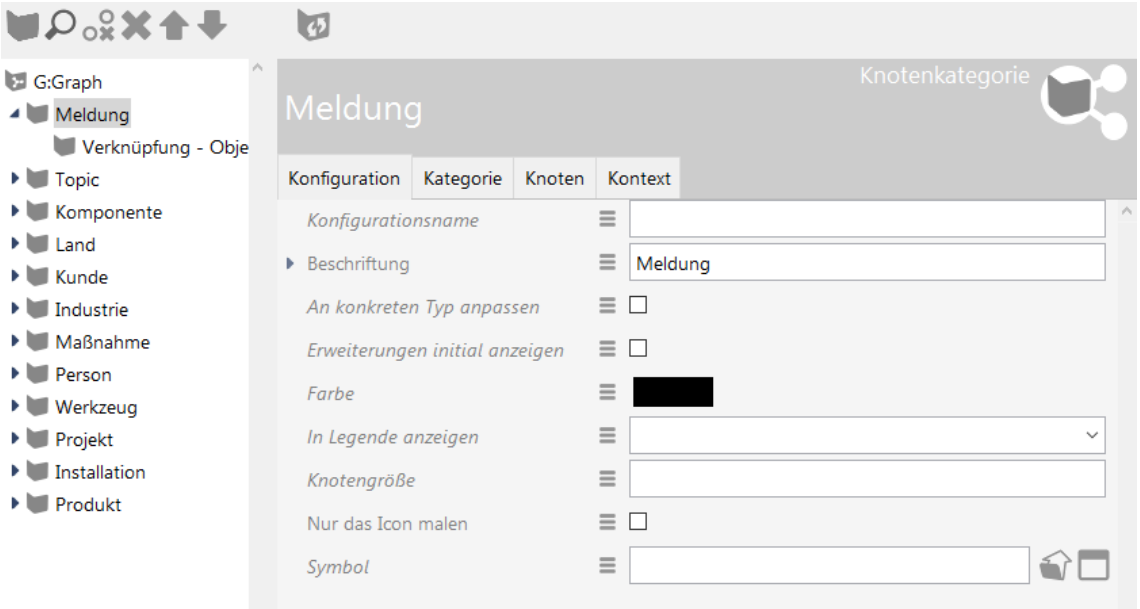


Für jeden Typen, dessen Objekte (oder auch Typen) angezeigt werden sollen, muss eine Knotenkategorie angelegt werden. Diese werden standardmäßig als Legende im Graphen angezeigt.

Im Graphen werden Objekte angezeigt, die direkt am Typen oder an dessen Untertypen hängen. Über *An konkreten Typ anpassen* werden die Untertypen gesondert in der Legende angezeigt, ohne das diese einzeln als Knotenkategorien angelegt werden müssen.

Um Typen statt Objekte anzuzeigen, muss im Reiter Kontext der Haken bei *anwenden auf Untertypen* gesetzt sein.

Im Reiter Knoten kann unter Menüs ein Satellitenmenü vergeben werden, um weiter im Graphen zu arbeiten (siehe *Knowledge-Builder* > *View-Konfiguration* > *Aktionen* > *Aktionen für den Viewconfiguration-Mapper* > *NN-Expand/NN-Hide/NN-Pin Aktionen).*



Um die Relationen zwischen den Knoten anzuzeigen, wird eine *Verknüpfung* unter jeder *Knotenkategorie* benötigt. Hier wird festgelegt, welche Relationen für diesen Typen angezeigt werden sollen. Die Relationen können über eine Abfrage, ein Skript oder über die direkte

Bestimmung der Relation festgelegt werden. *Benutzerrelation* kann vergeben werden, wenn alle Relationen (außer Systemrelationen) angezeigt werden sollen.

Für weitere Details siehe Kapitel *vcm-plugin-net-navigator*

3.4.11 Text

Mit der Text-View kann Text dargestellt werden, der entweder statisch vorgegeben oder durch ein Skript berechnet wird.

Text	Statischer, mehrsprachiger Text
Skript für Text	Skript zur Berechnung des Textes
Beschriftung	Optionale Überschrift
Skript für Beschriftung	Optionales Skript zur Berechnung der Überschrift

Beispiel für ein Text-Skript:

```
function text(element)
{
    return "Durch ein Skript im Netz " + $k.volume() + " generierter Text";
}
```

3.4.12 Image

Stellt ein im Wissensetz gespeichertes Bild dar, das entweder statisch vorgegeben oder durch ein Script berechnet wird.

Bild	Statisches Bild
Skript für Bild	Skript zur Berechnung des Bildes. Als Rückgabewert wird ein Blob-Attribut erwartet. Dynamische Blobs (z.B. durch Download mittels Http-Client) sind nicht möglich.
Beschriftung	Optionale Überschrift
Skript für Beschriftung	Optionales Skript zur Berechnung der Überschrift
Breite / Höhe	Fest vorgegebene Breite / Höhe des Bildes

3.4.13 Script generated HTML

Dieser View erzeugt per Skript HTML. Sowohl Knowledge-Builder als auch ViewConfigMapper zeigen dies ungefiltert dar, d.h. es ist Aufgabe des Script-Entwicklers, Nutzerinhalte nicht ungefiltert auszugeben. Im Knowledge-Builder sind die Anzeigemöglichkeiten stark eingeschränkt (u.a. kein CSS).

Bei umfangreicherem HTML sollte man besser einen scriptgenerierten View verwenden.

Als Parameter werden an das Skript folgende Argumente übergeben:

element	$k.SemanticElement	Das Element, in dessen Kontext der View dargestellt wird
document	$k.TextDocument	Dokument, auf das HTML ausgegeben wird

Zur Ausgabe des HTML kann man zwei Ansätze wählen:

- Ausgabe des HTML-Quellcodes per Funktion print() des Dokuments
- Strukturierte Ausgabe per XMLWriter

Das nachfolgende Beispiel zeigt die Verwendung eines XMLWriters, um eine Überschrift auszugeben:

```
/**
```

```
 * 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 Skriptgenerierte View

Ein scriptgenerierte View ermöglicht es, eigene View-Komponenten zu definieren. Die Daten werden durch ein Skript erzeugt und per JSON weitergegeben. Es ist Aufgabe des Frontends, diese darzustellen.

view-Type	Frei wählbarer Bezeichner, der im JSON ausgegeben wird. Dieser wird dazu verwendet, im Frontend die eigene Komponente zuzuordnen.
Skript	Liefert die Daten, die im JSON ausgegeben werden.

An das Skript werden zwei Parameter übergeben:

element	$k.SemanticElement	Das Element, in dessen Kontext der View angezeigt wird
view	object	Vorbefülltes Objekt mit den Viewdaten. Konfigurationselemente wie z.B. Styles sind hier bereits enthalten.

Nachfolgendes Skript liefert die Daten für einen View, der im Plugin *vcm-plugin-timeline* enthalten ist:

```
/**
 * 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 Plugins

Allgemein sollte dokumentiert werden:

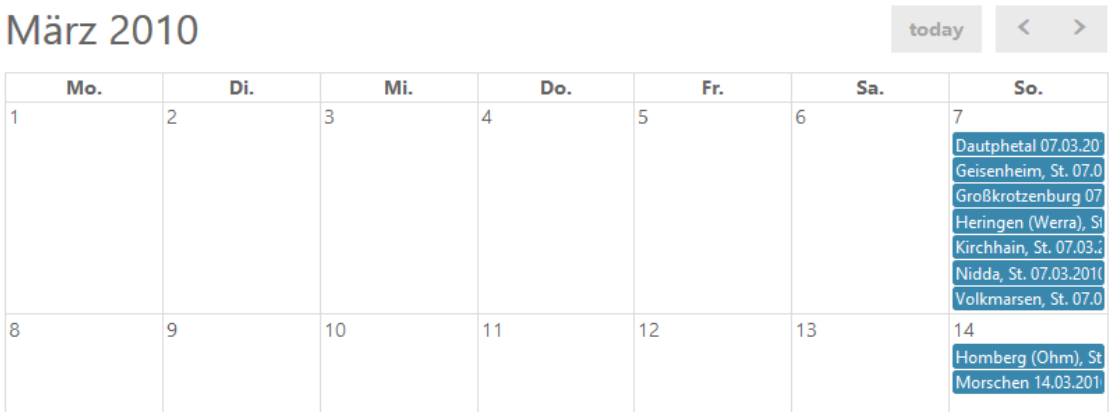
- Wozu dienen Plugins
- Code-Beispiel zur Einbindung von Plugins in den View-Config-Mapper sollen **NICHT** hier dokumentiert werden, sondern unter dem Kapitel zur Erstellung von Anpassungsprojekten

Folgendes sollte zu jedem einzelnen Plugin dokumentiert werden:

- Allgemeine Beschreibung der Funktionsweise und Screenshot
- View-Konfigurationselemente an die ein Plugin gebunden ist (bspw. vcm-plugin-calendar -> Tabellen-View)
- Welches Plugin-spezifischen Styles werden unterstützt
- Plugin-spezifische Aktionen (bspw. nn-expand für vcm-plugin-net-navigator)

3.5.1 vcm-plugin-calendar

Mit dem vcm-plugin-calendar können Daten in einem Kalender dargestellt werden.



Um die Daten als Kalender anzuzeigen, muss an der Tabellenkonfiguration ein Style-Element hinzugefügt werden, das den renderMode *calendar* enthält. Der Wert unter Anzahl Zeilen

(Page Size) gibt an, wieviele Kalendereinträge maximal pro Ansicht (in diesem Fall pro Monat) angezeigt werden können. Die Tabelle muss die folgenden Spalten haben:

- *start*: Ein Datum an dem der Kalendereintrag startet.
- end: Enddatum des Eintrags (optional)
- title: Der Titel des Eintrags
- allDay: Boolean-Wert der angibt, ob der Eintrag für den gesamten Tag gilt (optional)
- Weitere Möglichkeiten für Spalten sind in der fullcalendar.io Event_Object Dokumentation zu finden.

Es kann auch eine Auswahlaktion auf die Spalten der Tabelle konfiguriert werden. Diese wird dann beim Klick auf einen Kalendereintrag ausgeführt.

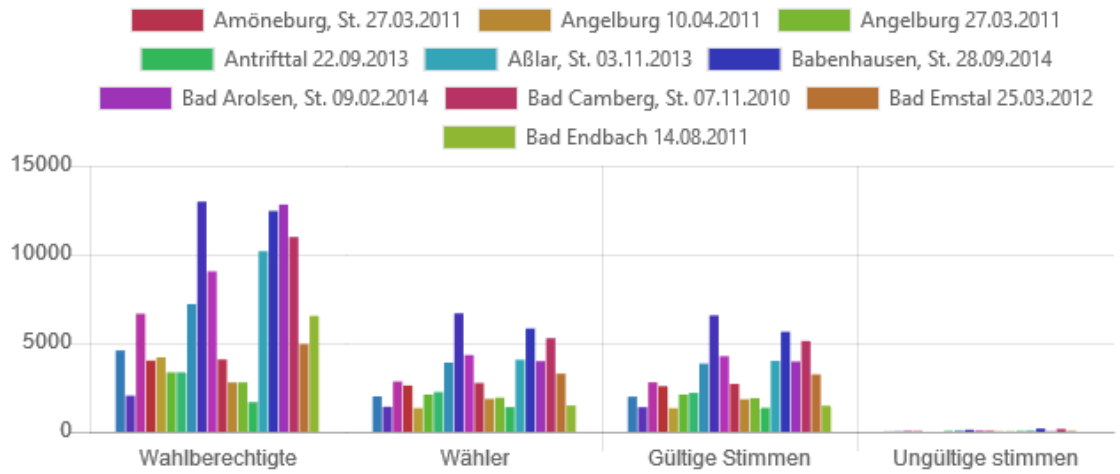
Außerdem lassen sich mit dem Style Attribut vcmPluginCalendarOptions weitere Konfigurationen vornehmen.

Weiterführende Informationen zum Plugin sind unter fullcalendar.io abrufbar.

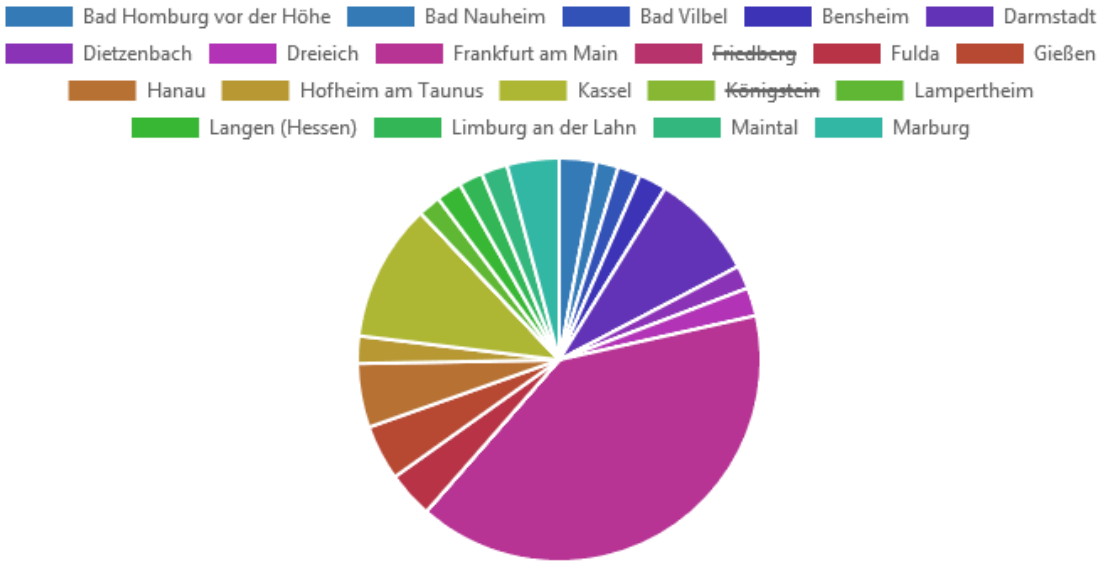
3.5.2 vcm-plugin-chart

Das vcm-plugin-chart dient dazu, Daten aus einer Tabellenkonfiguration in Form eines Diagramms im Web-Frontend anzeigen zu können. Es stehen verschiedene Diagrammtypen zur Verfügung: Linien-, Balken-, Torten-, Ring- und Radardiagramme.

Beispiel eines Balkendiagramms:



Beispiel eines Tortendiagramms:



3.5.2.1 Konfiguration auf Basis einer Tabelle

Für die generierung eines Diagramms muss ein Style an einer Tabellenkonfiguration angelegt werden, der als *renderMode* die Option "chart" hat.

Wird beispielsweise in der zugrundeliegenden Tabellenkonfiguration eine Aktion mit der Option "Graphisch darstellen" angefügt, so kann durch Klick auf Teile des Diagramms der jeweilige Datensatz zussätzlich im Net-Navigtor angezeigt werden.

Das Plugin benutzt chart.js für die Generierung der Charts.

Für den vcm-plugin-chart gibt es mehrere Optionen zur Darstellungsanpassung, die über Styles festgelegt werden können:

- **vcmPluginChartDataColumns:**
  Zeichenkette mit Spaltennummern die als Datenquelle dienen. Default: columns 1-n
- **vcmPluginChartDataMode:**
  'rows' oder 'columns'. Default: 'rows'
- **vcmPluginChartHeight:**
  Angabe der Diagrammhöhe in Pixel. Default: 'auto'
- **vcmPluginChartWidth:**
  Angabe der Diagrammbreite in Pixel. Default: 'auto'
- **vcmPluginChartLabelColumn:**
  Spaltennummer für Beschriftungen. Default: 0
- **vcmPluginChartOptions:**
  Optionen für Anpassung von Legenden-Anzeige und Skalierung von Achsen, die an chart.js übergeben werden.
- **vcmPluginChartType:**
  Angabe zu Diagrammtyp: 'line', 'bar', 'radar', 'pie' oder 'doughnut'. Default: 'line'

3.5.2.2 Konfiguration auf Basis eines skriptgenerierten Views

Charts lassen sich anstelle von Tabellen auch über einen Skriptgenerierten View anzeigen.

Voraussetzung hierzu ist, dass als „viewType“ im Konfigurationsreiter des Skriptgenerierten Views „chart“ angegeben sein muss.

Außerdem muss wie auch bei der Tabellenkonfiguration ein Style vergeben werden, der über die Eigenschaft vcmPluginChartType den gewünschte chartType angibt (line', 'bar', 'radar', 'pie' oder 'doughnut'. Default: 'line').

Im Folgenden ist ein Beispiel-Skript, welches Aufgaben nach ihrem Status zählt und die Menge in einem Tortendiagramm darstellt. Es ist zu beachten, dass dieses Skript ein Beispiel für den chartType „pie“ ist. Nutzen Sie die Dokumentation des chart.js um die Unterschiede der Datenformate der anderen chartTypes festzustellen: https://www.chartjs.org/docs/latest/

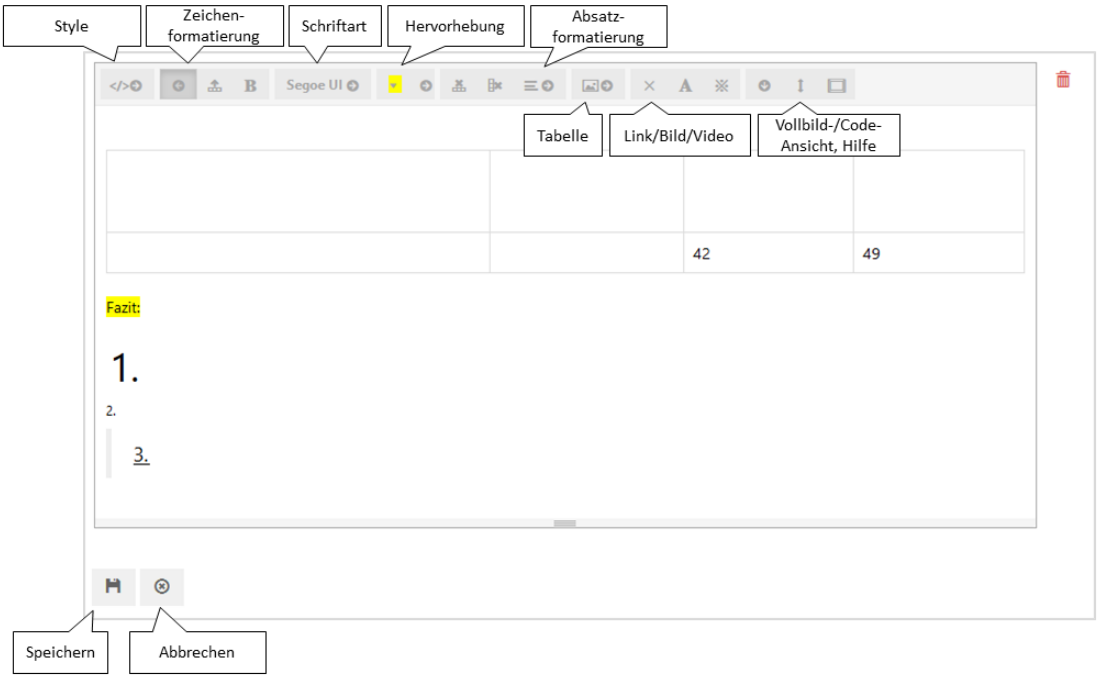
```
function customizeView(element, view) {
  var aufgabenCount= $k.Registry.type("aufgabe").allInstances().reduce(function (result, aufgabe,
        var status = aufgabe.attributeValueString("statusAufgabe");
        result[status]= (result[status]||0)+1
        return result;
    }, {})

   view.chartData = {
    datasets: [{
         data:Object.keys(aufgabenCount).map(function(key) {return aufgabenCount[key]}),
         backgroundColor: ['red', 'green']
    }],
    labels: Object.keys(aufgabenCount)
  }

  view.type =  'pie'

  return view;
}
```

Dieses Tortendiagramm wurde über ein Skript generiert, welches das Plugin chart.js benutzt.

3.5.3 vcm-plugin-html-editor

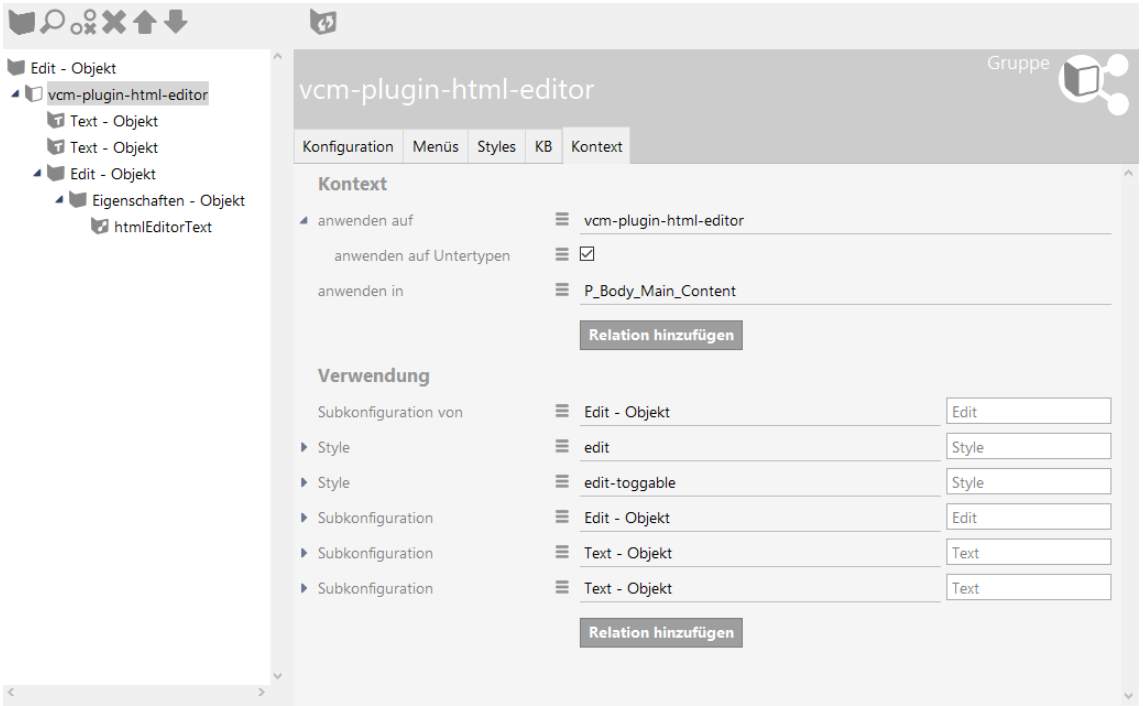
Web-Frontend

Das vcm-plugin-html-editor ermöglicht es, HTML-formatierten Text zu bearbeiten. Es verwendet hierfür den WYSIWYG-Editor summernote.

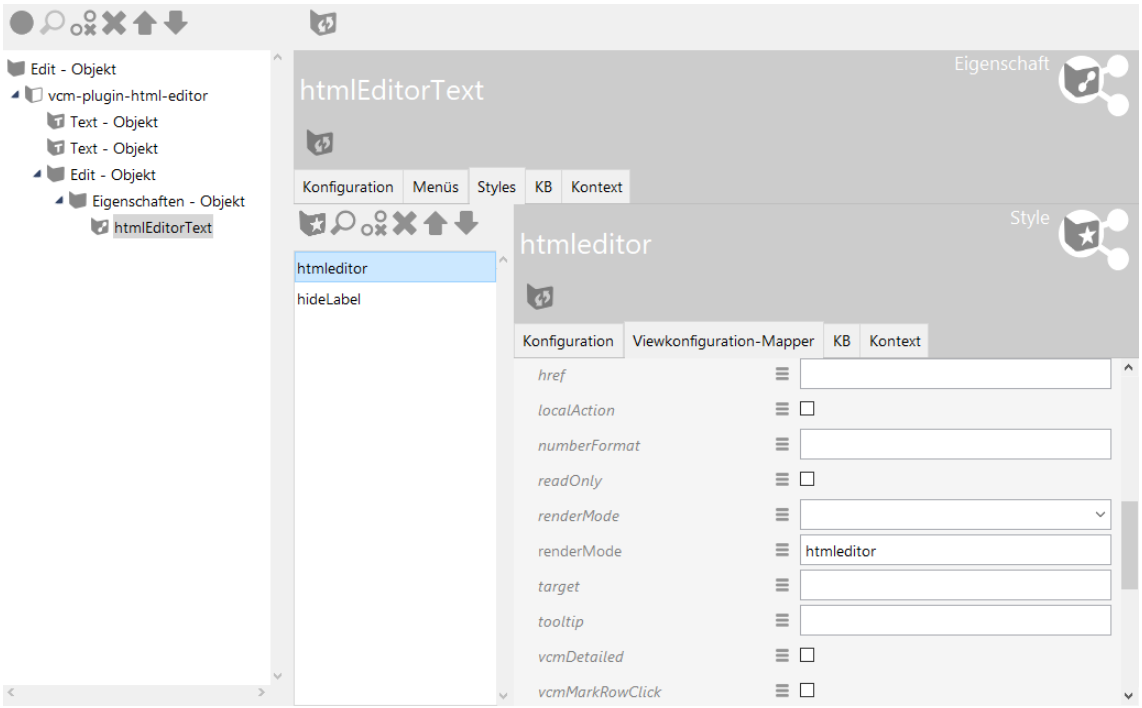


Konfiguration

Für die View-Konfiguration des HTML-Editors wird eine Grupe benötigt: Im Reiter "Kontext" wird hierzu unter "anwenden auf" der Eintrag "vcm-plugin-html-editor" benötigt.



Nach dem der Konfiguration der Gruppe muss eine Eigenschafts-Konfiguration angelegt werden. Diese ist mit einem Style zu versehen, welchen den renderMode "htmleditor" enthält:

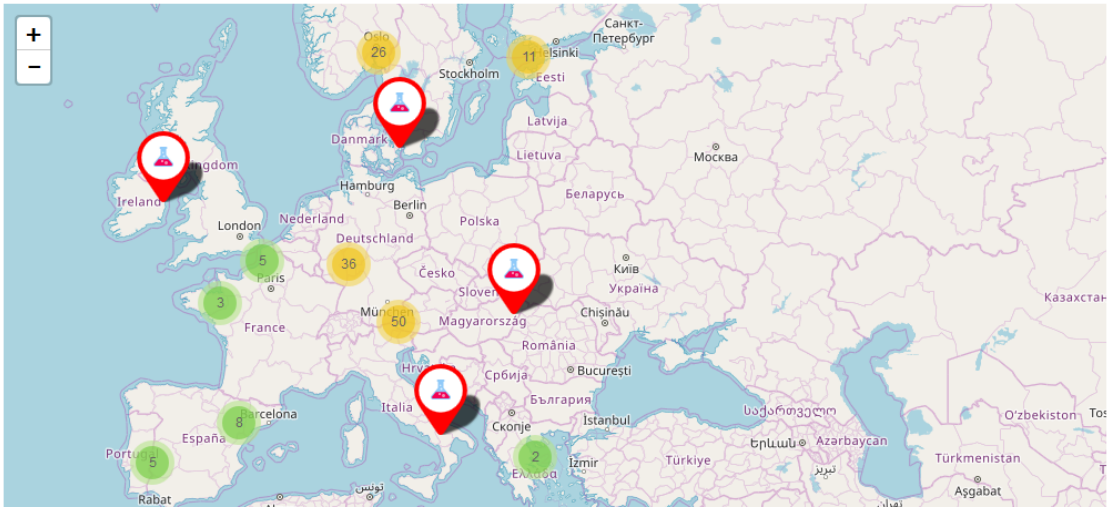


Die Eigenschafts-Konfiguration benötigt darüber hinaus ein Zeichenketten-Attribut, welches den Text enthalten soll.

Damit der Inhalt vom Web-Frontend aus bearbeitbar ist, muss ein zusätzlicher Style mit dem renderMode "edit" an der übergeordneten Gruppe der Eigenschafts-Konfiguration angelegt werden. Alternativ kann hierzu eine übergeordnete Edit-Konfiguration angelegt werden.

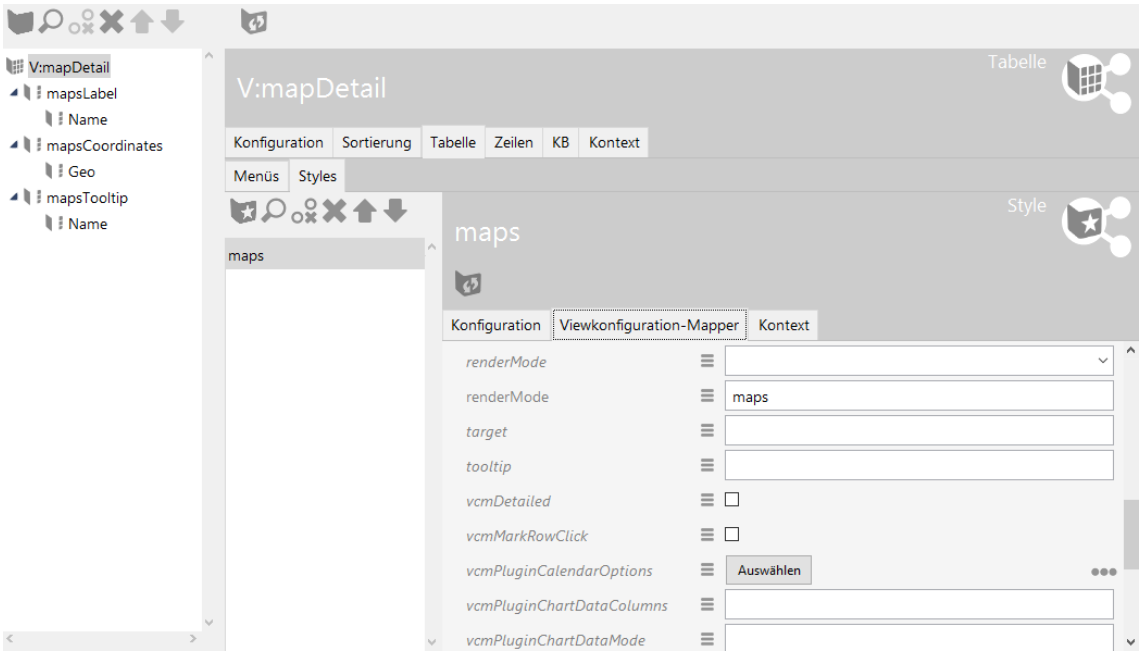
3.5.4 vcm-plugin-maps

Das Karten-Plugin ermöglicht das Einbinden einer Karte in das Frontend. Dafür müssen die Objekte, die angezeigt werden sollen, ein Attribut vom Typ "Geographische Position" besitzen.



Die Karte kann als Skriptgenerierte View oder Objektliste konfiguriert werden.

Für die Nutzung über Objektlisten wird in einer Tabellen-View am Reiter "Tabelle" ein Style mit renderMode "maps" angebracht.



Über Spalten wird die Karte weiter konfiguriert. Die Spalten mit der Beschriftung "mapsLabel" (enthält den Namen des Objekts) und "mapsCoordinates" (enthält das Attribut mir den geographischen Koordinaten) sind obligatorisch, da über sie die anzuzeigenden Objekte und deren Koordinaten ermittelt werden. Dabei ist zu beachten, dass diese exakte Beschriftung zu verwenden ist.

Optionale Spalten bzw. Funktionen sind:

- "mapsPopup" - ruft durch Klick auf das Icon ein Popup mit dem Inhalt dieser Spalte auf (akzeptiert html). Wenn eine Auswahl-Aktion vorhanden ist, wird diese Spalte deaktiviert.
- "mapsTooltip" - zeigt die konfigurierte Eigenschaft als Tooltip an.
- "mapsColor" - bestimmt die Farbe des Markierungselementes auf der Karte.
- "mapsIconLocator" - als Standard wird das Icon des Typs für die Anzeige der Objekte auf der Karte verwendet. Hier ist eine Anpassung durch Angabe eines anderen Icon-Ortes in Form der ID des entsprechenden Datei-Attributs möglich.

An der Tabelle kann eine Auswahl-Aktion angebracht werden, die bei Klick auf das Markierungselement aktiviert wird.

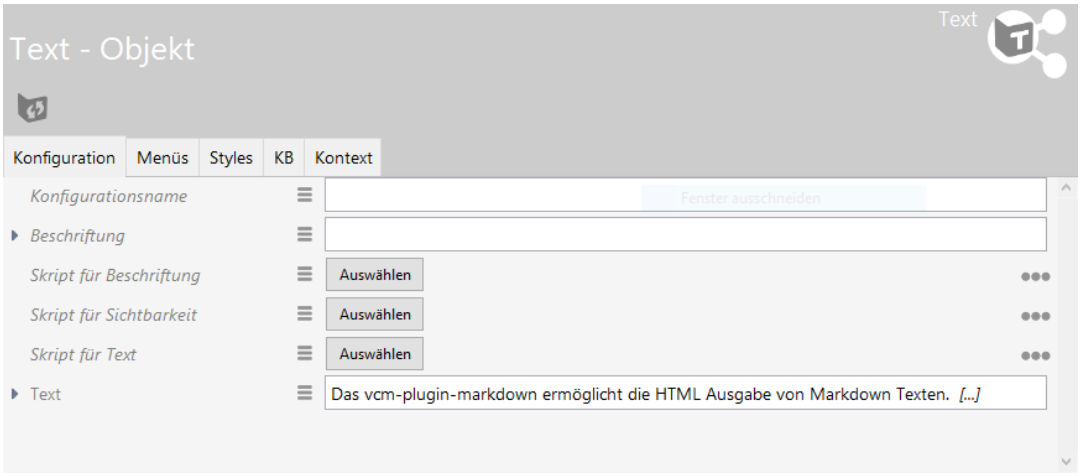
3.5.5 vcm-plugin-markdown

Das vcm-plugin-markdown ermöglicht die HTML Ausgabe von Markdown Texten.

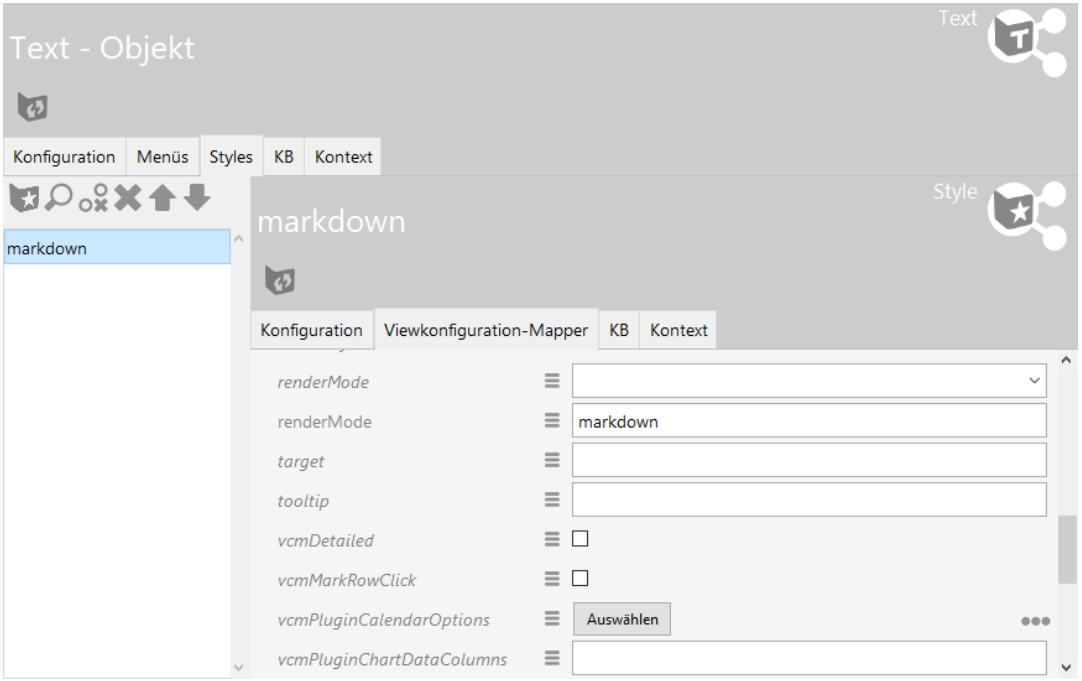
Es lässt sich verwenden indem man einen Style mit dem renderMode *markdown* an eines der folgenden Konfigurationselemente anbringt:

- *Statischer Text* : Hierbei wird die Viewconfig-Eigenschaft *Text* des Konfigurationselements

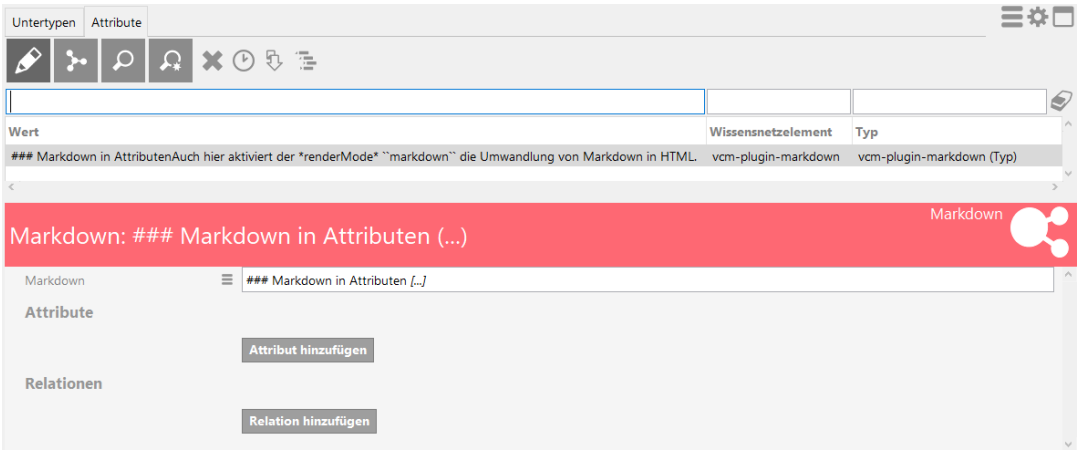
als Markdown interpretiert.



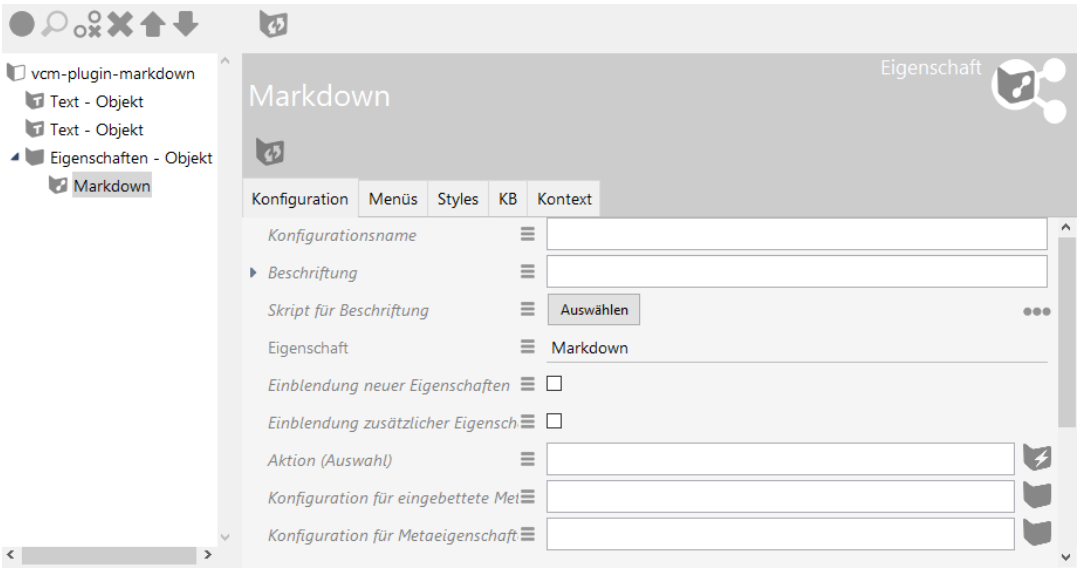
Für die Anwendung des Plugins muss im Reiter "Style" der RenderMode mit der Bezeichnung "markdown" eingegeben werden:



- *Eigenschaft* : hierbei wird der Wert des Attributes als Markdown interpretiert.



Die Konfiguration der View für das Zeichenketten-Attribut "Markdown" findet mithilfe einer Eigenschafts-View statt:



Die Eigenschaft erhält wie ein Text-Objekt gleichermaßen den RenderMode "markdown".

Nach dem Rendern erhält der Text im Web-Frontend die optischen Hervorhebungen:

Auch hier aktiviert der *renderMode* `markdown` die Umwandlung von Markdown in HTML.

Weitere Konfiguration des Plugins kann über das Style-Attribut *vcmPluginMarkdownOptions* vorgenommen werden.

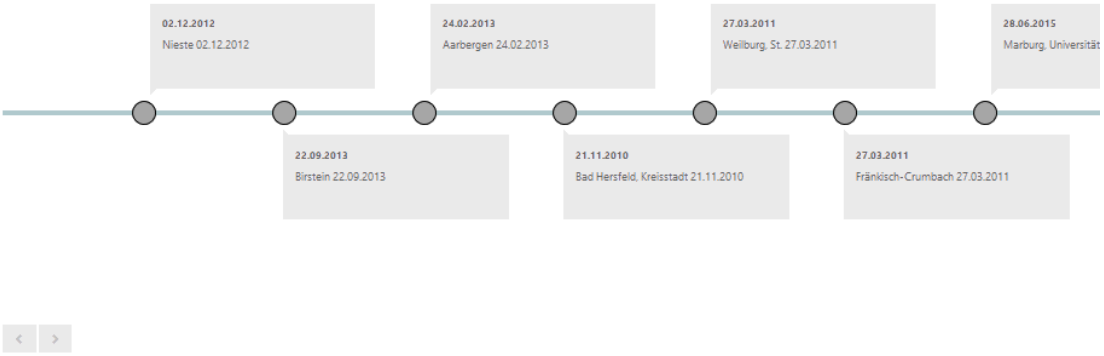
Das Plugin verwendet das Modul markdown-it

3.5.6 vcm-plugin-timeline

Mit dem Plugin vcm-plugin-timeline lassen sich Ereignisse chronologisch auf einem Zeitstrahl anzeigen.

Der Zeitstrahl kann horizontal oder vertikal ausgerichtet werden. In der horizontalen Variante bietet die Timeline zusätzlich zwei Knöpfe zum scrollen an, sollte die Zeitleiste breiter

als der zur Verfügung stehende Platz sein. Bei der vertikalen Variante sollte in diesem Fall vom Browser eine Scrollbar angeboten werden.



3.5.6.1 Configuration

Es muss zunächst eine "Skriptgenerierte View" angelegt werden und ihr viewType-Attribut auf "timeline" gesetzt werden. Außerdem muss ein Skript an der View angebracht werden, welches die Daten für die Timeline bereit stellt, beispielsweise:

```
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
}
```

Über das Skript können mit den folgenden Parameter unter 'view.options' das Erscheinungsbild der Timeline angepasst werden:

- 'layout': Bestimmt die Richtung des Zeitstrahls, entweder 'horizontal' oder 'vertical'.
- 'itemHeight': Höhe der Elemente auf der Zeitleiste in Pixeln. Falls nicht gesetzt, erhalten alle Elemente die Höhe des Elements mit dem größten Platzbedarf.

Unter 'view.events' muss ein Array angelegt werden, welches die Ereignisse als Objekte enthält. Diese benötigen jeweils die Attribute 'name', 'date' und 'elementId'.

3.5.6.2 Styling

Mittels CSS-Regeln lässt sich der Default-Style der Timeline anpassen.

Hierfür steht, je nach konfigurierter Ausrichtung der Zeitleiste, die folgende Klassenhierarchie zur Verfügung:

Die Textfelder der Ereignisse lassen sich mit den folgenden Selektoren anpassen:

```
.timelineVertical ul li
.timelineHorizontal ul li
```

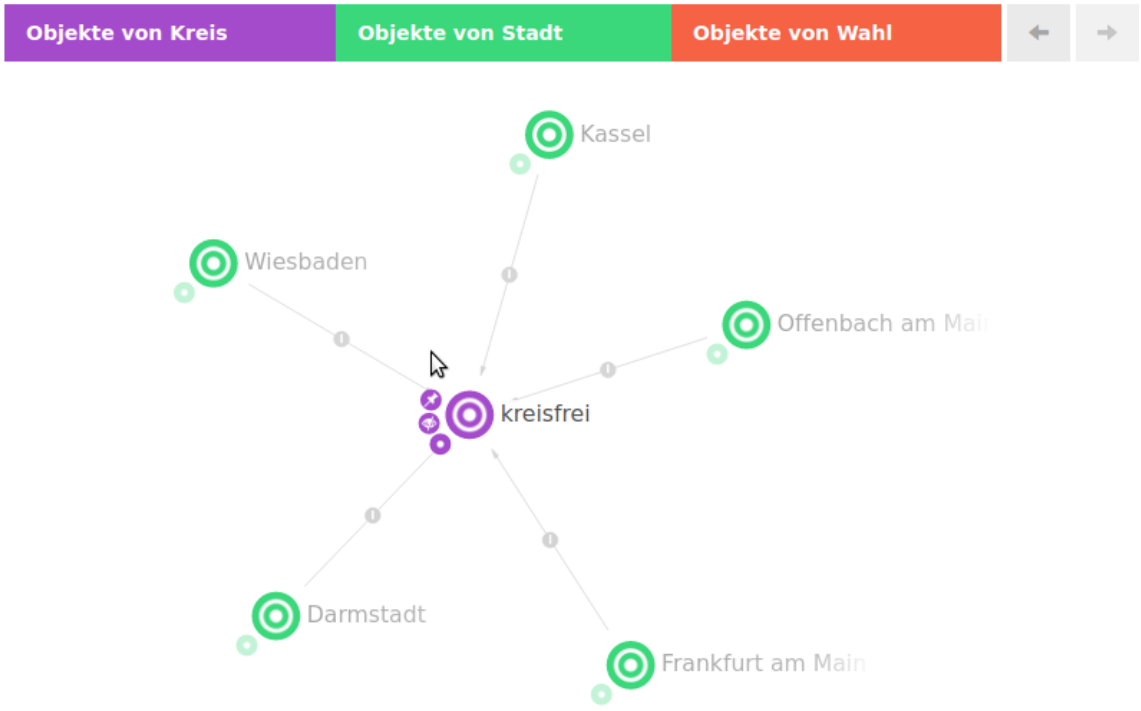
Die Markierungspunkte der Ereignisse lassen sich über folgenden Selektor anpassen:

```
.timelineVertical ul li::after
.timelineHorizontal ul li::after
```

3.5.7 vcm-plugin-page

3.5.8 vcm-plugin-net-navigator

Das vcm-plugin-net-navigator visualisiert Elemente in einer graphartigen Ansicht.



3.5.8.1 Configuration

Das Plugin kann über Styles konfiguriert werden.

Styles der View

Style	Beschreibung

vcmPluginNetNavigatorOptions	Ein JSON Objekt für die View Optionen. Details s. unten
extra	Alternativ zu *vcmPluginNetNavigatorOptions*

Optionen

Option	Beschreibung
vcmPluginNetNavigatorOptions.categories.hideLabel	Ein-/Ausblenden der Kategorielabels
vcmPluginNetNavigatorOptions.categories.embeddedActions	Konfiguration wo die Aktionen angezeigt werden sollen. Bei true werden sie neben den Kategorien angezeigt
vcmPluginNetNavigatorOptions.categories.compactActions	Aktionen in einem Menü zusammenfassen
vcmPluginNetNavigatorOptions.history.enabled	Aktiviert/Deaktiviert die Navigationshistorie
vcmPluginNetNavigatorOptions.enableEditing	Aktiviert-/Deaktiviert die Möglichkeit Elemente im Graph neu zu verknüpfen
vcmPluginNetNavigatorOptions.nnOptions	Optionen für die Net-Navigator Komponente
vcmPluginNetNavigatorOptions.nnOptions.overload.maxExpandNodes	Anzahl der gleichzeitig zu öffnenden Knoten, bevor ein Rückfragedialog zu den zu öffnenden Relationen erscheint. Default Wert ist 5.

Styles der Knoten

extra	Ein JSON Objekt für die Knoten Optionen. Details s. unten

Optionen der Knoten

color	Überschreibt die Hintergrundfarbe des Knotens
label	Überschreibt das Label des Knotens
icon	Überschreibt das Icon des Knotens

Styles der Kanten

extra	Ein JSON Objekt für die Kanten Optionen. Details s. unten

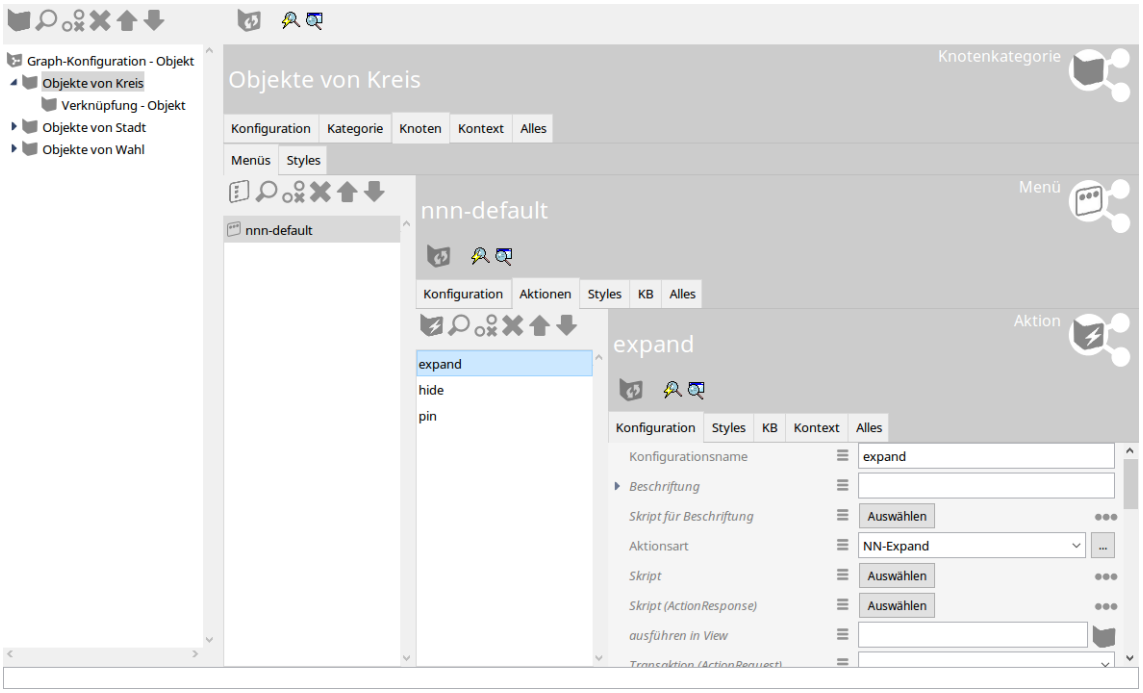
Optionen der Kanten

color	Überschreibt die Hintergrundfarbe der Kante
label	Überschreibt das Label der Kante

3.5.8.2 Actions

Knoten und Relationen können um Aktionen erweitert werden. Diese werden Kreisförmig um einen Knoten bzw. die Relation angeordnet.

Aktionen werden in der Graph-Konfiguration innerhalb von einer Knotenkategorie bzw. Verknüpfung konfiguriert.



Vorkonfigurierte Aktionen

Aktionsart	Beschreibung
NN-Expand	Über ein kleines Plus Symbol können benachtbarte Knoten (für die es eine Konfiguration gibt) angezeigt werden
NN-Hide	Ausblenden eines Knotens
NN-Pin	Festpinnen eines Knotens

Eigene Aktionen

Für die Darstellung wird immer ein Symbolbild benötigt

3.5.8.3 Followups

Die Graphansicht reagiert auf folgende Followups:

Followup	Data	Beschreibung
graph-show	{elementId: ["ID123_456"]}	Zeigt die Elemente im Graph an. Bereits angezeigte Elemente werden ausgeblendet
graph-join	{elementId: ["ID123_456"]}	Fügt die Elemente dem Graph hinzu. Bereits angezeigte Elemenete bleiben erhalten
graph-hide	{elementId: ["ID123_456"]}	Entfernt Elemente aus dem Graph
graph-back		Geht in der Graph-Historie einen Schritt zurück
graph-forward		Geht in der Graph-Historie einen Schritt vorwärts
graph-reload		Aktualisiert die Elemente im Graph

Beispiel: ActionResponse Skript, welches den Wurzelbegriff der Graphansicht hinzufügt:

```
function actionResponse (element, context, actionResult) { var actionResponse = new $k.ActionRespo

  actionResponse.setFollowup('graph-join')
  actionResponse.setData({
    elementId: [$k.rootType().idString()]
  })
  return actionResponse
}
```

3.6 Spezielle Konfigurationen

Dieses Kapitel behandelt spezielle Anwendungsfälle im Viewconfiguration-Mapper, die eine Kombination aus Viewconfig-Element, Suche und/oder Skript erfordern.

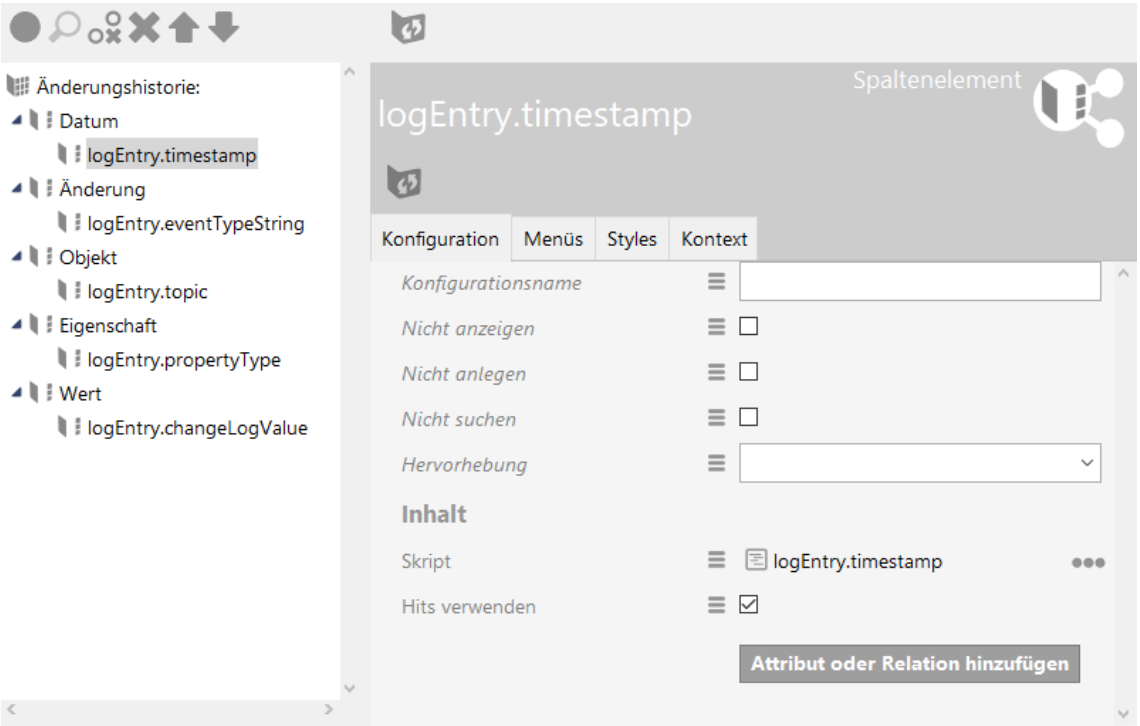
3.6.1 Anzeigen einer Änderungshistorie im Web-Frontend

Voraussetzung: Eingerichtete Änderungshistorien-Aufzeichnung

Damit Änderungen an Elementen aufgezeichnet werden, muss ein Metaattribut mit dem internen Namen „changeLog“ vom Wertetyp „Zeichenkette“ eingerichtet werden. Siehe hierzu Kapitel „ChangeLog Trigger“.

View-Konfiguration

Die View-Konfiguration von ChangeLog-Einträgen für das Web-Frontend kann via Viewconfiguration-Mapper in Form einer Tabelle umgesetzt werden:



Aus der Änderungshistorie können Werte wie Datum, Änderung, betroffenes Element, geänderte Eigenschaften etc. ausgelesen werden.

Für jede dieser Werte ist eine Spaltenkonfiguration anzulegen, die jeweils als Spaltenelement ein Skript enthält. Das Skript verarbeitet die Einträge gemäß der Klasse $k.HistoryChangeLogEntry und gibt den jeweiligen Wert nach Wertetyp gefiltert für das Spaltenelement zurück (zur Syntax siehe Java-Script API). Für Skript-Beispiele siehe nachfolgende Abschnitte.

Da für jedes zu protokollierende Wissensnetzelement vom ChangeLog-Attributtyp nur ein einzelnes Attribut-Element generiert wird, werden sämtliche Einträge der Änderungshistorie als Attributwert in die Zeichenkette geschrieben. Daher müssen die Einträge der Zeichenkette mithilfe des Skripts einzeln ausgelesen werden.

Damit die Einträge überhaupt verfügbar sind, muss die Option „Hits verwenden“ aktiv (ange-

hakt) sein. Für mehr Informationen hierzu siehe Kapitel „Inhaltsmodell 'Hit'".

Zu beachten:

1. Die ChangeLog-Einträge können in der Viewkonfiguration wie die „Hits" einer Abfrage behandelt werden. Wenn die Option „Hits verwenden" nicht aktiviert wird, erfolgt eine eigenschaftslose Ausgabe des übergeordneten Elements ohne zugehörigen ChangeLog-Eintrag (Ergebnis: Leere Spaltenelemente in der Anzahl der ChangeLog-Einträge).
2. Damit die View-Konfiguration der Tabelle mitgeteilt bekommt, zu welchem Attribut die ChangeLog-Einträge angezeigt werden sollen, ist eine Beeinflussung durch eine andere View notwendig, anhand derer das Kontextelement an die Tabelle weitergereicht wird.

Die Ausgabe-Tabelle im Web-Frontend sieht wie folgt aus:

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 diesem Beispiel wurde ein Objekt namens "Roadster" erstellt, eine Relation "hat Ausstattung" zum Objekt "Verdeck" gezogen, dann "Roadster" in "Convertible" umbenannt und schließlich wurde das Objekt nach "Cabriolet" umbenannt. In der Spalte "Objekt" wird aufgrund des Skriptes in jeder Zeile der *aktuelle* Name des Objektes dargestellt, an dem die Änderungen vorgenommen wurden.

Skript-Beispiele für die ChangeLog-Ausgabe

Änderungsdatum

```
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))
}
```

Änderung

```
function cellValues (logEntry, queryParameters) {
    return [ logEntry.eventTypeString() ]
```

```
}

function filter (elements, queryParameters, columnSearchValue) {
    return elements
}
```

Objekt

```
function cellValues (logEntry, queryParameters) {
    return [ logEntry.topic() && logEntry.topic().name() ]
}

function filter (elements, queryParameters, columnSearchValue) {
    return elements
}
```

Eigenschaft

```
function cellValues (logEntry, queryParameters) {
    return [ logEntry.propertyType().name() ]
}

function filter (elements, queryParameters, columnSearchValue) {
    return elements
}
```

Wert

```
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
}
```

3.7 Installation

- ViewConfig-Mapper als ZIP-Datei über Static-REST-Ressource bereitstellen
- Verweis auf VCM-Demo mit Bezugsmöglichkeit (Link)
- Über (andere) Web-Server
- Produktivbetrieb/ Testbetrieb

3.8 Customized project

3.8.1 IDE

- Node.js/Webpack/etc.

3.8.2 Technical details

- Schaubild Informationsfluss bei Aktionen
- Komponenten-State

4 k-infinity services

4.1 Overview

4.1.1 Command line parameters

Falls es zu einem Aufrufparameter auch einen Eintrag der Ini-Datei gibt, hat der der Aufrufparameter höhere Priorität.

`-inifile <Dateiname>, -ini < Dateiname >`

Name der Ini-Datei, die statt dem Standard-Ini-Datei verwendet wird.

4.1.2 Configuration file

Einige Einstellungen können über eine Konfigurationsdatei (*.ini) festgelegt werden. Der Aufbau der Datei sieht folgendermaßen aus:

```
[Default]
parameterName1=parameterWert1
parameterName2=parameterWert2
...
```

Im folgenden sind Konfigurationen aufgeführt, die für jeden Dienst verwendet werden können. Für dienstspezifische Einstellungen siehe den Abschitt "Konfigurationsdatei" des entsprechenden Dienstes.

Logging-Einstellungen

`loglevel = <LogLevel>`

Konfiguriert, welche Meldungen im Log erscheinen sollen:

- FATAL ERROR: Nur kritische Fehlermeldungen
- ERROR: Nur Fehlermeldungen
- WARNING: Nur Warnungen und Fehlermeldungen
- NORMAL (Standardwert): Alle Meldungen außer Debug-Ausgaben
- NOTIFY: Alle Meldungen inklusive einiger Debug-Ausgaben
- DEBUG: Alle Meldungen inklusive aller Debug-Ausgaben

`debug = true/false`

Veraltet. Setzt den Log-Level bei true auf DEBUG, bei false auf NORMAL. Wird nur noch ausgewertet, wenn logLevel nicht gesetzt wird

`nolog = true/false`

Veraltet. Entspricht bei true einem logtargets=null. Wird nur noch ausgewertet, wenn logtargets nicht gesetzt wird

`channels = <Channel1> [,<Channel2>,...]`

Namen von Channelfiltern. Mit Hilfe der Channelfilter werden nur Log-Meldungen ausgegeben, die zu den angegebenen Channelfiltern gehören. Der Name eines Channelfilters deutet darauf hin, zu welchem Themengebiet die Log-Ausgaben gehören. Welche Channelfilter möglich sind, erfährt man in der Kommandozeile mit Hilfe des Parameters -availableChannels.

`channelLevels = <Channel1>:<Level1> [,<Channel2>:<Level2>,...]`

Gezielte Konfiguration des Loglevels für den jeweiligen Channel.

`logTargets = <Name1> [,<Name2>,...]`

Namen von Log-Targets. Für die Konfiguration siehe Abschnitt „Log-Targets“.

`logprefix = <Prefix1> [, <Prefix2>,... ]`

Zusätzliche Daten, die bei jeder Log-Ausgabe hinzugefügt werden:

- pid : Prozess-ID der Anwendung
- $proc$: ID des aktuellen Smalltalk-Threads
- $alloc$: belegter Speicher der VM (in Megabyte)
- $free$: Freier Speicher der VM (in Megabyte)
- $incGC$: Status inkrementelle GCs
- os : Information über OS

- cmd : Kommandozeile
- $build$: Build-Version
- $coast$: COAST-Version

Bei einem Präfix, der nicht in dieser Liste enthalten ist, wird der Präfix unverändert ausgegeben.

`logTimestampFormat = <FormatString>`

Formatierungsangabe für den Timestamp des Log-Eintrags, z.B. "hh:mm:ss".

`exceptionLogSize = <Integer>`

Setzt die maximale Größe des bei einer Fehlermeldung mitgelieferten StackTrace.

Log-Targets

Über Log-Targets lassen sich verschiedene Ziele für das Logging festlegen, für die sich jeweils Log-Level, Channels, Formatierung und mehr konfigurieren lassen. Für jeden angegebenen Namen aus der logtargets-Liste muss eine Konfiguration im Abschnitt [<Konfigurationsname>] angegeben werden:

```
[Default]
logTargets=errorausgabe

[errorausgabe]
type=stderr
format=json
loglevel= ERROR
```

konfiguriert zum Beispiel eine Ausgabe aller Fehlermeldungen im JSON-Format auf dem Standard-Error-Stream.

Eine Ausnahme stellt das Log-Target null dar: bei einer Konfiguration von logtargets=null muss kein Konfigurationsabschnitt erstellt werden. Fehlt dieser, so ist dies gleichbedeutend mit folgender Konfiguration

```
[Default]
logTargets=null

[null]
type=null
```

Es ist jedoch möglich, null als Bezeichner für eine beliebige Log-Target Konfiguration zu verwenden.

Grundsätzlich lassen sich genau wie in der allgemeinen Konfiguration loglevel, debug, channels, channelLevels, logprefix und logTimestampFormat festlegen (siehe oben). Die Konfiguration am Log-Target hat immer Vorrang, wenn keine angegeben ist, wird auf die allgemeine Konfiguration zurückgegriffen.

Zusätzlich gibt es noch einige weitere Konfigurationsmöglichkeiten:

`format = <Format>`

legt das Ausgabeformat fest. Mögliche Werte sind:

- **plain**: Die Standardformatierung in möglichst menschenlesbarer Form
- **json**: einzeilige Ausgabe als JSON-String, vor allem für Maschinenverarbeitung

`type = <Zieltyp>`

legt den Typ der Ausgabe fest. Diese Konfiguration MUSS angegeben werden, sonst wird das Log-Target ignoriert. Im folgenden sind Beschreibung und weitere Konfigurationsmöglichkeiten der verschiedenen Typen angegeben:

file

Ausgabe in eine Log-Datei.

`file = <Dateiname>`

legt den Dateinamen der Ziel-Datei fest.

`maxLogSize = <size>`

Die maximale Größe des Logfiles, ab der die alte Logdatei archiviert wird und eine neue geschrieben wird. Bei Werten kleiner als 1024 wird die Angabe als in MB verstanden.

`maxBacklogFiles = <amount>`

Die maximale Anzahl an archivierten Logdateien. Beim Anbruch einer neuen wird die älteste gelöscht.

transcript

Ausgabe in das Transcript, kann außerdem in eine Log-Datei umgeleitet werden und akzeptiert daher die gleichen Konfigurationen wie **file**.

stdout

Ausgabe auf den Standard-Out-Stream.

stderr

Ausgabe auf den Standard-Error-Stream.

mail

Versendet die Log-Ausgabe per Mail.

```
[errorMail]
type = mail
loglevel =  ERROR
;Absender-Adresse:
sender = mail@example.org
;Empfänger-Adresse:
recipient = rec@example.org
;Mail-Server:
smtpHost = stmp.example.org
;Port des Mail-Servers:
smptPort = 465
;Aktiviert bei true die gesicherte Verbindung (TLS/SSL).
;Bei true muss username und password gesetzt sein.
tls = true
```

```
username = mail@example.org
password = 12345abc
;Anzahl der Versuche, die Mail bei einem Fehlschlag erneut zu senden:
retries = 3
;Wartezeit zwischen den Versuche in Sekunden:
retryDelay = 5
```

mailfile

Wie mail, allerdings werden Ausgaben mit niedrigem Log-Level zunächst angesammelt und erst per Mail versendet, wenn ein Eintrag mit hohem Level geloggt wird.

`mailSendLevel = <LogLevel>`

setzt das Log-Level, ab dem die Mail gesendet wird.

syslog

Ausgabe als UDP-Datagramm an einen Syslog-Client.

`format = <Format>`

Anders als bei anderen Log-Targets werden json und plain als Formatierung nicht unterstützt, stattdessen kann hier die Syslog-Version angegeben werden:

- **rfc5424**: Formatiert die Nachricht nach RFC 5424. Die meisten Daten werden strukturiert im Structured-Data-Feld hinterlegt. Nur die eigentliche Log-Message wird im Message-Feld übertragen.
- **rfc3164**: Formatiert die Nachricht nach RFC 3164. Da dieser Standard kein Structured-Data-Feld hat, werden die entsprechenden Daten in der gleichen Formatierung an den Anfang des Message-Felds gestellt. Achtung: Der Zeitstempel ist standardkonform in Lokalzeit des sendenden Rechners angegeben.

`facility = <Integer>`

Die Facility als Ganzzahl. Für detailierte Informationen siehe https://tools.ietf.org/html/rfc5424#section-6.2.1

`targetHostname = <Hostname>`

Der Hostname des Zielsystems. Falls nicht angegeben wird localhost verwendet.

`targetPort = <Integer>`

Der Port, an den gesendet werden soll. Falls nicht angegeben, wird der Syslog-Standard-Port 514 verwendet.

`hostname = <Hostname>`

Der Hostname des Senders. Falls nicht angegeben, wird des Hostname des Systems ausgelesen.

`appname = <Name>`

Name der sendenden Anwendung. Falls nicht angegeben, wird der Name der EXE verwendet.

`maxMessageSize = <Integer>`

Die maximale Nachrichtengröße in Bytes. Falls nicht angegeben, wird die maximale Größe für UDP verwendet. Zum Kürzen der Nachricht wird zunächst stückweise Structured Data entfernt und im Notfall die Message abgeschnitten. Die Nachricht ist auch nach der Kürzung in validem Syslog-Format.

null

Zum Unterdrücken der Logausgaben. Es werden keinerlei Optionen ausgelesen.

4.1.2.1 Text extraction

Für die Extraktion von Texten und Metadaten aus Dateiinhalten muss die Verwendung von Apache-Tika eingerichtet werden:

- Von der Webseite http://tika.apache.org/ die aktuelle tika-app (z.B. tika-app-1.18.jar) herunterladen und ins Verzeichnis des jobclients legen.
- Die Konfigurationsdatei (z.B. jobclient.ini oder bridge.ini) um folgenden Eintrag ergänzen:

```
[text-extraction]
tikaJavaParams=-Xmx1024M
tikaJarPath=tika-app-1.18.jar
; Optional: Maximale Größe der Binärdateien,
;  für die eine Textextraktion durchgeführt wird
; extractedTextSizeLimit=100000
;
; Optional: Java-Pfad, Standardwert ist 'java'
; extractorPath=C:\Program Files\Java\jdk-9\bin\java.exe
```

4.2 Mediator

4.2.1 Overview

Der i-views-Server sorgt für konsistente und persistente Datenhaltung und für die Aktualität der Daten auf den angeschlossenen i-views-Clients.

Die Datenhaltung erfolgt in einer objektorientierten Datenbank, die durch ein optimistisches Transaktionssystem kooperatives Arbeiten auf dem Wissensnetz ermöglicht.

In seiner Funktion als Kommunikationszentrale sorgt der i-views-Server für die Synchronisation von Clients und Services. Als Basismechanismus stellt er hierfür einen geteilten Objektraum und aktive Updates zur Verfügung.

Technische Daten:

- Multi-Platform Executable auf Basis der VisualWorks Smalltalk Virtual Machine (mediator.exe bzw. mediator.im).
- Konfigurierbarer TCP/IP Server-Port für die Kommunikation mit den Clients, Standard bei i-views 5.1 ist 30064.

Der i-views-Server kann in drei Modi betrieben werden:

1. Klassisch / kompakt: In diesem Modus startet der Server als einzelner Prozess - dem sogenannten "mediator".
2. Multiprozess: In diesem Modus startet der Server mindestens zwei Prozesse. Der Speicherverbrauch ist dadurch höher als im kompakten Betrieb, aber viele Aufgaben können parallelisiert werden.
3. Verteilt: In diesem Modus müssen die Server-Komponenten "stock" und "dispatcher" separat konfiguriert und betrieben werden. Es ist dann möglich, die Server-Komponenten auf unterschiedliche Rechnerknoten zu verteilen.

4.2.2 System requirements

Der i-views-Server ist Plattform-unabhängig und läuft auf allen gängigen Betriebssystemen, z. B. Windows und Linux. Andere Systeme auf Anfrage.

OS	Version	Prozessor	Unterstützt	64 Bit VM
Windows	alle aktuell von Microsoft unterstützen Versionen, Server und Client	x86	ja	ja
Linux	RedHat, SLES, u.a. (Kernel >= 2.4, glibc >= 2.5)	x86	ja	ja
	Kernel >= 2.6, glibc >= 2.5	PPC	nein	
		ARM	nein	
Mac	OSX 10.9+	x86	ja	ja

4.2.3 Betriebsmodi

Folgende Startparameter unterscheiden zunächst grundsätzlich über den Modus, in dem der Server gestartet wird. Ohne Parameter starter der Server im kompakten "mediator"-Modus.

`-stock`

Startet die Serverkomponente "Stock", die für die persistente Datenhaltung verantwortlich ist.

`-dispatcher`

Startet die Serverkomponente "Dispatcher", die für die Synchronisation der Clients bzw. für die Verteilung der "active updates" verantwortlich ist.

`-server`

Startet den vollständigen Server im Multiprozess-Modus.

4.2.3.1 Multi-Process Mode (-server)

Mit dem Startparameter **-server** wird automatisch ein Stock und ein Dispatcher gestartet. Der Dispatcher öffnet einen Server auf dem Standard-Port (30064). Der Port des Stock wird automatisch ausgewählt. Authentifizierungs-Tokens zwischen den beiden Prozessen werden automatisch erzeugt und müssen nicht konfiguriert werden.

Achtung: Es ist wichtig, dass alle Clients (Knowledge-Builder, Bridge, BatchTool etc) Zugriff auf Stock und Dispatcher haben.

Falls ein dies nur für bestimmte Ports möglich ist, muss eine explizite Konfiguration von Stock und Dispatcher erfolgen. Es werden die gleichen Konfigurations-Dateien im lokalen Verzeichnis verwendet wie im echten verteilten Modus

- **dispatcher.ini** konfiguriert den Dispatcher-Prozess
- **stock.ini** konfiguriert den Stock-Prozess

Es ist aktuell nicht möglich, andere Konfigurations-Dateien zu verwenden.

4.2.3.2 Stock-Configuration

Der Stock ist für die Speicherung der Daten auf der Festplatte verantwortlich. Ein einfaches Beispiel für die Konfiguationsdatei **stock.ini** ist

```
[Default]

interfaces=cnp://0.0.0.0:4998
```

Diese Konfiguration sorgt dafür, dass der Stock auf Port 4998 horcht und über das Native Coast-Protokoll kommuniziert.

Die Konfigurations-Datei kann folgende Einträge enthalten:

```
[Default]
parameterName1=parameterWert1
parameterName2=parameterWert2
...
```

Folgende Parameter sind an dieser Stelle einsetzbar:

`port=<portnummer>`

Startet den Stock mit der Portnummer <num>. Ohne diese Angabe wird Port 30064 verwendet.

Dieser Parameter ist veraltet. Er wird durch den Parameter "interfaces" ersetzt. Dabei entspricht ein Eintrag von "port=1234" dem Eintrag "interfaces=cnp://0.0.0.0:1234". Im Unterschied zum Startparameter sind hier mehrere Werte zulässig, die, durch Komma getrennt, hintereinander aufgezählt werden.

`interfaces=<interface-1>,<interface-2>,...<interface-n>`

Dieser Parameter bestimmt, unter welchen Adressen und Protokollen der Server erreichbar

ist. Mehrere Werte sind zulässig, und werden durch Komma getrennt. Mögliche Protokolle sind: http, https, cnp, cnps. Dabei steht "cnp" für "Coast Native Protocol" bzw. "Coast Native Protocol Secure". Der syntaktische Aufbau einer Interface-Definition entspricht der einer URL mit Schema, Host und Port. Über den Host-Teil steuert man, über welche Netzwerkadresse(n) der Server erreicht werden kann. z.B: "0.0.0.0"=IPv4 alle Interfaces, "[::1]"=IPv6 nur Loopback.

Die Protokolle "http" und "https" lassen sich über Proxies umleiten, so dass man den Server bspw. über einen auf Port 443 laufenden IIS erreichen kann.

`baseDirectory=<Verzeichnis>`

Setzt das Verzeichnis, in dem sich das Verzeichnis volumes befindet. Sollte dieser Wert auf volumes enden, so wird dieses Verzeichnis direkt verwendet, ohne noch zusätzlich darunter ein Verzeichnis volumes anzulegen.

`volumesDirectory=<Verzeichnis>`

In diesem Verzeichnis liegen die Wissensnetze. Als Standardwert ist an dieser Stelle 'volumes' eingetragen.

`backupDirectory=<Verzeichnis>`

Gibt das Verzeichnis an, in welches Wissensnetz-Backups geschrieben und zum Wiederherstellen auch gelesen werden. Nur vollständige Verzeichnisnamen erlaubt, keine Relativ-Angabe.

`networkBufferSize=<Größe in Bytes>`

Gibt die Größe des Puffers an, der für das Senden/Empfangen von Daten verwendet wird. Der Standardwert ist 20480. In manchen Infrastrukturen kann man durch Angabe von

networkBufferSize=4096

einen höheren Durchsatz erreichen.

`flushJournalThreshold=<Anzahl der Cluster>`

gibt den Maximalwert an, den "veränderte Cluster" + "Indexcluster" in einem Speichervorgang erreichen dürfen. Wenn der Wert für "veränderte Cluster" bereits überschritten ist, werden keine "Indexcluster" gespeichert sondern diese werden mit Journal geführt.

Ein niedriger Wert (z.B. 50) garantiert schnelle Speicherzeiten, baut aber potentiell ein großes Journal auf.

Ein Wert von "0" deaktiviert Journaling. Standardwert ist "2000".

Anmerkung: Ein "flush" des Journals wird immer spätestens bei einer vollständigen Speicherung ausgeführt. Diese wiederum wird ausgelöst, wenn:

- der Mediator beendet wird
- der letzte Client des entsprechenden Volumes abgemeldet wird
- eine Speicherung durch einen full-save-job (siehe jobs.ini) ausgelöst wird

`autoSaveTimeInterval=<Warteintervall in Sekunden>`

gibt an in Sekunden, wie lang nach dem letzten Cluster-Speichern maximal gewartet wird, bis wieder automatisch gespeichert wird. Standardwert ist 15.

`clientTimeout=<Timeout in Sekunden>`

gibt die Zeit in Sekunden an, die ein verbundener Client maximal keine Alive-Nachricht geschickt haben darf, bevor der Mediator ihn als inaktiv erachtet und ihn ausschließt.

```
password.flavour=190133293071522928001864719805591376361
password.hash=1119954518245866070549559980205262417173496579142708063869495424703551323984 4
```

Das Mediator-Passwort wird zusammen mit einem zufälligem flavour zu einen (SHA256) hashwert berechnet. Diese beiden Informationen genügen dann, um dem Mediator eine Authentifizierungsanfrage zu überprüfen. Beim Authentifizieren am Server muss der Benutzername mit "Server:admin" angegeben werden. Um diese Werte zu ermitteln kann man mit

```
password.update=neues_passwort
```

den Server veranlassen, beim Start einen neuen Flavour und einen passenden Hashwert zu berechnen und in die ini-Datei zurückzuschreiben. Der Eintrag password.update wird dabei entfernt.

```
password=<String>
```

Die veraltete, noch unterstützte Art, das Mediator-Passwort zu setzen. Diese Variante darf nicht gleichzeitig mit der SHA256-Hash Variante verwendet werden.

Geändert

```
skipVolumesCheck=<true|false>
```

gibt an, ob die normalerweise nach dem Start des Mediators durchgeführte Überprüfung der vorhandenen Volumes ausgelassen wird

Geändert

Logging-Einstellungen:

Zu den Konfigurationsmöglichkeiten des Loggings siehe Logging-Einstellungen im Kapitel 11.1.2 Konfigurationsdatei.

Speicher-Einstellungen:

Die folgenden drei Parameter dienen zur Konfiguration der Speicherzuteilung und -nutzung. Erlaubt ist die Angabe von Werten entweder in Megabyte oder in tatsächlichen Byte, wobei die Annahme gilt, dass sich Werte kleiner als 1048576 auf Megabyte-Angaben beziehen.

```
maxMemory=<Integer, in MB>
```

Maximal erlaubte Hauptspeicherbelegung. Minimal 50 MB, standardmäßig gesamter physikalisch vorhandener Hauptspeicher (unter Windows) bzw. 512 MB.

```
baseMemory=<Integer, in MB>
```

Hauptspeicherbelegung ab der verstärkt versucht wird, Speicher freizugeben. Standardmäßig 0.6 * maxMemory. (alias: "growthRegimeUpperBound")

```
freeMemoryBound=<Integer, in MB> [10]
```

Falls belegter, aber nicht mehr benötigter Speicher diese Grenze überschreitet, wird er

wieder freigegeben.

BLOB-Service-Konfiguration

Soll der Mediator mit einem integrierten BLOB-Service gestartet werden, damit die BLOBs getrennt von der Datenbank auf der Festplatte gespeichert werden, so muss die folgende Einstellung in der "mediator.ini" eingetragen werden:

```
startBlobService=true
```

Nähere Informationen dazu finden Sie in der Dokumentation des BLOB-Service (siehe Link unten).

4.2.3.3 Dispatcher-Configuration

Der Dispatcher ist verantwortlich für Transaktionssteuerung und Koordination mehrere Clients. Eine einfacher Konfigurations-Datei ist

```
[Default]

interfaces=cnp://0.0.0.0:5000

stockAddress=cnp://localhost:4998
stockAuthentication=dsfkhvqw3n9485z432504
```

Diese Konfiguration öffnet einen Server auf Port 5000 zu dem sich Clients verbinden können. Den Stock sucht der Dispatcher unter localhost:4998. Diese Adresse ist auch die Adresse, die von den Clients verwendet wird um Daten vom Stock zu

Falls Dispatcher und Stock auf dem gleichen Server laufen, teil der Dispatcher seinen Clients eigenen Hostname mit, damit auch Verbindungen über das Netzwerk funktionieren.

Für die Authentifizierung des Dispatchers beim Stock wird das Token dsfkhvqw3n9485z432504 verwendet. Dieses Token muss in der Stock-Konfiguration über die "password.*"-Schlüssel eingestellt sein.

4.2.4 Installation

Der i-views-Server benötigt prinzipiell keine spezielle Installation, d.h. er ist ad hoc aus einem beliebigen Verzeichnis startbar.

Es ist dabei darauf zu achten, dass die notwendigen Zugriffsrechte (lesen/schreiben/erzeugen) für das Arbeitsverzeichnis des Servers und alle Unterverzeichnisse gesetzt sind.

4.2.4.1 start parameters

Dem Mediator Prozess können beim Start diverse Parameter mit übergeben werden. Die meisten Parameter können aber auch in der mediator.ini angegeben werden, sodass man

den Mediator mit einer einfachen Kommandozeile starten kann. Dabei gilt, dass auf der Kommandozeile angegebene Parameter Präzedenz vor evtl. doppelt in der .ini-Datei angegebenen Parametern haben.

Die komplette Liste der möglichen Startparameter gibt der Mediator beim Aufruf mit dem Parameter "-?" aus.

`-interface <interface-1>`

Dieser Parameter bestimmt, unter welchen Adressen und Protokollen der Server erreichbar ist. Mögliche Protokolle sind: http, https, cnp, cnps. Dabei steht "cnp" für "Coast Native Protocol" bzw. "Coast Native Protocol Secure". Der syntaktische Aufbau einer Interface-Definition entspricht der einer URL mit Schema, Host und Port. Über den Host-Teil steuert man, über welche Netzwerkadresse(n) der Server erreicht werden kann. z.B: "0.0.0.0"=IPv4 alle Interfaces, "[::1]"=IPv6 nur Loopback.

Die Protokolle "http" und "https" lassen sich über Proxies umleiten, so dass man den Server bspw. über einen auf Port 443 laufenden IIS erreichen kann.

`-clientTimeout <sec>`

Setzt die Zeit, innerhalb der sich ein Client automatisch melden muss, auf <sec> Sekunden. Der Wert sollte mindestens auf 600 gesetzt werden (was auch der Standardwert ist).

`-baseDirectory <directory>`

Setzt das Verzeichnis, in dem sich das Verzeichnis "volumes" befindet. Neben dem Unterverzeichnis "volumes" werden hier auch standardmäßig die Verzeichnisse für Backups und Downloads angelegt. Dieser Parameter hieß früher "-volumes".

Die folgenden Parameter stellen Kommandos an das Mediator Executable, um bestimmte Aufgaben auszuführen, ohne danach als Server für Wissensnetze zu fungieren.

`-quickRecover <volume> -recover <volume>`

Falls der Mediator unordnungsgemäß beendet wird (z.B. Absturz des Rechners), bleiben in Volumes, die in Benutzung waren, Lock-Dateien stehen. Das Volume kann dann nicht mehr betreten werden. Um das Lock aufzuheben, kann man mit dem Aufruf von -quickRecover <volume> das Lock entfernen. Der Aufruf schlägt fehl, wenn (mögliche) Inkonsistenzen gefunden wurden. In diesem Fall muss der Startparameter -recover verwendet werden.

Achtung:

Das Arbeitsverzeichnis beim Aufruf muss hier das Verzeichnis sein, welches das „volumes“-Verzeichnis enthält. Der „-volumes“-Parameter wirkt hier also nicht.

`-bfscommand <volume> <command>`

Führt Kommandos aus, die vom BlockFileSystem erkannt werden.

Kommandozeilen-Parameter für das Logging:

`-nolog`

Schaltet Logging ab

`-loglevel <Integer>`

Konfiguriert, welche Meldungen im Log erscheinen sollen:

- 0: Alle Meldungen inklusive Debug-Ausgaben

- 10 (Standardwert): Alle Meldungen außer Debug-Ausgaben
- 20: Nur Warnungen und Fehlermeldungen
- 30: Nur Fehlermeldungen

`-logfile <Dateiname>, -log <Dateiname>`

Name der Log-Datei, die statt der Standard-Log-Datei verwendet wird. Dieser Parameter muss auf jeden Fall verändert werden, wenn mehrere Clients im selben Arbeitsverzeichnis gestartet werden sind.

`-debug`

Schaltet das Logging auf debug-mode

`-log <logname>`

Setzt die Logdatei auf <logname>.

4.2.4.2 Configuration file "mediator.ini"

Einige Mediator-Einstellungen können auch in der Konfigurationsdatei mediator.ini festgelegt werden. Der Aufbau der Datei sieht folgendermaßen aus:

```
[Default]
parameterName1=parameterWert1
parameterName2=parameterWert2
...
```

Folgende Parameter sind an dieser Stelle einsetzbar:

Netzwerkkommunikation

`port=<portnummer>`

Startet den Server mit der Portnummer <num>. Ohne diese Angabe wird Port 30061 verwendet.

Dieser Parameter ist veraltet. Er wird durch den Parameter "interfaces" ersetzt. Dabei entspricht ein Eintrag von "port=1234" dem Eintrag "interfaces=cnp://0.0.0.0:1234". Im Unterschied zum Startparameter sind hier mehrere Werte zulässig, die, durch Komma getrennt, hintereinander aufgezählt werden.

`interfaces=<interface-1>,<interface-2>,...<interface-n>`

Dieser Parameter bestimmt, unter welchen Adressen und Protokollen der Server erreichbar ist. Mehrere Werte sind zulässig, und werden durch Komma getrennt. Mögliche Protokolle sind: http, https, cnp, cnps. Dabei steht "cnp" für "Coast Native Protocol" bzw. "Coast Native Protocol Secure". Der syntaktische Aufbau einer Interface-Definition entspricht der einer URL mit Schema, Host und Port. Über den Host-Teil steuert man, über welche Netzwerkadresse(n) der Server erreicht werden kann. z.B: "0.0.0.0"=IPv4 alle Interfaces, "[::1]"=IPv6 nur Loopback.

Die Protokolle "http" und "https" lassen sich über Proxies umleiten, so dass man den Server bspw. über einen auf Port 443 laufenden IIS erreichen kann.

Für ssl-Kommunikation (cnps:// bzw. https://) müssen in der Konfigurationsdatei zusätzlich die Dateipfade für Zertifikat und Private Key angegeben werden:

`certificate=`*`Name der .crt-Datei`* `privateKey=`*`Name der .key-Datei`*

Verzeichnisse

`baseDirectory=<Verzeichnis>`

Setzt das Verzeichnis, in dem sich das Verzeichnis volumes befindet. Sollte dieser Wert auf volumes enden, so wird dieses Verzeichnis direkt verwendet, ohne noch zusätzlich darunter ein Verzeichnis volumes anzulegen.

`volumesDirectory=<Verzeichnis>`

In diesem Verzeichnis liegen die Wissensnetze. Als Standardwert ist an dieser Stelle 'volumes' eingetragen.

`backupDirectory=<Verzeichnis>`

Gibt das Verzeichnis an, in welches Wissensnetz-Backups geschrieben und zum Wiederherstellen auch gelesen werden. Nur vollständige Verzeichnisnamen erlaubt, keine Relativ-Angabe.

`networkBufferSize=<Größe in Bytes>`

Gibt die Größe des Puffers an, der für das Senden/Empfangen von Daten verwendet wird. Der Standardwert ist 20480. In manchen Infrastrukturen kann man durch Angabe von

networkBufferSize=4096

einen höheren Durchsatz erreichen.

`journalMaxSize=<Maximale Größe des Journals>`

Mit journalMaxSize=0 kann man das normalerweise aktive Journalling deaktivieren. Der Standardwert ist 5242880 (5 MB).

autoSaveTimeInterval=<Warteintervall in Sekunden>

gibt an in Sekunden, wie lang nach dem letzten Cluster-Speichern maximal gewartet wird, bis wieder automatisch gespeichert wird. Standardwert ist 15.

`clientTimeout=<Timeout in Sekunden>`

gibt die Zeit in Sekunden an, die ein verbundener Client maximal keine Alive-Nachricht geschickt haben darf, bevor der Mediator ihn als inaktiv erachtet und ihn ausschließt.

```
password.flavour=190133293071522928001864719805591376361
password.hash=111995451824586607054955998020526241717349657914270806386949542470355132398 44
```

Das Mediator-Passwort wird zusammen mit einem zufälligem flavour zu einen (SHA256) hashwert berechnet. Diese beiden Informationen genügen dann, um dem Mediator eine Authentifizierungsanfrage zu überprüfen. Beim Authentifizieren am Server muss der Benutzername mit "Server:admin" angegeben werden. Um diese Werte zu ermitteln kann man mit

`password.update=neues_passwort`

den Server veranlassen, beim Start einen neuen Flavour und einen passenden Hashwert zu berechnen und in die ini-Datei zurückzuschreiben. Der Eintrag password.update wird dabei entfernt.

`password=<String>`

Die veraltete, noch unterstützte Art, das Mediator-Passwort zu setzen. Diese Variante darf

nicht gleichzeitig mit der SHA256-Hash Variante verwendet werden.

Geändert

`skipVolumesCheck=<true|false>`

gibt an, ob die normalerweise nach dem Start des Mediators durchgeführte Überprüfung der vorhandenen Volumes ausgelassen wird

Logging

Zu den Konfigurationsmöglichkeiten des Loggings siehe Logging-Einstellungen im Kapitel 4.1.2 "Konfigurationsdatei".

Arbeitsspeicher

Die folgenden drei Parameter dienen zur Konfiguration der Speicherzuteilung und -nutzung. Erlaubt ist die Angabe von Werten entweder in Megabyte oder in tatsächlichen Byte, wobei die Annahme gilt, dass sich Werte kleiner als 1048576 auf Megabyte-Angaben beziehen.

`maxMemory=<Integer, in MB>`

Maximal erlaubte Hauptspeicherbelegung. Minimal 50 MB, standardmäßig gesamter physikalisch vorhandener Hauptspeicher (unter Windows) bzw. 512 MB.

`baseMemory=<Integer, in MB>`

Hauptspeicherbelegung ab der verstärkt versucht wird, Speicher freizugeben. Standardmäßig 0.6 * maxMemory. (alias: "growthRegimeUpperBound")

`freeMemoryBound=<Integer, in MB> [10]`

Falls belegter, aber nicht mehr benötigter Speicher diese Grenze überschreitet, wird er wieder freigegeben.

BLOB-Service-Konfiguration

Soll der Mediator mit einem integrierten BLOB-Service gestartet werden, damit die BLOBs getrennt von der Datenbank auf der Festplatte gespeichert werden, so muss die folgende Einstellung in der "mediator.ini" eingetragen werden:

```
startBlobService=true
```

Nähere Informationen dazu finden Sie in der Dokumentation des BLOB-Service (siehe Link unten).

4.2.4.3 Safety Concept of the mediator

Der i-views-Server ist eine generische Komponente, die nicht nur für i-views verwendet werden kann. Neben der Einschränkungen über die Authentifizierungen am Server oder in der Datenbank kann man auch kontrollieren, welche Anwendungen sich verbinden dürfen.

Jede Anwendung (Client und Server) enthält ein RSA-Schlüsselpaar, das je ausgelieferter Anwendung eindeutig ist. Den öffentlichen Schlüssel kann man über die Information erhalten (KB: Menü „Werkzeuge“, „Info“, dann die Schaltfläche „RSA-Key kopieren“) bzw. für Konsolen-Anwendungen per Aufruf mit dem Parameter -showBuildID. Die hierdurch exportierte Build-Information enthält den öffentlichen RSA-Exponenten (rsa.e_1) und RSA-Modul (aufgeteilt auf mehrere rsa.n_x) sowie eine MD5 Prüfsumme dieser Informationen (buildID).

Beispiel einer Build-Information:

```
[buildID.90A1203EFB957A58C2268AD8FE3CC5A3] rsa.n_1=93D516DF61395258AA21A91B33E8EE67 rsa.n_2=B07C6
```

Möchte man nun, dass sich nur eine bestimmte Menge Client-Anwendungen mit dem Server verbinden kann, so muss man im Server die jeweiligen Abschnitte in die mediator.ini übertragen. Beim Verbindungsaufbau überträgt der Client seine buildID. Wenn der Mediator einen passenden Eintrag enthält, so wird er die Client-Authentizität prüfen. Andernfalls wird er eine Verbindung nur aufbauen, wenn es gar keine Einträge zu Build-Informationen in seiner Ini-Datei gibt. Somit kann beispielsweise verhindert werden, dass sich veraltete Client-Anwendungen oder modifizierte Client-Anwendungen mit dem Mediator verbinden.

Umgekehrt können auch in der Client-Anwendung entsprechende buildIDs für die Mediatoren in die jeweilige ini-Datei eingetragen werden, um eine Verbindung zu einem kompromittierten oder veralteten Server zu verhindern.

So kann man eine Umgebung einrichten, in der nur mit der aktuellsten Software auf die Produktivdaten zugegriffen werden kann, aber auf die Server mit den Testdaten auch von einer Entwicklungsumgebung aus. Die Anwendersoftware wiederum kann nur auf den Produktivserver oder auf den Testserver zugreifen.

Konfiguriert man weder Server noch Client, so verhält sich die Installation wie in den Vorgängerversionen: Jede Anwendung kann sich mit jedem Server verbinden (sofern die Protokollversion übereinstimmt).

Seit der Version 5.4 des Servers benötigt man zum Durchführen administrativer Befehle das Server-Passwort als Parameter (über die Rest-Schnittstelle oder über die Verwaltung per Administrationswerkzeug). Für Aktionen, die sich auf eine existierende Datenbank beziehen (backup, download, garbage collection usw) genügt hierfür seit Version 6.2 eine Authentifizierung als Administrator im Volume.

Umgekehrt ist es mit dem Serverpasswort möglich, sich in einem Volume anzumelden. Details hierzu finden sich im Admin-Tool.

Ist am Server kein Passwort konfiguriert, so kann man sich mit einem beliebigen Passwort am Server anmelden. Die Anmeldung im Volume ist dann jedoch nicht möglich.

4.2.4.4 Audit Log

In einigen Anwendungsszenarien kann es gefordert sein, alle Zugriffe auf ein Wissensnetz in einem Zugriffs- oder Audit-Log zu protokollieren. Dieses Audit-Log enthält Einträge für alle An- und Abmeldevorgänge, schreibende und lesende Zugriffe auf Wissensnetz-Inhalte, gestellte Suchanfragen, Ausdrucke, Exporte etc.

Damit das Zugriffslog aktiviert werden kann, muss eine Konfigurationsdatei 'log.ini' im Verzeichnis des zu überwachenden Wissensnetzes angelegt werden:

```
[Default]
applicationLog=CoastJSONApplicationLogger

[CoastJSONApplicationLogger]
backupInterval=14
maxLogSize=5
```

Zusätzlich muss das Log im Admin-Tool in der Rubrik 'Systemkonfiguration / Audit-Log' aktiviert werden. Die Aktivierung bzw. Deaktivierung des Logs resultiert wiederum in einem Eintrag im Audit-Log.

Für die Auswertung der Zugriffslogs steht Administratoren eine Funktion im Administrations-Menue des Knowledge-Builders zur Verfügung.

4.2.5 Operation

4.2.5.1 Shut down of the server

Der i-views-Server lässt sich lokal durch das Strg-C Abbruchsignal herunterfahren.

Bei der Installation als Windows-Dienst muss der Server mit der Diensteverwaltung gestoppt werden.

Unter UNIX sowie beim Betrieb als Windows-Dienst, wird der Server beim Herunterfahren des Betriebssystems ordnungsgemäß beendet.

4.2.5.2 Storage and backup of knowledge networks

Verzeichnisstruktur

Das Basisverzeichnis des i-views-Servers weist folgende Struktur auf:

```
volumes/
   wissensnetzName/
     wissensnetzName.cbf
     wissensnetzName.cdr
     wissensnetzName.cfl
     wissensnetzName.lock (wenn das Wissensnetz geöffnet ist)

backup/
  wissensnetzName/
    <zehnstellige Nummer>/
      wissensnetzName.cbf
      wissensnetzName.cdr
      wissensnetzName.cfl
```

Speicherung von Wissensnetzen

Wissensnetze werden im Dateisystem im Unterverzeichnis "volumes" des Basisverzeichnisses des i-views-Servers abgelegt. In diesem Verzeichnis wird für jedes Wissensnetz ein Unterverzeichnis mit entsprechendem Namen angelegt. Eine Datei mit der Dateiendung '.lock' zeigt an, dass ein Wissensnetz gerade in Verwendung ist.

Backup von Wissensnetzen

Die Wissensnetzverzeichnisse dürfen auf keinen Fall direkt kopiert werden, so lange der Server läuft. Zu diesem Zweck besitzt der Server einen Backup-Service, der einen konsistenten Stand des Wissensnetzes in einen Backup-Bereich kopiert. Dieser Backup-Bereich muss in regelmäßigen Abständen gesichert werden (z.B. im Rahmen einer allgemeinen Backup-Strategie).

Der Ort, an dem die Backups angelegt werden kann über den Eintrag

`backupDirectory=<Verzeichnis>`

in der Datei "**mediator.ini**" festgelegt werden. Ohne diese Angabe wird das Unterverzeichnis

"backup" des Basisverzeichnisses verwendet.

Der Backup-Service des K-Infinity-Servers kann auf zwei Arten angestoßen werden:

1. Durch einen direkten Request an den Serverprozess (z.B. vom Administrationstool aus)
2. Durch Einträge in der Datei '**jobs.ini**' im Arbeitsverzeichnis des Servers. Diese Datei kann pro Wissensnetz eine Rubrik [Name_des_Netzes] mit folgenden Einträgen enthalten:

Beispiel jobs.ini

```
[volume1]
;Backup des Netzes "volume1"

;Uhrzeit, zu dem das Backup gestartet wird
backupTime=00:45

;Turnus in Tagen - hier täglich
backupInterval=1

;Die letzten 5 Backups dieses Wissensnetzes aufheben
backupsToKeep=5
```

'backupsToKeep' gibt die Anzahl der aufzuhebenden Backups an. Dies beinhaltet auch Backups, die manuell erstellt worden. Der Standardwert ist 3.

Bei der Angabe der Netznamen in eckigen Klammern ist die Verwendung der Platzhalter "*" und "?" erlaubt, Groß- und Kleinschrift wird ignoriert.

4.2.5.3 Garbage Collection

Ohne Garbage Collection wächst das Wissensnetz kontinuierlich mit der der Verwendung. Folglich ist es sinnvoll, von Zeit zu Zeit eine Bereinigung (Garbage Collection) durchzuführen. Wie die Datensicherung kann die Garbage Collection jederzeit manuell (z.B. mit einem speziellen Administrationswerkzeug) oder automatisch gestartet werden.

Die Garbage Collection kann - je nach Netzgröße - viel Zeit und Arbeitsspeicher in Anspruch nehmen. Bei der Durchführung auf großen Netzen empfiehlt es die Garbage Collection ohne verbundene Clients (z.B. KBuilder und JobClients) und ohne weitere aktive Prozesse (z.B. Backup) zu starten.

Automatische Garbage Collection: Aufbau der Datei jobs.ini

Die automatische Garbage Collection wird durch einen Eintrag in der Datei '**jobs.ini**' konfiguriert, z.B.

```
[volume1] garbageCollectTime=00:55 garbageCollectInterval=7
```

Dieser Eintrag in der jobs.ini sorgt dafür, dass das Netz mit Namen "volume1" im Abstand von "7" Tagen jeweils um "00:55" Uhr garbage-collected wird. Für das Intervall ist der Standardwert "1" (also täglich), der Zeitpunkt muss angegeben werden.

Bei der Angabe der Netznamen in eckigen Klammern ist die Verwendung der Platzhalter "*" und "?" erlaubt, Groß- und Kleinschrift wird ignoriert.

Manueller Start der Garbage Collection

Alternativ kann die Garbage Collection auch durch spezielle Aufrufparameter des i-views-Servers gesteuert werden:

-startGC <volume> -host <hostname>	Startet die Garbage Collection auf dem Netz mit Namen <volume> auf einen ggfs. entfernten Mediator auf Rechner <hostname> (optional inkl. Portangabe).
-stopGC <volume> -host <hostname>	beendet eine ggfs. auf dem Mediator <hostname> laufende Garbage-Collection des Netzes mit dem Namen <volume>.
-infoGC <volume> -host <hostname>	Informiert über den aktuellen Stand der Garbage Collection.

Diese Kommandos werden mit Hilfe eines Mediator-Executables an einen anderen bereits laufenden Mediator übermittelt.

Als weitere Möglichkeit bietet sich das Starten der Garbage Collection über das Admin-Tool an.

Zum Ausführen dieser Befehle muss mittels Parameter -password das richtige Serverpasswort übermittelt werden.

4.2.5.4 Unix operation

Unter UNIX reagiert der Server auf folgende Signale:

`SIGTERM/SIGHUP`

Beendet den Server

`SIGUSR2`

Der Server startet einen sofortigen Backup aller Wissensnetze, die in der jobs.ini-Datei für Backup spezifiziert sind (siehe auch Abschnitt über Backup).

4.2.5.5 Cluster operating

Der Mediator kann in einem Cluster betrieben werden. In einer Cluster-Umgebung wird i.d.R. eine laufende Spiegelung der Verzeichnisse und damit des Wissensnetzes vorgenommen. Fällt der Teil des Clusters, auf dem der Mediator läuft aus, wird automatisch ein neuer Mediator gestartet, der dann den Zugriff auf das Wissensnetz verwaltet.

Bei Ausfall des ersten Mediators kann es passieren, dass der Mediator keine Zeit mehr hat, das Wissensnetz in einen konsistenten Zustand zu bringen, das Netz damit eine Inkonsis-

tenz aufweist und die "lock"-Datei des alten Mediators noch im entsprechenden Verzeichnis bestehen bleibt. Damit der neue Mediator in der Lage ist, die "lock"-Datei zu löschen, muss folgender Parameter in die mediator.ini hinzugefügt werden.

```
host=NameDesClusters
```

In diesem Fall können alle Mediatoren mit diesem ini-Eintrag auch gesperrte Volumes anderer Mediatoren, die beim Start den selben Wert in der mediator.ini ausgelesen hatten, entsperren. "NameDesClusters" ist frei wählbar, muss aber den Regeln entsprechen, die für Hostnamen gelten (keine Leerzeichen, Doppelpunkte, o.ä.)

Eine Konsistenzprüfung des Volumes läuft beim Starten des Mediators automatisch ab. Soweit möglich, wird das Wissensnetz in einen konsistenten Zustand versetzt und der Betrieb läuft normal weiter.

4.2.5.6 Problem solving

Falls der i-views-Server während des Betriebs nicht ordnungsgemäß heruntergefahren wurde (z.B. Absturz des Rechners), bleiben bei geöffneten Wissensnetzen die Sperren bestehen. Beim Öffnen eines gesperrten Wissensnetzes wird diese Sperre erkannt und - falls möglich - entfernt.

Falls der Mediator eine Inkonsistenz erkennt, kann in der Kommandozeile durch den Aufruf des Mediators mit den Parametern -quickRecover / -recover das Wissensnetz geprüft und Inkonsistenzen soweit möglich repariert werden.

Sollte eine Auflösung der Inkonsistenzen wider Erwarten nicht möglich sein, muss auf eine Sicherungskopie zurückgegriffen werden.

4.2.5.7 Commands of the BlockFileSystems

Die Befehle hinter -bfscommand ermögliche Oerationen auf dem BlockFileSystem und sind für Supportfälle vorgesehen. Ein solcher Befehl könnte zBsp so aussehen:

```
-bfscommand quickCheck {target volume}
```

Die mit {target volume} addressierte Datenbank wird einer schnellen Strukturanalyse unterzogen. Analog kann mit deepCheck eine komplettanalyse ausgeführt werden.

4.3 Bridge

4.3.1 Overview

Die Bridge ermöglicht den externen Zugang zu Wissensnetzen auf drei Arten/Betriebsmodi:

- Über eine RESTful Services-Architektur (REST-Bridge). Die Schnittstelle steht als HTTP- oder HTTPS-Version zur Verfügung (KHTTPRestBridge)
- Über KEM-RPC (KEMBridge): Zugang über KEM. Falls Binärdaten im Wissensnetz gespeichert werden ist zusätzlich eine REST-Bridge erforderlich, die einen REST-Service mit

einem Blob-Resource-Handler bereitstellt.

- Betriebsmodus "Lastverteiler für andere Bridges" (KLoadBalancer).

ACHTUNG: KLoadBalancer und KEMBridge/KHTTPRestBridge dürfen nicht gleichzeitig in einer Bridge aktiviert werden, weil diese sich gegenseitig behindern.

Die Bridge und alle in ihr zu aktivierenden Zugänge lassen sich über eine ini-Datei konfigurieren. Einstellungen für die Zugänge sind dabei in Abschnitten gebündelt. Die wichtigsten dieser Parameter lassen sich aber auch über die Kommandozeile spezifizieren. Ist dies der Fall, so haben die Werte des Kommandozeilenaufrufs Vorrang vor denen in der ini-Datei. Die einzelnen Parameter werden nun erläutert.

4.3.2 Common command line parameters

Wird die Bridge ohne jegliche Parameter gestartet, so werden die erforderlichen Parameter aus der Ini-Datei bridge.ini gelesen und die Fehlermeldungen in die Datei bridge.log geschrieben.

Falls es zu einem Aufrufparameter auch einen Eintrag der Ini-Datei gibt, hat der der Aufrufparameter höhere Priorität.

`-inifile <Dateiname>, -ini < Dateiname >`

Name der Ini-Datei, die statt dem Standard-Ini-Datei verwendet wird. Standard ist bridge.ini

`-host <hostname:port>, -hostname <hostname:port>`

Name des Mediators, der als Datenserver fungiert. Dieser gilt für alle aktivierten Bridgeclients

`-port |<ClientName> <portnumber>`

Der Parameter -port ist eigentlich für jeden Klienten in der ini-Datei zu setzen. Will man dieses aber bereits in der Kommandozeile tun, so lassen sich die unterschiedlichen Klienten durch Voranstellen des Klientennamens vor die Portnummer spezifizieren. Die obige Zeile gilt für einen Klienten, entsprechend muss der Parameter -port wiederholt werden, sollen mehrere Klienten konfiguriert werden.

Beispiele für den Aufruf der Bridge:

```
bridge -host server01:30000 -port KEMBridge 4713 -port KEMStreamingBridge 4714

bridge -ini bridge2.ini -port KMultiBridge 3030
```

Kommandozeilen-Parameter für das Logging:

`-nolog`

Schaltet Logging ab

`-loglevel <Integer>`

Konfiguriert, welche Meldungen im Log erscheinen sollen:

- 0: Alle Meldungen inklusive Debug-Ausgaben

- 10 (Standardwert): Alle Meldungen außer Debug-Ausgaben
- 20: Nur Warnungen und Fehlermeldungen
- 30: Nur Fehlermeldungen

`-logfile <Dateiname>, -log <Dateiname>`

Name der Log-Datei, die statt der Standard-Log-Datei verwendet wird. Dieser Parameter muss auf jeden Fall verändert werden, wenn mehrere Clients im selben Arbeitsverzeichnis gestartet werden sind.

`-debug`

Schaltet das Logging auf debug-mode

`-log <logname>`

Setzt die Logdatei auf <logname>.

`-stop <hostname>`

Ruft man die Bridge mit dem obigen Parameter auf, so wird die auf dem angegebenen Host laufende Bridge zum Beenden aufgefordert. Alle in ihr gestarteten Klienten werden heruntergefahren und die Bridge beendet.

4.3.3 Configuration file "bridge.ini"

Alle der folgenden Einträge befinden sich unterhalb des ini-Datei-Abschnitts [Default]. Die Einträge für die einzelnen Klienten schließen daran an. Durch das Einfügen klientenspezifischer Konfigurationsabschnitte wird zusätzlich definiert, welche Klienten in der zu konfigurierenden und zu startenden Bridge aktiviert sind. Im Moment mögliche Klienten sind dabei:

- KEMBridge
- KHTTPRestBridge

Zusätzlich kann noch der KLoadBalancer als Klient der Bridge gestartet werden, dann enthält die ini-Datei nur den Abschnitt

- KLoadBalancer

host = <hostname:portnumber>

siehe Kommandozeilenparameter -host

Speicher-Einstellungen:

Die folgenden drei Parameter dienen zur Konfiguration der Speicherzuteilung und -nutzung. Erlaubt ist die Angabe von Werten entweder in Megabyte oder in tatsächlichen Byte, wobei die Annahme gilt, dass sich Werte kleiner als 1048576 auf Megabyte-Angaben beziehen.

`maxMemory=<Integer, in MB>`

Maximal erlaubte Hauptspeicherbelegung. Minimal 50 MB, standardmäßig gesamter physikalisch vorhandener Hauptspeicher (unter Windows) bzw. 512 MB.

`baseMemory=<Integer, in MB>`

Hauptspeicherbelegung ab der verstärkt versucht wird, Speicher freizugeben. Standardmäßig 0.6 * maxMemory. (alias: "growthRegimeUpperBound")

`freeMemoryBound=<Integer, in MB> [10]`

Falls belegter, aber nicht mehr benötigter Speicher diese Grenze überschreitet, wird er wieder freigegeben.

`minAge=<Integer> [30]`

Mindestdauer (in Sekunden), die ein Cluster im Speicher bleibt. Ein Cluster ist eine Menge von Objekten, die immer zusammen am Stück geladen werden (z.B. ein Individuum mit all seinen (Meta)eigenschaften. Cluster, die längere Zeit nicht mehr verwendet werden, werden bei Bedarf ausgelagert.

`unloadInterval=<Integer> [10]`

Mindestdauer (in Sekunden) zwischen zwei Cluster-Auslagerungen

`unloadSize=<Integer> [4000]`

Mindestanzahl an geladenen Cluster, ab der ausgelagert wird

`keepSize=<Integer> [3500]`

Zahl der Cluster, die beim Auslagern behalten werden

`useProxyValueHolder=true/false`

Um den Mediator bei Suchen zu entlasten, kann die Option useProxyValueHolder=false verwendet werden. Der Client lädt dann Indizes in den Hauptspeicher, statt per RPCs den Mediator abzufragen.Der Nachteil dieser Option ist, dass dann nur noch lesender Zugriff möglich ist.

`loadIndexes=true/false`

Über diese Option werden Indizes ebenfalls in den Speicher geladen. Es ist aber auch weiterhin schreibender Zugriff möglich. Die Option kann bei allen Clients inkl. Knowledge-Builder aktiviert werden.

Logging-Einstellungen:

Zu den Konfigurationsmöglichkeiten des Loggings siehe Logging-Einstellungen im Kapitel 11.1.2 Konfigurationsdatei.

4.3.4 REST-Bridge

4.3.4.1 Introduction

Die REST-Bridge Application ermöglicht den lesenden und schreibenden Zugriff auf i-views über eine RESTful Services-Architektur. Die Schnittstelle steht als HTTP- oder HTTPS-Version zur Verfügung.

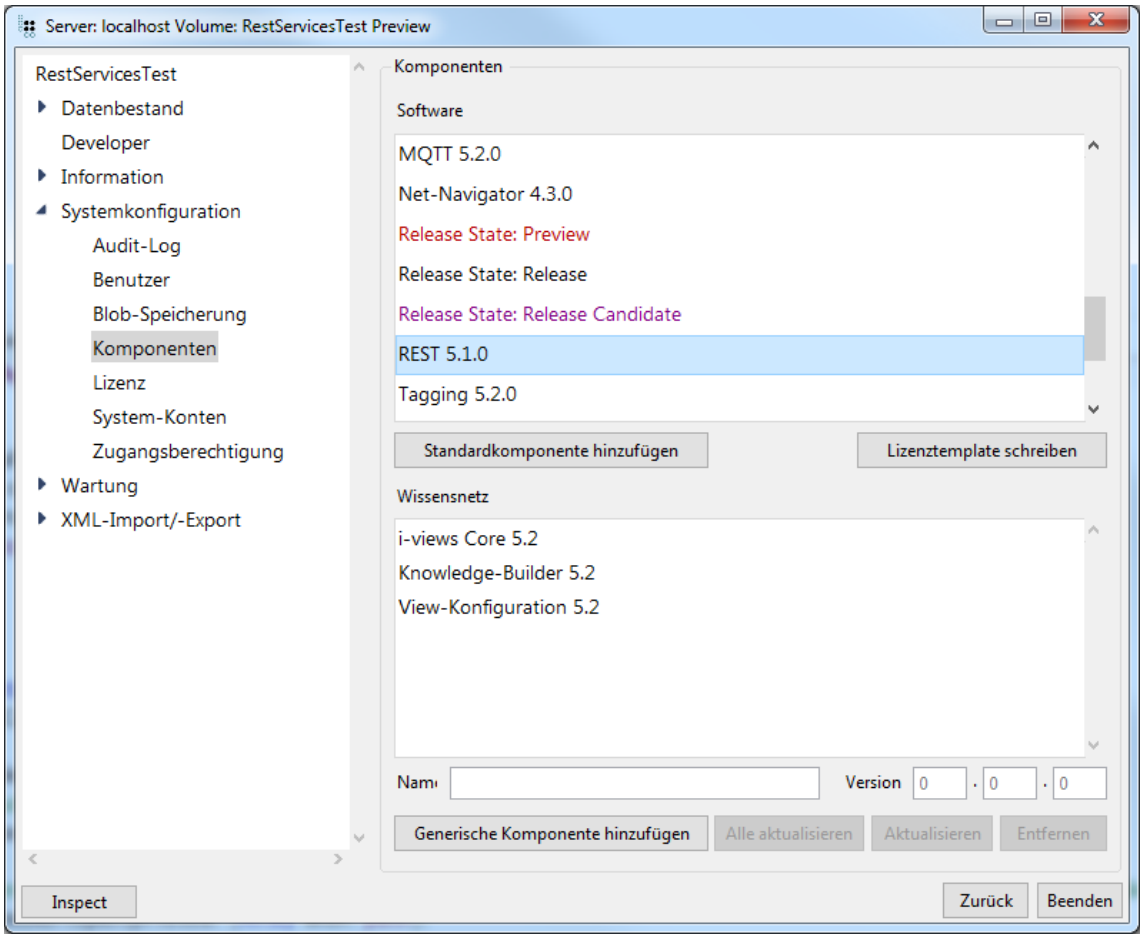
Die REST-Bridge läuft innerhalb der Standard-Bridge von i-views (bridge.exe).

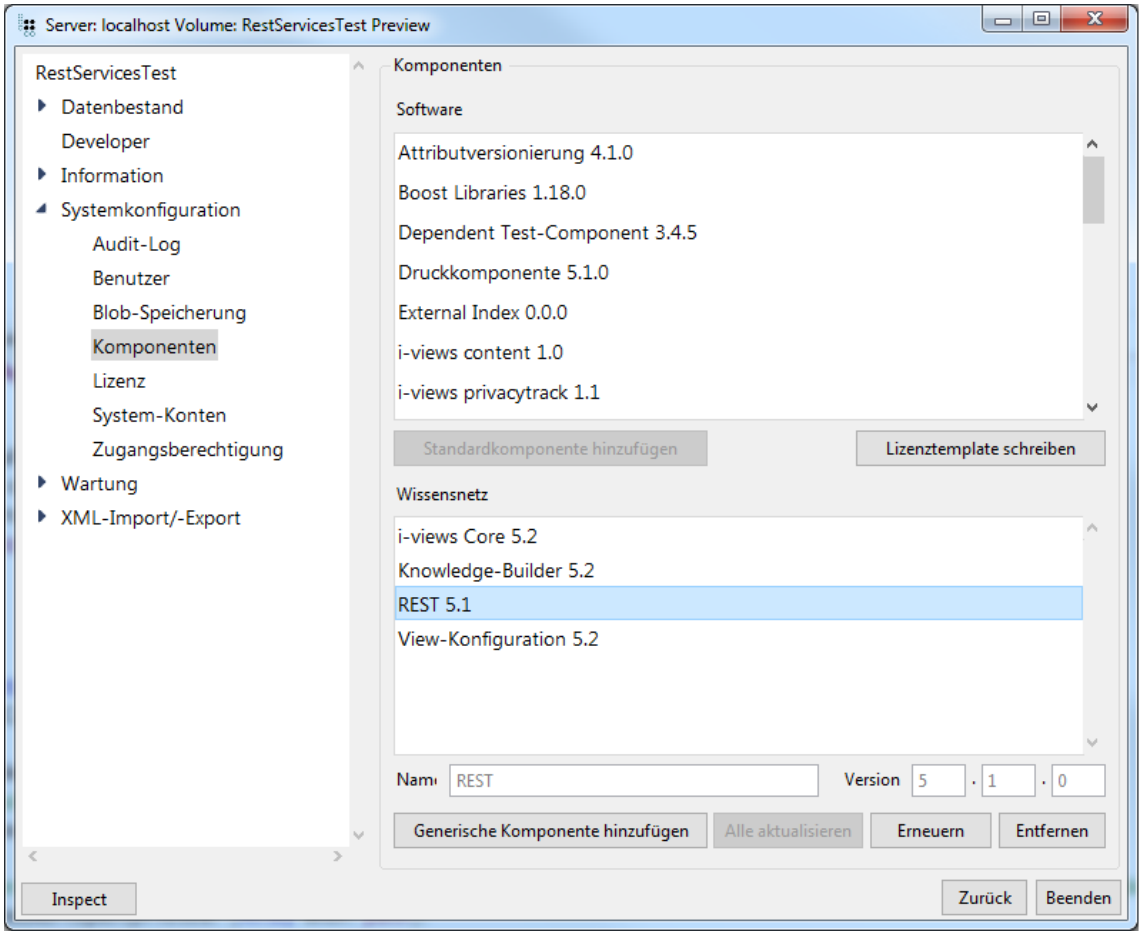
Die Schnittstelle wird vollständig durch Konfigurations-Individuen im Wissensnetz konfiguriert. Der Rückgabewert eines REST-Aufrufes ist eine beliebige Zeichenkette, in der Regel in einem Format, das der aufrufende Client gut weiterverarbeiten kann (z.B. XML oder JSON).

4.3.4.2 Installation

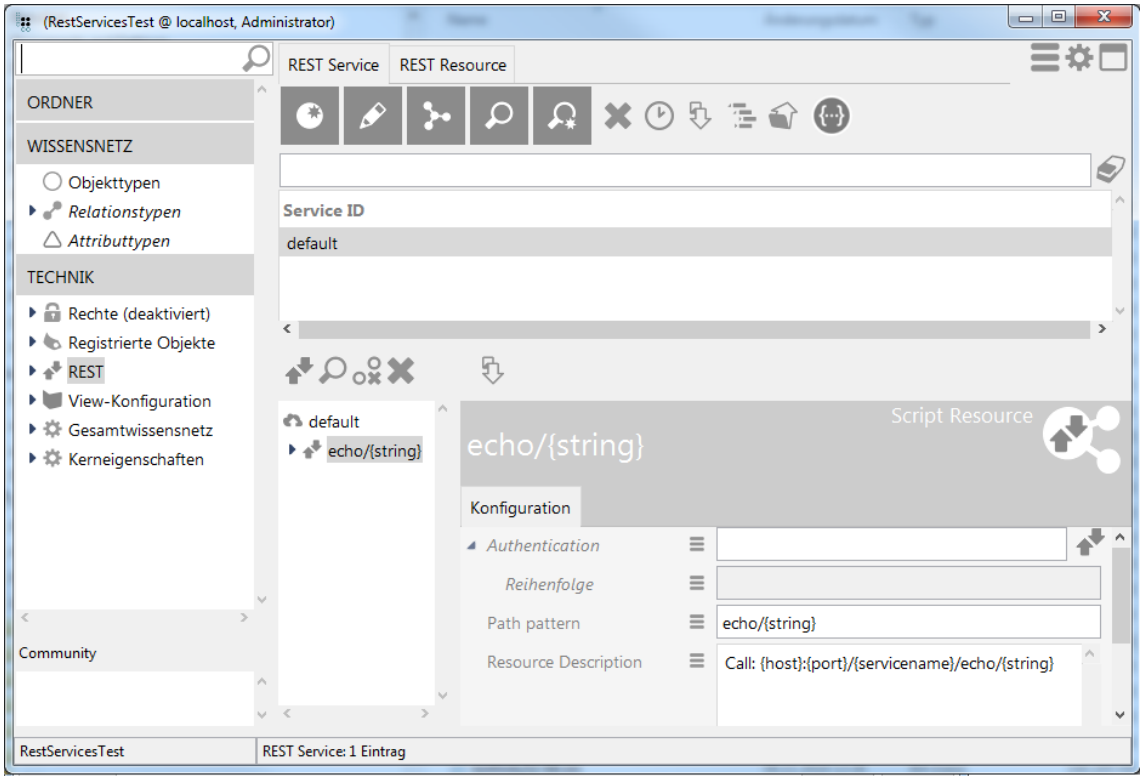
4.3.4.2.1 Prepare Volume

Durch das Hinzufügen der Softwarekomponente "REST" im Admin-Tool wird im Wissensnetz das benötigte Schema angelegt.





Das Schema wird als Teilnetz des Wissensnetzes namens "REST" angelegt, das nur als Administrator im Technikteil bearbeitet werden kann:



4.3.4.2.2 Bridge set up

Die REST-Schnittstelle wird durch die Standard-Bridge-Komponente von i-views bereitgestellt, sofern in der zugehörigen Konfigurationsdatei **bridge.ini** eine Kategorie **KHTTPRestBridge** bzw. **KHTTPSRestBridge** eingetragen ist:

```
[KHTTPRestBridge] volume=name des Wissensnetzes port=port, unter dem der Service erreichbar sein s
```

Für die HTTPS-Version müssen in der Konfigurationsdatei zusätzlich die Dateipfade für Zertifikat und Private Key angegeben werden:

```
[KHTTPSRestBridge] volume=name des Wissensnetzes port=port, unter dem der Service erreichbar sein
```

Im Konfigurationsabschnitt "KHTTPRestBridge" oder "KHTTPSRestBridge" können außerdem noch die folgenden speziellen Konfigurationsoptionen eingetragen werden:

Name	Beschreibung
realm	Name, der bei aktivierter Authentifizierung als Realm-Name an den Client zurückgegeben wird. Web-Browser zeigen den Realm-Namen typischerweise in Dialogfenstern zur Authentifizierung als Applikationsnamen an, damit der Benutzer weiß, wer die Authentifizierung fordert. Standardwert: REST

4.3.5 KEM-Bridge

KEMBridge

Abschnittsname:

[KEMBridge]

port = <portnumber>

Angabe des Ports, unter welchem die KEMBridge reagiert. Bei Nichtangabe gilt der Defaultwert von 4713.

ldapHost = <hostname:portnumber>

Angabe des LDAP-Hostes, welcher kontaktiert werden soll, für den Fall, dass die Authentifikation über LDAP stattfinden soll. Ist dieser Parameter angegeben, so muss die Authentifikation über LDAP abgewickelt werden.

maxLoginCount = <number>

Angabe der maximalen Fehlversuche beim Einloggen, bis der entsprechende Benutzer im Netz gesperrt wird. Danach ist ein Einloggen nur nach Entsperren per Knowledge-Builder möglich. Falls der Wert nicht gesetzt ist, ist ein fehlerhaftes Einloggen praktisch beliebig oft möglich.

Um ein Sperren des Benutzers im Wissensnetz zu ermöglichen, muss für Individuen des Personenkonzepts ein boolesches Attribut mit internem Namen userlock und Defaultwert false definiert sein.

KEMrestrictToIPAddress = <IP-Adresse>

Wenn dieser Parameter gesetzt ist, werden nur Verbindungen von dem hier angegebenen Host akzeptiert.

trustedLoginEnabled = <true/false>

Erlaubt ein Einloggen ohne Passwort mittels des Requests: „newAuthenticatedUser(username)“.

preventSessionReplay=<true/false>

[default=false]

Dieser Parameter gibt an, dass jede schreibende Session ihren eigenen geschützten Wissensnetzzugriff erhält, so dass der sonst übliche Mechanismus, die Aktionen einer deaktivierten Session beim Reaktivieren erneut auszuführen, um den letzten aktuellen Editorstand zu erhalten, unnötig wird.

KEMStreamingBridge

Abschnittsname:

[KEMStreamingBridge]

port = <portnumber>

Angabe des Ports, unter welchem die KEMStreamingBridge reagiert. Bei Nichtangabe gilt der Defaultwert von 4714.

4.3.6 KLoadBalancer

Der KLoadBalancer kann eingesetzt werden, um die Services und Verfügbarkeit der KEM-Bridge und KEMStreamingBridge zu skalieren.

Im Abschnitt [KLoadBalancer] können/müssen die folgenden Angaben gemacht werden, um den gewünschten Betriebsmodus zu erreichen:

- allowRemoteShutdown (Default-Wert false)
- autoRestart (Default-Wert true)
- directory (Default-Wert aktuelles Arbeitsverzeichnis, in dem der KLoadBalancer gestartet wurde)
- executable (Default-Wert 'bridge.exe')
- image (Default-Wert 'bridge.im')
- vm (Default-Wert 'visual')
- hostname (Default-Wert Localhost)
- **configNames (benötigter Wert, nicht optional)**
- parameters (Default-Wert leer)

Der Parameter #configNames dient der weiteren Konfiguration der zu startenden KEM-Bridges und KEMStreamingBridges, je Einzel-Konfiguration wird ein Bridge-Typ gesteuert. Die Konfigurationsnamen sind durch Komma zu trennen.

Hier ein Beispiel für eine KLoadBalancer-ini-Datei:

```
[Default] [KLoadBalancer] hostname=ws01 port=30003 directory=C:\3.2\balancing executable=bridge.ex
```

Beim Start werden gemäß der beiden Konfigurationen KEMBridges und KEMStreaming-Bridges gestartet. Da zum Betrieb dieselbe Software wie für den Betrieb des KLoadBalancers verwendet wird, sind in diesem Abschnitt die Angabe der Parameter #executable, #image und #vm (für Linux-Betrieb), #hostname, #directory und #parameters nötig.

executable / image, vm; directory: Angaben, wie die einzelnen Bridges gestartet werden können. Unter Windows wird die Angabe von #executable und #directory benötigt, unter Linux die Angabe von #image, #vm und #directory.

hostname / port: Der Hostname, der den zu startenden Bridges als für Verwaltungszwecke zu kontaktierender KLoadBalancer genannt wird. Falls hier keine Angabe gemacht wird, wird der Rechnername ermittelt und dieser verwendet. Der Port gibt an, unter welchem Port die Bridges den Balancer ansprechen, Default-Wert ist 4715.

Vorsicht: Der Name des jeweiligen Mediators, den die Bridges zum Abrufen von Daten kontaktieren, ist in den jeweiligen ini-Dateien gemäß Konfigurationsabschnitt einzutragen!

parameters: Ein Feld, mit dem zusätzliche Angaben in die Kommandozeile der zu startenden Bridges eingefügt werden können, ist für alle zu startenden Bridges gleich.

allowRemoteShutdown: Parameter, der angibt, ob der KLoadBalancer per shutdown-Request per remote-Zugriff zu beenden ist.

autoRestart: Parameter, der angibt, ob eine gestoppte KEMBridge nach dem shutdown erneut zu starten ist, mit neuer ID.

In jedem Konfigurationsabschnitt müssen zusätzliche Angaben gemacht werden:

- bridgeClientClassName (nicht optional, nur eine Angabe je Abschnitt möglich. Bitte

obige Schreibweise beachten!)

- inifile (ini-Datei mit Einstellungen für diesen Typ zu startende Bridge)
- bridgeLogfile (Muster eines Logfile-Namens, in den ein Platzhalter eingefügt wird, <id>, über den sich die Log-Dateien der einzelnen Bridges auseinanderhalten lassen, wird mit der laufenden Nummer der gestarteten Bridge ersetzt)
- maxBridges (Anzahl der maximal zu startenden Bridges des angegebenen Typs, nicht optional!)
- sslEnabled (Angabe, ob die Bridges dieses Typs SSL für den Verbindungsaufbau verwenden sollen, Default-Wert false)

Zur Beachtung: Der Parameter #directory gibt das Arbeitsverzeichnis an, in dem die in den Konfigurationsabschnitten angegebenen Dateien gesucht und ggfs. angelegt werden. Software und ini-Datei für den Start des KLoadBalancers können sich an anderer Stelle befinden.

Die ini-Dateien der jeweiligen Bridges müssen wie gewohnt aufgebaut werden. Ein Beispiel für die im obigen Konfigurationsabschnitt KEM referenzierte ini-Datei ist hier angefügt:

```
[Default] host=mediator-hostname:30053 [KEMBridge] trustedLoginEnabled=true preventSessionReplay=t
```

Für Details sei auf Kapitel 5 "Konfigurationsdatei bridge.ini" verwiesen.

4.4 Jobclient

4.4.1 Overview

Der Job-Client erbringt zum einen Dienste für andere i-views-Clients, um diese von rechenzeit- oder datenintensiven Aufgaben zu entlasten. Zum anderen dient er als Brücke zwischen i-views-Clients und externen Systemen.

Zu seinen wichtigsten Aufgaben gehört die Ausführung aller Arten von Suchen sowie die Auslieferung der Suchergebnisse an die Clients (Sortierung, textuelle Aufbereitung, Rechtefilterung).

Im Normalfall wartet der Client auf die Fertigstellung eines Auftrags (Synchronbetrieb).

Für die Ausführung komplexer Suchen, das Erstellen von Statistiken, Batch-Abgleiche, Datenaufbereitungen, Datenbereinigungen, etc. muss der Client nicht auf die Fertigstellung warten (Asynchronbetrieb). Das Ergebnis wird vom Service bereitgestellt und der Client wird benachrichtigt. Das Ergebnis kann dann beliebige Zeit später eingesehen werden. Da das Ergebnis auch persistent gemacht wird, ist es auch nach einem Neustart des Systems bzw. im Falle eines Fail-Overs weiterhin verfügbar.

Funktionsweise:
In dem vom i-views-Mediator bereitgestellten geteilten Objektraum werden die Aufträge der Clients an die Services in sogenannten Pools abgelegt. Alle i-views Job-Clients werden über neue Aufträge notifiziert und bewerben sich - sofern sie aktuell frei sind - für die Bearbeitung des neuen Auftrags. Nach Bearbeitung des Auftrags wird das Resultat wieder im geteilten Objektraum bereitgestellt, der beauftragende Client wird benachrichtigt und das Ergebnis kann abgerufen und zur Anzeige gebracht werden. Somit beauftragt der Client zwar logisch einen Job-Client, physikalisch läuft die Kommunikation aber immer über den i-views-Server. Für den Client ist es transparent, welcher Job-Client seinen Auftrag ausführt, sowie es für den Job-Client transparent ist, wo der Auftrag herkommt und wie viele parallele Job-Clients zurzeit aktiv sind. Für Administratoren ist die Installation und Wartung der Job-Clients daher sehr einfach und flexibel. Job-Clients lassen sich beliebig skalieren, auf verschiedene Rechner verteilen und dynamisch zu- und abschalten. Eine externe Clusterung oder sonstige

Orchestrierung ist nicht erforderlich.

Technische Daten:
Multi-Platform Executable auf Basis der VisualWorks Smalltalk Virtual Machine (jobclient.exe bzw. jobclient.im)
Benötigt eine TCP/IP-Verbindung zum i-views-Server
Automatische Lastverteilung zwischen den Services
Job-Clients können zu jeder Zeit zugeschaltet oder heruntergefahren werden
Standby-Modus bei zeitweiliger Nicht-Verfügbarkeit benötigter Ressourcen

4.4.2 Configuration of the Job-Client

4.4.2.1 Configuration file "jobclient.ini"

Die Konfiguration des Job-Clients wird in der Ini-Datei vorgenommen. Falls diese nicht durch den Aufrufparameter "-inifile" beim Start des JobClients spezifiziert ist, wird "jobclient.ini" als Konfigurationsdatei verwendet.

4.4.2.1.1 Allgemeine Parameter

Die folgenden Parameter können konfiguriert werden:

Parameter:	**Beschreibung:**	**Syntax:**
host	Name / IP-Adresse und Port des Servers.	`host=<Hostname:Portnummer>`
volume	Der Name des Wissensnetzes, auf dem gearbeitet werden soll.	`volume=<Volumename>`
jobPools	Angabe, welche Jobs der Job-Client abarbeiten soll. Die Namen der zu startenden Job-Pools sind hier kommagetrennt anzugeben. Alternativ kann auch die Kategorie (z.B. "index") angegeben werden. Es werden dann alle Job-Pools dieser Kategorie ausgewählt. Die möglichen Typen werden in den Unterkapiteln dargestellt.	`jobPools=<Jobname1> [,<Jobname2>, ...]` Beispiel: `jobPools=KScriptJob, query`

cacheDir	Beschreibung des Ortes, an dem der Cache für den Job-Client angelegt wird.	`cacheDir=<Verzeichns>`

volumeAccessor	Beschreibung der Speicherart des Caches. Wenn nicht angegeben wird CatBSBlockFileVolumeAccessor verwendet. Diese Speicherart ist vor allem bei großen Netzen zu empfehlen, da CatCSVolumeFileStorageAccessor eine große Anzahl an Dateien anlegen würde.	Beispiel:
		`volumeAccessor=CatBSBlockFileVolumeAccessor`
		oder
		`volumeAccessor=CatCSVolumeFileStorageAccessor`

maxCacheSize	Zielgröße des Caches	`maxCacheSize=<Größe in MB>`
shutDown-Timeout	Dauer, auf die beim Herunterfahren des Job-Clients auf die Beendigung der aktiven Jobs gewartet wird. Nach Ablauf werden die Jobs abgebrochen. Der Standardwert ist 10 Sekunden.	`shutDownTimeout=<Sekunden>`
enableLowSpaceHandler	Mit dieser Option wird der LowSpaceHandler eingeschaltet. Dieser sollte auf jeden Fall bei großen Netzen eingeschaltet werden.	`enableLowSpaceHandler=true/false`

useProxyValue-Holder	Mit dieser Option kann gesteuert werden, ob der Job-Client Indexzugriffe per RPC durchführt (true), oder Indizes in den Speicher lädt (false). Diese Option sollte ausgeschaltet werden, wenn der Mediator entlastet werden soll. Dabei sollte allerdings darauf geachtet werden, dass der Job-Client genug Speicher zur Verfügung hat. Falls der Job-Client für schreibende Jobs konfiguriert wurde, hat diese Option keinen Effekt, da dann der Indexzugriff immer per RPC durchgeführt wird. Es wird beim Start im Log eine Meldung ausgegeben, falls man den Wert auf false gesetzt hat.	`useProxyValueHolder=true/false`
loadIndexes	Seit 4.2 gibt es die Option loadIndexes=true. Indizes werden dann ebenfalls in den Speicher geladen. Im Gegensatz zur Option useProxyValueHolder ist aber auch weiterhin schreibender Zugriff möglich. Die Option kann bei allen Clients inkl. KnowledgeBuilder aktiviert werden.	`loadIndexes=true/false`
name	Dieser Name wird verwendet, um den Job-Client im Admin-Tool in der Übersichtsliste aller Job-Clients zu identifizieren.	`name=<Job-Client-Name>`
scheduledJobs	Eine kommagetrennte Liste mit Namen von Jobs, deren Ausführung geplant werden soll.	`scheduledJobs=<Job-Name1> [, <Job-Name2>, ...]`

Speicher-Einstellungen:

Die folgenden drei Parameter dienen zur Konfiguration der Speicherzuteilung und -nutzung. Erlaubt ist die Angabe von Werten entweder in Megabyte oder in tatsächlichen Byte, wobei die Annahme gilt, dass sich Werte kleiner als 1048576 auf Megabyte-Angaben beziehen.

Parameter:	**Beschreibung:**	**Syntax:**
maxMemory	Maximal erlaubte Hauptspeicherbelegung. Minimal 50 MB, standardmäßig gesamter physikalisch vorhandener Hauptspeicher (unter Windows) bzw. 512 MB.	`maxMemory=<Integer, in MB>`
baseMemory	Hauptspeicherbelegung ab der verstärkt versucht wird, Speicher freizugeben. Standardmäßig 0.6 * maxMemory. (alias: "growthRegimeUpperBound")	`baseMemory=<Integer, in MB>`
freeMemoryBound	Falls belegter, aber nicht mehr benötigter Speicher diese Grenze überschreitet, wird er wieder freigegeben.	`freeMemoryBound=<Integer, in MB> [10]`
minAge	Mindestdauer (in Sekunden), die ein Cluster im Speicher bleibt. Ein Cluster ist eine Menge von Objekten, die immer zusammen am Stück geladen werden (z.B. ein Individuum mit all seinen (Meta)eigenschaften. Cluster, die längere Zeit nicht mehr verwendet werden, werden bei Bedarf ausgelagert.	`minAge=<Integer> [30]`
unloadInterval	Mindestdauer (in Sekunden) zwischen zwei Cluster-Auslagerungen	`unloadInterval=<Integer> [10]`

unloadSize	Mindestanzahl an geladenen Cluster, ab der ausgelagert wird	`unloadSize=<Integer> [4000]`
keepSize	Zahl der Cluster, die beim Auslagern behalten werden.	`keepSize=<Integer> [3500]`

Job Konfiguration:

Für die Konfiguration von einzelnen Jobs in der Konfigurationsdatei, muss jeweils ein neuer Abschnitt angelegt werden. Diese werden jeweils mit dem Namen des Jobs in einem Paar eckiger Klammern begonnen. Danach folgen die jeweiligen Parameter des Jobs.

Beispiel:

```
[Job-Name1]
<Parameter>=<Wert>
...

[Job-Name2]
...
```

Logging-Einstellungen:

Zu den Konfigurationsmöglichkeiten des Loggings siehe Logging-Einstellungen im Kapitel 11.1.2 Konfigurationsdatei.

Lucene-Server-Konfiguration:

Die Einbindung von Lucene erfolgt über einen Job Client, dessen jobclient.ini-Datei dafür entsprechend konfiguriert werden muss. Nachfolgend eine Beispielkonfiguration:

```
[lucene]
directory=lucene-index
port=5100
pageSize=100
; Wildcards am Wortanfang sind standardmäßig verboten, da sehr langsam
; In dieser Konfiguration erlauben
allowLeadingWildcards=true

[JNI]
classPath=lucene-6.4.1\core\lucene-core-6.4.1.jar;lucene-6.4.1\analysis\common\lucene-analyzers-co
```

Das Verzeichnis *lucene-6.4.1* enthält die Binaries von Lucene. ImVerzeichnis *lucene-index* wird der Index gespeichert.

4.4.2.1.2 Job-spezifische Parameter

Allgemein:

Parameter:	Beschreibung:	Syntax:
jobPool	JobPool für die Ausführung des Jobs.	`jobPool=<Job-Pool-Name>`
...?		

scheduledJobs:

Parameter:	Beschreibung:	Syntax:

time	Zeitpunkt an dem zum ersten mal der Job ausgeführt werden soll.	`time=<Uhrzeit>`
		Beispiel:
		`time=22:15`

interval	Angabe wie häufig der Job ausgeführt werden soll. (d=Tage, h=Stunden, m=Minuten, s=Sekunden)	`interval=<Zeitangabe>`
command	**Nur bei KExternalCommandJob.** Name einer externen Batchdatei, die vom Job ausgeführt werden soll.	`command=<Dateiname.cmd>`
scriptName	**Nur bei KScriptJob**. Registrierungsschlüssel eines internen Skripts, das vom Job ausgeführt werden soll.	`command=<Skriptressource>`
unique	(?)	`unique=true/false`
user	(Nur ?) Interner Name einer Benutzerinstanz unter der der Job ausgeführt werden soll.	`user=<Username>`

arguments	(Nur **KExternalCommandJob**?) Argumente die beim Skriptaufruf übergeben werden.	`arguments=<Argument1 [Argument2 ...]>`

4.4.2.2 JobPool Typen

Die folgenden Typen an JobPools stehen zur Verfügung:

4.4.2.2.1 index jobs

- Kategorie(n): **index**

Werden für den Jobpool die unten angezeigten Jobklassen oder **index** angegeben, dann werden die Indexierungsaufträge vom Job-Client ausgeführt. Die Indexierungsaufträge sollten nur von einem einzigen Job-Client durchgeführt werden. Statt alle Jobklassen einzeln im JobPool aufzuzählen, kann auch der symbolische Name **index** verwendet werden.

KAddAllToIndexJob

- Bezeichnung: Attribute zum Index hinzufügen

KLightweightIndexJob

- Bezeichnung: Externen Index aktualisieren

  Ein externer Index wird über den KLightweightIndexJob gepflegt.

KLuceneAdminJob

- Bezeichnung: Lucene Verwaltungsaufgabe

  Der KLuceneIndexJob verwaltet einen extern aufgebauten Lucene-Index.

KRemoveIndexJob

- Bezeichnung: Attribute aus dem Index entfernen

KSyncIndexJob

- Bezeichnung: Index synchronisieren

KAddAllToIndexJob, KRemoveIndexJob und KSyncIndexJob werden benötigt, um die internen Indizes zu pflegen.

4.4.2.2.2 KBrainbotJob

- Kategorie(n): <keine>
- Bezeichnung: KBrainbotJob

Der KBrainbotJob führt Aktionen zur Pflege eines Brainbot-Indexes aus.

Falls innerhalb der Konfiguration im Admin-Tool angegeben wird, dass Pflegeaktionen von einem Jobclient ausgeführt werden sollen ("Jobclient benutzen"), so muss ein Jobclient gestartet werden, damit die Pfege des externen Index ausgeführt wird.

Der KBrainbotJob hat keine weiteren Konfigurationsparameter in der ini-Datei, da die gesamte Konfiguration im Admin-Tool stattfindet.

4.4.2.2.3 KExternalCommandJob

- Kategorie(n): <keine>
- Bezeichnung: Externer Aufruf

Mit Hilfe des **KExternalCommandJobs** ist es möglich ausführbare Programme, die sich mit der Abarbeitung oder Veränderung von Dateien beschäftigen oder einfach nur aufgerufen werden sollen, anzusteuern. Eine Konfiguration in der INI-Datei des JobClients ist nicht notwendig. Der Job wird durch einen Skriptaufruf eingeworfen.

Das Hauptelement des Skriptaufrufes ist das Element ***ExternalCommandJob***. Mit dem Attribut *execution* kann eingestellt werden, ob der Job lokal ohne JobClient (Wert: *local*) oder mit JobClient (Wert: *remote*) ausgeführt werden soll. Der Standardwert ist *remote*.

Anmerkung zur Remote-Ausführung:

Eine Kontrolle über den Zugriff auf lokale Programme findet über den Aufruf einer Batchdatei statt. Bevor sich der JobClient einen KExternalCommanJob zur Ausführung nimmt, überprüft er, ob er diesen Job ausführen kann. Das ist der Fall, wenn im aktuellen Verzeichnis des JobClients die Batchdatei vorhanden ist, die im Element *Command*angegeben ist. Wird der aktuell anstehende Job von keinem JobClient zur Bearbeitung angenommen, ist die Job-Warteschlange für den Benutzer, der den Job eingeworfen hat, blockiert. Dieser Job muss von Hand gelöscht werden.

Das notwendige, erste Unterelement im Skript:

- **Command**: gibt an welche Batchdatei aufgerufen werden soll

```
<Command>convert.bat</Command>
```

In dem Element Command wird der Name der Batchdatei angeben. In der Batchdatei ist das Verzeichnis und das auszuführende Programm selbst angegeben. ***Wichtig:*** *Die Batchdatei muss auf der gleichen Ebene wie das Programm (z.B. JobClient oder KB) liegen. Verzeichnisangaben im Element Command werden ignoriert.*

Die weiteren Unterelemente werden von oben nach unten abgearbeitet. Falls die Reihenfolge der Parameter im externen Programm eine Rolle spielt, sollte dies berücksichtigt werden.

Skriptelemente, die die Parameter für den Aufruf bilden:

- **OptionString**: kann mehrfach verwendet werden. Es werden Parameter des aufzurufenden externen Programms als Zeichenketten angegeben. Die Parametereinträge

müssen vollständig angegeben werden.
<OptionString>-size 100x100</OptionString>

- **OptionPath**: der angegebene Path-Ausdruck wird ausgewertet und als Zeichenkette in den Kommandoaufruf eingebaut
<OptionPath path="./topic()/concept()/@$size$"/>

Skriptelemente, die sich mit dem Handling von Attributen beschäftigen

- **SourceBlob**: Angabe des Blobattributes, das als Datenquelle verwendet wird
<SourceBlob><Path path="$bild$"/></SourceBlob> <SourceBlob path="$bild$"/>
- **ResultAttribute**: Angabe der Parameter für die Erzeugung eines neuen oder die Veränderung eines bestehenden Blobattributes mit dem Inhalt der Datei bzw. der Datei selbst, die das Ergebnis des extern aufgerufenen Programms ist.
Attributwerte:
name: Name bzw. interner Name des anzulegenden Attributes
topic: Zielindividuum des anzulegenden Attributes
modifyExisting: verändern (*true*) oder neu anlegen (*false,* Standardwert)
filename: Dateiname des anzulegenden Blobattributes
<ResultAttribute name="$bild2$" topic="./topic()" modifyExisting="true" filen
<Path path="$bild2$"></ResultAttribute>

Beispiel 01:

Skript:

<Script> <ExternalCommandJob execution="local"> <Command>convert.bat</Command> <O

Inhalt der Batchdatei unter Windows:

```
"C:\Program Files\ImageMagick-6.2.6-Q16\convert.exe" %*
exit /B %ERRORLEVEL%
```

Inhalt der Batchdatei unter Linux:

```
#!/bin/bash
convert $*
```

Beispiel 02:

Skript:

<Script> <ExternalCommandJob execution="local"> <Command>convert2.bat</Command> <

Inhalt der Batchdatei unter 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%
```

Inhalt der Batchdatei unter Linux:

```
#!/bin/bash

convert -size 100x100 $1 -geometry +5+10 $2 -geometry +35+30 -composite $3
```

Anmerkung: Die beiden Beispiele liefern als Ergebnis die gleiche Datei. In den Windows-

Batchfiles dient der Exit-Befehl dazu, den Exit-Code von "convert" an den Aufruf zurückzuliefern.

Hier noch ein Beispiel für ein erweitertes Konvertierungsskript, welches mit den Parametern "Quelldatei", "Bildbreite" und "Zieldate" aufgerufen werden kann und welches nur breitere Bilder auf die angegebene Breite verkleinert. Das Script schreibt außerdem eine Protokolldatei über die Konvertierung wobei auch Fehlermeldungen von Image Magick in die Logdatei geschrieben werden:

```
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%
```

Und hier noch die Version für 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

- Kategorie(n): <keine>
- Bezeichnung: Blob in ein Textattribut umwandeln. Aus dem Blobattribut wird mithilfe der im Admin-Tool auf dem Reiter "Indexkonfiguration -> Externer Volltextfilter" angegebenen Batch-Datei der Textinhalt extrahiert und in einem neuen Attribut des angegebenen Textattributtyps abgelegt. Weitere mögliche Parameter für den Job sind das Topic, an dem der Extrakt angelegt werden soll, sowie die Sprache des anzulegenden Attributs, in dem Fall, dass das angegebene Textattribut mehrsprachig ist. Dieser Job wird von einem Trigger eingeworfen, der so angelegt sein sollte, dass er auf Erzeugen und Modifizieren von Blobattributen reagiert. Die dabei anzugebende KSkript-Regel lautet "ExtractBlobText" und gestattet die Angabe der oben genannten Parameter.

4.4.2.2.5 KQueryJob

- Kategorie(n): query
- Bezeichnung: Suche

Dient der ausgelagerten Ausführung von einfachen und Expertensuchen auf einem Jobclient. Wird je nach Bedürfnissen der betrachteten Suche ausgestattet und ausgeführt.

4.4.2.2.6 KScriptJob

- Kategorie(n): script
- Bezeichnung: KScriptJob

Mithilfe des KScriptJobs lassen sich KSkripte aus KSkript heraus so aufrufen, dass sie auf dem Job-Client ausgeführt werden. Dabei geschieht das Erzeugen des Jobs durch die KSkript-Regel "ScriptJob", welche ausgestattet mit Script und den zu diesem Zeitpunkt errechneten Startobjekten als Ausgangspunkt, den resultierenden KScriptJob in die Job-Queue einstellt. So lassen sich Arbeiten asynchron auf Job-Clients verteilen. Anwendungsbeispiele sind die Auslagerung von Tätigkeiten, die bei sequenzieller Ausführung den aufrufenden Klienten zu lange blockieren würden.

Der Parameter "scriptName" muss hierfür auf den Registrierungsschlüssel eines im Netz hinterlegten Skripts verweisen. Das Skript wird automatisch in einer Transaktion gekapselt.

4.4.2.3 Example for an ini-file

```
volume=MeinNetz
host=localhost
jobPools=query, index
cacheDir=jobcache
logfile=jobclient01.log
maxMemory=400
name=jobclient01
```

4.4.2.4 Performance tuning

Vorab laden

Die JobClients können beim Hochfahren durch die Konfiguration auswählbare Strukturen vorab laden. Durch diesen Vorgang steigt der Speicherbedarf des JobClients. Im Gegenzug kann der JobClient Jobs schneller ausführen.

In der Ini-Datei des JobClients muss der Eintrag **keepClusterIDs** angegeben werden. Mögliche Werte für diesen Eintrag sind:

- **index** - Bei den Einstellungen der zusammensteckbaren Indexer gibt es die Möglichkeit, das Häkchen bei *Jobclient soll Index in den Hauptspeicher laden* zu setzen. Für die aktivierten Indexer wird ein Teil Ihrer Indexstruktur geladen.
- **protoOfSizes** - Die Anzahl der Individuen jeden Typs wird bereits beim Start ermittelt.
- **accessRights** - Das Root-Objekt des Rechtesystems wird in den Speicher geladen.

Wichtig: Für den Eintrag *useProxyValueHolder* muss der Wert *false* gesetzt sein. Sonst versucht der JobClient RPCs (Anfragen, die der Mediator beantworten kann) an den Mediator abzusetzen. Der Client soll jedoch die Cluster selber laden und unter Umständen auch im Speicher behalten.

Anmerkung: Es ist ebenfalls von Vorteil, für eine Performanceverbesserung den Festplattencache für den JobClient einzuschalten.

Beispiel für die Einträge in der INI-Datei:

```
[Default]
...
useProxyValueHolder=false
keepClusterIDs=index,protoOfSizes,accessRights
cacheDir=jobcache
maxCacheSize=1000
...
```

4.5 BLOB-Service

4.5.1 Introduction

Der Blob-Service dient der Datenhaltung von großen Dateien außerhalb des Wissensnetzes, aber verknüpft mit den Datei-Attributen, in denen diese Dateiinhalte abgelegt werden sollen. Dies hat mehrere Vorteile:

- Das Wissensnetz enthält dadurch nur noch die semantischen Informationen, die auf den Dateien aufsetzen und bleibt gut sicher- und übertragbar.
- Speicherorte von Wissensnetz und Dateiinhalten können unterschiedlich konfiguriert werden.
- Es lassen sich mehrere Blob-Services an ein Wissensnetz anschließen, so dass theoretisch je Attributdefinition ein Speicherort vorgehalten werden kann.

Im folgenden Kapitel wird das Einrichten und der Betrieb von Blob-Services erläutert.

4.5.2 Configuration

Um festzulegen, unter welcher Netzwerk-Adresse (Host und Port) der Blobservice erreichbar sein soll, muss in der Datei "blobservice.ini" die Option "interfaces" eingetragen werden. Prinzipiell gibt es dabei zwei Möglichkeiten:

1. Der BLOB-Service soll nur von dem Rechner aus erreichbar sein, auf dem der BLOB-Service installiert ist
2. Der BLOB-Service soll über das Netzwerk auch von anderen Rechnern aus erreichbar sein.

Hier ein Konfigurationsbeispiel für Variante 1, wobei der BLOB-Service-Port (30000) auch frei wählbar ist:

```
interfaces=http://localhost:30000
```

Zur Konfiguration von Variante 2 muss man anstelle von "localhost" die IP-Adresse des Netzwerk-Adapters eintragen, über den der BLOB-Service aus dem Netzwerk ansprechbar sein soll. Möchte man, dass der BLOB-Service über alle Netzwerk-Adapter erreichbar ist, die auf dem Rechner aktiv sind, so muss man als IP-Adresse "0.0.0.0" eintragen. Beispiel:

```
interfaces=http://0.0.0.0:30000
```

Wird der BLOB-Service über das Netzwerk angesprochen, so sollte die Kommunikation verschlüsselt werden. Die verschlüsselte Kommunikation über HTTPS kann ebenfalls in der Option "interfaces" konfiguriert werden, indem "http://" durch "https://" ersetzt wird. Beispiel:

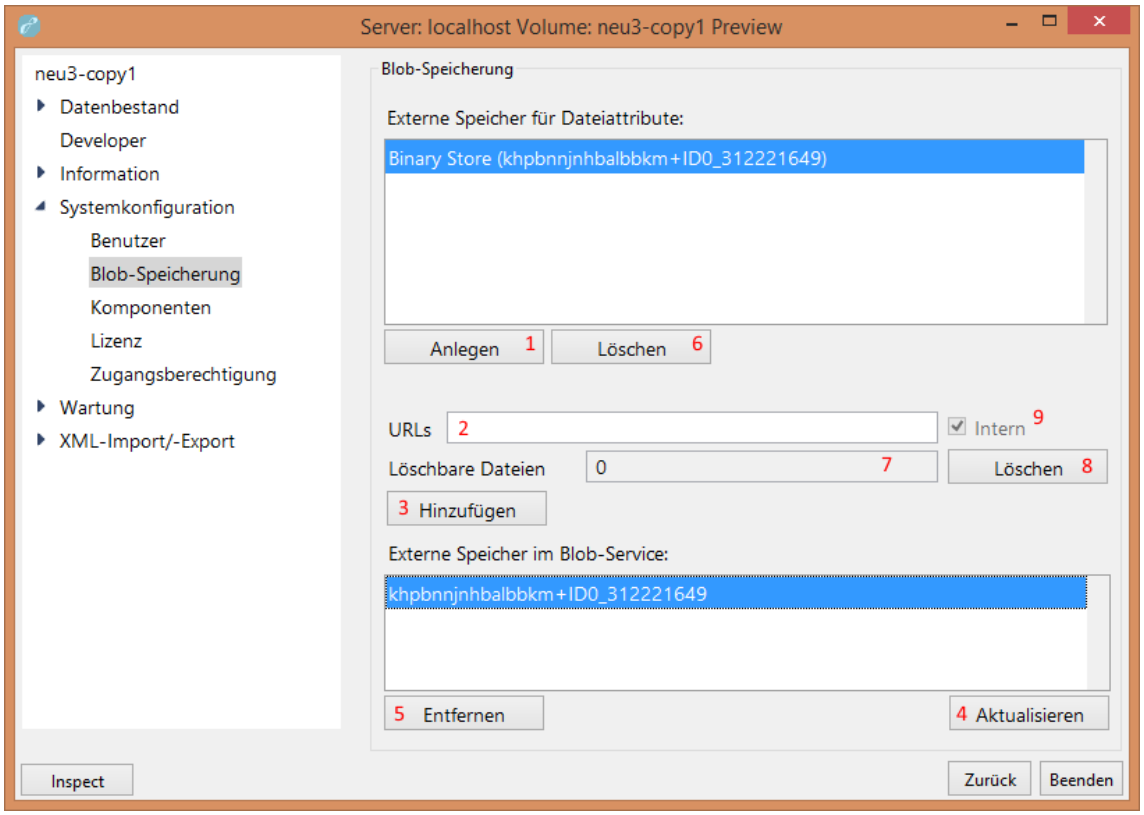
```
interfaces=https://0.0.0.0:30000
```

Für den verschlüsselten Fall siehe auch das nachfolgende Kapitel SSL Zertifikate.

Um den Betrieb zu gewährleisten, muss zusätzlich im Arbeitsverzeichnis die DLL des SQLite Frameworks "sqlite3.dll" vorhanden sein. Ohne diese DLL kann die intern benötigte Verwaltungsstruktur nicht aufgebaut und gepflegt werden.

Danach kann der Blobservice gestartet werden und steht ab sofort zur Verfügung.

Um den Blobservice mit einem Blobstore in der semantischen Graph-Datenbank zu verknüpfen,bietet das Admin-Tool unter "Systemkonfiguration -> Blob-Speicherung" die nötigen Werkzeuge:



Durch Klicken auf "Anlegen" (1) wird eine neuer logischer Store erzeugt. Danach muss in das Eingabefeld "URL" (2) die in der ini-Datei angegebene URL des Blobservices eingetragen werden und dann auf "Hinzufügen" (3) geklickt werden. Der neu gebaute Blobstore für externe Haltung von Dateiattributen ist danach mit dem Blobservice verknüpft, was durch Klicken auf "Aktualisieren" (4) im unteren Darstellungsbereich kontrolliert werden kann.

Im Bereich "URLs" (2) kann auch eine Liste alternativer URLs per Komma getrennt angegeben werden. i-views bevorzugt bei alternativen URLs wenn möglich eine Verbindung über ein Loopback-Device.

Der Bereich "Löschbare Dateien" (7) zeigt die Zahl der aus Sicht des Wissensnetzes nicht mehr benötigten Dateien an. Mit "Löschen" (8) werden diese im Blob-Service dereferenziert

und ggf. entfernt.

Der Indikator "Intern" (9) zeigt an, dass es sich um einen in einen Mediator integrierten Store handelt. Interne Stores werden bei Volume-Transfers (upload, download, copy, backup, recover) automatisch mit dem Volume transferiert.

Will man die Verknüpfung eines Blobstores zu einem Blobservice aufheben, so selektiert man den gewünschten Blobstore in der Liste "Externe Speicher im Blob-Service" und klickt "Entfernen" (5). Danach kann man den Blobstore im oberen Bereich "Externe Speicher für Dateiattribute" selektieren und durch Klicken auf "Löschen" (6) ganz entfernen, oder man kann durch Angabe einer neuen URL den Blobstore mit einem anderen Blobservice wie oben beschrieben neu verknüpfen.

ACHTUNG!

Durch das Auflösen der Verknüpfung eines Blobstores zu einem Blobservice gehen alle dort gespeicherten Dateien verloren!

4.5.3 SSL certificates

Zur Konfiguration der HTTPS-Verbindung müssen das Zertifikat und der Private-Key abgelegt werden.

Das Zertifikat muss unter **certificates/server.crt** liegen.

Der Private-Key muss unter **private/server.key** liegen. Es ist darauf zu achten, dass server.key als RSA-Key vorliegt, d.h. die erste Zeile der Datei muss
—–*BEGIN RSA PRIVATE KEY*—–
lauten. Wenn der Key in einem anderen Format vorliegt, muss er konvertiert werden. Mittels OpenSSL ist dies bspw. mittels "*openssl rsa -in input.kez -out private/server.key -outform PEM*" möglich.

4.6 Install as OS-service

Die Dienst-Programme können unter den verschiedenen unterstützten Betriebssystemen die Möglichkeit als OS-Dienst eingerichtet zu werden.

Für Unix-artige Betriebssysteme sind die auf der jeweiligen Plattform unterstützten Mechanismen zu verwenden, einige Beispiele finden sich im versions-unabhängigen Handbuch von i-views.

Für MS-Windows bieten die Dienste die Parameter -installAsService NAME und -deinstallService NAME an, um aus einer administrativen Shell heraus einen von Windows verwalteten Dienst einzurichten bzw. zu entfernen. Beim Installieren werden alle Parameter, die nach dem Servicenamen angegeben werden, dem installierten Dienst als Kommandozeilenparameter übergeben. Z.B. richtet

```
bridge -installAsService iviews-bridge-rest -inifile bridge-rest.ini
```

einen Dienst mit dem Namen "iviews-bridge-rest" ein, der als Aufrufzeile

```
PFAD\bridge.exe bridge.exe -serviceName iviews-bridge-rest -ini bridge-rest.ini
```

erhält.