Technisches Handbuch i-views 6.1

Inhalt

1. Knowledge-Builder

Dieses Technische Handbuch umfasst jegliche fortgeschrittene Konfiguration zu Knowledge-Builder, Admin-Tool, Viewconfiguration Mapper und den Services. Die Grundlagen zur Benutzung des Knowledge-Builders sind im Anwenderhandbuch beschrieben.

1.1. Globale Aktionen und Einstellungen

Alle Aktionen und Einstellungen, welche unabhängig sind vom Kontext des Knowledge-Graphen, sind sogenannte "Globale Aktionen" oder "Globale Einstellungen". Diese können in der rechten oberen Ecke des Knowledge-Builders aufgerufen werden, solange der Startbildschirm sichtbar ist oder wenn ein Element auf der linken Seiten im Organizer ausgewählt ist:

- Globales Kontextmenü (Global context menu): Stellt Aktionen für administrative Vorgänge zur Verfügung.
- Globale Einstellungen (Global settings): Enthält nutzerspezifische Einstellungen oder allumfassende Einstellungen, welche nur durch den Administrator geändert werden können.
- Neues Fenster (New window): Dient zum Öffnen eines ausgewählten Inhaltes in einem neuen Fenster (bspw. Import-Mapping, View-Configuration etc.).

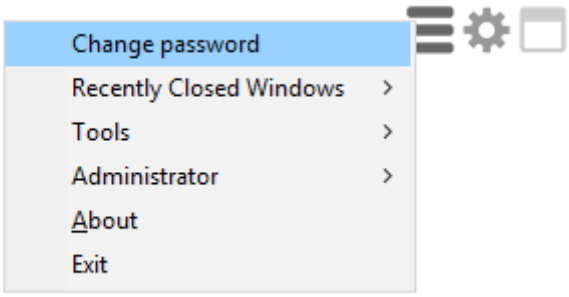
  Vorteile:

  - Die Ansicht geht nicht verloren wenn ein anderer Inhalt im Hauptfenster des Knowledge-Builders ausgewählt wird
  - Die Ansicht wird ohne Organizer geöffnet, wodurch mehr Anzeigefläche zur Verfügung steht

1.1.1. Globales Kontextmenü

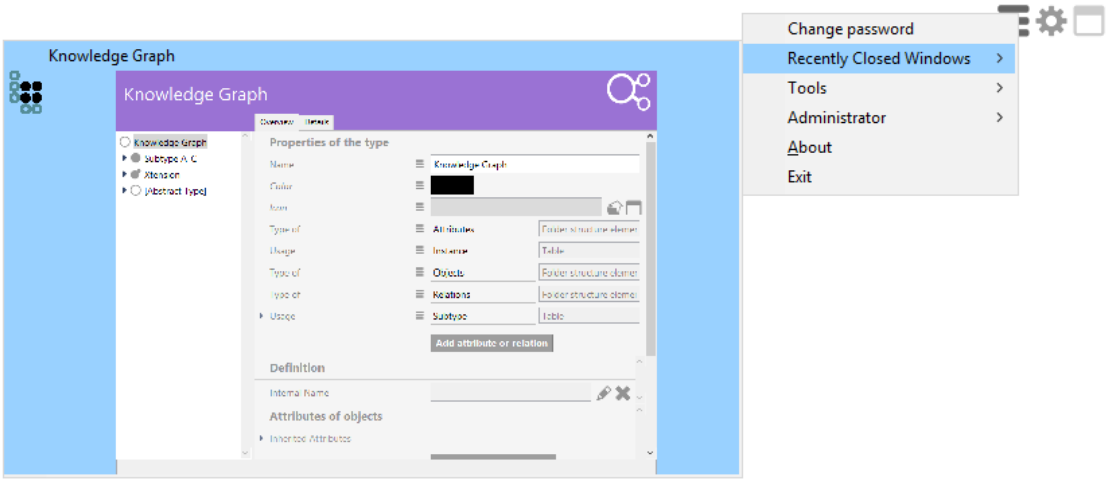
Passwort ändern (Change password)

Für das angemeldete Konto (administrativ und nicht-administrativ) kann hier das Passwort für den Backend-Zugang zum Knowledge-Graph per Knowledge-Builder geändert werden.



Kürzlich geschlossene Fenster (Recently closed windows)

Seit i-views 5.4 ist diese Funktion standardgemäß vorhanden. Kürzlich geschlossene Fenster können wieder geöffnet werden, ohne dass nach der betreffenden Ansicht im Knowledge-Graph gesucht werden
muss.



Werkzeuge (Tools)

Das Werkzeug-Menü enthält folgende Aktionen:

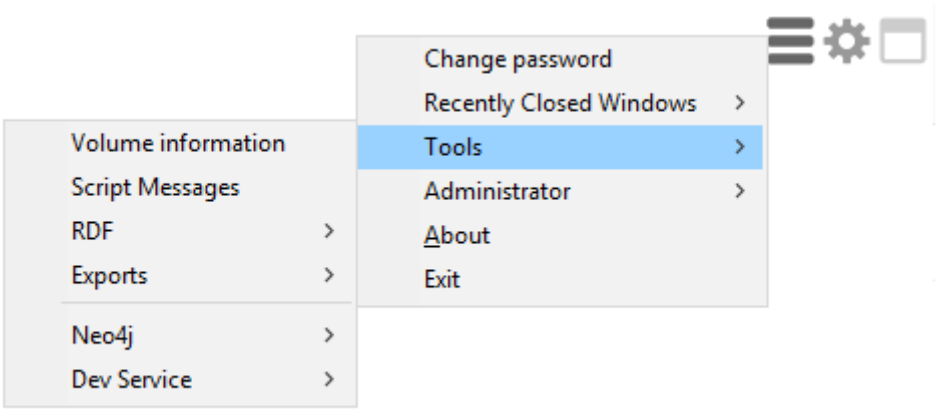
- **Volume-Information (Volume information):** Zeigt ein Dialogfenster mit detaillierten Informationen zu Typen und Instanzen des Knowledge-Graphen an, inklusive der Größe des Volumes, in welchem der Knowledge-Graph abgespeichert ist.
- **Skriptmeldungen (Script messages):** Wenn JavaScript-Code benutzt oder per Debugging untersucht wird, zeigen die Skriptmeldungen die Rückmeldungen an, welche durch die Methode $k.log() im Skrpt ausgegeben werden.

HINWEIS	Die Sichtbarkeit der Skriptmeldungen hängt von der Konfiguration der Bridge ab.

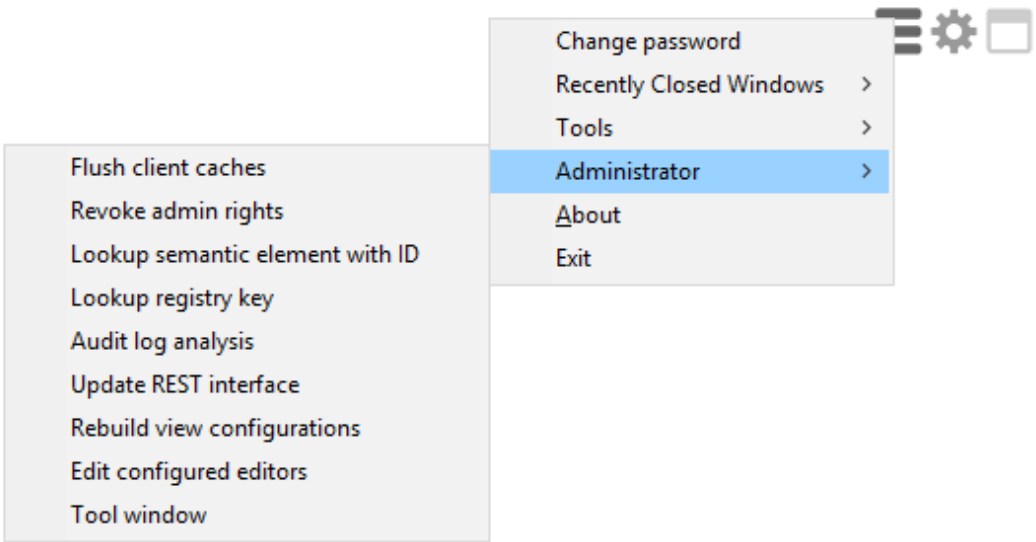
- **RDF:** Enthält die Aktionen "RDF-Import" und "RDF-Export". Für mehr Informationen siehe "RDF-Import und -Export" im Anwenderhandbuch.

- **Exporte (Exports):** Enthält Export-Aktionen zu JavaScript-API, Viewconfig JSON-Schema, REST-API als OpenAPI 2.0 und KScript XML Schema.
- **Dev Service:** Mithilfe des DEV-Services können weitere Tools verwendet werden wie bspw. die i-views Browser-Extension, mit welcher durch Rechtsklick auf Bereiche des i-views Web-Frontends die zugehörigen Elemente/Views/Panels des Viewconfiguration Mappers aufgerufen werden können.

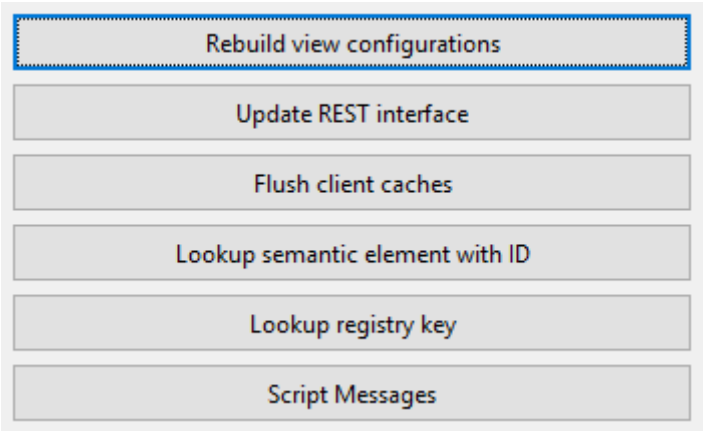
Mithilfe des DEV-Services kann das i-views Browser Extension Tool verwendet werden, um durch Rechtsklick auf einen relevanten Teil der Browser-Oberfläche die entsprechenden Elemente/Views/Panels des Viewconfiguration-Mappers aufzurufen. Die i-views Browser-Extension ist ein gesondertes Werkzeug, welches auf Nachfrage erhältlich ist. Zu beachten ist, dass mehrere gleichzeitig geöffnete Knowledge-Builder den DEV-Service nicht zur selben Zeit verwendet werden können, wenn sie den in den globalen Einstellungen definierten gleichen DEV-Service Port verwenden.



Administrator

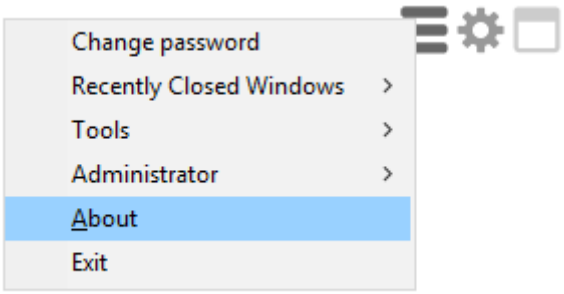


- **Client-Caches zurücksetzen** (Flush client caches): Da der Knowledge-Builder selbst ein Client ist, welcher auf das Knowledge-Graph Volume zugreift, können Daten aus zuvor durchgeführten Transaktionen den Cache blockieren. Ein Zurücksetzen des Client-Caches kann die Reaktionsschnelligkeit des Knowledge-Builders wieder verbessern.
- **Adminrechte entziehen** (Revoke admin rights): Diese Option ermöglicht es einem Administrator, sich die administrativen Rechte temporär zu entziehen, um das Rechtesystem des Knowledge-Builders zu testen. Der administrative Zugriff kann durch Deaktivieren der Option wiederhergestellt werden.
- **Wissensnetzelement mit ID nachschlagen** (Lookup semantic element with ID): Erlaubt das Nachschlagen eines semantischen Elements anhand dessen ID ("= frame ID"). Dies kann bei der Analyse etwaiger Fehlerrückmeldungen hilfreich sein.
- **Registrierungsschlüssel nachschlagen** (Lookup registry key): Ermöglicht eine Suche nach registrierten Objekten im Knowledge-Builder (bspw. registrierte Abfragen, Skripte oder registrierte Typen).
- **Audit-Log-Analyse** (Audit log analysis)
- **REST-Schnittstelle aktualisieren** (Update REST interface): Global verfügbare Aktion zum Aktualisieren der REST-Schnittstelle. Dient als Ersatz wenn der lokale REST-Update Button  aufgrund eines Ansichtswechsels momentan nicht verfügbar (sichtbar) ist.
- **Rebuild view configurations / View-Konfigurationen aktualisieren:** Global verfügbare Aktion zum Aktualisieren der View-Konfiguration; dient als Ersatz wenn der lokale Aktualisierungs-Button  der View-Konfiguartion momentan nicht sichtbar ist.
- **Geöffnete Editoren konfigurieren** (Edit configured editors): Falls Detail-Editoren für Elemente des Knowledge-Graphs konfiguriert sind, können sie hier zentral verwaltet werden.
- **Tool-Fenster** (Tool window): Öffnet ein gesondertes Menüfenster mit oft benötigten Optionen. Dies kann hilfreich sein, wenn viele Fenster zur gleichen Zeit geöffnet sind:



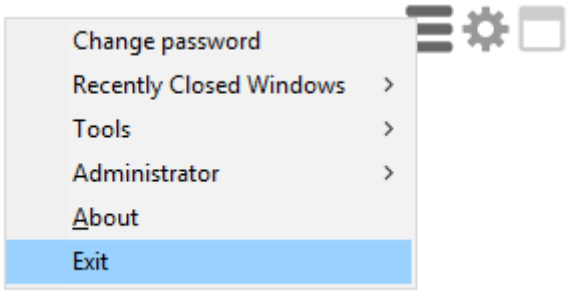
Info (About)

Ruft Informationen zu Konfiguration, Lizenzierung und Komponenten des Knowledge-Graphen auf. Diese Informationen können im Login-Fenster des Knowledge-Builders aufgerufen werden.



Beenden (Exit)

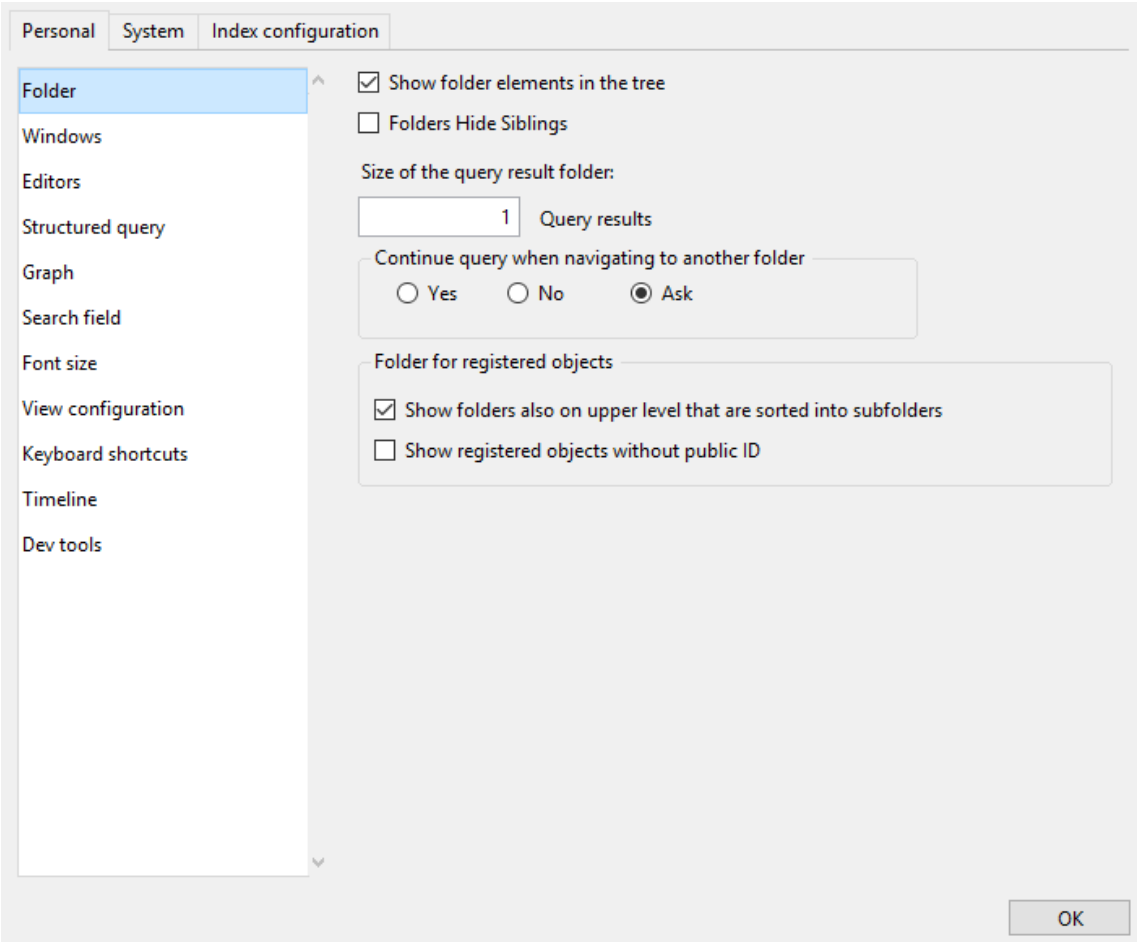
Beendet den Knowledge-Builder.



1.1.2. Persönliche Einstellungen

Persönliche Einstellungen sind exklusiv für den eingeloggten Knowledge-Builder Benutzer verfügbar. Diese Optionen werden in den folgenden Unterkapiteln im Detail beschrieben.

1.1.2.1. Ordner



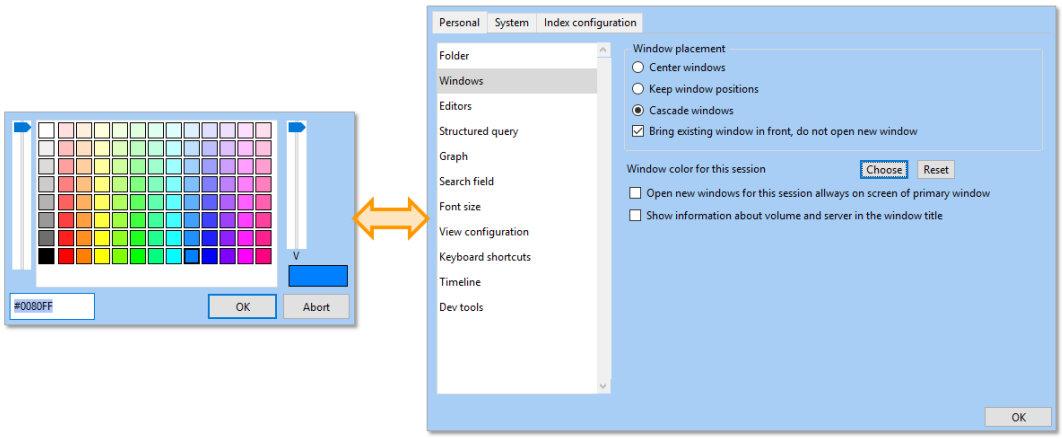
- **Ordnerelemente im Baum anzeigen (Show folder elements in the tree):** Bestimmt, ob der Inhalt des Ordners als Unterknoten im Ordner-Baum angezeigt werden soll. Diese Option ist bspw. nützlich, um bei vielen Ordnerelementen die Übersichtlichkeit des Baums zu verbessern.
- **Geschwisterordner werden ausgeblendet (Folders hide siblings):**
- **Größe des Abfrageergebnis-Ordners (Size of the query result folder):** Anzahl von Suchergebnismengen der zuletzt im KB ausgeführten Strukturabfragen, welche unter ORDNER > Suchergebnisse aufgelistet werden sollen. Ein Suchergebnis-Eintrag besteht aus der mit Zeitstempel versehenen Auflistung der aufgefundenen semantischen Elemente, welche zusammen mit ihren Ursachen im Graph dargestellt werden können. Das reduzieren der Anzahl wirkt sich erst beim Ausführen der nächsten Suche aus.
- **Suchen bei Ordnerwechsel weiter ausführen (Continue query when navigating to another folder):**
- **Ordner, die in Unterordnern einsortiert sind, ebenfalls auf der oberen Ebene anzeigen (Show folders also on upper level that are sorted into subfolders):**
- **Registrierte Objekte ohne öffentliche ID anzeigen (Show registered objects without public ID):**

1.1.2.2. Fenster

Die Fenster-Einstellungen bestimmen das Verhalten des Knowledge-Builders selbst und dessen

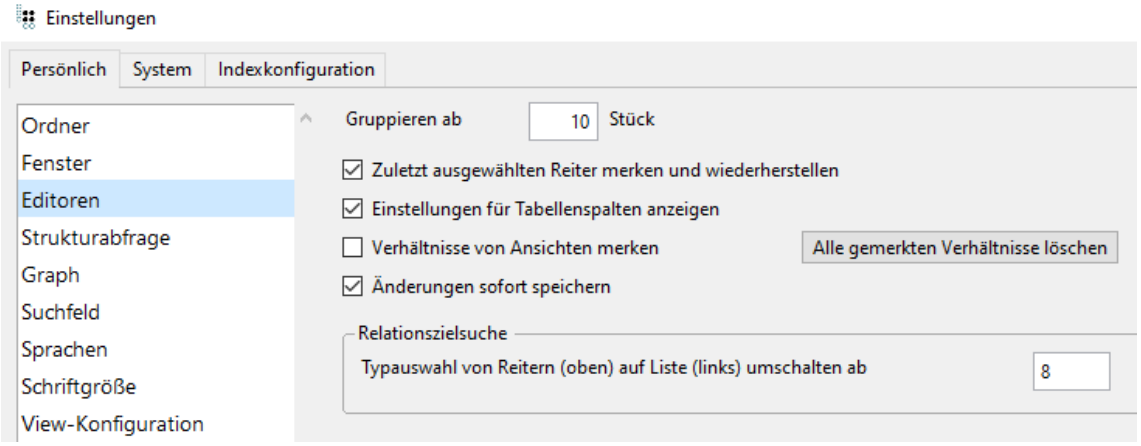
Dialogfenster.

- **Fenster zentrieren (Center windows):** Neue Fenster werden stets auf dem Bildschirm zentriert geöffnet.
- **Fensterpositionen beibehalten (Keep window positions):** Öffnet dasselbe Fenster an derselben Position.
- **Fenster kaskadieren (Cascade windows):** Stapelt alle Fenster desselben Fenstertyps in einer kaskadierenden Art, sodass all ihre Titel auf einmal sichtbar sind.
- **Bestehendes Fenster in den Vordergrund bringen, kein neues Fenster öffnen (Bring existing window in front, do not open new window):** Verwendet Fenster wieder anstatt sie neu zu öffnen, wodurch die Übersichtlichkeit bewahrt bleibt.
- **Fensterfarbe für diese Sitzung (Window color for this session):** Wenn mehrere Knowledge-Builder zur selben Zeit geöffnet sind, ermöglicht diese Option durch Einfärbung der Fenster je Knowledge-Builder und Sitzung eine bessere Unterscheidung zwischen den geöffneten Anwendungen:



- **Neue Fenster für diese Sitzung immer auf dem Monitor des Hatupfensters öffnen (Open new windows for this session always on screen of primary window):** Wenn mehrere Bildschirme verwendet werden, öffnen sich neue Fenster stets auf dem Hauptbildschirm.
- **Im Fenstertitel Informationen über den Knowldege Graph und Server anzeigen (Show information about volume and server in the window title):** Um die geöffneten Fenster aus mehreren Knowledge-Builder Anwendungen inhaltlich voneinander unterscheiden zu können, werden Knowledge-Graph Volume-Name und Server in allen Fenstertiteln mit angezeigt. Dient wie die Einfärbung der Fenster zur besseren Übersichtlichkeit.

1.1.2.3. Editoren



Gruppieren ab […] Stück

Diese Option bewirkt ein Bündeln von Eigenschaften in einem Dropdown-Akkordeon, wenn die Anzahl der Eigenschaften in einem Detail-Editor die angegeben Zahl überschreitet.

Zuletzt gewählten Reiter merken und wiederherstellen

Ermöglicht das wiederholte Anzeigen des Detail-Editors mit demselben, zuletzt gewählten Reiter innerhalb derselben Sitzung.

Einstellungen für Tabellenspalten anzeigen

Wenn aktiviert, werden bei allen Tabellen, rechts neben den Filtern, der Knopf zum zurücksetzen der Filter durch ein Menü ersetzt. Über dieses Menü können dann die Filter zurückgesetzt werden. Außerdem kann hier konfiguriert werden, welche Splaten diese Tabelle anzeigen soll. Dort kann auch definiert werden, wie lange diese Konfiguration wirkt: Bis die Ansicht geändert wird, der KB geschlossen wird oder bis sie wieder geändert wird.

Verhältnisse von Ansichten merken

Wenn aktiviert, speichern die meisten Ansichten, die man verschieben kann, ihre Verhältnisse, um sie bei erneutem Öffnen wiederherzustellen.

Alle gemerkten Verhältnisse löschen

Setzt alle Verhältnisse zurück, welche durch die vorige Option gespeichert wurden.

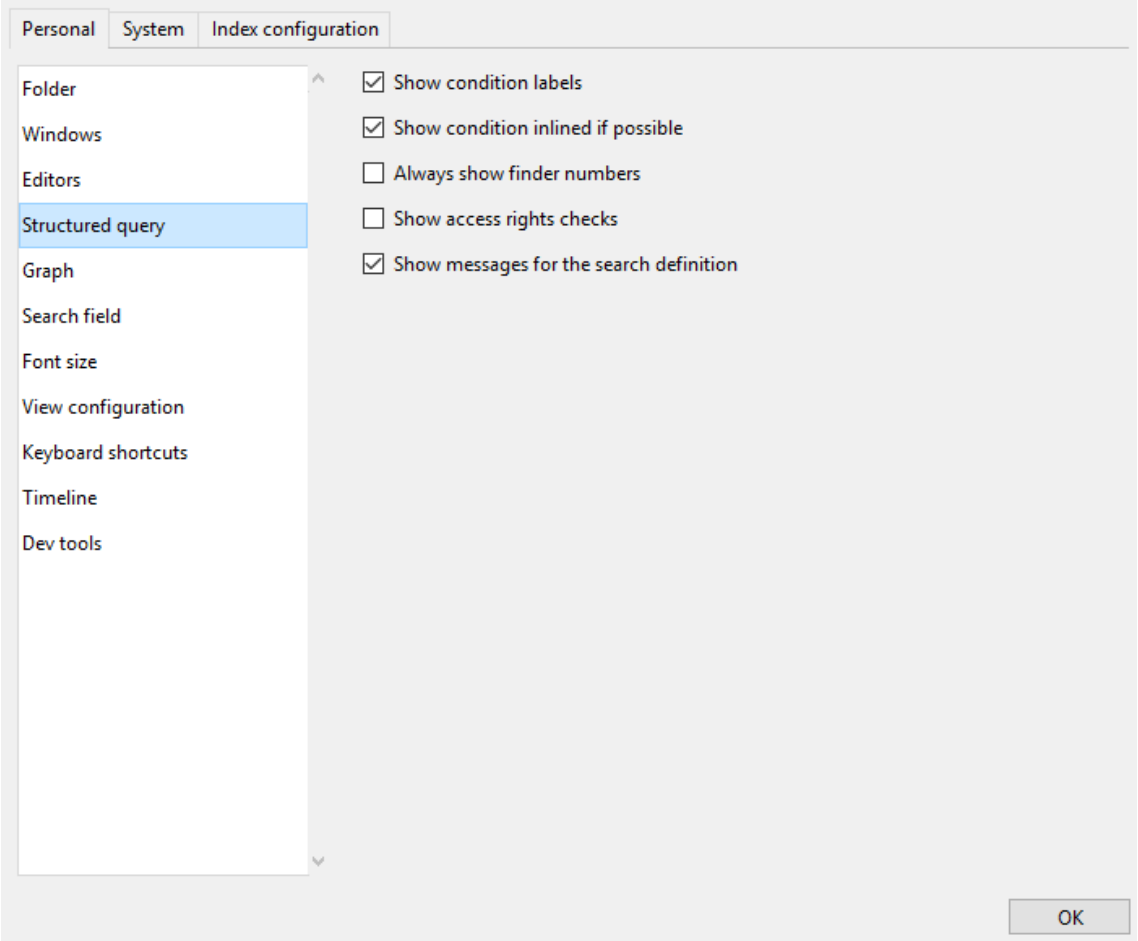
Änderungen sofort speichern

Diese Option wirkt sich nur auf das Backend (Knowledge-Builder) aus. Wenn Elementeigenschaften editiert werden, werden die Änderungen normalerweise sofort in den Knowledge-Graphen geschrieben. Wenn diese Option aktiviert ist, können Elementeigenschaften editiert werden, ohne dass die Änderungen sofort in den Knowledge-Graph geschrieben werden, damit sie zuvor gegen Schema-Regeln validiert werden können. In diesem Fall erscheint ein "Anwenden"-Button am unteren Rand der Editor-Ansicht. Im Web-Frontend hingegen dienen Aktionen (Buttons) mit dem Aktionstyp "Validieren" oder dem Aktionstyp "Speichern" diesem Zweck.

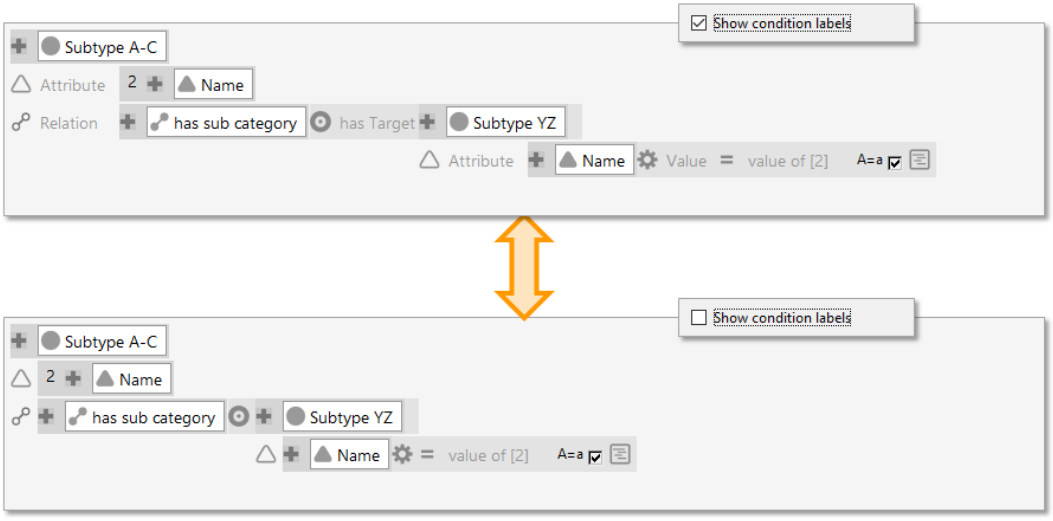
Typauswahl von Reitern (oben) auf Liste (links) umschalten ab […]

Wenn der Relationsziel-Auswahldialog geöffnet wurde, um eine Relation zu bearbeiten, dann werden die Relationen normalerweise durch Reiter im oberen Rand getrennt aufgelistet. Diese Option bestimmt die Anzahl an Relationen, ab welcher diese in Form von einzelnen Kategorien am linken Rand dargestellt werden.

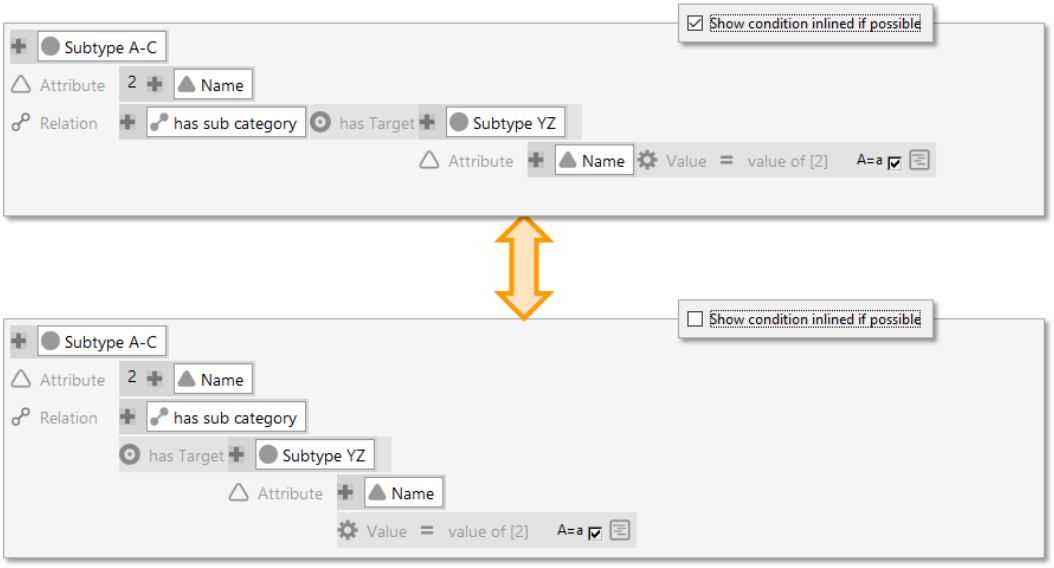
1.1.2.4. Strukturabfrage



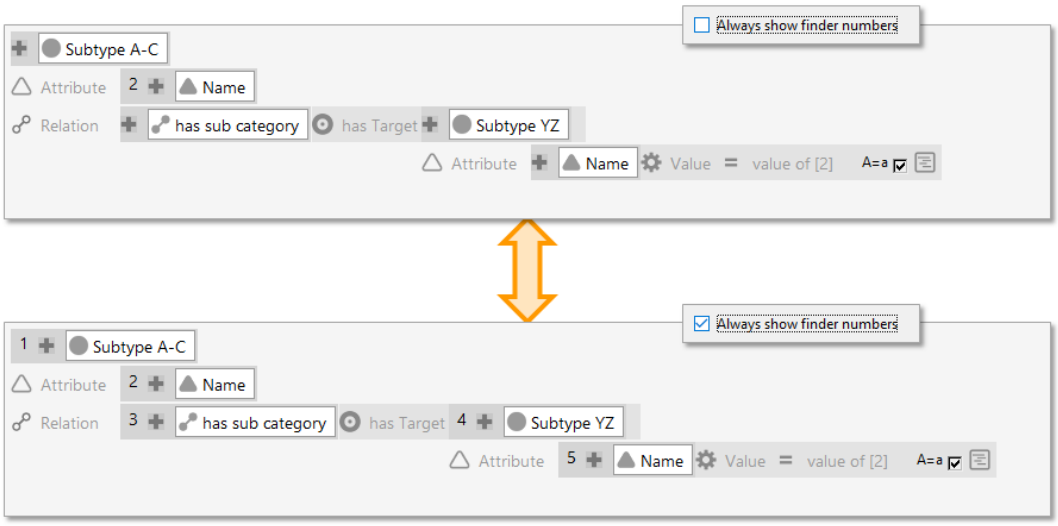
- **Beschreibung der Bedingungen anzeigen (Show condition labels):** Wenn aktiviert, werden die Beschriftungen für Eigenschaften zusätzlich zum Symbol angezeigt:



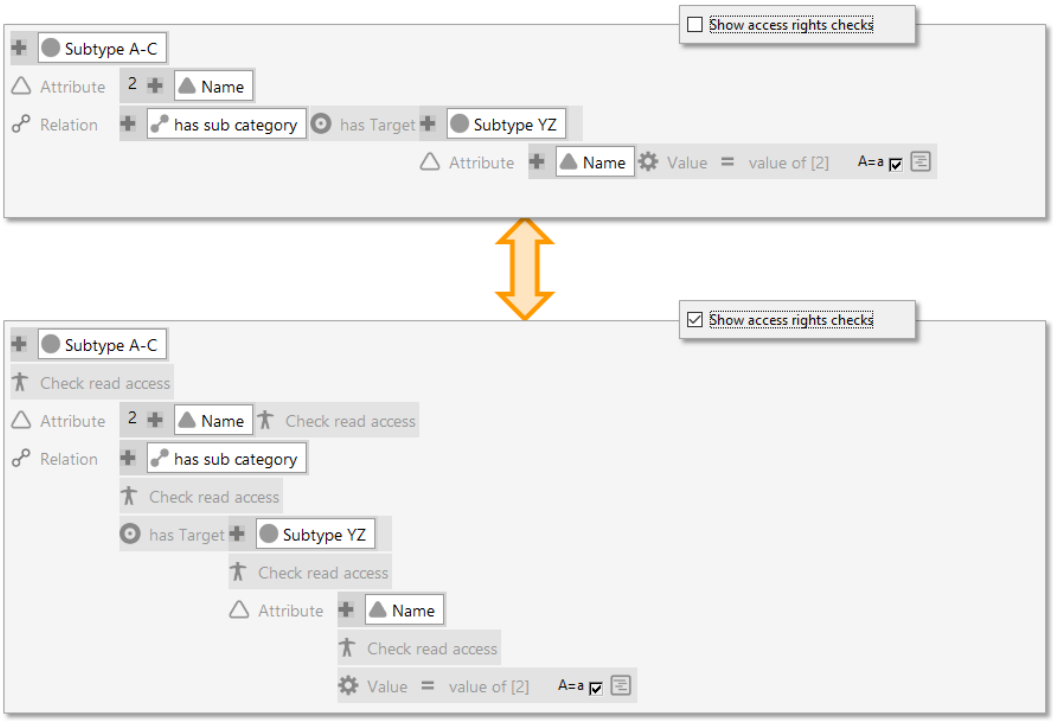
- **Bedingungen wenn möglich einzeilig anzeigen (Show condition labels inlined if possible):** Wenn aktiviert, werden Relationsziele und Attributwerte bevorzugt in einer Reihe mit ihren Eigenschaftstypen angezeigt statt kaskadiert:



- **Teilsuchennummer immer anzeigen (Always show finder numbers):** In Strukturabfragen werden alle Elemente mihilfe einem inhärenten Nummerierungssystem identifiziert. Normalerweise wird die Nummer eines Elements nur dann angezeigt, wenn ein anderes Element sich darauf bezieht, wie bspw. eine hinzugefügte Ergebnisspalte in der Suchergebnisliste. Wenn diese Option aktiviert ist, wird die Nummerierung persistent angezeigt:

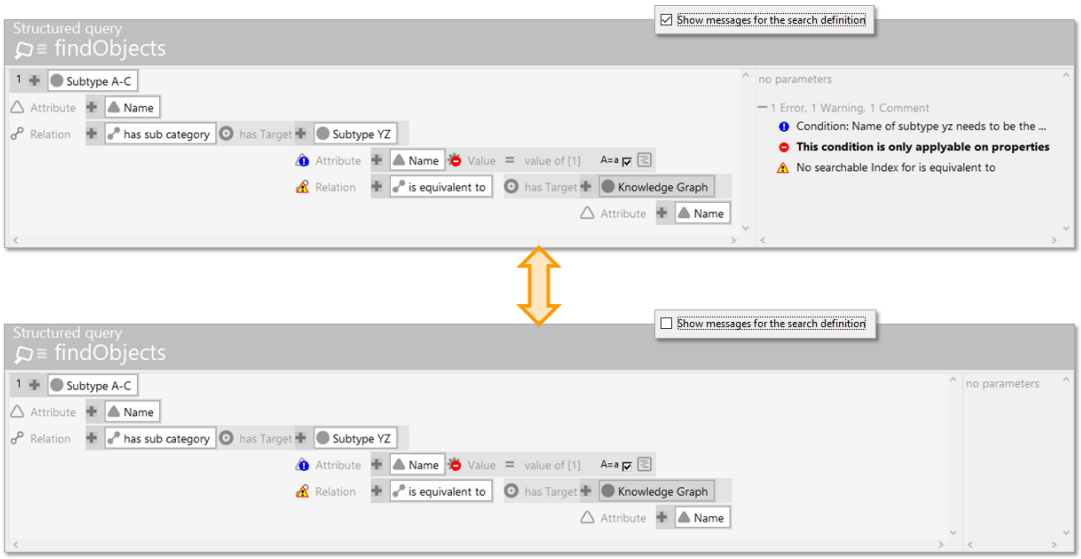


- **Leserechtprüfungen anzeigen (Show access rights checks):** Zeigt zusätzlich die Zugriffsrechte betreffend der jeweiligen Eigenschaft an.



- **Meldungen zur Suchdefinition anzeigen (Show message for the search definition):** Diese Option ermöglicht das Anzeigen von Meldungen für Kommentare, Warnungen und Fehlern in der Legende am rechten Rand der Strukturabfrage.

HINWEIS	Zusätzlich zu dieser globalen Einstellung ist die Option "Warnung unterdrücken" per Kontextmenü am jeweiligen Abfrage-Label lokal verfügbar.



1.1.2.5. Graph

Die Graph-Optionen gelten ausschließlich für den Graph-Editor des Knowledge-Builders. Für Einstellungen des Graphen in Form der Net-Navigator Komponente siehe "vcm-plugin-net-navigator" im technischen Handbuch.

Einige dieser Einstellungen können auch im Graph-Editor direkt überschrieben werden, gelten dann aber nur für dieses spezielle Graph-Editor-Fenster.

Tooltips mit Details anzeigen

Wenn angekreuzt, wird, wenn der Mauszeiger über einen Knoten gehalten wird, ein Fenster mit den Eigenschften des Objekts angezeigt.

Max. Anzahl Zeilen für Tooltips

Bestimmt nach wie vielen Zeilen das Fenster abgeschnitten wird.

Knoten automatisch ausblenden

Blendet automatisch überschüssige Knoten aus, sobald mehr als die gewünschte Anzahl an Knoten sichtbar ist. Die Anzahl kann im Eingabefeld "max. neue Knoten" in der Symbolleiste eingestellt werden.

Knoten automatisch positionieren

Führt für neu eingeblendete Knoten automatisch die Layoutfunktion aus.

Positionierungshilfe

Sorgt dafür, dass Knoten, die im Umkreis von anderen Knoten bewegt werden, auf deren x oder y Koordinate einrasten, wenn sie in deren Nähe kommen.

Cairo-Bibliothek zur Darstellung verwenden

Kann nur aktiviert werden, wenn die Cairo Bibliothek der KB Anwendung beigelegt ist. Wenn angekreuzt, wird diese Bibliothek zur Darstellung bestimmter Dinge verwendet.

Standard Zoom

Das Zoomlevel mit dem der Graph-Editor sich öffnet.

Max. neue Knoten

Wenn ein Knoten/Objekt viele Nachbarobjekte hat, ist es oft nicht sinnvoll, alle beim Klick auf den Anfasser gleich einzublenden. Hiermit wird definiert, wie viele neue Knoten auf einmal ohne Nachfrage eingeblendet werden können.

Max. Textlänge

Definiert, nach wie vielen Zeichen die Labels von Knoten abgeschnitten werden.

Knotengröße

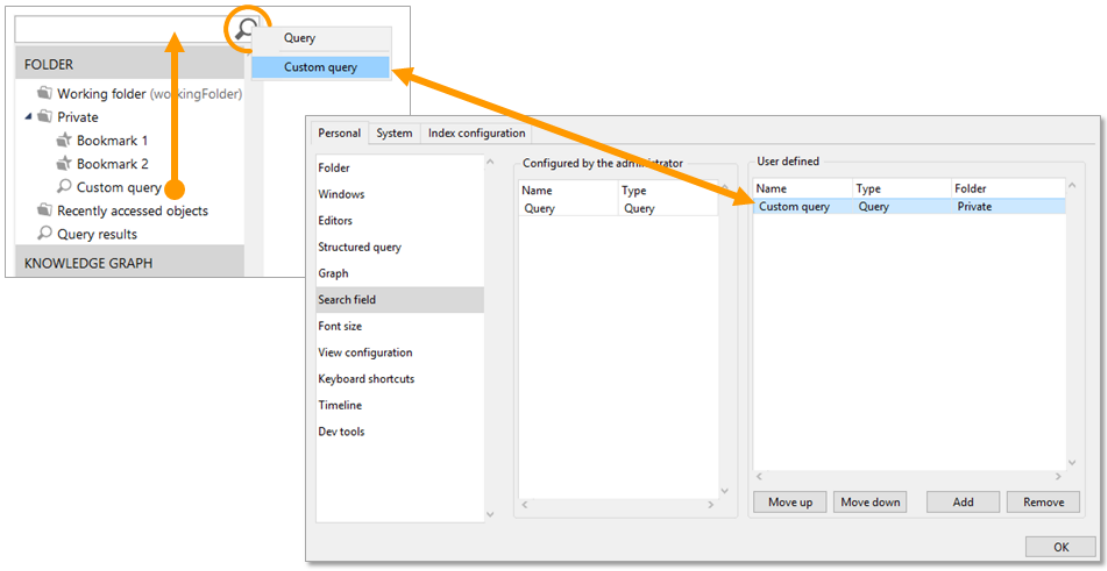
Bestimmt mit welcher Größe Knoten einem Graph hinzugefügt werden.

Konfiguration der Legende

Hier können Typen definiert werden, welche immer beim Öffnen eines Graph-Editors in der Legende angezeigt werden.

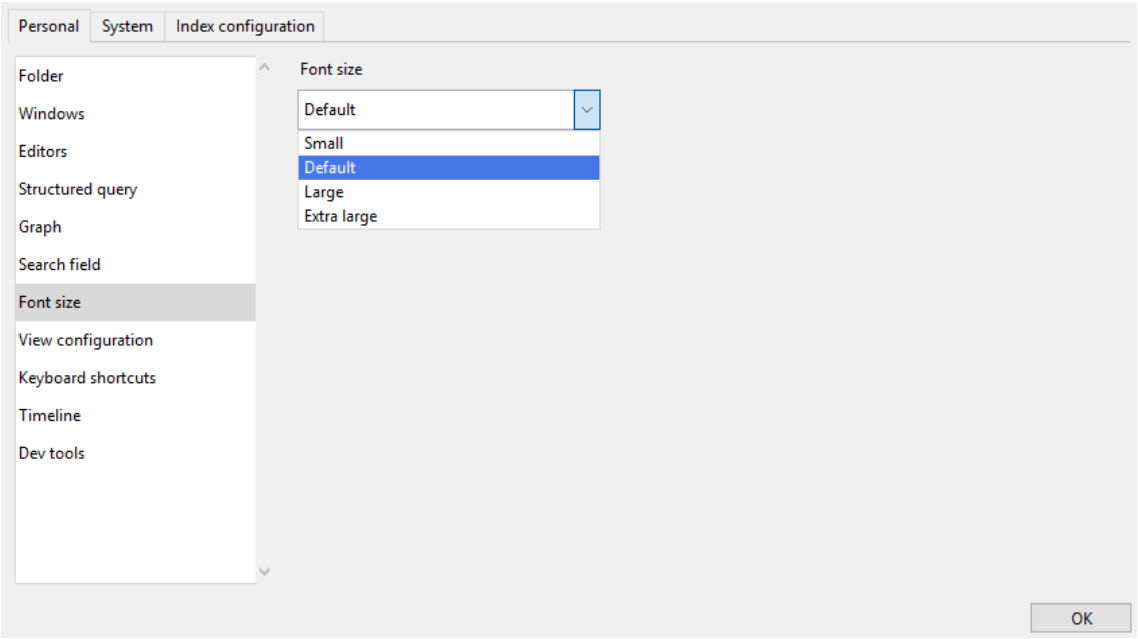
1.1.2.6. Suchfeld

Für das Suchfeld des Knowledge-Builders können Abfragen aus dem Arbeitsordner oder dem privaten Ordner per Drag&Drop hinzugefügt werden. Die Suchfeld-Einstellungen dienen zur Verwaltung der Abfragen (bspw. um sie wieder zu entfernen). Hinzugefügte Abfragen stehen in einer Dropdown-Auswahl zu Verfügung, das durch Klick auf den Abfrage-Button erscheint:



1.1.2.7. Schriftgröße

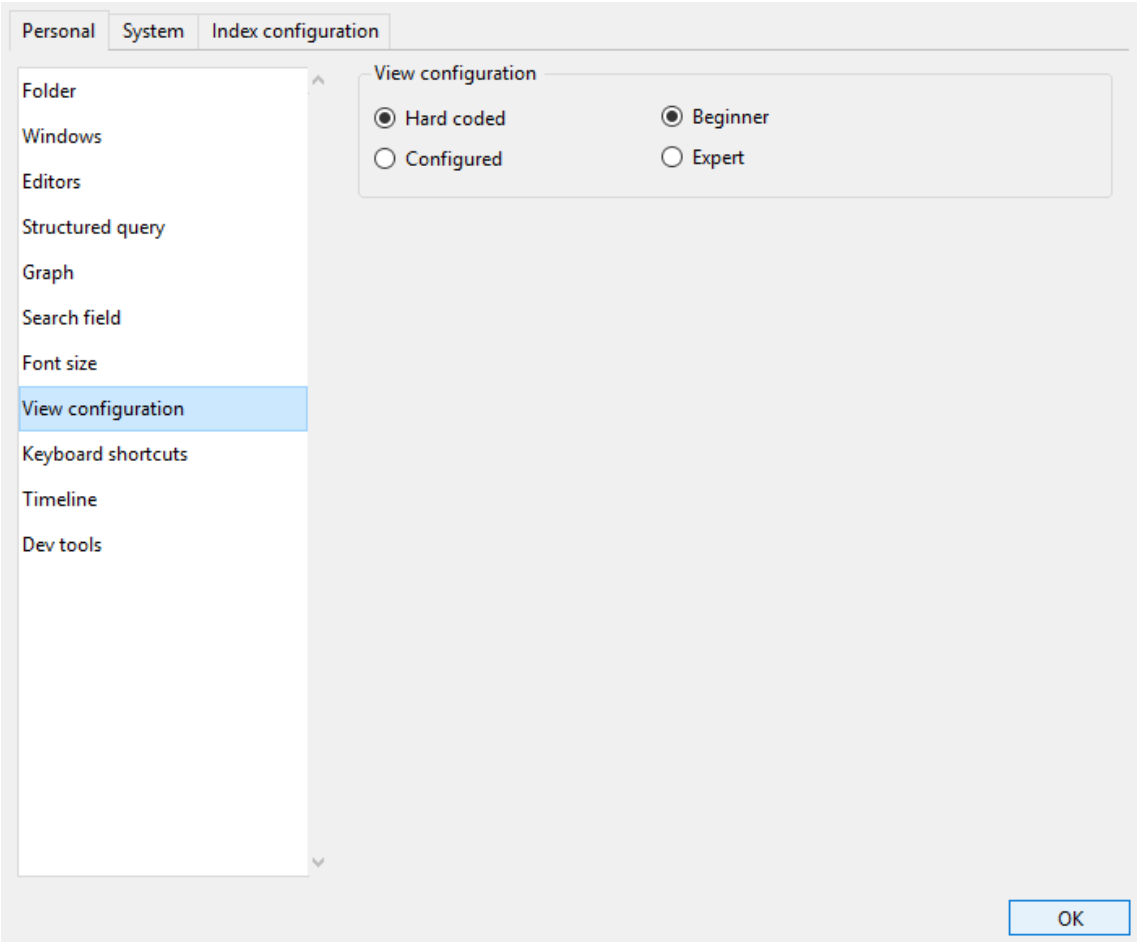
Diese Option ermöglicht es, die Schriftgröße für den Knowledge-Builder zu ändern. Wenn die Schriftgröße geändert wurde, wird ein Beispieltext hierzu angezeigt. Die Änderungen der Schriftgröße werden erst nach einem Neustart des Knowledge-Builders wirksam und bleiben permanent erhalten.



1.1.2.8. View-Konfiguration

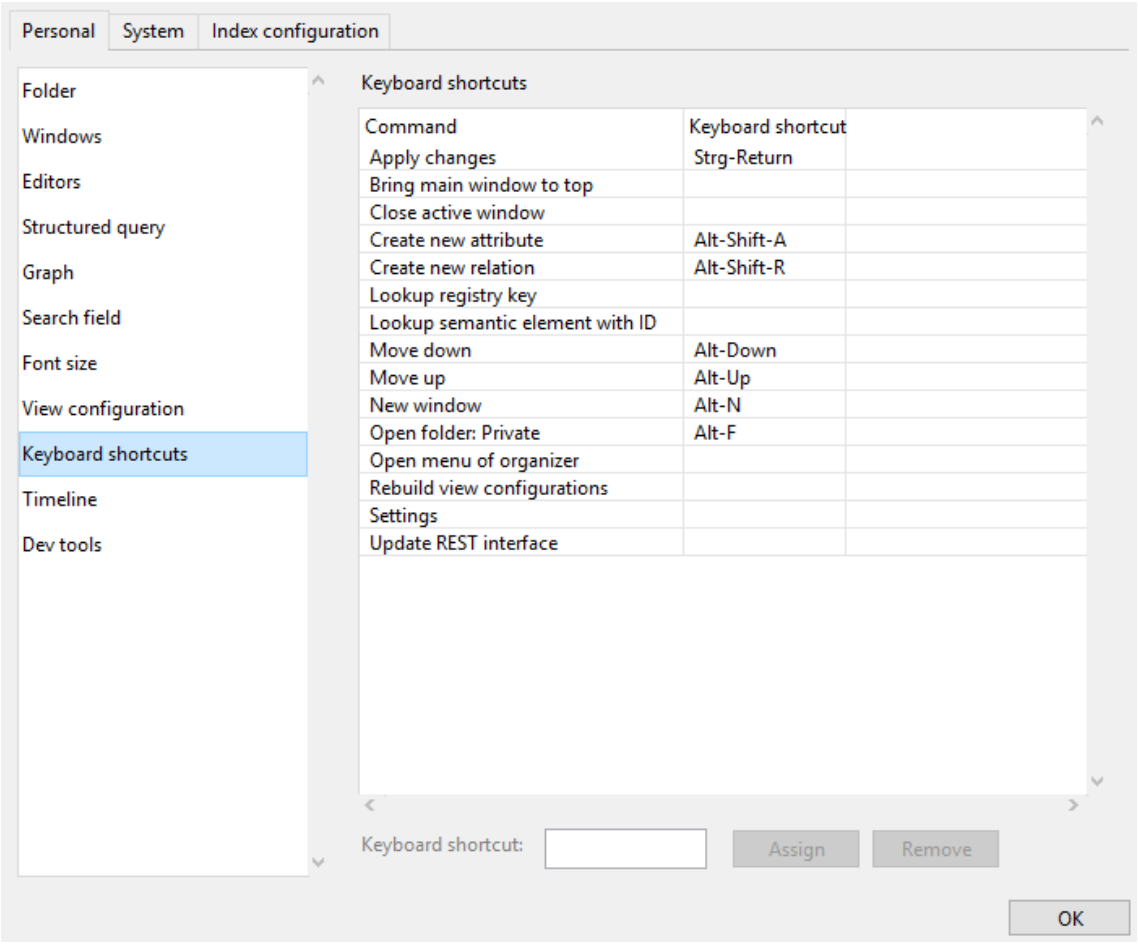
Die View-Konfigurations-Optionen wirken sich ausschließlich auf das Verhalten der View-Konfiguration des Knowledge-Builders. Optionen für die View-Konfiguration des Web-Frontends werden mithilfe der Einstellungen des Viewconfiguration-Mappers konfiguriert.

- **Fest vorgegeben / Konfiguriert (Hard coded / Configured):** Für die Ordnerstruktur innerhalb des Organizers des Knowledge-Builders können typabhängige View-Konfigurationen oder auch rollenspezifische View-Konfigurationen erstellt werden. Die Option "Fest vorgegeben" (Hard coded) und "Konfiguriert" (Configured) erlauben das Umschalten zwischen der Standardansicht und der konfigurierten Ansicht des Knowledge-Builders. Wenn bestimmte Typen eine View-Konfiguration enthalten, welche für die Detailansicht und für die Ordnerstruktur gleichermaßen definiert wurden, dann wird beim Umschalten auf "Konfiguriert" die Darstellung in der Ordnerstruktur vorrangig angezeigt.
- **Anfänger/Experte (Beginner/Expert):** Betreffend des Viewconfiguration-Mappers gibt es zwei Arten der nutzerorientierten Ansicht: "Anfänger" (Beginner) bewirkt eine Aufteilung der Konfigurationsreiter des Detaileditors in "Konfiguration" und "Erweitert"; die Option "Experte" (Expert) listet alle Konfigurationsoptionen unter einem Reiter auf.



1.1.2.9. Tastaturkürzel

Zur Erleichterung der Bedienung können Tastaturkürzel für Aktionen definiert werden wie in der folgenden Abbildung dargestellt:



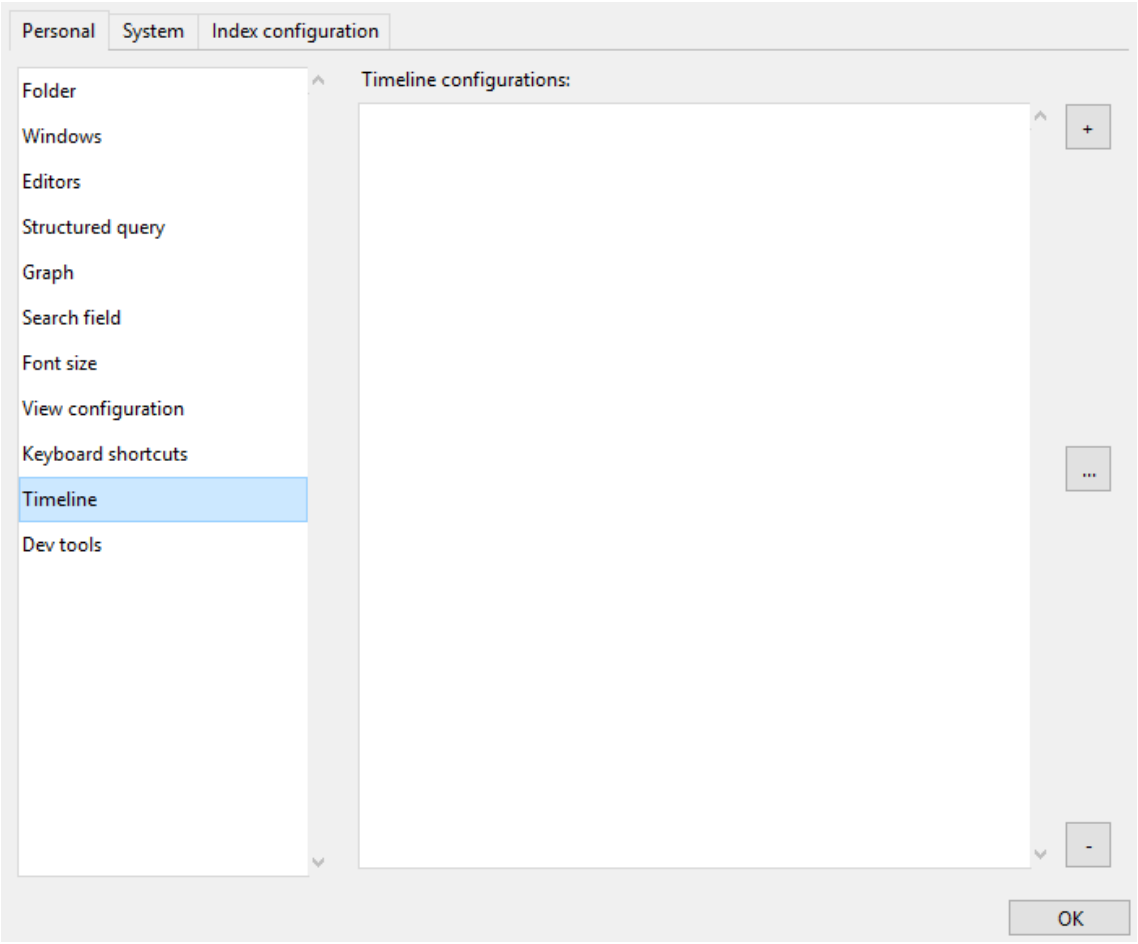
Oftmals sind auch inhärente Tastaturkürzel vorhanden. Wenn diese verfügbar sind, werden sie in Form des Hinweises **Shortcut** im betreffenden Kapitel beschrieben.

Innerhalb des Knowledge-Builders gibt es ein allgemein gültiges Prinzip zu Tastaturkürzeln: Die Kombination von [Strg] + Klick entfernt Elemente oder blendet diese aus (z. B. Entfernen von Elementen in einer Strukturabfrage, Entfernen von Eigenschaften in einem Detail-Editor oder Ausblenden von Elementen im Graph-Editor).

Im JavaScript-Editor kann die Detailansicht zu einem referenzierten Element per [Strg] + o aufgerufen werden (vorausgesetzt, dass der Schlüssels oder Konfigurationsname im Graph registriert ist). In einem JavaScript-Code kann zwischen gleichlautenden Begriffen gewechselt werden durch Markieren des Begriffes und anschließendem Drücken der Tastenkombination [Strg] + g.

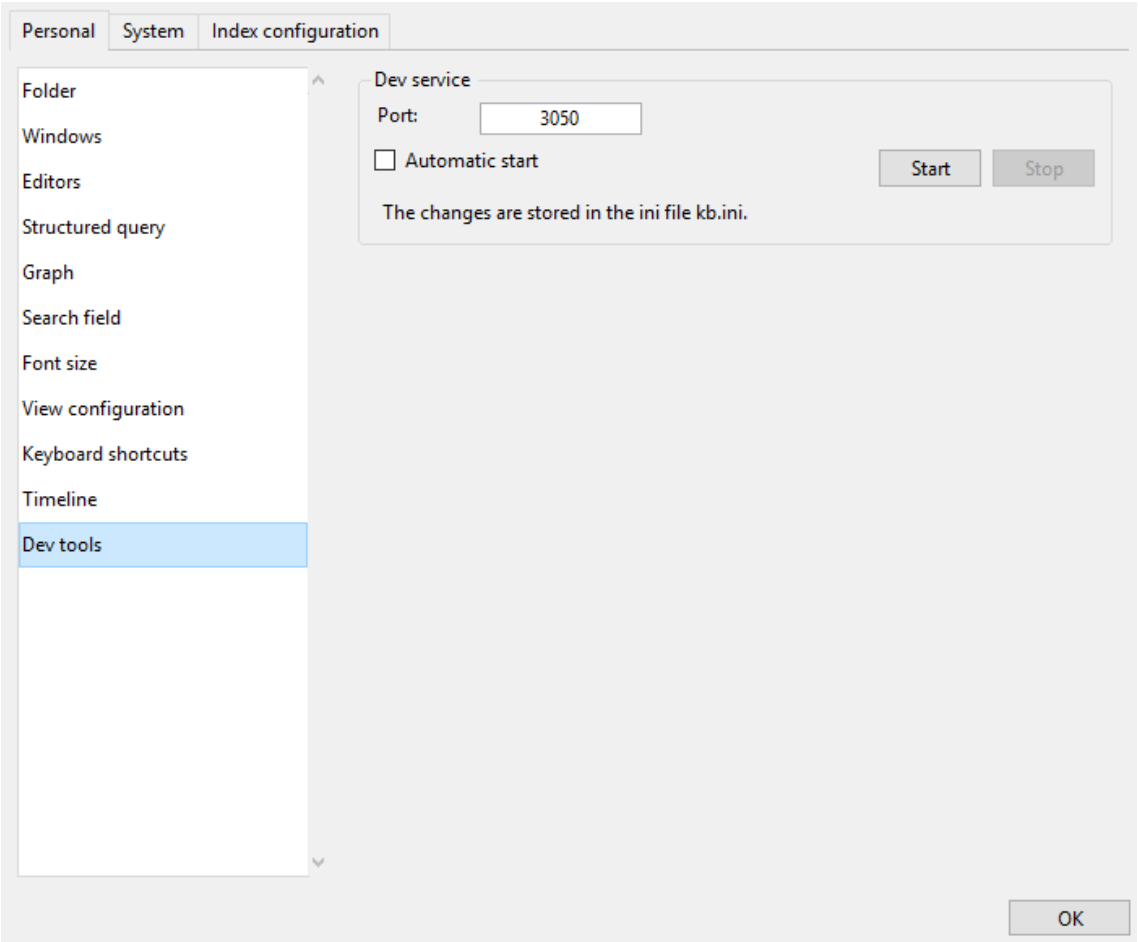
1.1.2.10. Timeline

Die Timeline-Funktion erlaubt das Konfigurieren einer Zeitstrahl-Ansicht für den Knowledge-Builder mithilfe einer Strukturabfrage. Für die Timeline können mehrere Elementtypen als Dimension angegeben werden, damit die Instanzen anhand von Datumswerten, Zeitangaben oder Zeitintervallen angezeigt werden können. Die Timeline-View muss dann als View-Konfiguration für den Knowledge-Builder definiert werden, damit sie angewendet werden kann.



1.1.2.11. Dev-Tools

Diese Option ermöglichen die Einstellung des Ports für den Dev-Service und ob der Dev-Service mit dem Start des Knowledge-Builders gestartet werden soll. Wenn die Dev-Services mehrerer Knowledge-Builder zur selben Zeit verwendet werden sollen, dann sind für die jeweiligen Dev-Service individuelle Portnummern zu vergeben.



1.1.3. System-Einstellungen

Die System-Einstellungen sind ausschließlich für Benutzer mit Administrator-Status verfügbar und ermöglichen eine übergreifende Konfiguration systemweiter Einstellungen des Knowledge-Builders.

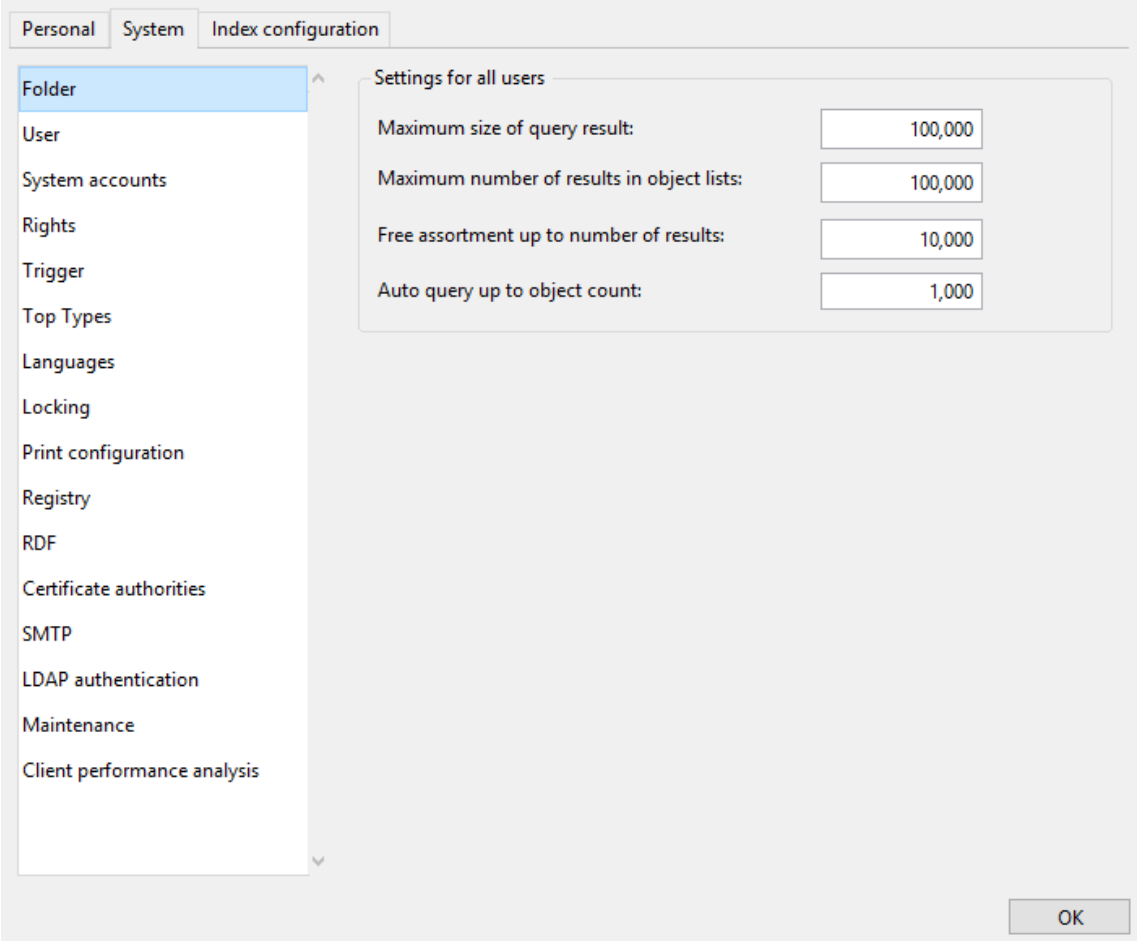
1.1.3.1. Ordner

Die Ordner-Optionen dienen zur Optimierung der Listenansichten für bestimmte Anwendungszwecke wenn mit großen Datenmengen umgegangen werden muss. Durch Limitierung zusätzlicher, nicht benötigter Optionen kann eine Verbesserung der Performance und somit eine Verbesserung der Usability erreicht werden.

- **Maximale Anzahl der Abfrageergebnisse (Maximum size of query result):** Bestimmt die maximale Anzahl an Treffern, die bei der Anzeige der Abfrageergebnisse aufbereitet und gerendert werden.
- **Maximale Ergebnisanzahl in Objektlisten (Maximum number of results in objects lists):** Bestimmt die maximale Anzahl an Objekten, welche in einer Objekteliste anzeigbar ist. Wenn die Anzahl den Grenzwert überschreitet, wird anstatt der Objekte ein entsprechender Hinweis in der Objektliste angezeigt.
- **Freie Sortierung bis Ergebnisanzahl (Free sorting up to number of results):** Die Einträge der

Objektlisten können sortiert werden durch Klick auf die Spaltenüberschriften und/oder durch Festlegen von Spaltenfilter-Optionen an der Spalte. Für große Objektmengen kann die Sortierung deaktiviert werden, um unnötige Belastungen des Systems zu verhindern.

- **Automatische Abfrage bis Anzahl Objekte (Auto query up to object count):** Bestimmt die Anzahl an Listenobjekten, bis zu welcher die Listenabfragen automatisch ausgeführt werden sollen. Wenn die Anzahl der anzuzeigenden Objekte den Grenzwert überschreiten, dann wird die Abfrage nur ausgeführt wenn der Nutzer die Suche durch Klick auf den Suche-Button aktiviert. Darüber hinaus sind für Objektlisten im KB in den Tabellen-Konfigurationen gesonderte Optionen zur automatischen Ausführung der Suche verfügbar (Reiter "KB").



1.1.3.2. Benutzer

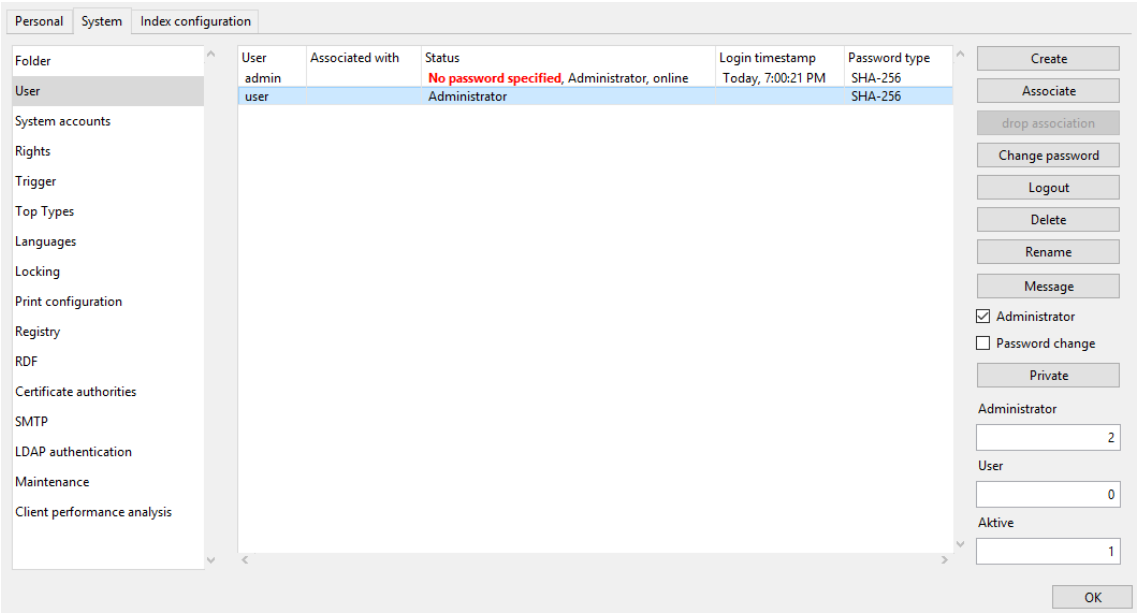
Diese Options-Kategorie dient zur Administrierung der Backend-Nutzer, welche per Knowledge-Builder Zugriff zum Knowledge-Graphen erhalten.

- **Erstellen (Create):** Erstellt einen Backend-Nutzer für den Knowledge-Builder.
- **Verknüpfen (Associate):** Verknüpft den Backend-Nutzer mit einem Frontend-Nutzer-Kontenobjekt.
- **Verknüpfung aufheben (Drop association):** Entfernt wieder die Zuweisung des Backend-Nutzers zum Frontend-Nutzer-Kontenobjekts.

- **Passwort ändern (Change password):** Ermöglicht die Änderung bzw. das Zurücksetzen des eigenen Passwortes oder des Passwortes eines anderen Backend-Nutzers. Zusätzlich kann eine Passwortänderung beim ersten/nächsten Login erzwungen werden.
- **Abmelden (Logout):** Führt zum Log-out des gewählten Benutzers.
- **Löschen (Delete):** Entfernt das Nutzerkonto des gewählten Nutzers. **Achtung:** Das Löschen des eigenen Nutzerkontos ist gleichermaßen möglich, was zu einer sofortigen Löschung und gleichzeitigem Logout führt!
- **Umbennen (Rename):** Dient zu Umbenennung des gewählten Nutzers.
- **Mitteilung (Message):** Sendet eine Mitteilung zum gewählten Nutzer, ähnlich dem Senden einer Nachricht über die Community-Funktion in der linken, unteren Ecke des Knowledge-Builders. Wenn die Person derzeit abgemeldet und deshalb nicht erreichbar ist, kann trotzdem eine Nachricht gesendet werden, die bei der nächsten Anmeldung erscheint.
- **Administrator:** Bestimmt, ob der gewählte Nutzer ein Administrator ist.

HINWEIS	Damit Nutzer ohne Administratorrechte einen Zugriff auf die jeweils benötigten Inhalte und Funktionen erhalten, muss zuvor eine gesonderte View-Konfiguration für die KB-Ordnerstruktur eingerichtet werden.

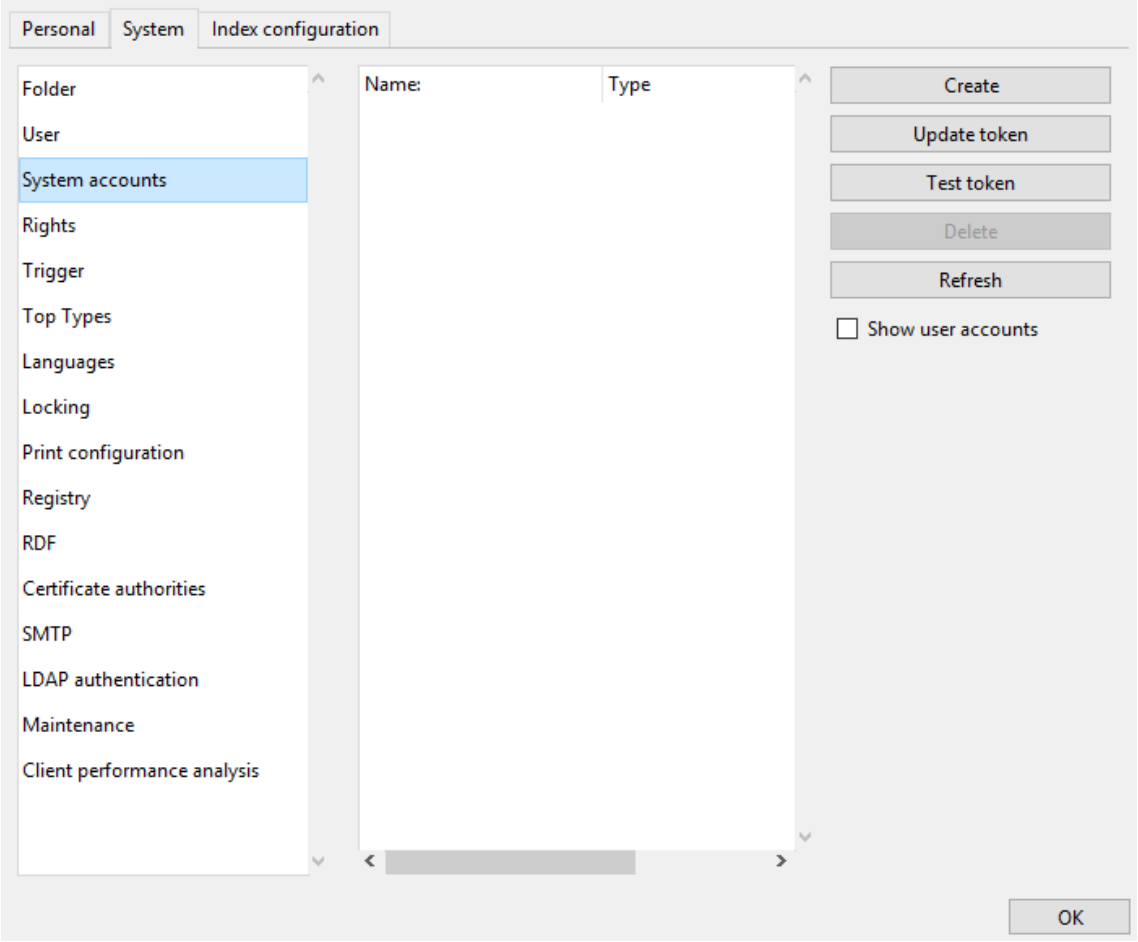
- **Passwortänderung (Password change):** Erzwingt eine Passwortänderung des jeweiligen Accounts beim nächsten Login.
- **Privatordner (Private):** Zeigt den Inhalt des Privatordners des gewählten Nutzerkontos.
- **Administrator:** Zeigt die Anzahl der Nutzer mit Administrator-Status.
- **Benutzer (User):** Zeigt die Anzahl der Nutzer mit herkömmlichem (nicht-administrativem) Nutzer-Status.
- **Aktive (Active):** Zeigt die Anzahl der aktuell eingeloggten Nutzer/Administratoren.



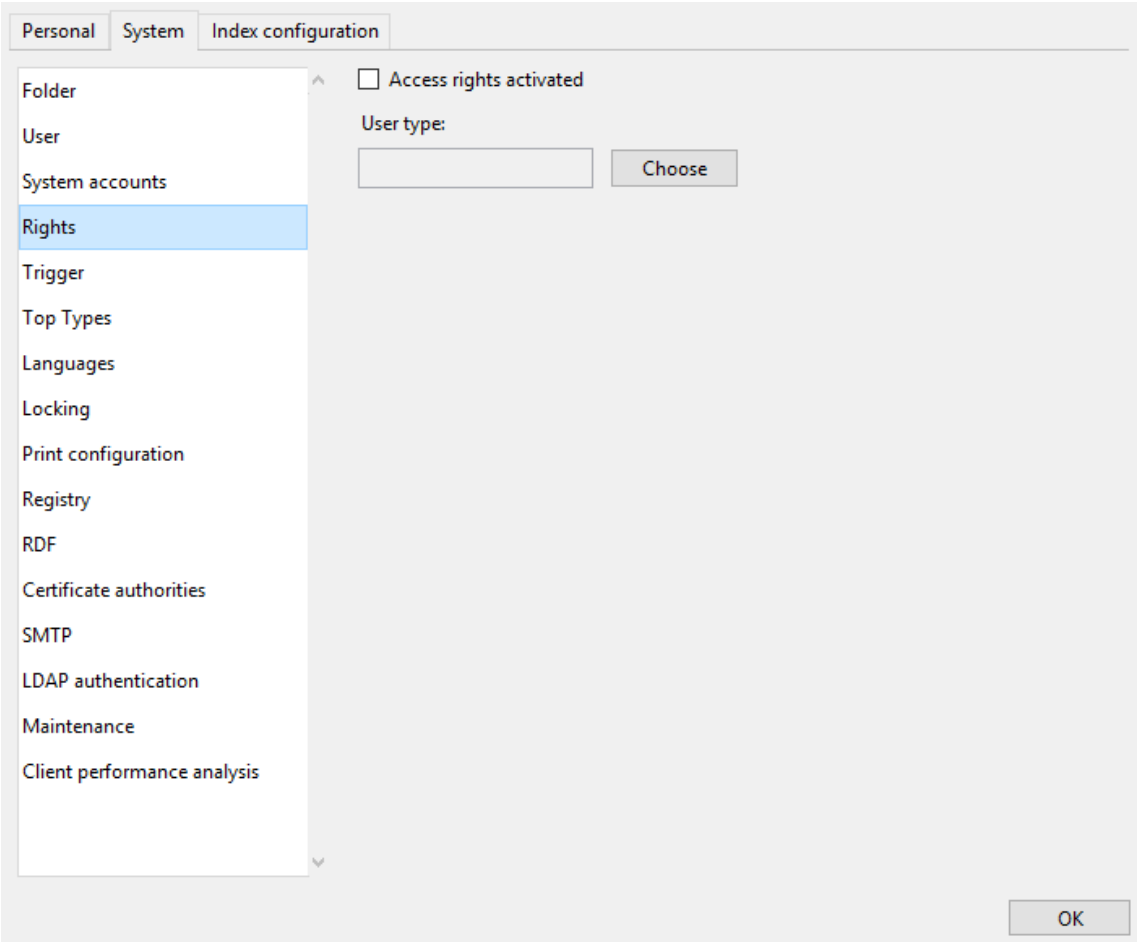
1.1.3.3. System-Konten

Systemkonten werden benötigt für die Authentifizierung von externen Services, welche per TCP/IP oder REST-Schnittstelle angeschlossen werden (bspw. Brigde für Web-Frontend).

- **Erstellen (Create):** Erzeugt ein Systemkonto; nach dem Festlegen eines Namen wird einmalig ein Token angezeigt, welcher für die weitere Benutzung herauskopiert werden kann (bspw. für Bridge *.ini-Dateien).
- **Token erneuern (Update token):** Erneuert den Token und zeigt den neuen Token einmalig an, damit er kopiert werden kann.
- **Token überprüfen (Test token):** Ermöglicht das Testen eines Token, ob dieser noch gültig ist.
- **Löschen (Delete):** Löscht das gewählte Systemkonto.
- **Aktualisieren (Refresh):** Lädt die Systemkonten-Ansicht neu.
- **Benutzerkonten anzeigen (Show user accounts):** Zeigt die Nutzerkonten zusätzlich zu den Systemkonten an.



1.1.3.4. Rechte

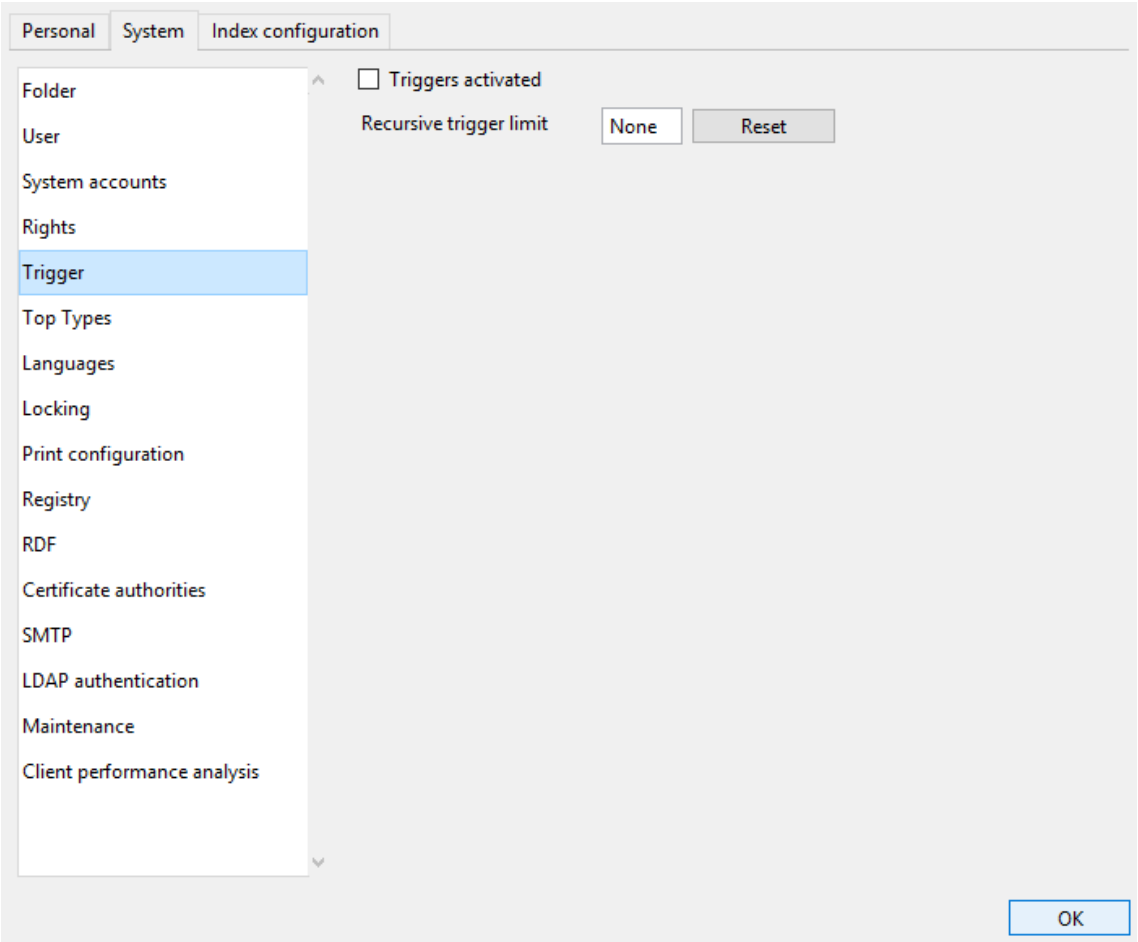


- **Rechtesystem aktiviert (Access rights activated):** Das Zugriffsrechtesystem und dessen Zugriffsrechteprüfung sind nur aktiv, wenn diese Option aktiviert ist. Das Zugriffsrechtesystem umfasst die Zugriffsrechteprüfung von Web-Frontend Nutzern.
- **Benutzertyp (User type):** Spezifiziert, welcher Objekttyp für die Zugangsrechteprüfung verwendet wird für die Nutzer-Instanzen, welche mit dem Nutzerkonten im Administrationsabschnitt "Benutzer" verknüpft werden können.

1.1.3.5. Trigger

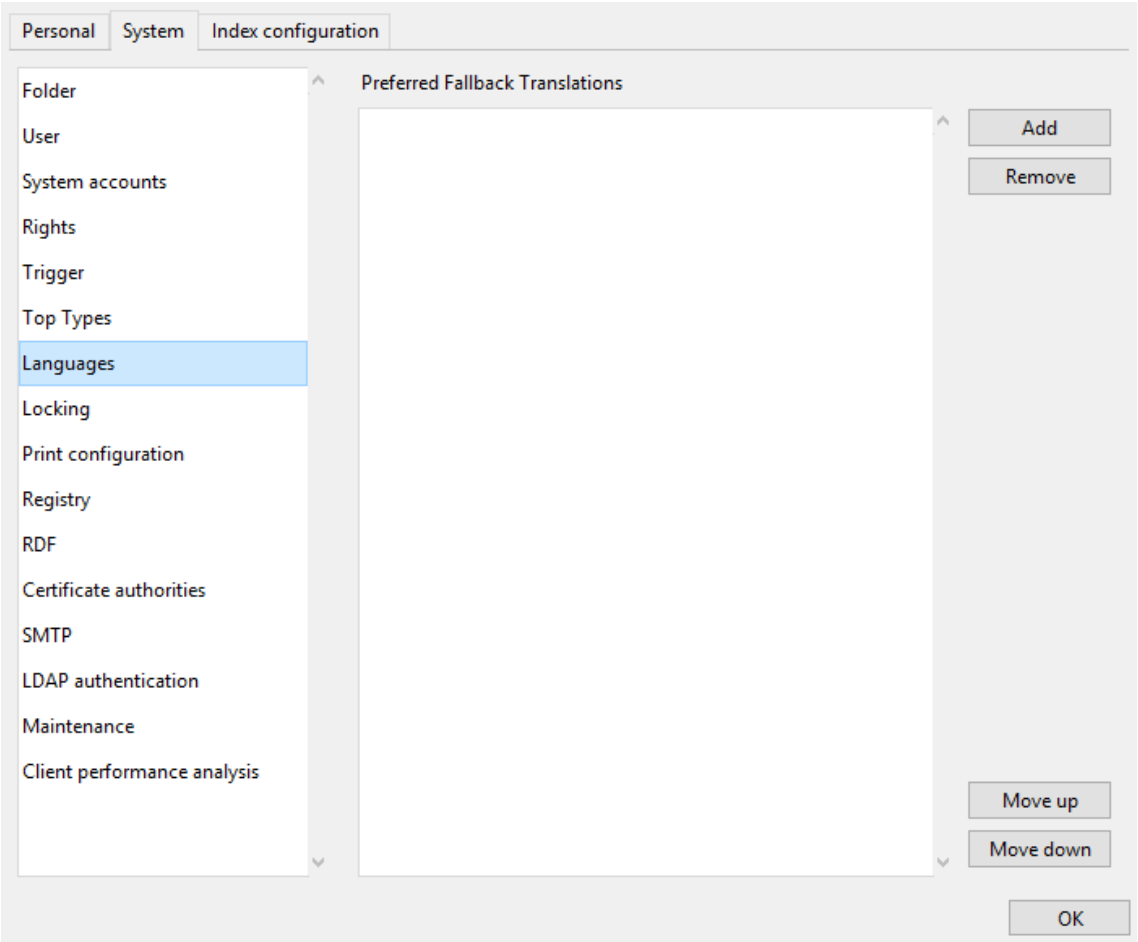
Diese Option aktiviert oder deaktiviert den Trigger-Mechanismus.

HINWEIS	Der Trigger-Abschnitt ist nur dann innerhalb des TECHNIK-Haupttyps verfügbar, wenn der Trigger-Mechanismus mit dieser Option aktiviert wurde.

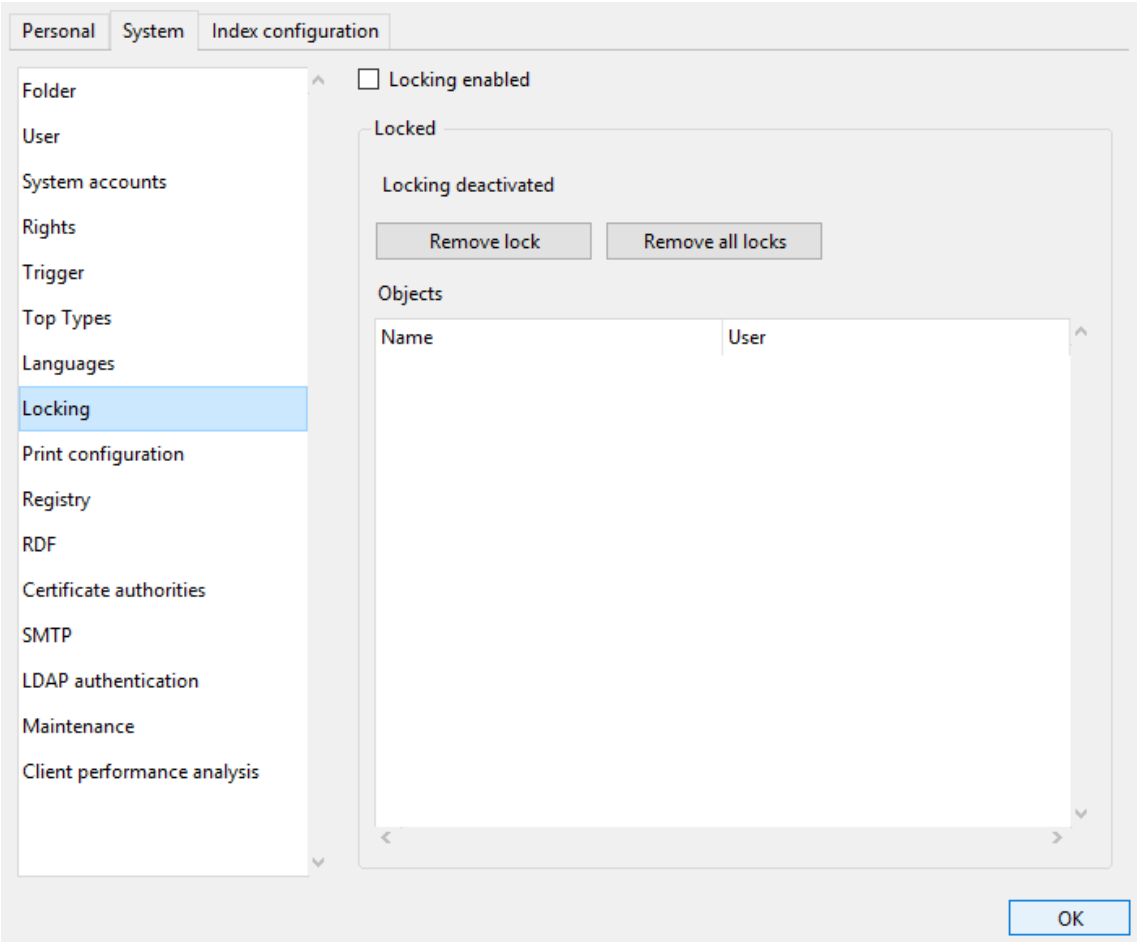


1.1.3.6. Sprachen

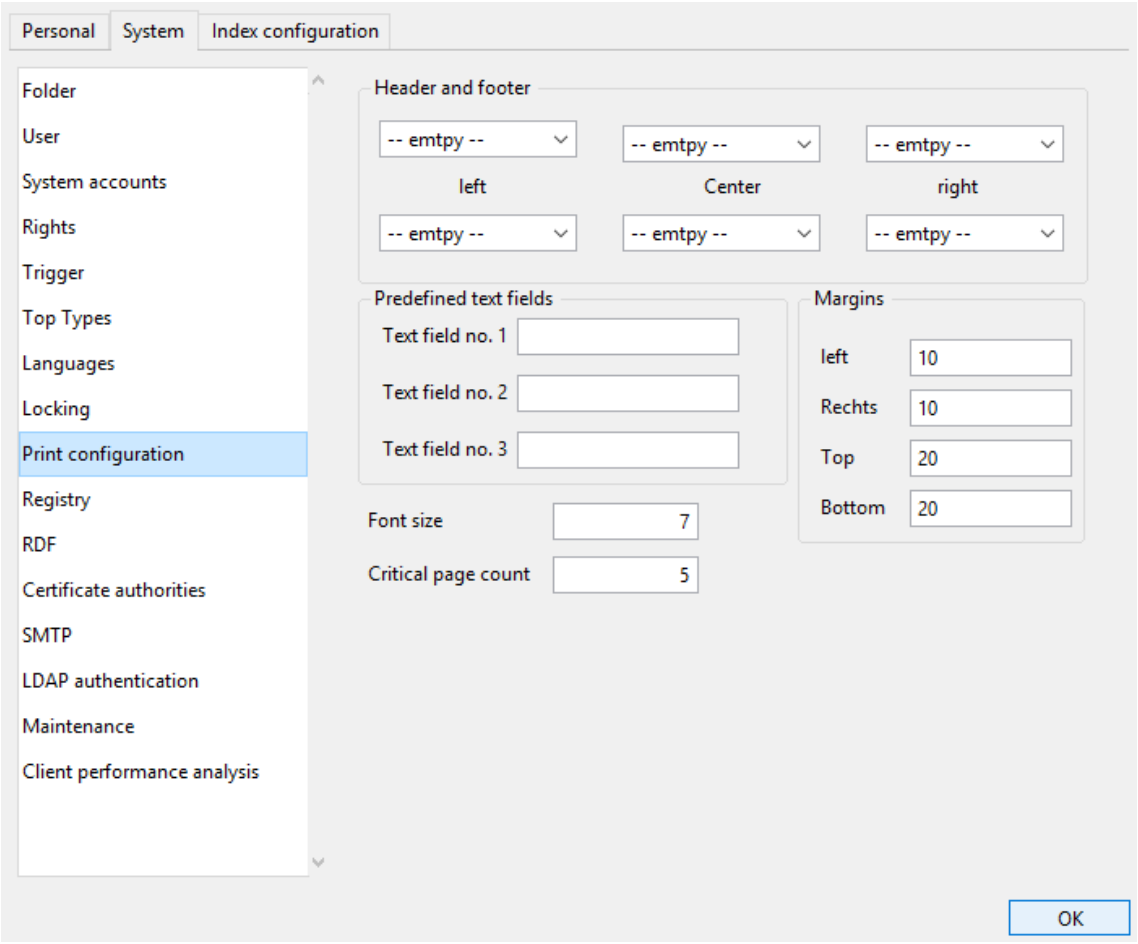
Wenn ein Wert eines gegebenen, übersetzten Attributs nicht in der Sprache der aktuellen Sitzung vorhanden ist, so definiert diese Liste die Abfolge der Sprachen, welche als Ersatzwert angezeigt werden sollen.



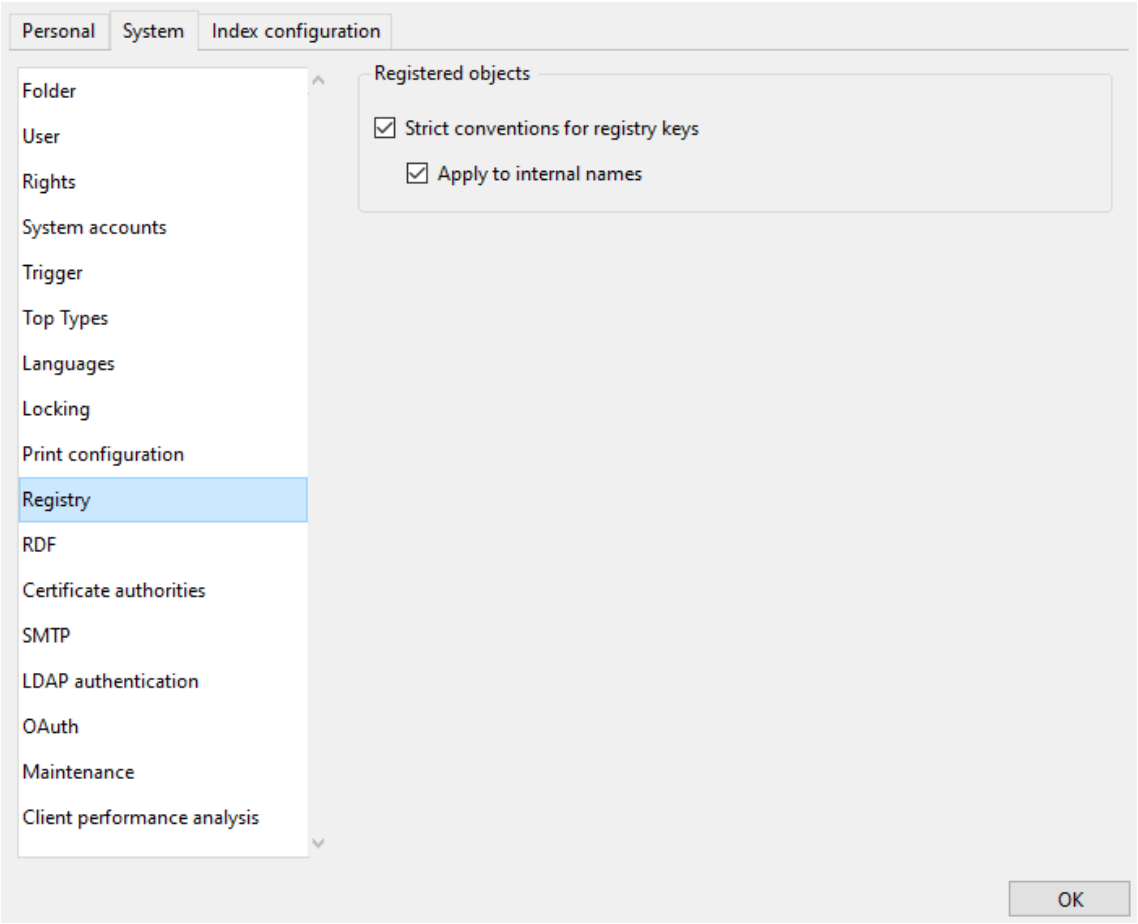
1.1.3.7. Sperren



1.1.3.8. Druckkonfiguration



1.1.3.9. Registratur



Strikte Konventionen für Registrierungsschlüssel (Strict conventions for registry keys): Die Konventionen finden Anwendung beim Erzeugen eines Registrierungsschlüssels und bspw. beim XML-Schematransfer per Admin-Tool. Die strikten Konventionen sind wie folgt:

- Alle 26 Buchstaben der ASCII Code-Tabelle (Klein- und Großbuchstaben)
- Zeichen wie Punkt ".", Unterstrich "_" und Bindestrich "-"
- Das erste Zeichen sollte ein Buchstabe sein

HINWEIS	Die Konventionen sind **case-insensitiv**, d. h. eine Unterscheidung von Registrierungsschlüsseln anhand Klein- und Großschreibweise ist nicht möglich. Beispiel: "myVolume.myQuery1" und "myVolume.MYQuery1" können nicht gleichzeitig in einer Volume verwendet werden. Dies betrifft auch den XML-Schematransfer von einer Volume in eine andere.

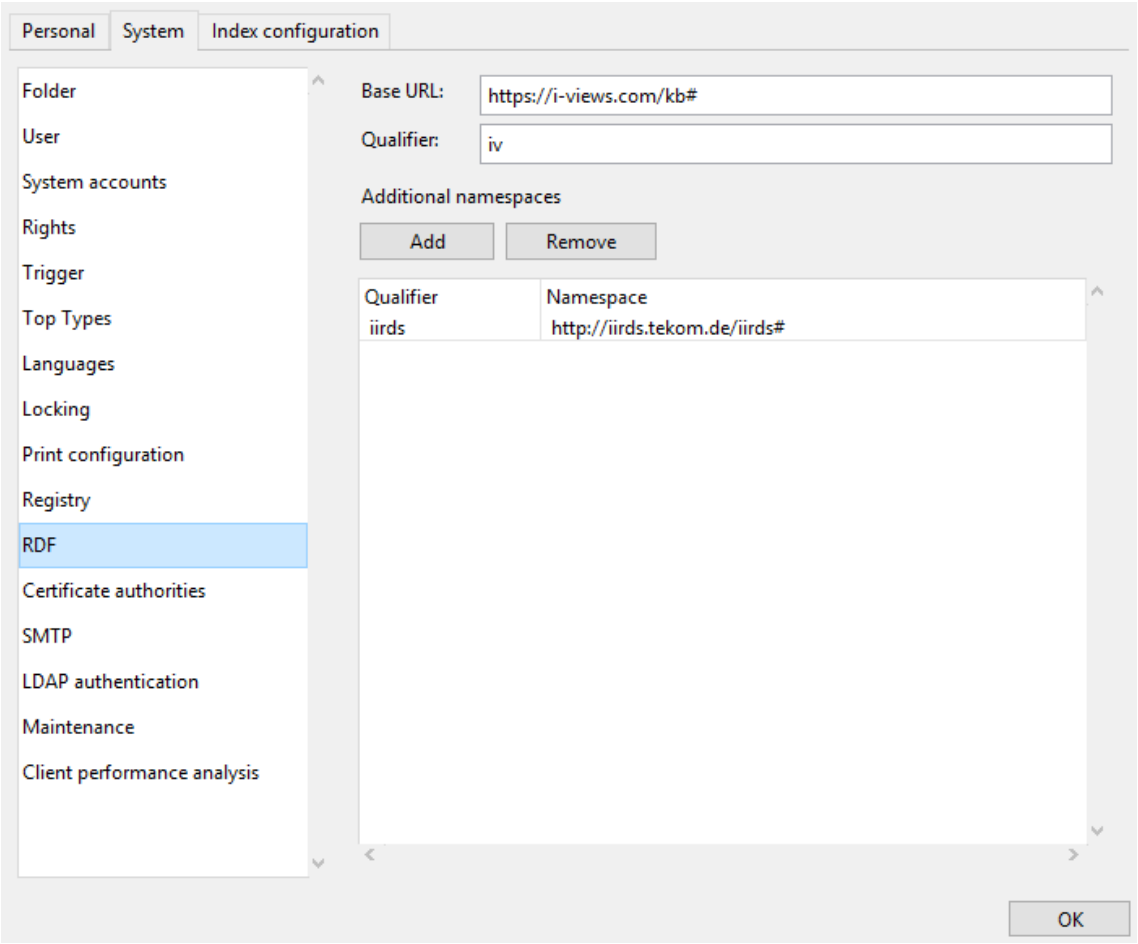
Auf interne Namen anwenden (Apply to internal names): Wenn aktiviert,werden die strikten Konvenvtionen auch auf interne Namen angewendet.

1.1.3.10. RDF

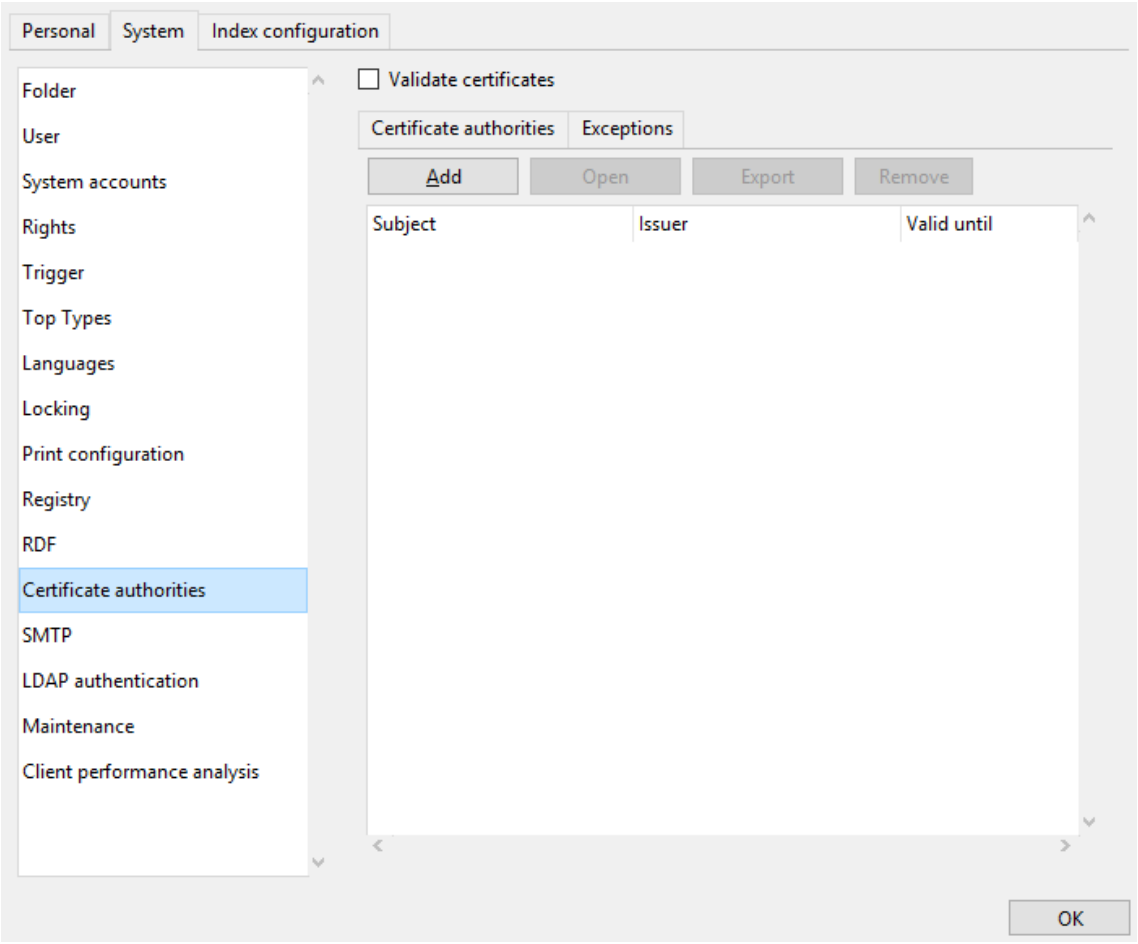
Die RDF-Optionen umfassen die Einstellungen für die Basis-URL, den Qualifier und zusätzlichen Namensräumen, welche zur Identifizierung der Ausgangsknoten bei Import oder Export von RDF-

Dateien herangezogen werden.

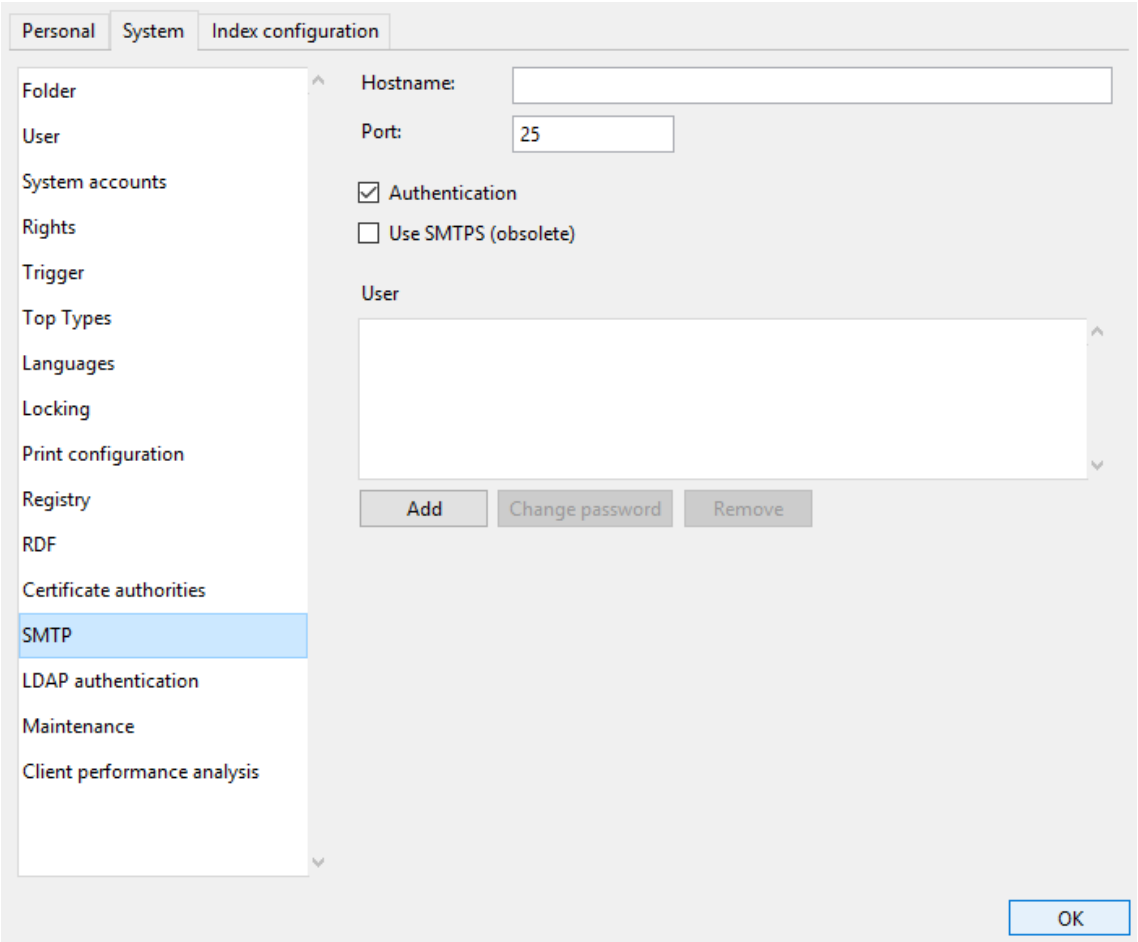
HINWEIS	Die zusätzlichen Namensräume sind nur für den Export bestimmt. Für weitergehende Informationen hierzu siehe "RDF-Import und -export" im Anwenderhandbuch.



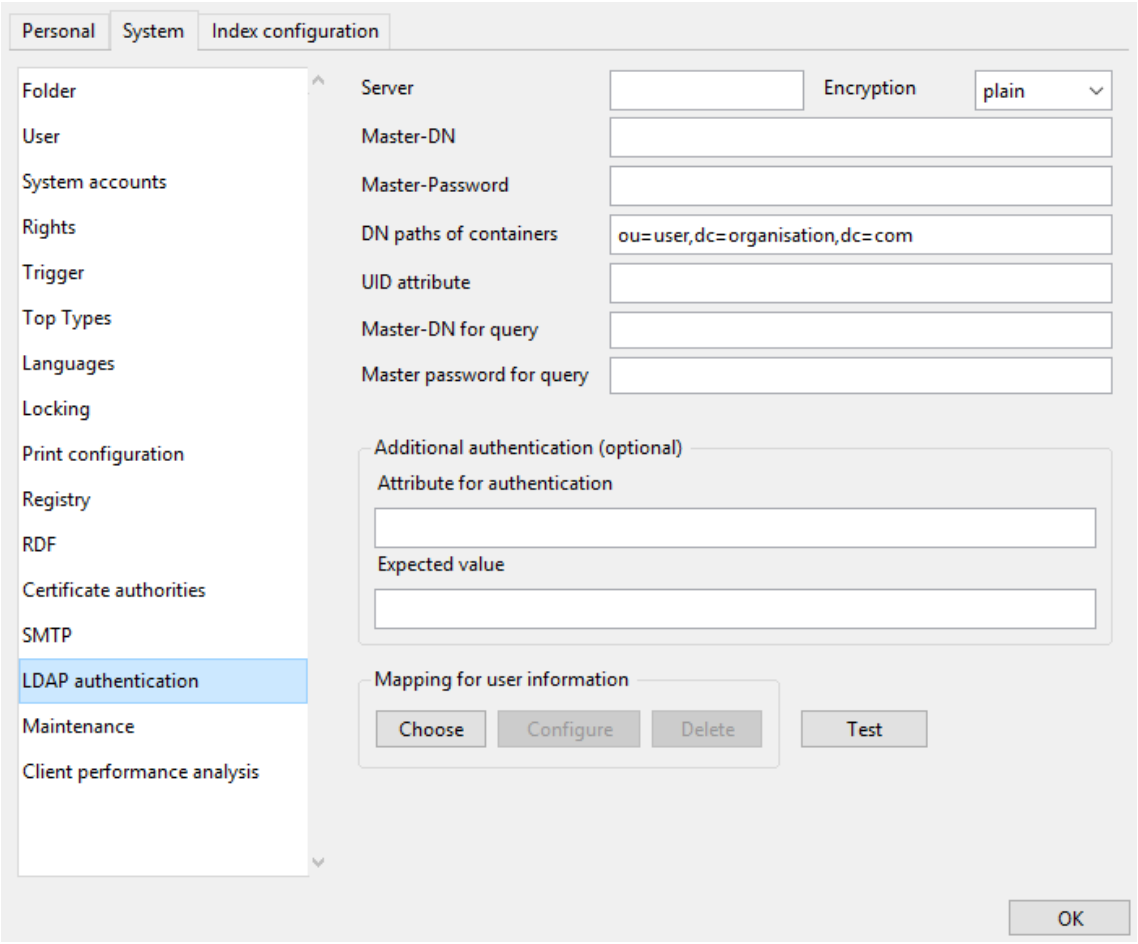
1.1.3.11. Zertifizierungsstellen


Personal
System
Index configuration
Folder
User
System accounts
Rights
Trigger
Top Types
Languages
Locking
Print configuration
Registry
RDF
Certificate authorities
SMTP
LDAP authentication
Maintenance
Client performance analysis
Validate certificates
Certificate authorities
Exceptions
Add
Open
Export
Remove
Subject
Issuer
Valid until
OK

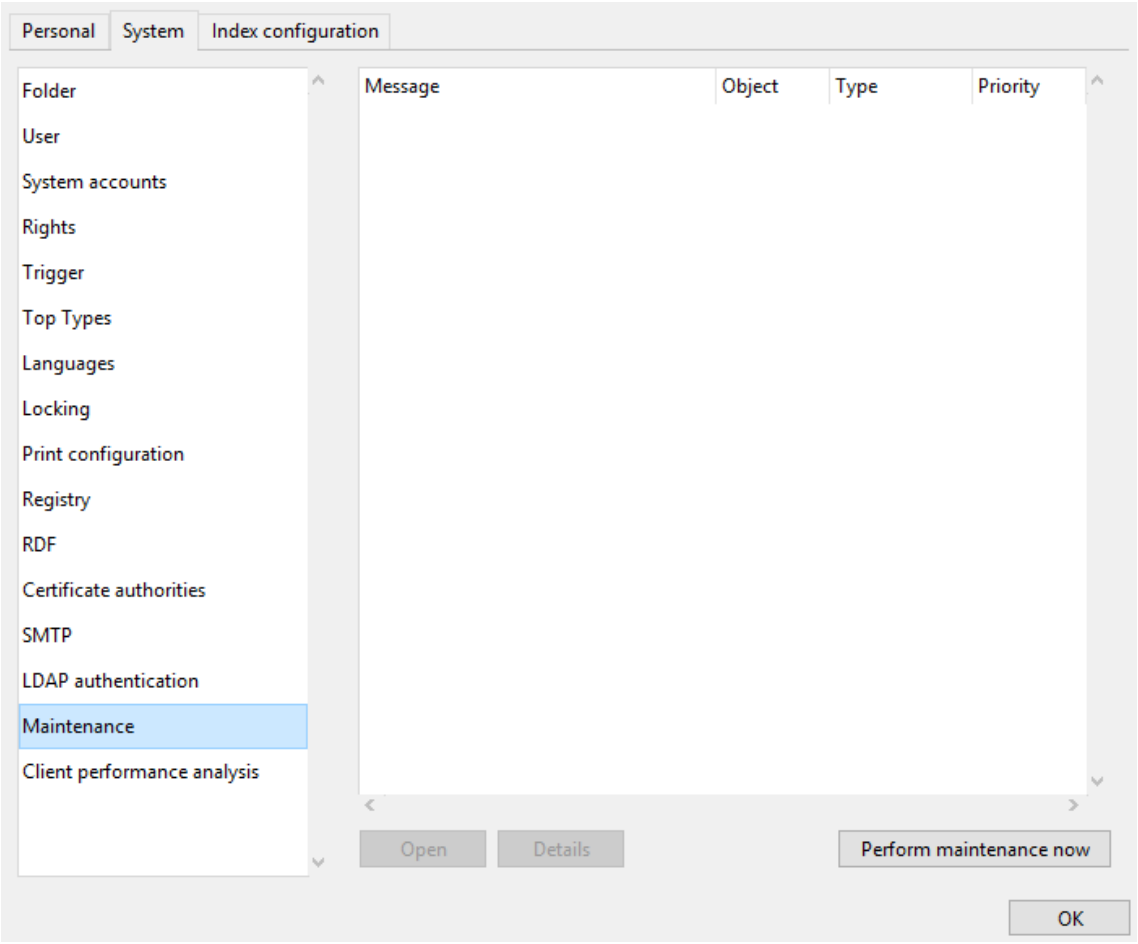
1.1.3.12. SMTP



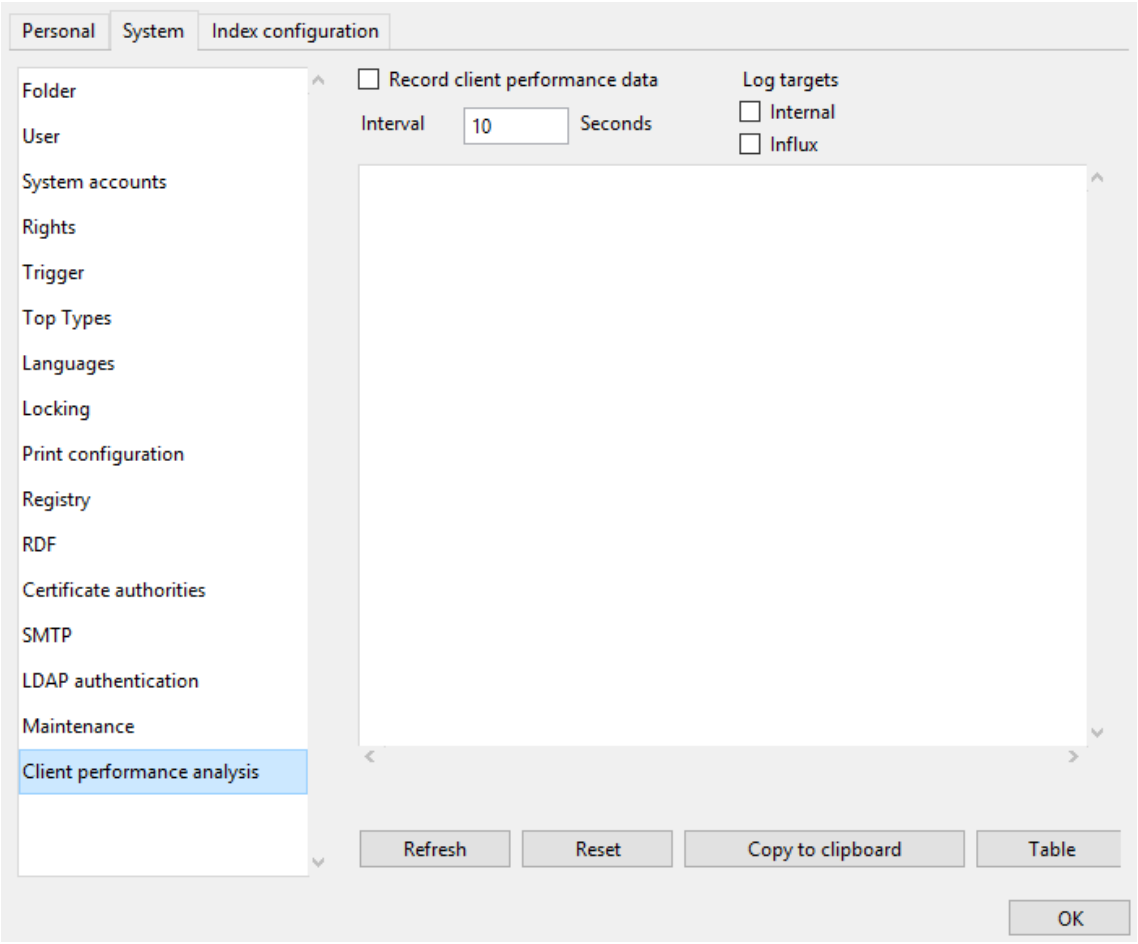
1.1.3.13. LDAP-Authentifizierung



1.1.3.14. Wartung



1.1.3.15. Client-Leistungsanalyse

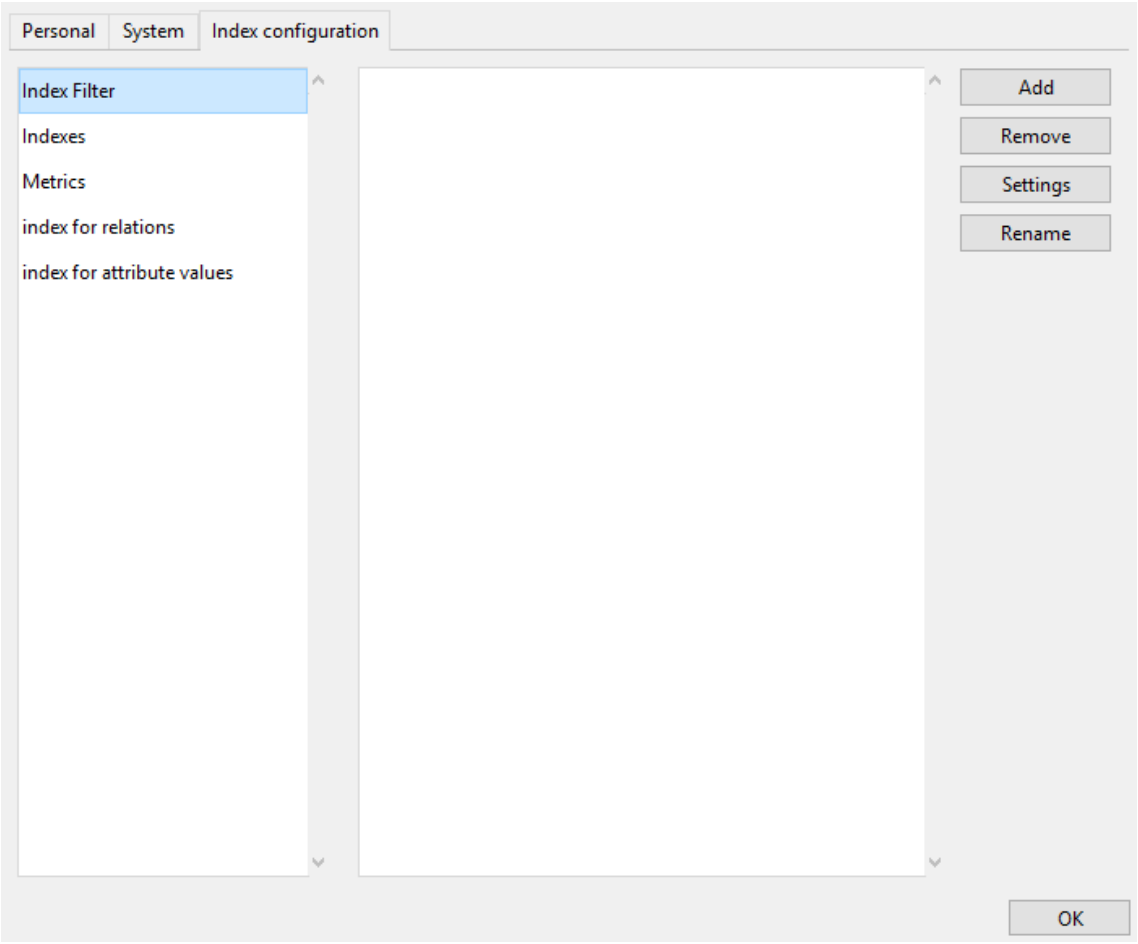


1.1.4. Indexkonfiguration

Die Konfiguration des Indexes für semantische Elemente im Knowledge-Graphen kann hier vorgenommen werden. Die Anwendung von Indizes auf einen Untertypen (= Indexierung) kann entweder hier in den globalen Einstellungen oder im Detail-Editor des betreffenden Typs vorgenommen werden.

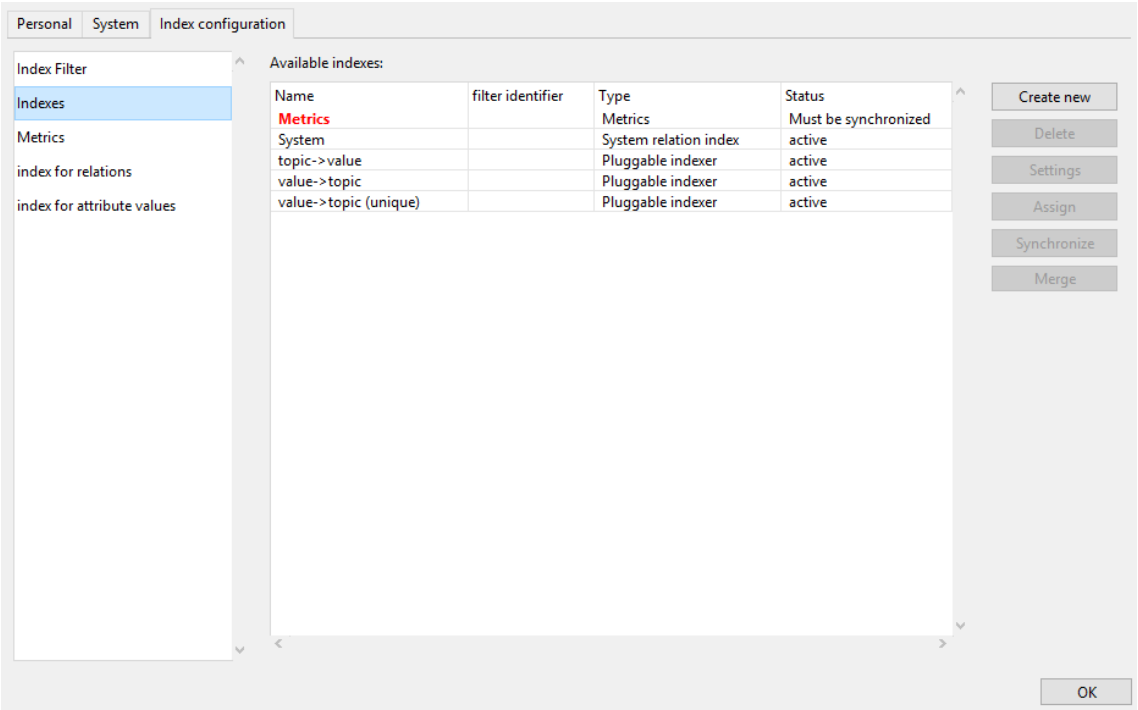
Index-Filter (Index filter)

Index-Filter werden für Volltext-Indizes benötigt und umfassen die Einstellungen für PreFilter (Vorfilterung), Tokenizer (Zeichen-Ersetzung) und sonstige Filter (Nachfilterung).



Indizes (Indexes)

- **Metriken (Metrics):** Die Metriken umfassen klassifizierte Einträge zu den Mengengerüsten von Objekten und bewirken eine Verbesserung der Performance in Hinsicht von Abfragen. Abhängig vom Umfang der Änderungen im Knowledge-Graphen (erzeugen/entfernen semantischer Elemente) müssen die Metriken von Zeit zu Zeit aktualisiert werden.
- **System:** Der Systemindex ist für Systemeigenschaften reserviert (Relationen und Attribute für Kernfunktionalitäten); sie sind persistent und können nicht geändert werden.
- **Weitere Indizes:** In den meisten Fällen sind dies zusammensteckbare Indizes, welche je nach Bedarf aufgebaut werden können.



1.1.4.1. Der Indexreport

Der Indexreport analysiert welche Indexzuweisungen benötigt werden. Vergleicht man diesen "Bedarf" mit den tatsächlichen Indexzuweisungen, können fehlende und unnötige Zuweisungen erkannt werden. Dabei werden Strukturabfragen, Suchkonfigurationen, Viewkonfigurationen und Java-Skripte untersucht. Da Java-Skripte nur bzgl. Referenzen überprüft werden, kann hier nicht erkannt werden, ob und wie das Referenzierte verwendet wird.

HINWEIS	Strukturabfragen können durch Vererbung selten verwendete Eigenschaften einbinden. Diese Eigenschaften benötigen keine Indexierung, werden aber von der Suche selbst dennoch mit einer Warnung versehen.

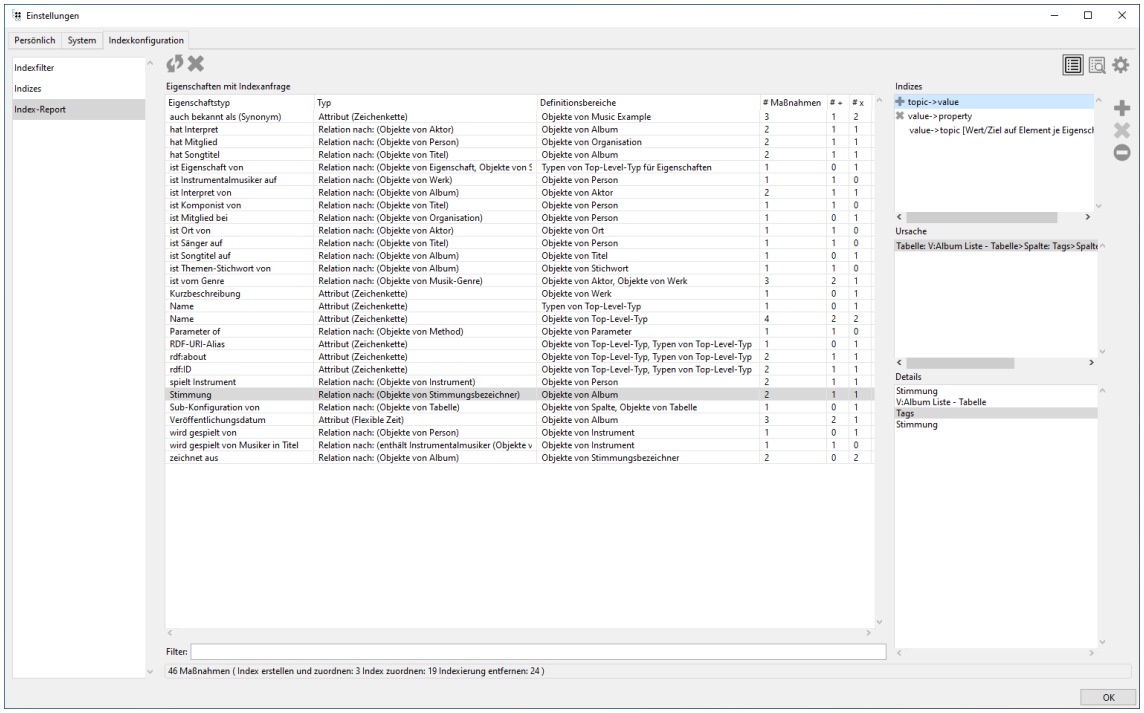
Wo der Indexreport zu finden ist

Unter Einstellungen, Indexkonfiguration finden Administratoren den Index-Report. Zunächst öffnet sich die "einfache Ansicht", zur "detaillierten Ansicht" und den Optionen kann man über die jeweiligen Knöpfe oben rechts wechseln.

Wenn der Index-Report geöffnet wird, wird eine (den Optionen entsprechende) System-Analyse durchgeführt. Dabei wird angezeigt, welcher Bereich, auf welchem Weg und welche Eigenschaft gerade analysiert werden.

HINWEIS	Bei umfangreichen Graphen kann die Analyse eine Weile dauern.

1.1.4.1.1. Die einfache Ansicht



Hier werden alle, bei der Analyse erfassten, Eigenschaftstypen aufgelistet. In der Tabelle werden zum Namen (Eigenschaftstyp) auch die Art der Eigenschaft (Typ), die Definitionsbereiche und die Anzahl an vorgeschlagenen Maßnahmen (# Maßnahmen), bestehend aus Hinzufügen (# +) und Entfernen (# x) von Indizes, angezeigt. Ein Eigenschaftstyp kann per Doppelklick auf seine Zeile geöffnet werden.

Rechts sieht man zu der ausgewählten Eigenschaft die zugeordneten Indizes. Änderungsvorschläge haben dem Namen des Indexes ein "" oder "x" vorangestellt. "" bedeutet, dass dieser Index nicht dem ausgewählten Eigenschaftstypen zugewiesen ist aber verwendet werden sollte. Ein "x" bedeutet, dass der zugewiesene Index nicht benötigt wird. Mit den entsprechenden Knöpfen auf der rechten Seite kann ein ausgewählter Index entfernt oder hinzugefügt werden.

In den Einstellungen kann bestimmt werden, wann ein Index tatsächlich hinzugefügt werden soll, wenn man das "+" drückt.

Unter den Indizes wird für ausgewählte, noch nicht zugeordnete (mit "+" markierte) Indizes aufgelistet, wo die Analyse diese fehlende Indexverwendung entdeckt hat. Wählt man eine Verwendung aus, so wird jeder Schritt dieses Weges aufgelistet und kann per Doppelklick geöffnet werden.

Die Liste der Eigenschaftstypen kann, mithilfe des unteren Eingabefelds, gefiltert werden. Dieser Filter bezieht sich auf den Namen des Eigenschaftstyps, den Typ und die Definitionsbereiche. Es kann ebenfalls mit +/x/# einer Zahl nach Eigenschaftstypen gefiltert werden, welche so viele Vorschläge zum hinzufügen, entfernen oder beides zusammen von Indizes haben.

Falls für einen Index keine Zuweisung oder Entfernung vorgeschlagen werden soll, kann man mit der Schaltfläche "Durchfahrt verboten" (-) eine Ausnahme anlegen.

Anlegen einer Ausnahme

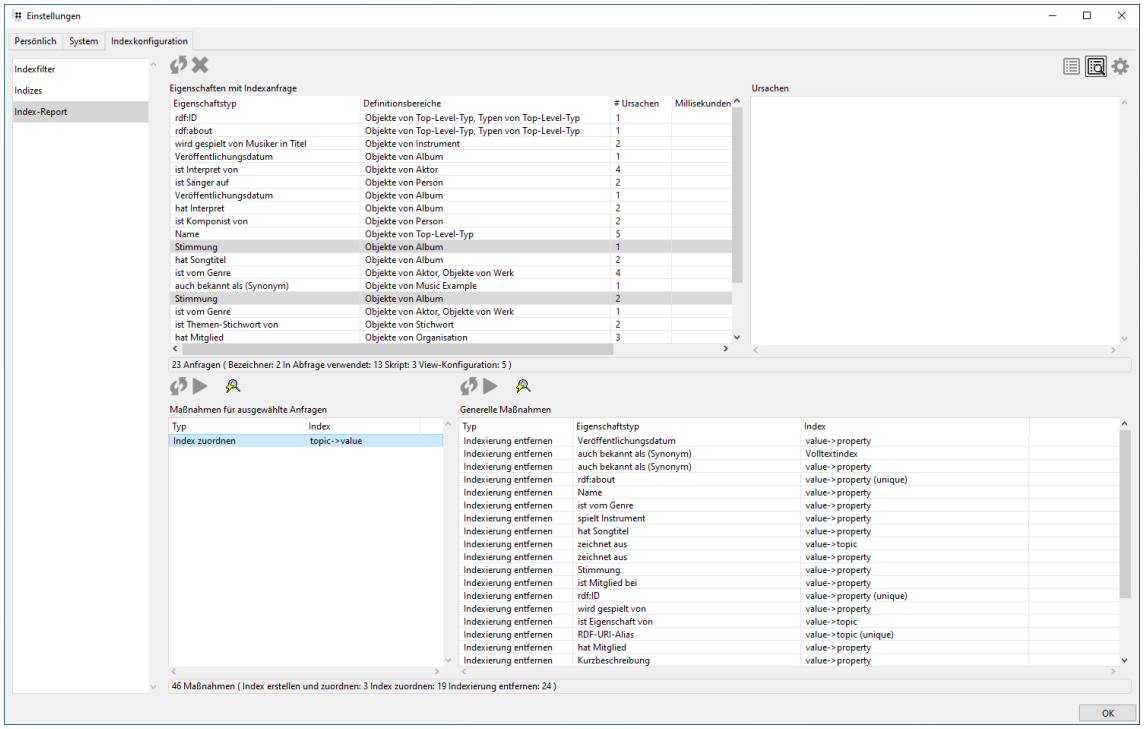
Ausnahmen legen fest, welche Eigenschaften, Indizes oder Referenzierende Elemente bei der Analyse ignoriert werden sollen.

Beim Anlegen einer neuen Ausnahme kann man auswählen, ob die Verwendung der Eigenschaft bzw. des Indexes ignoriert werden soll. Ebenso kann man mehrere Pfadelemente auswählen. Möglich sind damit z.B. folgende Ausnahmen: "Keine Indizes bei Abfragen in Skript xyz", "Keine Berücksichtigung von Eigenschaft abc", "Keine Berücksichtigung von Index ijk" oder auch alles zusammen "in Skript xyz keine Zuweisung von Index ijk an Eigenschaft abc". Ebenso kann mit einer Ausnahme auch verhindert werden, dass eine Eigenschaft Vorschläge erhält. Analog können auch Vorschläge zu einem kompletten Index verhindert werden oder eine Eigenschaft und ein Index aus der Vorschlagliste genommen werden. Pfadelemente wird es bei nicht verwendeten Indizes natürlich nicht geben.

Jede Ausnahme braucht einen Kommentar der aussagen sollte, wozu die Ausnahme da ist.

Ausnahmen verhindern auch die Prüfung, ob ein zugewiesener Index überhaupt benötigt wird.

1.1.4.1.2. Die detaillierte Ansicht



Mit dieser Ansicht kann man Fragen wie "Warum wird dieser Index für diese Eigenschaft vorgeschlagen?" beantworten. Statt Eigenschaften werden hier die einzelnen Indexanforderungen aufgelistet. Außerdem hat sie auch zusätzliche Spalten: Die Anzahl der Ursachen, aufgrund derer der Index vorgeschlagen wird, Millisekunden (der gemessene Wert der Leistungsanalyse) und Tag mit Werten "In Abfrage verwendet" (in registrierter Abfrage verwendet), "In Skript verwendet" (in registriertem JS verwendet), "In Mapping verwendet" (in registriertem Tabellen-Mapping

verwendet), "View-Konfiguration" (in der View-Konfiguration verwendet) und "Leistung" (in der Leistungsmessung verwendet). Falls eine Eigenschaft über einen konfigurieren Startpunkt erreicht wird, ist das Tag je nach Art des Startpunktes "Abfrage", "Skript" oder "Bezeichner" (letzteres wird auch für RDF-Systemeigenschaften verwendet).

Alle Referenzen zur Anforderung sind bei "Ursachen" aufgelistet.

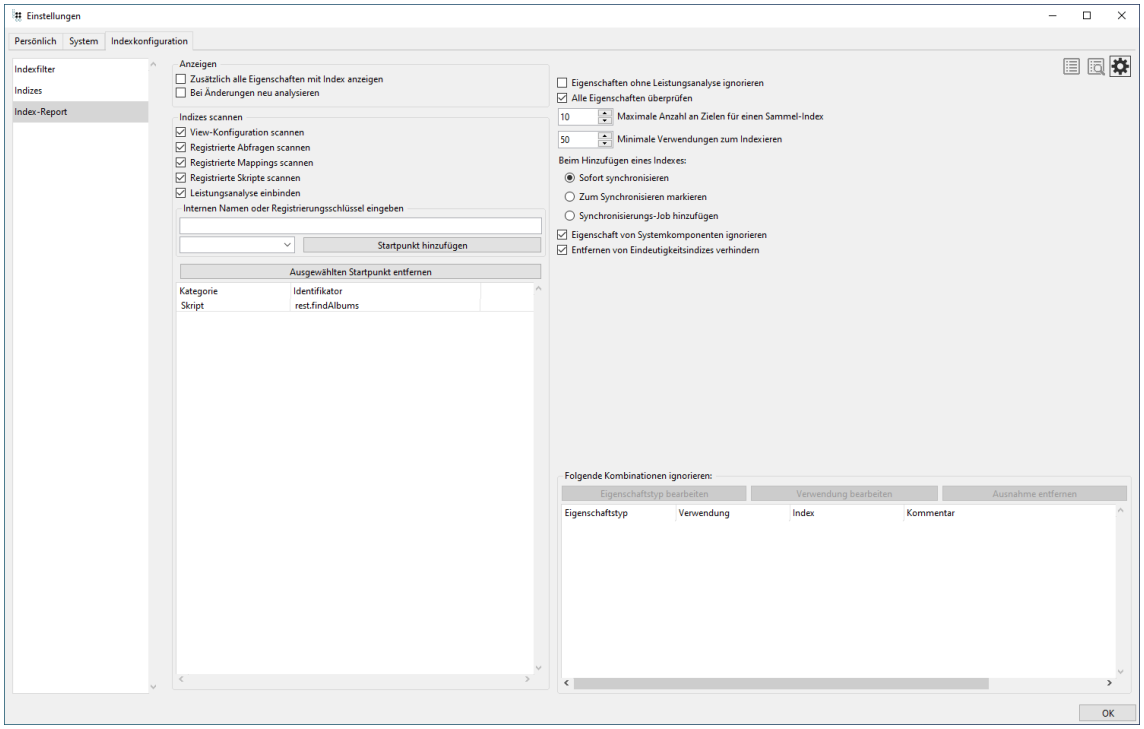
Nicht erfolgte Indexzuweisungen der gewählten Anforderungen werden unten links gelistet.

Schließlich werden zugewiesene Indizes ohne Anforderung (ohne Nachweis) unter "Generelle Maßnahmen" zum Entfernen gelistet.

In beiden Listen mit Maßnahmen können eine oder mehrere Maßnahmen markiert und über den darüber liegenden "Abspielen"-Knopf ausgeführt werden.

1.1.4.1.3. Einstellungen

Über die Optionen kann gesteuert werden, wo die Analyse erfolgt und welche Maßnahmen vorgeschlagen werden.



Manuell Startelemente konfigurieren

Wenn, zum Beispiel, Java-Skripte nur per Batchtool verwendet werden, aber für sie Indizes berechnet werden sollen, kann man sie als Startpunkt hinzufügen: Links in das Textfeld unter "Internen Namen oder Registrierungsschlüssel eingeben" können spezielle Elemente oder bei Bedarf auch mit Platzhalter eine größere Zielmenge adressiert werden (etwa "all*base*"). Zudem muss man angeben, ob als weiterer Startpunkt Skripte, Abfragen oder Typen verwendet werden sollen. Ist die Definition komplett, kann der zusätzliche Startpunkt mit der Schaltfläche "Startpunkt

hinzufügen" eingetragen werden. Nicht mehr benötigte Startpunkte kann man auswählen und über die Schaltfläche "Ausgewählten Startpunkt entfernen" löschen.

Ausnahmen

Die in der einfachen Ansicht angelegten Ausnahmen können unten rechts verwaltet werden. Man kann sie löschen (Schaltfläche "Ausnahme entfernen") oder die zugehörige Eigenschaft oder den konfigurierten Weg dorthin anzeigen (Schaltfläche "Eigenschaft" bzw. "Verwendung").

Sonstige Optionen

Option	Beschreibung
Zusätzlich alle Eigenschaften mit Index anzeigen (Standard: aus)	Zeigt auch die Eigenschaften ohne berechnete Änderung an.Diese Einstellung wird erst mit der nächsten Analyse wirksam.
Bei Änderungen neu Analysieren (Standard: aus)	Gibt an, ob nach dem Umsetzen eines Vorschlags ein Neustart der Analyse erfolgen soll.
View-Konfiguration scannen(Standard: an)	Analysiert die View-Konfiguration falls vorhanden.
Registrierte Abfragen scannen(Standard: an)	Wenn ausgeschaltet, werden registrierte Abfragen nicht analysiert.
Registrierte Mappings scannen(Standard: an)	Wenn ausgeschaltet, werden registrierte Mappings nicht analysiert.
Registrierte Skripte scannen(Standard: an)	Wenn ausgeschaltet, werden registrierte Skripte nicht analysiert.
Leistungsanalyse einbinden(Standard: an)	Wenn eine Messung zur Leistungsanalyse (siehe Client-Analyse) vorliegt, werden diese Messungen als Startpunkte hinzugenommen.
Eigenschaften ohne Leistungsanalyse ignorieren (Standard: aus)	Wenn keine Messwerte zu einer Eigenschaft vorliegen, werden Verwendungen nicht weiter berücksichtigt.(Kann nur eingeschaltet werden, wenn die Leistungsanalyse eingebunden wird.)
Alle Eigenschaften überprüfen (Standard: an)	Berücksichtigt auch Zuweisungen von Indizes, die nicht vom Report erfasst wurden, bezüglich ihrer Notwendigkeit.
Maximale Anzahl von Zielen für einen Sammel-Index(Standard: 10)	Relationen auf wenige Ziele sollten einen Index haben, der dies berücksichtigt. Üblicherweise betrifft dies Relationen auf Katalogwerte. Über diese Ganzzahl kann dafür der Grenzwert dafür vorgegeben werden.

Option	Beschreibung
Minimale Anzahl zum Indexieren(Standard: 50)	Eigenschaften, die nur selten vorkommen, benötigen keinen Index. Über diese Ganzzahl kann dafür der Grenzwert dafür vorgegeben werden.
Beim Hinzufügen eines Indexes:(Standard: Sofort Synchronisieren)	Bestimmt was tatsächlich passiert, wenn man einen Index hinzufügt.Möglichkeiten: * Sofort synchronisieren * Zum Synchronisieren markieren * Synchronisierungs-Job hinzufügen
Eigenschaften von Systemkomponenten ignorieren (Standard: an)	Gibt an, ob Eigenschaften von Systemkomponenten auch untersucht werden sollen.Nur für Entwickler sichtbar.
Entfernen von Eindeutigkeitsindizes verhindern(Standard: an)	Verhindert das Entfernen von Eindeutigkeitsindizes. Nur für Entwickler sichtbar.

1.1.5. Konfigurationsdatei kb.ini

Wie bei jedem i-views Produkt, kann auch für den Knowledge-Builder eine kb.ini-Konfigurationsdatei erzeugt werden. Im folgenden sind beispielhaft Auszüge für die Konfigurationsdatei des Knowledge-Builders aufgelistet:

```
; über die folgenden 3 Parameter kann man die entsprechenden Felder im
Anmeldefenster vorausfüllen lassen
host=demo-server.empolis.com
user=peter
volume=demo
logTargets=kb-log
; über die Angabe von cacheDir wird ein File-Caching der Daten
konfiguriert und aktiviert
; File-Caching erhöht die Geschwindigkeit beim erneuten Starten des KB
cacheDir=cache
; der Parameter maxCacheSize setzt das Limit des File-Cache (in MB)
; default ist 50 (MB)
maxCacheSize=200
; über den language Parameter kann man das Starten des KB in der
angegebenen Sprache erzwingen
; ohne diese Angabe wird die Spracheinstellung des Betriebssystems
verwendet
; mögliche Werte sind "eng" und "ger", fallback ist "ger"
;language=eng

[kb-log]
```

```
type=file
file=kb.log
```

1.2. Zugriffsrechte und Trigger

In diesem Abschnitt wird die Prüfung von Zugriffsrechten und Trigger behandelt:

- **Zugriffsrechte** regeln, welche Operationen am semantischen Modell bestimmte Nutzergruppen durchführen dürfen. Sie werden in i-views im Rechtesystem definiert. Das Rechtesystem befindet sich im Bereich *Technik > Rechte*.
- **Trigger** sind automatische Operationen, die bei einem bestimmten Ereignis ausgelöst werden und die zugehörigen Aktionen ausführen. Der Bereich Trigger befindet sich unter *Technik > Trigger*.

Das Rechtesystem und Trigger sind in einer neu angelegten semantischen Graph-Datenbank initial noch nicht aktiviert. Diese Bereiche müssen erst aktiviert werden, bevor sie eingesetzt werden können.

Bei der Erstellung von Rechten und Triggern ist die grundsätzliche Vorgehensweise identisch: Es werden Filter benötigt, die prüfen ob bestimmte Bedingen erfüllt sind oder nicht. Sind diese Bedingen erfüllt, wird beim Rechtesystem ein Zugriffsrecht oder -verbot erteilt sowie bei Triggern ein Log eingetragen oder ein Script ausgeführt. Im Rechtesystem wird die Anordnung der Filter als Rechtebaum und bei Triggern als Trigger-Baum bezeichnet.

HINWEIS	Damit die Abfrage-Filterbedingungen das gewünschte Ergebnis liefern, sind die Inhalte der Tabelle in Kapitel 1.2.5 "Operationen" zu berücksichtigen. Prinzipiell arbeiten die Operationsfilter in einer "UND"-Logik, wodurch alle Bedingungen eines Operationsfilters und die Bedingungen der Unterkomponenten des Operationsfilters erfüllt sein müssen. Daher wird empfohlen, eine möglichst exakte Bedingung zu wählen.

1.2.1. Die Prüfung von Zugriffsrechten

Mit Rechten regeln wir den Zugriff von Nutzern auf die Daten im semantischen Netz. Die zwei grundsätzlichen Ziele, deren Erreichung mit dem sogenannten Rechtesystem ermöglicht werden, sind:

- **Schutz von sensiblen Daten:** Es werden Nutzern oder Nutzergruppen, nur die Daten angezeigt, die sie auch lesen dürfen. Damit werden Geheimhaltungs- und Vertraulichkeitsbeschränkungen gewährleistet.
- **Arbeitsspezifische Übersicht:** Bestimmte Nutzer benötigen für ihre Arbeit mit dem System häufig nur einen Ausschnitt der Daten des Modells. Mit Hilfe des Rechtesystems ist es möglich ihnen nur die Elemente anzuzeigen, die sie für das Erledigen ihrer Aufgaben brauchen.

Das Rechtesystem von i-views zeichnet sich durch einen hohen Grad an Flexibilität aus. Es kann auf verschiedene Erfordernisse eines Projektes zielgenau konfiguriert werden. Durch die Definition von Regeln im Rechtebaum bestehend aus einzelnen Filtern und Entscheidern, entsteht ein Netz-spezifische Konfiguration des Rechtesystems. Es gibt vielfältige Möglichkeiten diese Regeln für das Rechtesystem zusammenzusetzen, wodurch hoch differenzierte Rechte erzeugt werden. Es ist nicht

möglich, alle möglichen Kombinationen von Konfigurationen aufzulisten; hier muss eine Beratung für den Einzelfall stattfinden.

Wie funktioniert das Rechtesystem?

Zugriffsrechte im System werden immer dann geprüft, wenn durch einen Nutzer eine Operation auf die Daten vorgenommen wird. Die grundsätzlichen Operationen sind:

- *Lesen* : Ein Element soll angezeigt werden.
- *Modifizieren* : Ein Element soll geändert werden.
- *Erzeugen* : Ein neues Element soll erstellt werden.
- *Löschen* : Ein Element soll gelöscht werden.

Soll in einer bestimmten Zugriffssituation das Zugriffsrecht geprüft werden, wird der **Rechtebaum** abgearbeitet, bis eine Entscheidung für oder gegen den Zugriff in dieser Situation getroffen werden kann. Der Rechtebaum besteht aus Bedingungen, gegen die die Zugriffssituation geprüft wird. Um die Bedingungen zu prüfen, werden **Filter** verwendet, die Elemente des Wissensnetzes und Operationen filtern. Am Ende eines Teilbaumes aus Filtern im Rechtebaum befinden sich die **Entscheider**. Von diesen wird der Zugriff entweder erlaubt oder abgewiesen.

In Bezug auf die Zugriffssituation werden Aspekte ausgewählt, die als Bedingung für die Erlaubnis oder das Verbot des Zugriffes eingesetzt werden. In Zugriffssituationen werden häufig folgende Aspekte für die Entscheidung herangezogen:

- die Operation (Erzeugen, Lesen, Löschen oder Modifizieren)
- das Element, auf das zugegriffen werden soll
- der aktuelle Nutzer

Es kann sein, dass nur ein Aspekt der Zugriffssituation als Bedingung ausgewählt wird, aber es kann auch eine Kombination der aufgeführten Aspekte abgefragt werden.

Beispiel: "Person A [Nutzer] darf keine Beschreibungen [Element] löschen [Operation]".

1.2.1.1. Die Aktivierung des Rechtesystems

In einem neu angelegten Wissensnetz ist das Rechtesystem standardmäßig deaktiviert. Damit es genutzt werden kann, muss es in den Einstellungen des Knowledge-Builders aktiviert werden.

Anleitung für die Aktivierung des Rechtesystems

1. Rufen Sie im Knowledge-Builder das Menü *Einstellungen* auf und wählen Sie den Reiter *System* aus. Wählen Sie dort das Feld *Rechte*.
2. Setzten Sie im Feld *Rechtesystem aktiviert* einen Haken.
3. Geben Sie im Feld Benutzertyp den Objekttyp an, dessen Objekte die Benutzer des Rechtesystems sind. Das ist i.d.R. der Objekttyp "Person". (Typ darf nicht abstrakt sein.)

4. Wenn Sie das Knowledge-Portal von i-views angebunden haben, geben Sie in dem Feld *Standard Web-Benutzer* einen Benutzer an (Objekt des zuvor definierten Personenobjekttyps).

Vor der Aktivierung des Rechtesystems heißt der Ordner *Rechte (deaktiviert)*. Wurde das Rechtesystem aktiviert heißt der Ordner *Rechte*. Durch eine Deaktivierung des Rechtesystem, werden keine Prüfungen der Zugriffsrechte mehr durchgeführt. Die definierten Regeln im Rechtebaum bleiben aber erhalten und werden bei einer erneuten Aktivierung des Rechtesystems wieder verwendet.

HINWEIS	Greift man vom Web-Frontend aus ohne spezielle Anmeldung auf ein Element zu, wird die unter *Standard Web-Benutzer* angegebene Person verwendet. Gewöhnlich legt man hier eine Scheinperson namens "anonymous" oder "Gast" an.

Damit das Rechtesystem auch im Knowledge-Builder funktioniert, muss der Benutzer-Accounts des Knowledge-Builders mit einem Objekt aus dem semantischen Modell verknüpft werden. Der Benutzer-Account kann nur mit Objekten des Typs verknüpft werden, der bei der Aktivierung des Rechtesystems im Feld Benutzertyp angegeben wurde.

Die Verknüpfung ist für die Verwendung des Operationsparameter *Benutzer* bei Suchfiltern bzw. bei für die Verwendung des Zugriffsparameters *Benutzer* bei Strukturabfragen generell notwendig, wenn das Rechtesystem bzw. die Suche nicht in einer Anwendung sondern im Knowledge-Builder selbst ausgeführt wird.

Anleitung für die Verknüpfung von Knowledge-Builder Nutzern mit Objekten des Personen Typs

1. Im Knowledge-Builder das Menü *Einstellungen* aufrufen und den Reiter *System* wählen. Dort das Feld *Benutzer* auswählen.
2. Den Nutzer auswählen, der verknüpft werden soll. Über *Verknüpfen* kann der Benutzer mit einem Personenobjekt verknüpft werden, das noch mit keinem Knowledge-Builder-Account verknüpft ist. Die Funktion *Verknüpfung aufheben* führt dazu, dass die Verknüpfung des Knowledge-Builder-Account mit dem Personenobjekt aufgehoben wird.

Beachte: Der aktuell angemeldete Nutzer kann nicht verknüpft werden.

Benutzer mit Administratorrechten dürfen generell alle Operationen durchführen, unabhängig davon welche Rechte im Rechtesystem definiert wurden. Die Definition als Administrator wird ebenfalls im Menü *Einstellungen* auf dem Reiter *System* im Feld *Benutzer* durchgeführt.

1.2.1.2. Der Rechtebaum

Traversierung des Rechtebaumes

Der Rechtebaum besteht aus Regeln, die in einem Baum definiert sind. Die Äste des Baumes, auch als Teilbaum bezeichnet, bestehen aus den Bedingungen, die geprüft werden sollen. Die Bedingungen werden im System als Filter definiert, die ineinander geschachtelt werden. Bei der Auswertung läuft das System den Baum von oben nach unten ab. Wenn eine Bedingung auf die

Zugriffssituation passt, dann geht die Prüfung zum nächsten Filter des Teilbaumes. Dieser Filter wird wiederum geprüft. Dies wird bis zum Ende des Teilbaumes durchgeführt, dort steht ein Zugriffsrecht oder -verbot. Trifft eine Bedingung nicht auf die Zugriffssituation zu, wird zum nächsten Teilbaum gewechselt. Wenn das System bei der Abarbeitung des Rechtebaumes auf ein Zugriffsrecht oder -verbot stößt, wird die Rechteprüfung mit diesem Ergebnis beendet. Die Äste (Teilbäume) des Baumes werden also nacheinander abgearbeitet, der Baum wird "traversiert", bis eine Entscheidung getroffen werden kann.

Filter und Entscheider werden in Form von Ordnern ineinander geschachtelt, so dass ein Baumkonstrukt entsteht, das aus verschiedenen Teilbäumen besteht. Ein Ordner kann mehrere Unterordner haben (mehrere Nachfolgefilter auf einer Ebene), wodurch Verzweigungen im Rechtebaum entstehen. Ordner, die auf einer Ebene definiert sind, werden nacheinander abgearbeitet (von oben nach unten).

Gestaltung des Rechtebaumes

Bei der Erstellung des Rechtebaumes ist es wichtig die Regeln sinnvoll zu gruppieren, denn wenn eine Entscheidung für eine Zugriffserlaubnis oder ein Zugriffsverbot getroffen wurde, werden keine weiteren Regeln mehr geprüft. Deswegen sollten Ausnahmen vor den globalen Regeln definiert werden.

Die zwei grundlegenden Fälle, die man unterscheiden muss, sind:

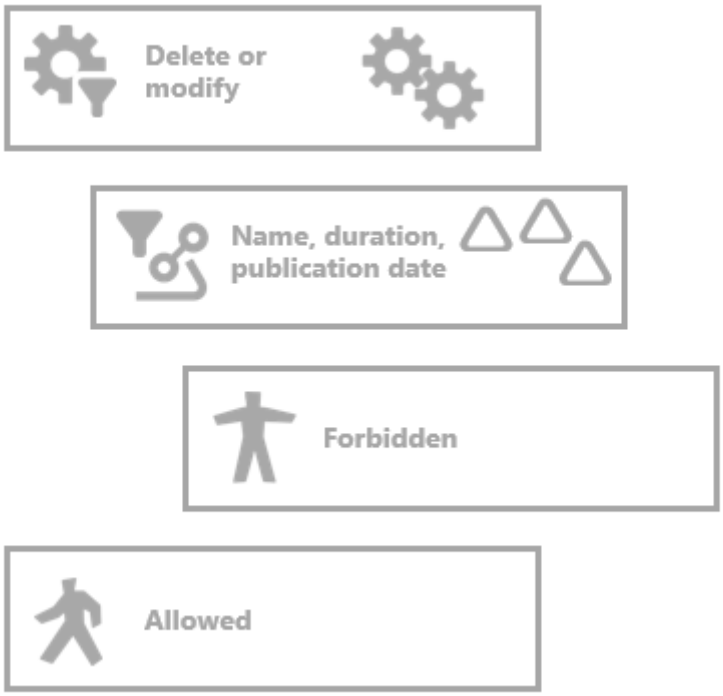
- **Negativ-Konfiguration** : Im untersten Teilbaum wird pauschal alles erlaubt, darüber werden Verbote formuliert.
- **Positiv-Konfiguration** : Unten ist pauschal alles verboten, außer dem, was weiter oben erlaubt ist.

Die Reihenfolge der Teilbäume ist also ausschlaggebend bei der Erstellung des Rechtebaumes. Die Reihenfolge der Bedingungen in einem Teilbaum dagegen (ob wir zuerst die Operation und dann die Eigenschaft prüfen oder umgekehrt) ist beliebig.

Für die Definition eines Teilbaumes des Rechtebaumes, ist es nicht unbedingt notwendig alle Filterarten zu verwenden. Ein Teilbaum besteht aus mindestens einem Filter und einem Entscheider. Eine Ausnahme ist der letzte Teilbaum der i.d.R. nur aus einem Entscheider besteht, der alle restlichen Operationen erlaubt (die vorher im Rechtebaum nicht verboten wurden) bzw. alle restlichen Operationen verbietet (die vorher im Rechtebaum nicht erlaubt wurden).

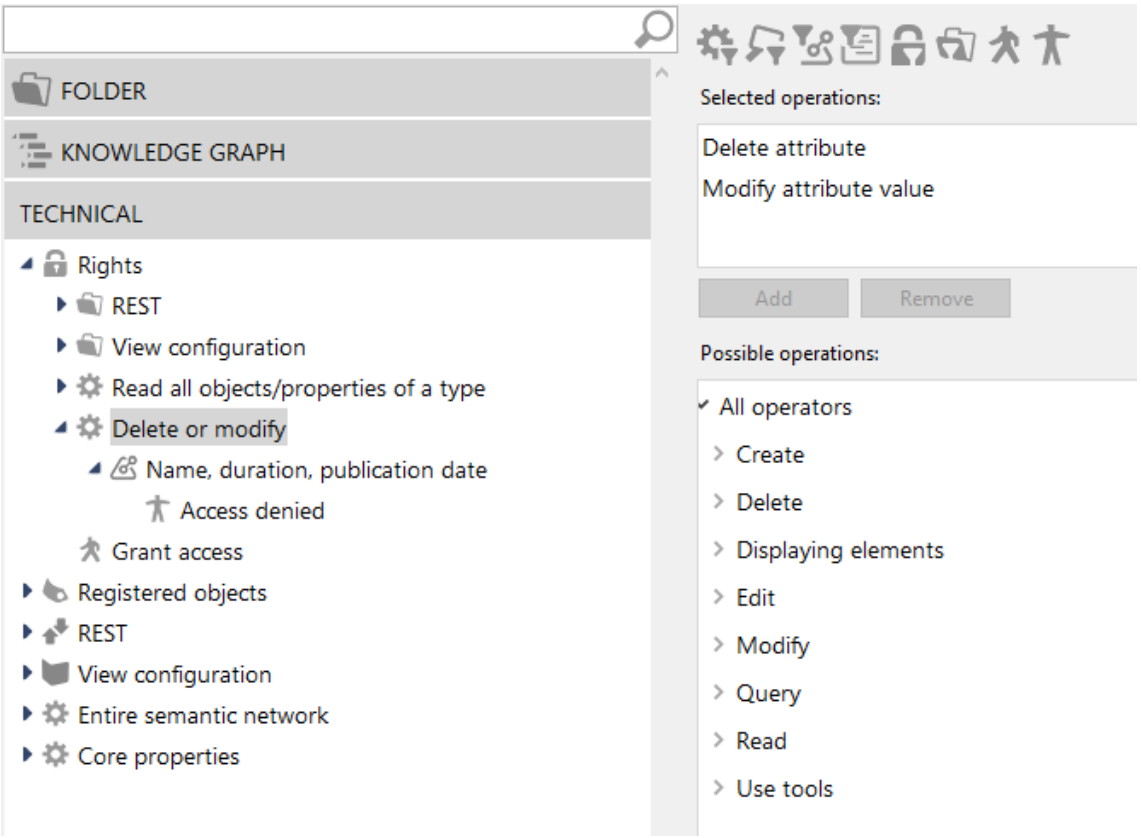
Beispiel: Rechtebaum

In dieses einfache Beispiel zeigt einen Rechtebaum bestehend aus einem Rechteteilbaum und einen Default-Entscheider, der alles erlaubt:

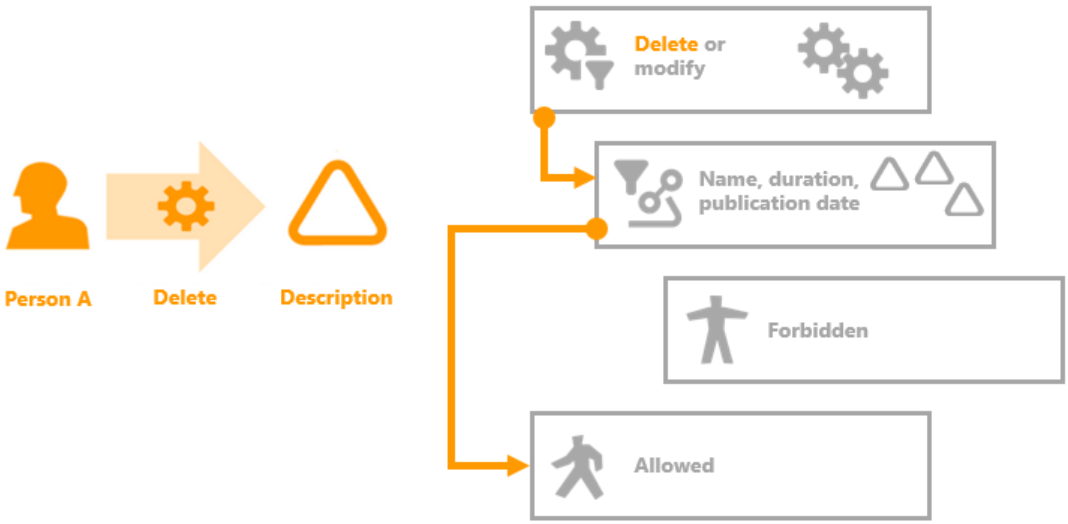


Im dem Rechteast wird das Löschen oder Modifizieren von den Attributen Name, Dauer und Erscheinungsdatum verboten. Dafür wird ein Operationsfilter verwendet, der die Operationen Löschen oder Modifizieren als Bedingung hat. Nur diese Operationen werden von diesem Operationsfilter durchgelassen. Der nächste Filter ist ein Eigenschaftsfilter, der auf bestimmte Eigenschaften filtert. In diesem Fall werden die Attribute Name, Dauer und Erscheinungsdatum gefiltert unabhängig davon, an welchem Objekt oder an welcher Eigenschaft diese gespeichert sind. Der letzte Knoten des Rechteast ist der Entscheider Verboten, der jede Zugriffsoperation verbietet, die auf die beiden vorgestellten Filter passt. Trifft eine der beiden Bedingungen nicht auf die Zugriffssituation zu, wird der Default Entscheider Erlaubt ausgeführt.

Dieser einfache Rechtebaum würde in i-views folgendermaßen aussehen:



Prüfung einer Operation anhand des Rechtebaum Beispiels:



Die linke Seite zeigt die zu prüfende Operation: Person A möchte das Attribut Beschreibung löschen. Auf der rechten Seite ist der Rechtebaum abgebildet. Die Prüfung der Bedingung des ersten Filters fällt positiv aus, da Person A die Operation Löschen durchführen möchte. Im Rechtebaum wird der nächste Filter des Rechteteilbaumes ausgeführt. Dies ist der Eigenschaftsfilter der Attribute Name,

Dauer und Erscheinungsdatum. Die Prüfung des Filters fällt negativ aus, da die Beschreibung keine der gefilterten Eigenschaften ist. Die Abarbeitung des Teilbaumes wird abgebrochen. Es wird zum nächsten Teilbaum des Rechtebaumes gewechselt. Dies ist bereits der Default-Entscheider "Erlaubt", der alles erlaubt, was nicht im Rechtebaum explizit verboten ist.

1.2.1.3. Entscheider im Rechtebaum

Entscheider stehen immer an der letzten Stelle eines Rechteteilbaumes. Durch die Kombination mit Filtern werden Zugriffssituationen 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.

HINWEIS	Wie alle anderen Labels des Rechtebaums auch, sind "Zugriff gewähren" und "Zugriff verweigern" Standard-Beschriftungen, welche je nach Bedarf geändert werden können.

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.

1.2.1.4. Zusammensetzen von Rechten

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.

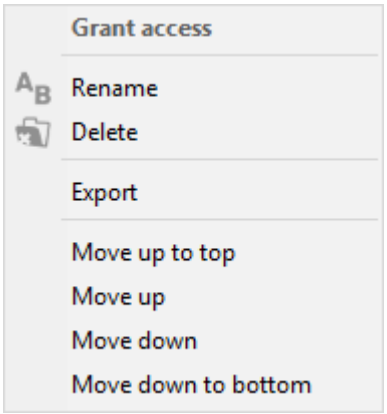
Um Regeln im Rechtesystem zu definieren stehen die folgenden Funktionen zur Verfügung:

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 Skriptfilter	Ein neuer Skriptfilter wird erstellt.
	Neuer Sperrfilter	Ein neuer Sperrfilter 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 gruppieren.

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

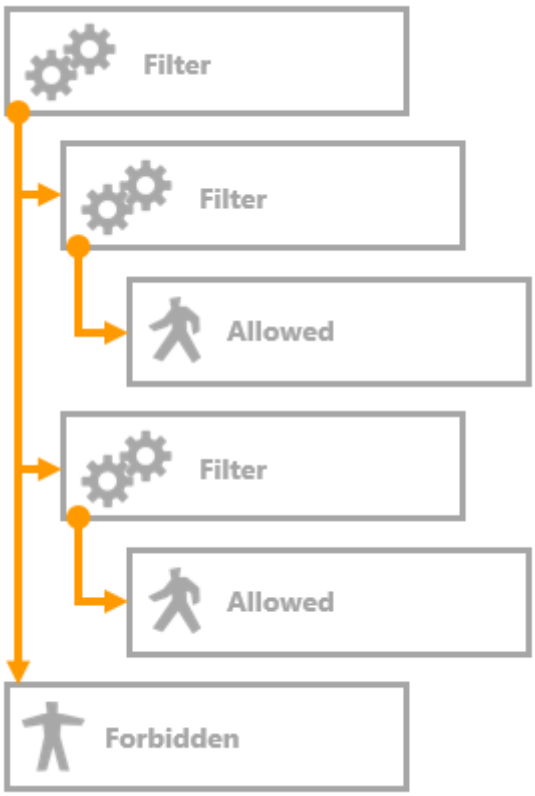
Durch das Zusammensetzen von Filtern und Entscheidern im Rechtebaum gibt es eine Vielzahl von Kombinationsmöglichkeiten um Rechte zu definieren. Es gibt grundsätzlich 3 verschiedene Vorgehensweisen um Rechte zu definieren:

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

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

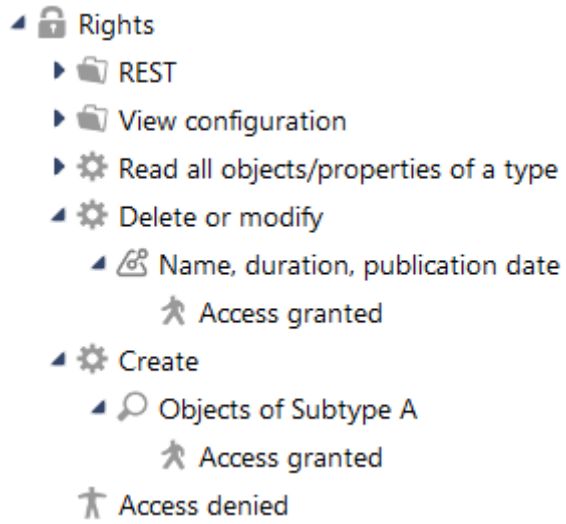
1.2.1.4.1. Positiv-Konfiguration von Rechten

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

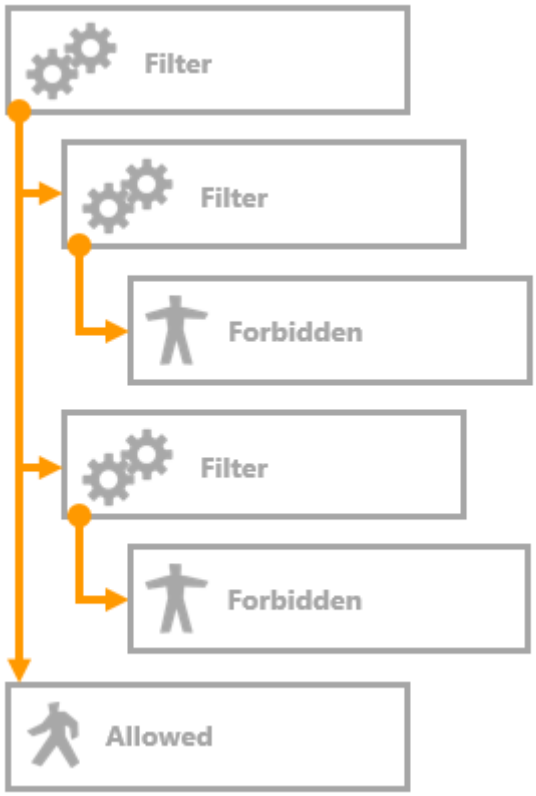
Dieses Beispiel zeigt, wie ein positiv formulierter Rechtebaum im Knowledge-Builder aussehen kann:



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.

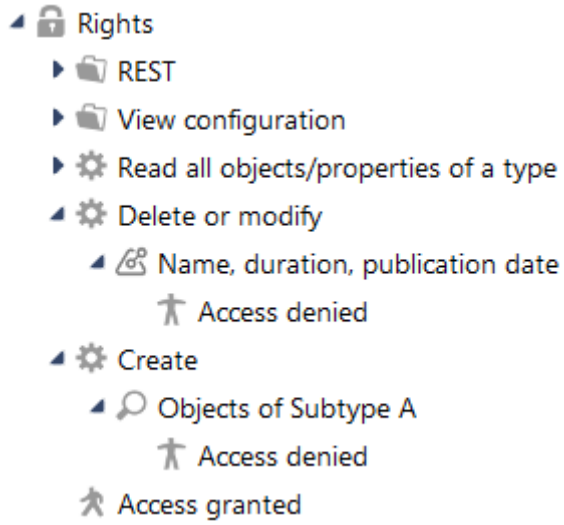
1.2.1.4.2. Negativ-Konfiguration von Rechten

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:

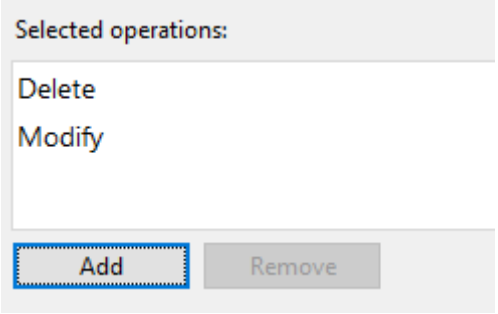


Der erste Teilrechtebaum verweigert im Gegensatz zum Beispiel Positiv-Konfiguration die Zugriffsrechte für das Löschen und Modifizieren der Attribute Name, Dauer und Erscheinungsdatum. Der Zweite Teilrechtebaum verbietet das Löschen der Relation die Songs mit dem Album verbindet, in dem sie enthalten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

1.2.1.4.3. Beispiel: Jeder Benutzer darf selbst erstellte Elemente ändern und löschen

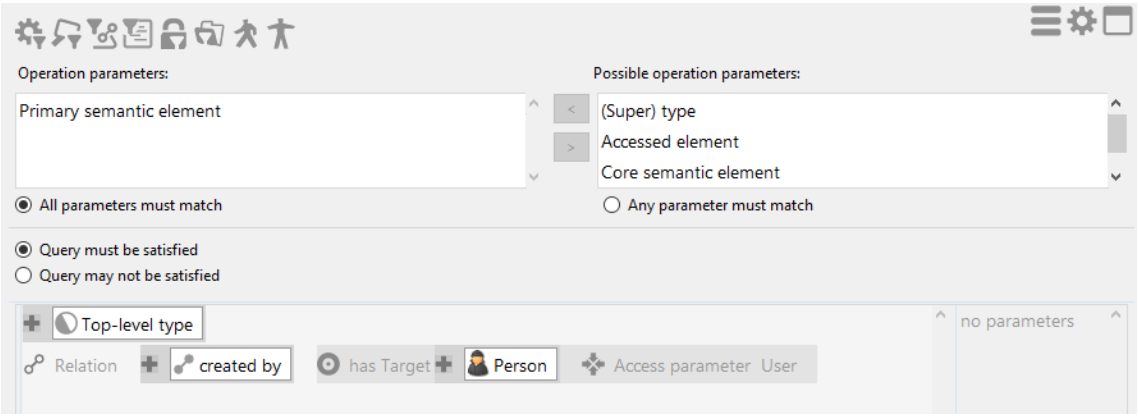
Was wird gebraucht um dieses Recht in i-views zu definieren? Zum einen wird ein Operationsfilter benötigt, da es um das Ändern und Löschen von Elementen geht. Zum anderen muss der Zusammenhang zwischen dem Benutzer und dem Element, an dem er eine Operation ausführen möchte, formuliert werden — das geht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

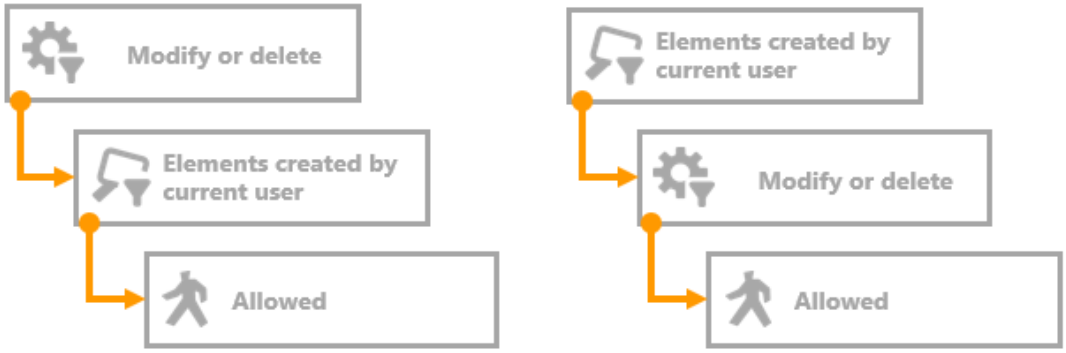
Ein Frage, die das Schema betrifft, ist: An welchen Elementen ist die Relation *wurde erstellt von* definiert? Es gibt verschiedene Möglichkeiten diese Relation in einem semantischen Netz umzusetzen:

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.

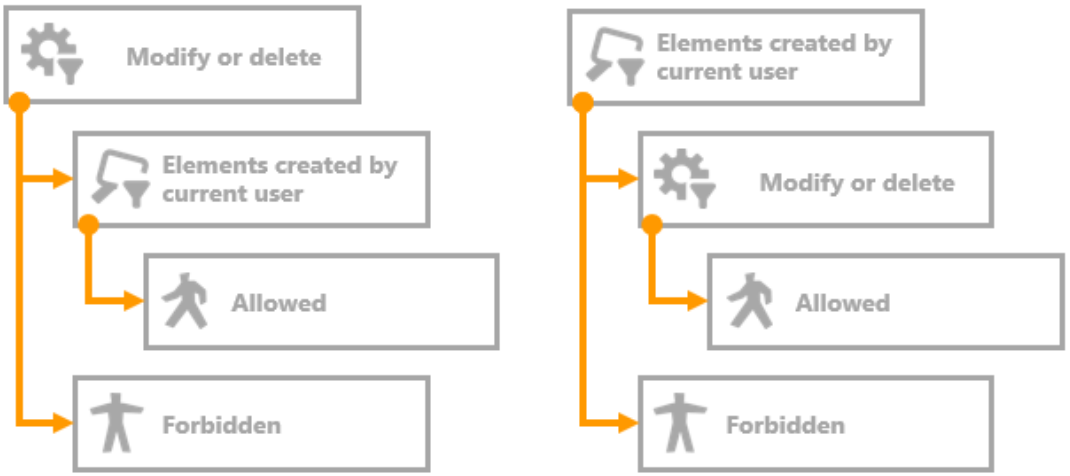


Die Graphik zeigt, die zwei möglichen Kombinationsweisen: Version 1 (links) erst Operationsfilter dann Suchfilter, Version 2 (rechts) erst Suchfilter dann Operationsfilter, als letztes folgt jeweils der Entscheider Erlaubt.

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 müssen 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.

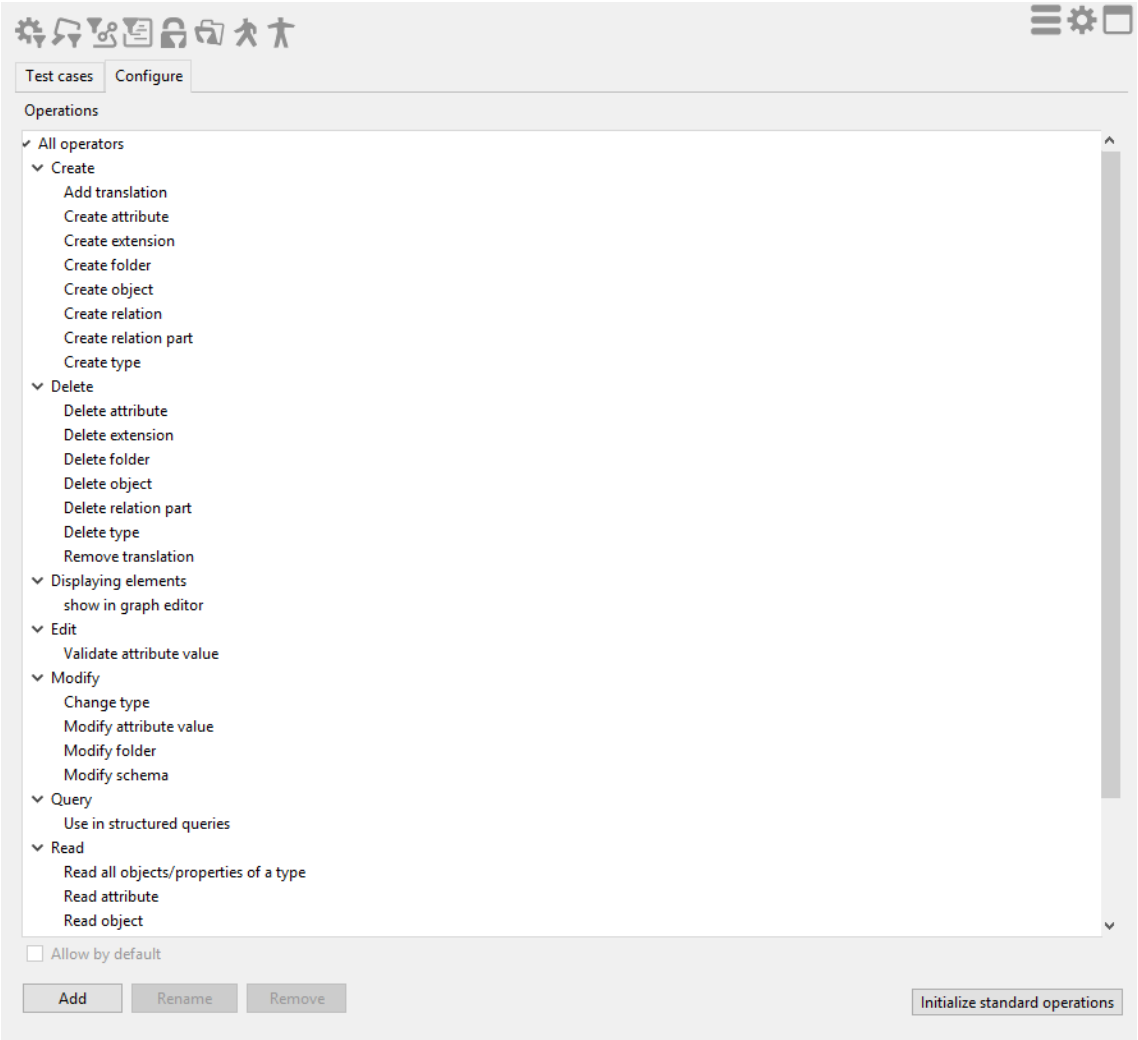
Version 2 (rechts)

- 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.

1.2.1.5. Konfiguration von eigenen Operationen

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.

1.2.2. 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 Trigger-Baum, 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 Zugriffssituation 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 Zugriffssituation, 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 Netz-spezifisch; Für den Einzelfall sollte eine Beratung durchgeführt werden.

1.2.2.1. Trigger aktivieren

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 des Trigger-Funktionalität heißt der Trigger Ordner im Technikbereich von i-views *Trigger (deaktiviert)*. Durch die Aktivierung wird der Ordner 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 Knowledge-Builder-Benutzer-Accounts mit einem Personenobjekt notwendig. Wie eine solche Verknüpfung erstellt wird, wird im Kapitel Aktivierung des Rechtesystems erklärt.

1.2.2.2. Der Triggerbaum

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 Trigger-Baum wird bei jeder Operation auf die Daten abgefragt — der Baum wird "traversiert". Passt ein Teilbaum auf die Zugriffssituation, 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 Trigger-Baum weiter durchlaufen, im Gegensatz zum Rechtesystem, dessen Abarbeitung mit dem Erreichen eines Entscheiders beendet ist. Um im Trigger-Baum 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 Trigger-Baumes wird beendet.

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

Symbol	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 Trigger-Baumes

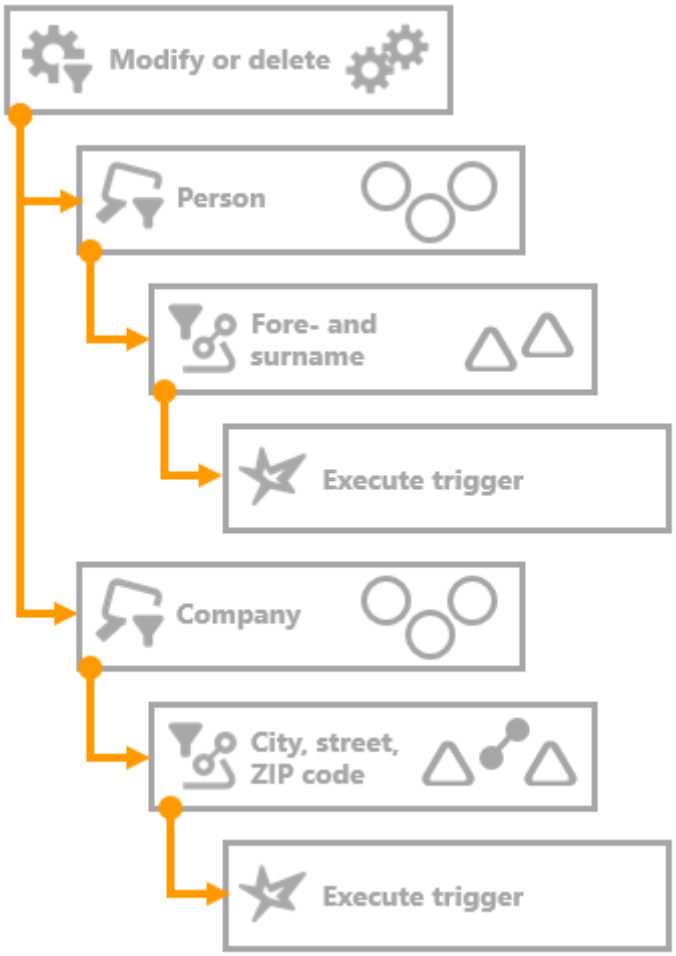
Bei der Gestaltung des Trigger-Baumes hat die Reihenfolge, in der man die Trigger definiert i.d.R. keinen Einfluss auf die Performance von i-views. Beim Rechtebaum gibt es Empfehlung zur Gestaltung, die aber nicht auf den Trigger-Baum übertragbar sind, da nach Ausführung einer Trigger-Aktion der Trigger-Baum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

Symbol	Funktion	Beschreibung
	Strukturordner	Strukturordner für die Gruppierung von Teilbäumen

Beispiel: Triggerbaum

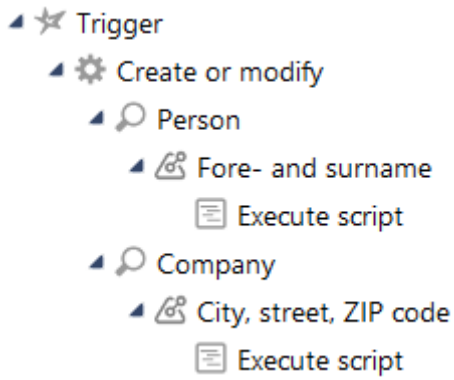
Dieses Beispiel zeigt einen Trigger-Baum, der die Namen von Personen und Konzerten automatisch aus Eigenschaften der Objekte zusammensetzt:



Dieser einfache Trigger-Baum 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:



1.2.2.3. Trigger erstellen

Wie im Abschnitt Trigger-Baum 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 Trigger-Baumes.

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 Trigger-Baum weiter durchlaufen. Alle Trigger-Aktionen der Teilbäume, die auf die Zugriffssituation zutreffen, werden ausgeführt.

1.2.2.4. Trigger-Aktionen

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 Trigger-Baum 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.

Logging-Aktionen

Prinzipiell stehen drei unterschiedliche Möglichkeiten zu Verfügung, durch das Trigger-System verursachte Änderungen zu loggen:

- **Log trigger:** Spezielles Logging-Element, welches zusätzlich zu einem Trigger-Element verwendet wird, um die Trigger-Vorgänge selbst zu loggen. Vorteil: Der Log trigger kann schnell zu jedem Skript Trigger hinzugefügt werden, jedoch muss hierzu im Voraus eine entsprechende Initialisierungs-Datei (*.ini) konfiguriert werden. Der Log Trigger ist im Unterkapitel "Log Trigger" beschrieben.
- **Skript Trigger mit Ausgabe in Form von "$k.log()":** Innerhalb jedes Trigger-Skripts können Einträge zum Logging mithilfe der Methode $k.log hinzugefügt werden. Vorteil: Die Logausgabe kann in höchst individueller Form definiert werden, lediglich begrenzt durch den Umfang der JavaScript API. Die Log-Informationen werden in den "Skriptmeldungen" ausgegeben und/oder in der betreffenden Log-Datei der Anwendung, welche entsprechend der Konfiguration der Initialisierungs-Datei entsteht. Für mehr Informationen hierzu, siehe i-views JavaScript API-Dokumentation.
- **changeLog-Trigger:** Die Anwendung eines vordefinierten, internen Namen auf ein Zeichenketten-Attribut erlaubt das Ablegen der Logging-Informationen als Attributwert des Attributs mithilfe von JavaScript-Methodenaufrufen. Vorteil: Die Log-Einträge werden in Form eines "changeLog"-Attributs direkt am semantischen Element erzeugt, das von den Änderungen betroffen ist, abhängig vom Definitionsbereich des changeLog-Attributtypen. Der changeLog-

Trigger ist um letzten Unterkapitel beschrieben.

1.2.2.4.1. Skript 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.

HINWEIS

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 durch geschrieben.

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**.

Vorsicht: 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.

Beispiel für Daten-Refactoring: Umwandlung in Einwegrelation. Das Umwandeln eines Relationstyps in eine Einwegrelation bewirkt ein Umspeichern von Relationszielen. Obwohl dies keine fachliche Änderung ist, kann dies ungewollt zur Ausführung eines Trigger-Skriptes führen, das ursprünglich nur dafür vorgesehen war, auf den Wechsel von Relationszielen zu reagieren.

Folgende Vorgänge gelten grundsätzlich als Daten-Refactoring:

Im Knowledge-Builder:

- "Eigenschaftsquelle neu wählen" (für Attribut)
- "Relationsquelle neu wählen" / "Relationsziel neu wählen" (an Relation)
- *Umkopieren*
- "Untertypen in Objekte umwandeln" (Kontextmenü "Überarbeiten")
- "Zusammenfassen" (von Knoten im Graph-Editor)
- *Relationen verschieben*
- Relationsquelle/-ziel im Graph-Editor per Drag&Drop ändern
- Umwandlung von Relationen von/zu Einwegrelationen

Allgemein:

- Änderung der Datenspeicherung bei Datei-Attributen
- Relationsquelle/-ziel ändern beim RDF -Import

Veraltet:

- Behavior-Function "adsorbRelationTarget" (wird nicht mehr benötigt)
- Relationsquelle/-ziel ändern in Edit-View im Web-UI (vor Version 5.4)

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 Attributwert 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)

1.2.2.4.2. 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 Zugriffssituation, 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 Zugriffssituation ein Trigger ausgelöst werden würde oder nicht, ohne dass die konkrete Zugriffssituation durchgeführt wird.

Die Durchführbarkeit des Log Triggers hängt im Eigentlichen davon ab, wie das Logging anhand der zugehörigen Initialisierungsdatei der Anwendung konfiguriert wurde (kb.ini, mediator.ini, jobclient.ini).

Beispiel: Minimal-Konfiguration einer "kb.ini" Datei für das Logging von Trigger-Aktionen eines lokalen Knowledge-Graph Volume ohne Mediator:

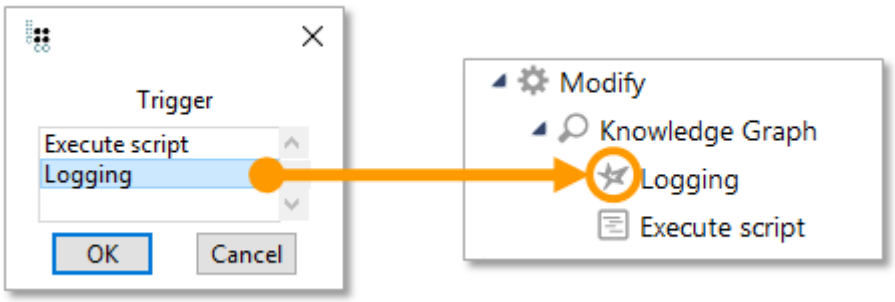
```
[Default]
logTargets = kblog

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

Diese Initialisierungsdatei erzeugt eine Logdatei "kb.log" im Knowledge-Builder Verzeichnis.

Für mehr Informationen zu Konfigurationsdateien, siehe Kapitel "i-views Services".

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:

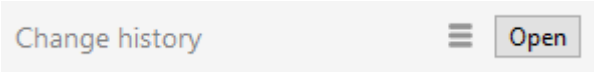
```
12.12.2019 14:15:51 #pre: Change value of attribute "e-mail" of "Person A" from "user123@iv.com" to "user123@iv.com"
12.12.2019 14:15:51 #post: Change value of attribute "e-mail" of "Person A" from "user123@iv.com" to "user1@iv.com"
12.12.2019 14:15:51 #end: Change value of attribute "e-mail" of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

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

1.2.2.4.3. ChangeLog Trigger

Möchte man die Aktivitäten von Nutzern an Objekten überwachen, sollte ein changeLog Trigger eingerichtet werden, auch bekannt als Änderungshistorie.

Dafür muss zunächst ein Zeichenketten-Attribut definiert werden, das den internen Namen "changeLog" erhält. Dieses changeLog Attribut muss für alle Elemente definiert werden, an denen es Nutzer-Aktivitäten dokumentieren soll.



Durch einen Klick auf "öffnen", öffnet sich die Tabelle, in der zu sehen ist wann, wer welche Änderung an welchem Wissensnetzelement mit welchem Wert getätigt hat.

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

HINWEIS

Weil Operationsfilter wie "Relation erzeugen", "Relationshälfte erzeugen" oder "Relationshälfte löschen" nur auf den Relationsursprung (dem semantischen Element selbst) angewendet werden, kann das Logging zu Änderungen der Relationsziele nicht getriggert werden. Für diesen Zweck kann stattdessen das Trigger-Skript verwendet werden, sofern entsprechend formuliert.

Modifikationen an Attributwerten werden nur dann geloggt, wenn sie erzeugt werden (zur gleichen Zeit, zu der das Attribut des Attributwert selbst erzeugt wird), aber nicht wenn der Attributwert gelöscht wird.

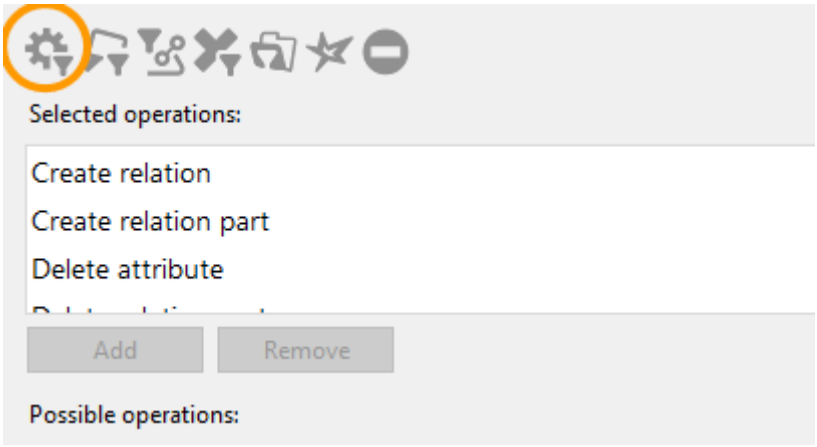
Der Trigger muss die Operationsfilter enthalten, die die Änderungshistorie protokollieren soll, und die Elemente, an denen das Attribut zu sehen sein soll.

Das Trigger-Skript sieht wie folgt aus:

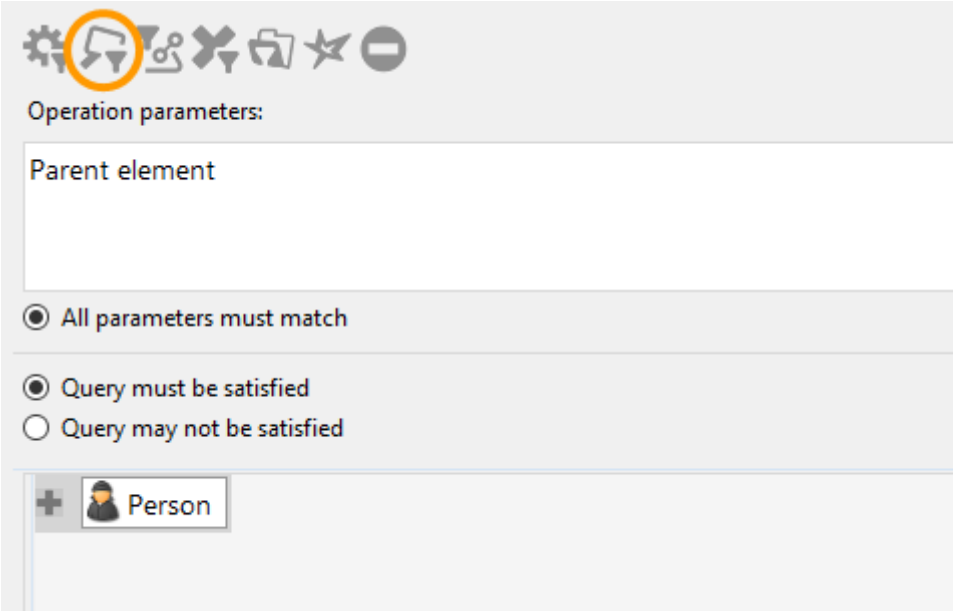
```
/**
 * Perform the trigger
 * @param parameter The chosen parameter, usually a semantic element
 * @param {object} access Object that contains all parameters of the
access
 * @param {$k.User} user User that triggered the access
**/

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

Beispiel

In einem Netz soll an allen Objekten aus dem Wissensnetz ein changeLog gespeichert werden. An den Objekten sollen Eigenschaften-Modifikationen, -erstellungen und -löschungen protokolliert werden. Es wurde dafür zunächst ein Operationsfilter angelegt, der auf die Operationen "Attribut löschen", "Attributwert modifizieren", "Relation erzeugen", "Relationshälfte erzeugen" und "Relationshälfte löschen" reagiert.

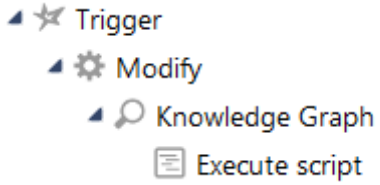


Im nächsten Schritt wurde ein Suchfilter definiert, der festlegt, welches die übergeordneten Elemente sind, an denen die Operationen getätigt werden.



Beim Trigger-Skript wurde der Operationsparameter "Übergeordnetes Element" eingestellt, weil dieser dem Suchfilter entspricht.

Die Trigger-Regeln (Operationsfilter, Suchfilter und Trigger-Skript) sind durch ihre Prüfabfolge wie folgt im Hierarchiebaum verortet:



1.2.3. Filterarten

Mithilfe von Filtern werden die Bedingungen im Rechtebaum bzw. im Trigger-Baum 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

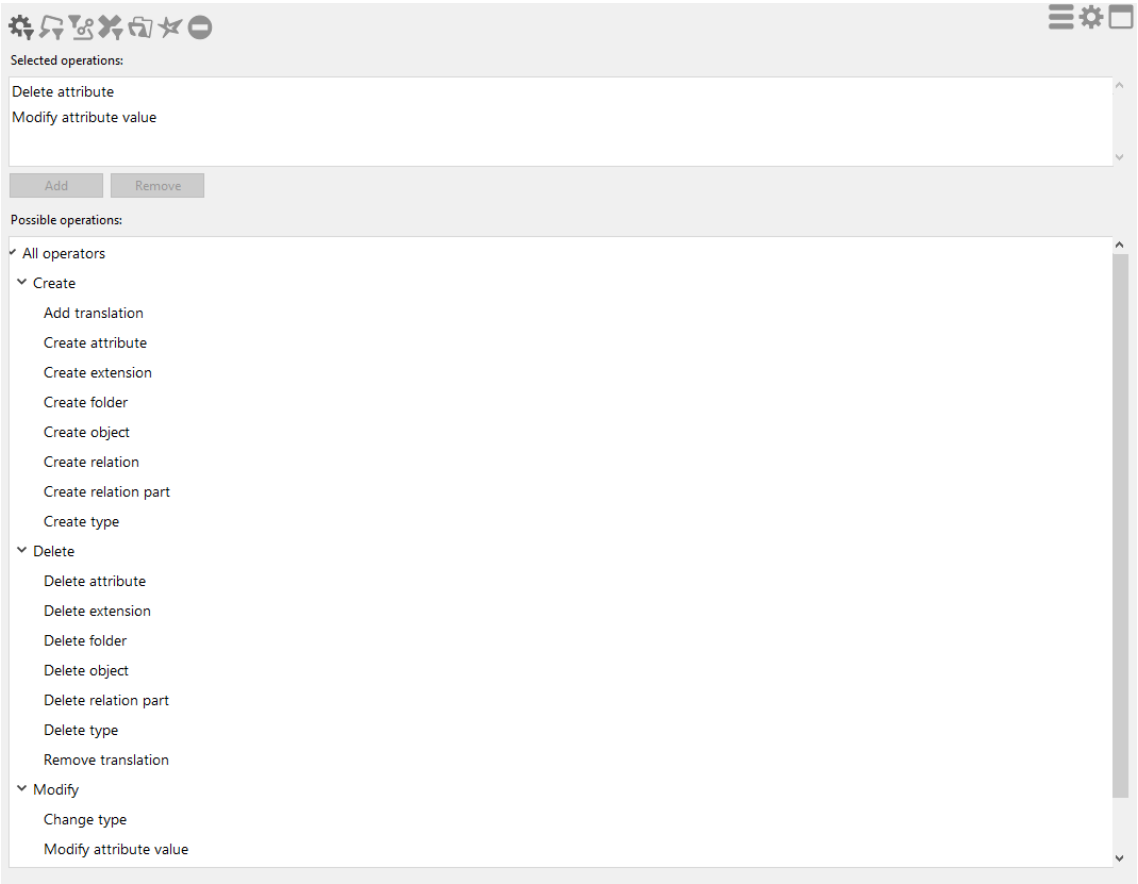
Operationen können nur mit einem Operationsfilter bestimmt werden. Benutzer können nur durch Suchfilter bestimmt werden. Eigenschaften können entweder mit Such- oder Eigenschaftsfiltern bestimmt werden. Die Verwendung von Eigenschaftsfiltern ist dann sinnvoll, wenn unabhängig von weiteren Eigenschaften 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. Trigger-Baum 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.

1.2.3.1. Operationsfilter

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.



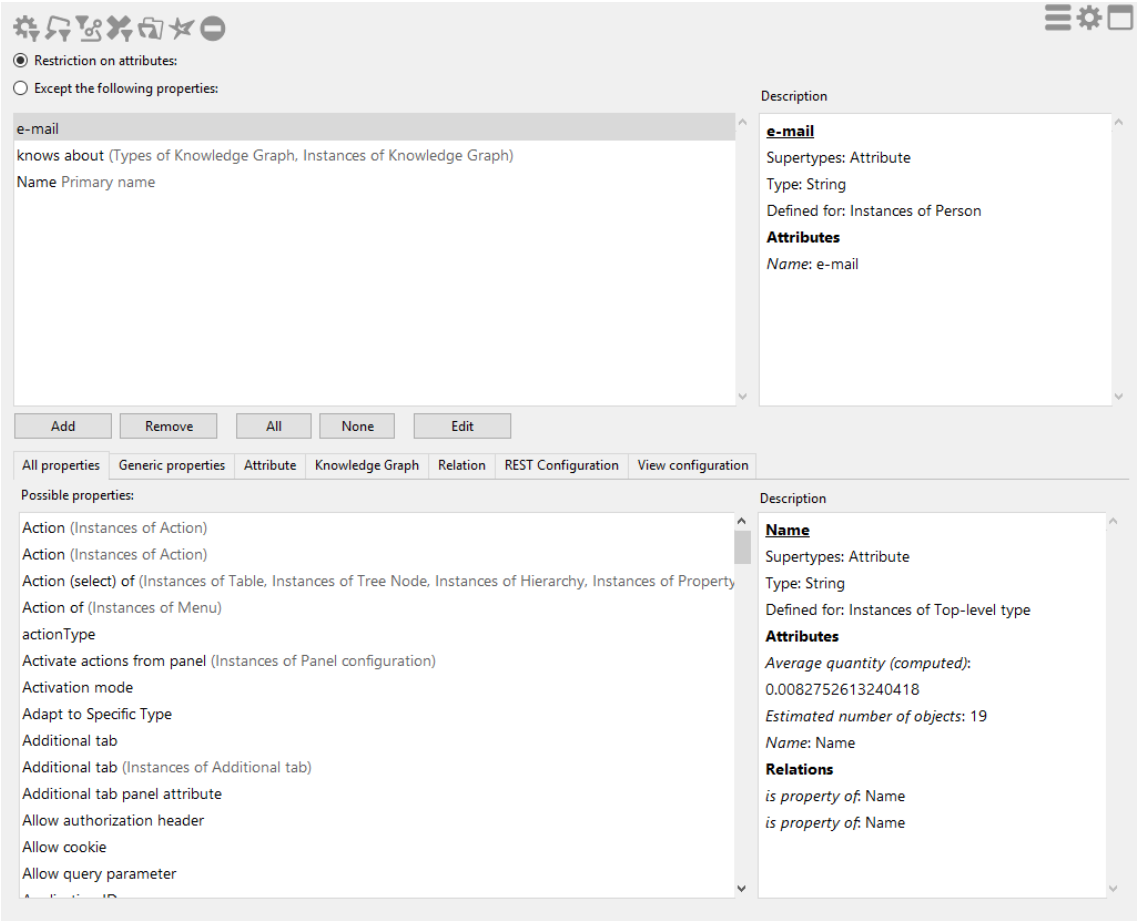
Die Operationen sind in Gruppen gegliedert. Wählt man den übergeordneten Knoten einer Gruppe aus, werden auch alle darunterliegenden Operationen mit gefiltert. Wenn beispielsweise die *Erzeugen* Operation ausgewählt wird, dann werden die Operationen *Attribut erzeugen*, *Erweiterung erzeugen*, *Ordner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

1.2.3.2. Eigenschaftsfilter

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.

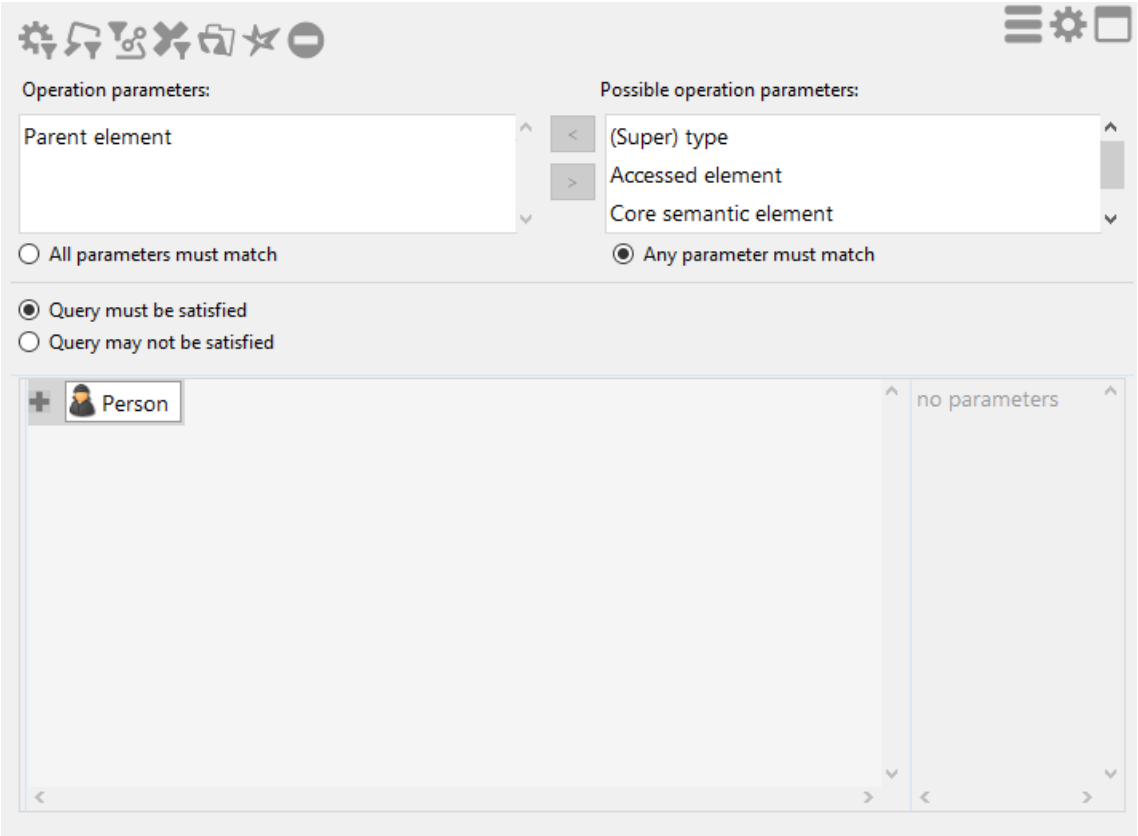
1.2.3.3. Suchfilter

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

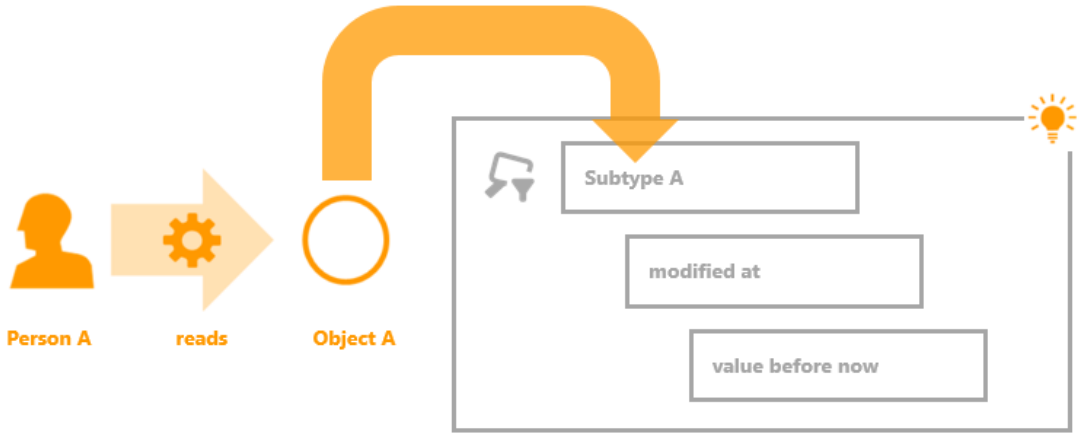
Bei der Auswahl der Operationsparameter kann konfiguriert werden, ob

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

HINWEIS	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.

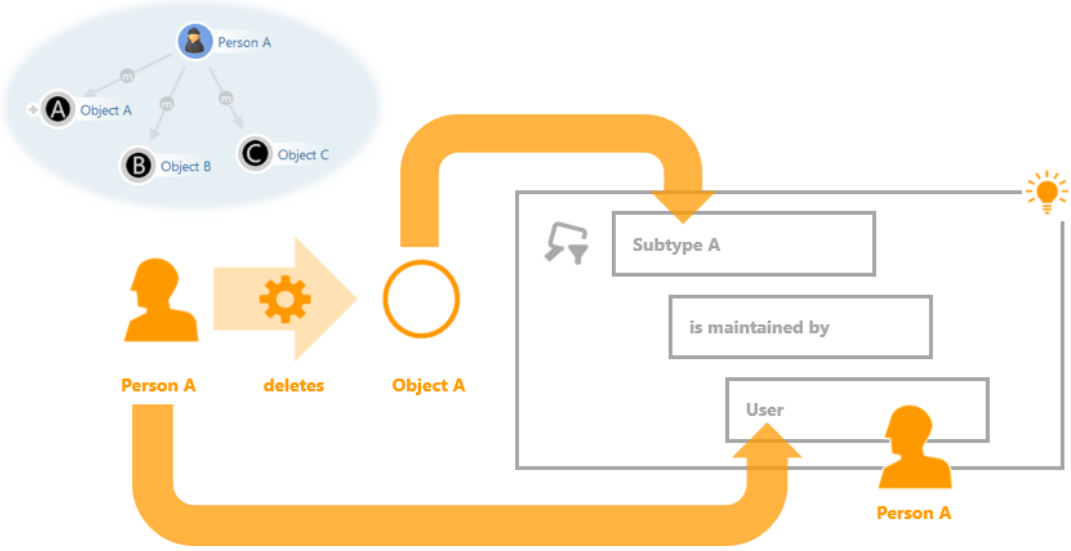


In diesem Beispiel möchte die Benutzer Person A das Objekt A lesen. Diese Operation wird nun vom Rechtesystem geprüft. Dort ist ein Suchfilter definiert, der prüft, ob das Objekt bereits verändert wurde. In der Strukturabfrage des Suchfilters werden Objekte vom Typ A gesucht, mit der Einschränkung, dass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

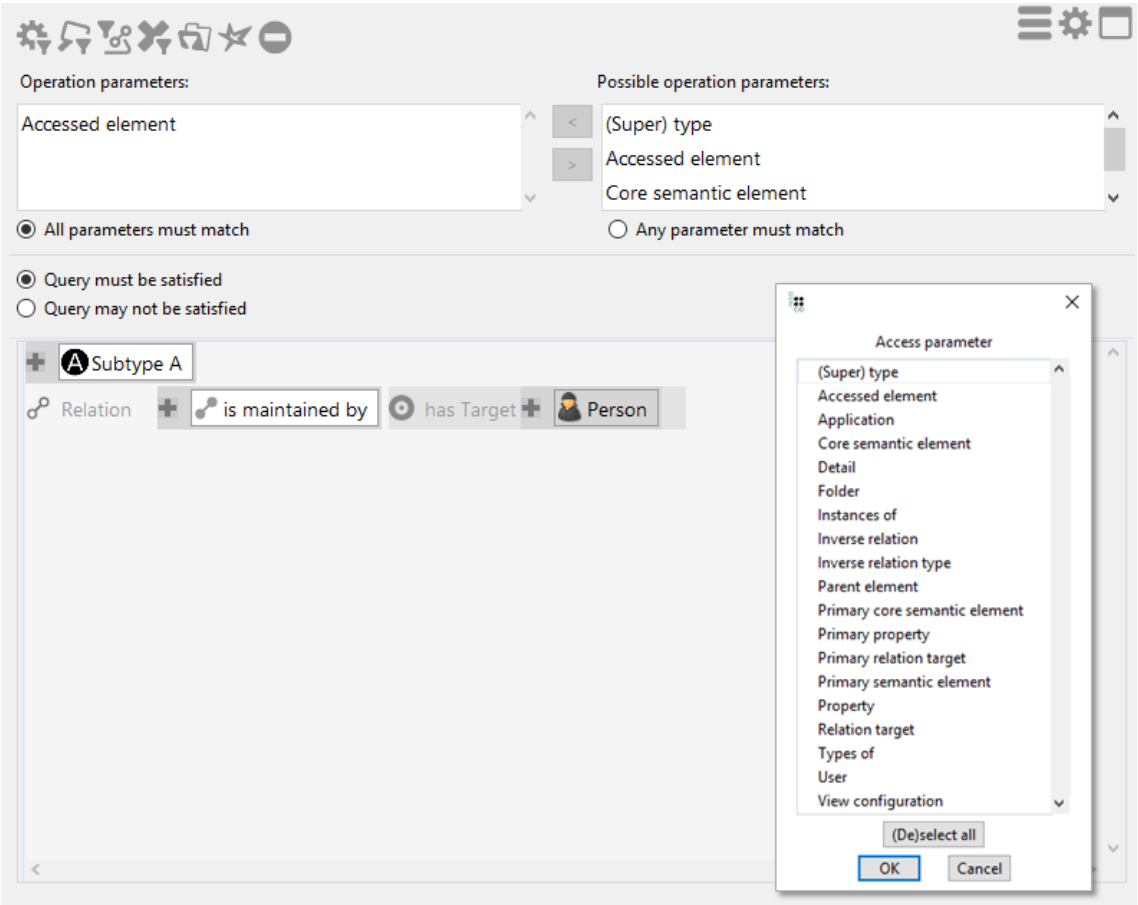
Bei dem Suchfilter wurden die Einstellungen "Suchbedingung muss erfüllt sein" und "Alle Parameter müssen zutreffen" ausgewählt.

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 ein Objekt pflegen, dürfen dieses bearbeiten und löschen".



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

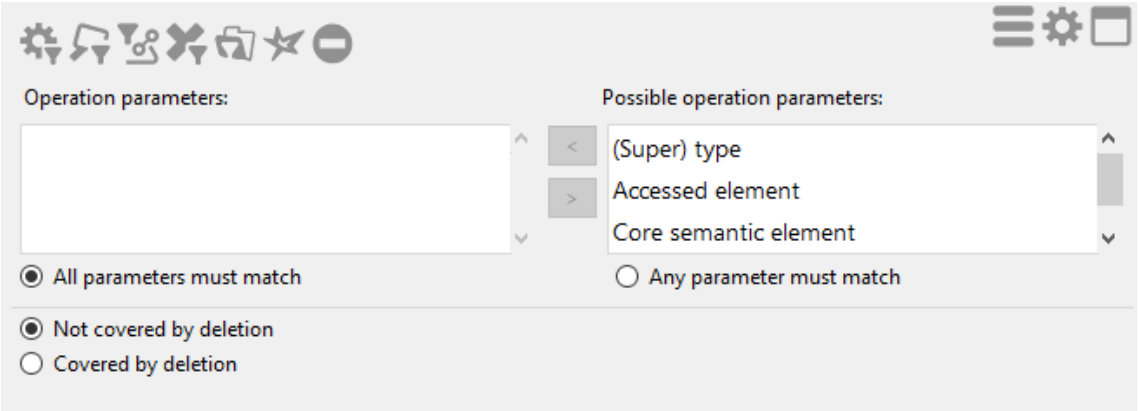


In diesem Beispiel möchte ein Benutzer ("Person") das Objekt vom Untertyp A löschen. Der dazugehörige Suchfilter liefert als Suchergebnis alle Objekte, die von einem bestimmten Benutzer gepflegt werden. Der aktuelle Benutzer wird als Zugriffsparameter in die Strukturabfrage übergeben. Zugriffsparameter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

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

1.2.3.4. Löschfilter

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.

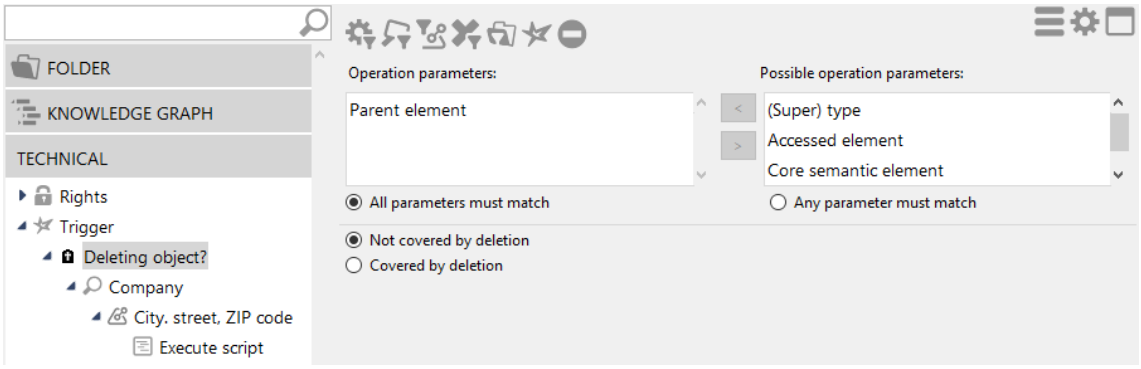
- *Alle Parameter müssen zutreffen* : Alle angegebenen Operationsparameter müssen zutreffen. Werden beispielsweise zwei Operationsparameter angegeben (Zugriffsobjekt und Primärobjekt), dann wird geprüft, ob der Löschvorgang sowohl für Zugriffsobjekt als auch für Primärobjekt gilt, das kann nur der Fall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

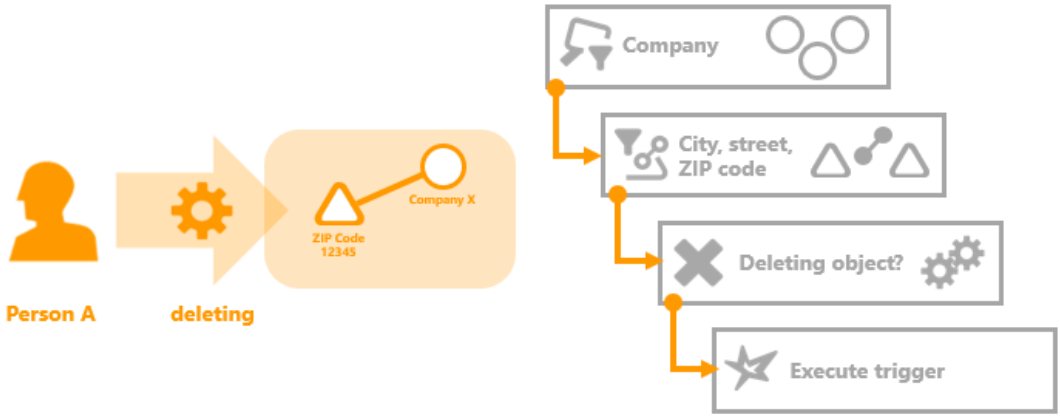
- *Nicht vom Löschvorgang erfasst* : Die Bedingung des Filters ist positiv, wenn das in Operationsparameter übergebene Element in dieser Transaktion nicht gelöscht wird.
- *Vom Löschvorgang erfasst* : Die Bedingung des Filters ist entsprechend positiv, wenn das in Operationsparameter übergebene Element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

Beispiel: Löschfilter bei Triggern

In diesem Beispiel soll ein Trigger nur dann ausgeführt werden, wenn die Stadt, die Straße oder die Postleitzahl eines Unternehmens 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 Unternehmens-Objekt selbst ist, dann wird die Prüfung des Teilbaumes, aufgrund des negativen Ergebnisses des Filters, abgebrochen.

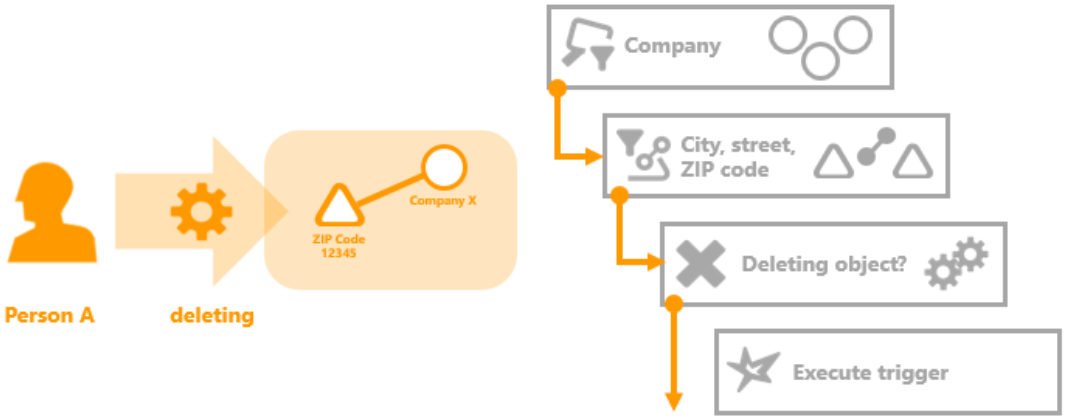


Verwendet wird der Operationsparameter Übergeordnetes Element und die Einstellung Nicht vom Löschvorgang erfasst.



In dieser beispielhaften Zugriffssituation wird das Attribut Postleitzahl mit dem Wert "12345" am Objekt "Unternehmen X" gelöscht. Das Objekt selbst wird nicht gelöscht. Der Suchfilter "Unternehmen", der mit dem Operationsparameter übergeordnetes Zugriffselement definiert ist, und der Eigenschaftsfilter "Stadt, Straße, Postleitzahl" 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 Zugriffssituation wird das Objekt "Unternehmen X" vom Nutzer "Person A" 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 "Unternehmen" und der Eigenschaftsfilter "Stadt, Straße, Postleitzahl" sind in der Prüfung des Triggerbaumes für den Löschvorgang des Attributs erfüllt. Der Löschfilter selbst ist in dieser Situation nicht erfüllt, da das Objekt "Unternehmen X" an dem die Eigenschaft "Postleitzahl 12345" 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.

1.2.4. Operationsparameter

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.
Zugriffselement	Das *Zugriffselement* ist das von der Operation betroffene Element.
Anwendung	Objekt des Typs "Anwendung" (zu finden unter TECHNIK > View-Konfiguration > Objekttypen > Anwendung).
Kernelement	Wenn das übergeordnete Element eine Erweiterung ist, dann ist das *Kernelement* das Objekt, an dem die Erweiterung gespeichert ist. Ansonsten ist das *Kernelement* identisch mit Zugriffselement.
Ordner	Der Operationsparameter *Ordner* ist der von der Operation betroffene Ordner.
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.
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. Wenn das Zugriffselement eine Metaeigenschaft ist und das übergeordnete Element eine Relation ist, dann ist folgendes zu beachten: • Aufgrund der symmetrischen Speicherung von Metaeigenschaften an Relationshälften ist bei beidseitigen oder symmetrischen Relationen die zurückgegebene Relationsrichtung nicht eindeutig. Hierbei ist die benötigte Relationsrichtung mittels Skript oder Strukturabfrage zu ermitteln. • Bei einseitigen Relationen ist das übergeordnete Element die reelle Relationshälfte (d. h. nicht die virtuelle Relationshälfte).

Operationsparameter	Beschreibung
Primäres Kernelement	Wenn das Primärelement eine Erweiterung ist, dann ist das *Primäre Kernelement* das Kernelement der Erweiterung. Ansonsten ist das *Primäre Kernelement* identisch mit Kernelement.
Primärelement	Falls das übergeordnete Zugriffselement eine Eigenschaft ist, dann 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 (d. h. wenn keine Eigenschaft vorliegt, sondern nur das Element).
Primäreigenschaft	Bei Metaeigenschaften ist die *Primäreigenschaft* die dem Objekt, Typ oder Erweiterung nächste Eigenschaft. Ansonsten ist *Primäreigenschaft* identisch mit Eigenschaft.
Primäres Relationsziel	Das *Primäre Relationsziel* ist das Primärelement des Relationsziels.
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.
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.)
Benutzer	Der *Benutzer* ist das Objekt des Benutzers, der die Operation ausführt.

1.2.4.1. Operationsparameter (Ober)typ

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

1.2.4.2. Operationsparameter Zugriffselement

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 Zugriffssituation 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.

1.2.4.3. Operationsparameter Anwendung

Der Operationsparameter "Anwendung" bezieht sich auf den Anwendungskontext, innerhalb dessen auf das Element zugegriffen wird. Beispiele für eine Anwendung sind der Knowledge-Builder oder der Viewkonfiguration-Mapper.

Zugriffselement	Anwendung
Objekt, Typ oder Erweiterung	Objekt der aktuell verwendeten Anwendung

1.2.4.4. Operationsparameter Kernelement

Das Kernelement wird verwendet, wenn mit Erweiterungen gearbeitet wird. Das Kernelement 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

1.2.4.5. Operationsparameter Ordner

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

1.2.4.6. Operationsparameter 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

1.2.4.7. Operationsparameter Inverser Relationstyp

Der inverse Relationstyp ist der Typ der inversen Relation.

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

1.2.4.8. Operationsparameter Übergeordnetes 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

1.2.4.9. Operationsparameter Primäres Kernelement

Wenn bei einem Zugriffselement das zugehörige Objekt oder der zugehörige Typ adressiert werden soll, muss das primäre Kernelement verwendet werden. Im Gegensatz zum Primärelement werden beim primären Kernelement keine Erweiterungen adressiert/zugelassen. Bei Erweiterungen als Zugriffselement wird das Kernobjekt ausgegeben.

Zugriffselement	Primäres Kernelement
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)

1.2.4.10. Operationsparameter Primärelement

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

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

1.2.4.11. Operationsparameter Primäreigenschaft

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

1.2.4.12. Operationsparameter Primäres Relationsziel

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

1.2.4.13. Operationsparameter Eigenschaft

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

1.2.4.14. Operationsparameter Relationsziel

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

1.2.4.15. Operationsparameter Benutzer

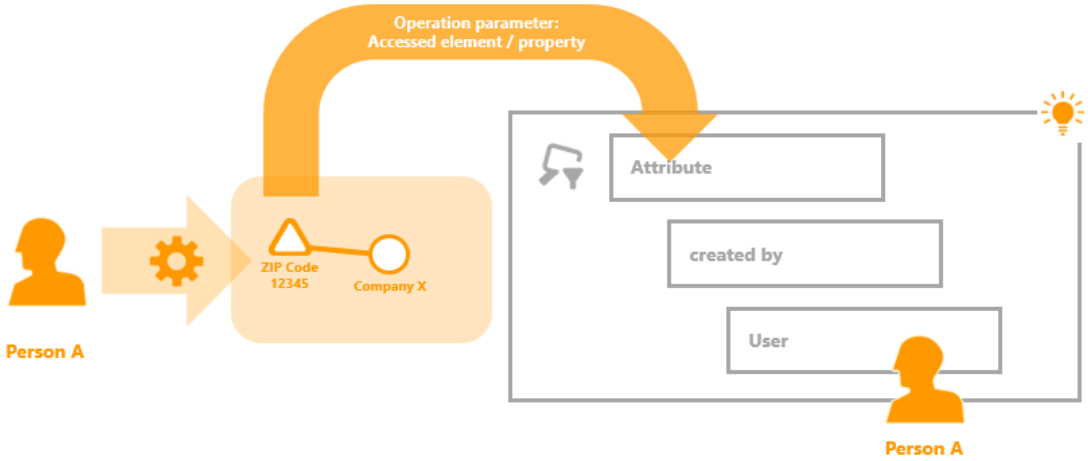
Der Parameter Benutzer ist unabhängig vom Zugriffselement immer das Benutzerobjekt des aktuell angemeldeten Nutzers. Hierfür muss der Knowledge-Builder-Account mit einem Wissensnetzobjekt verknüpft werden. Wie die Verknüpfung vorgenommen wird, wird im Kapitel Aktivierung des Rechtesystems vorgestellt.

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

1.2.4.16. Beispiele: Die Verwendung von Operationsparametern

Beispiel 1: Zugriffselement und Eigenschaft im Rechtesystem

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



Zugriffssituation: *Person A* möchte das Attribut *Postleitzahl* von *Unternehmen X* ändern.

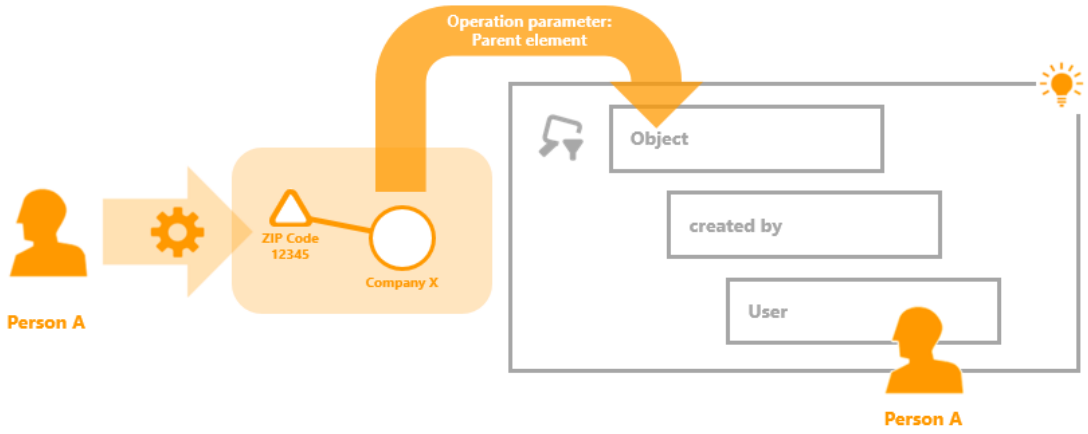
Suchfilter: Es werden alle Attribute gefiltert die, die von einem bestimmten Benutzer angelegt wurden. In der Strukturabfrage wird der Zugriffsparameter Benutzer verwendet, der die Objekte von Nutzer auf die Person einschränkt, welche die Operation ausführen möchte. Entsprechend sind das alle Attribute, die von *Person A* 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 *Postleitzahl* 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 Zugriffssituation und auf der rechten Seite den dazugehörigen Suchfilter.



Zugriffssituation: *Person A* nimmt eine Änderung des Attributes *Postleitzahl* vor, das aktuell den Wert *12345* annimmt und zum Objekt *Unternehmen X* gehört.

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 *Person A* angelegt wurden.

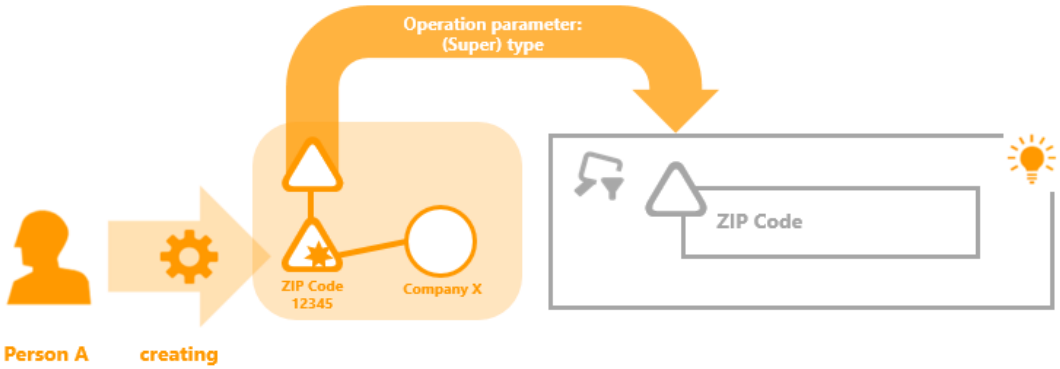
Prüfung der Zugriffsrechte: Ist in der Ergebnismenge des Suchfilters des *Unternehmen X* enthalten, wird der nachfolgende Ordner (Filter oder Entscheider) ausgeführt.

Operationsparameter: Die Verwendung des Operationsparameter übergeordnetes Element führt dazu, dass nicht das Attribut *Postleitzahl* an dem die Änderung stattfinden soll, an den Suchfilter übergeben wird, sondern das Objekt an dem es definiert wurde. Das ist in diesem Fall das *Unternehmen 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 Zugriffssituation dar und auf der rechten Seite wird der Suchfilter abgebildet, der in dieser Situation zum Einsatz kommt.



Zugriffssituation: *Person A* möchte das Attribut *Postleitzahl* am Objekt *Unternehmen X* erstellen. Es soll den Wert *12345* haben.

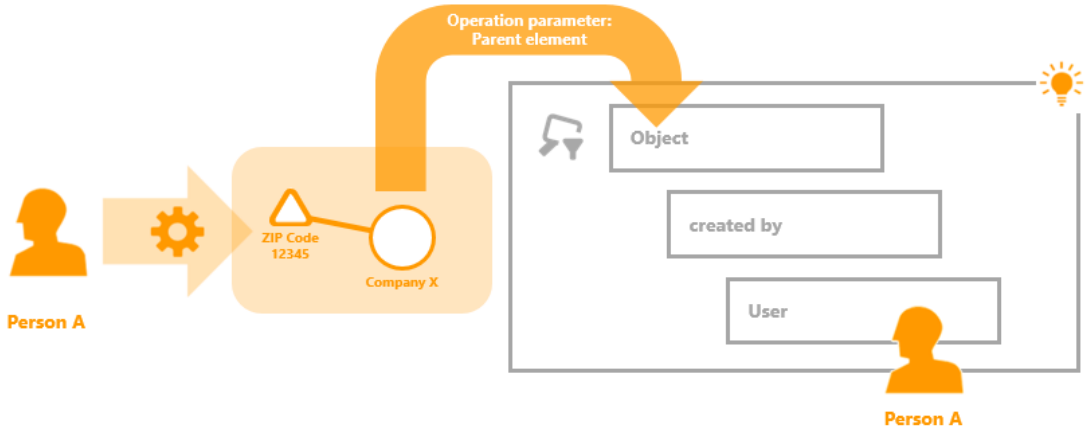
Suchfilter: Der Suchfilter liefert den Attributtyp *Postleitzahl*.

Prüfung der Zugriffsrechte: Ist das zu erstellende Attribut vom Typ *Postleitzahl*, 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 *Postleitzahl*.

Beispiel 2: Übergeordnetes Element und primäres semantisches Element im Rechtesystem

Diese Beispiel zeigt die Zugriffssituation auf der linken Seite und der zugehörige Abfragefilter auf der rechten Seite.



Zugriffssituation: Person A ändert das Postleitzahl-Attribut, welches momentan den Wert 12345 hat und Teil des Unternehmens X ist.

Suchfilter: Der Suchfilter wird so definiert, dass er nach allen Objekte sucht, welche durch einen bestimmten Nutzer angelegt wurde; der momentan eingeloggte Nutzer ist das "Zugriffselement". Dementsprechend findet der Suchfilter alle Objekte, welche von Person A erzeugt wurden.

Prüfen der Zugriffsrechte: Wenn die Ergebnismenge des Suchfilters das Unternehmen X enthält, wird der folgende Ordner (Filter oder Entscheider) ausgeführt.

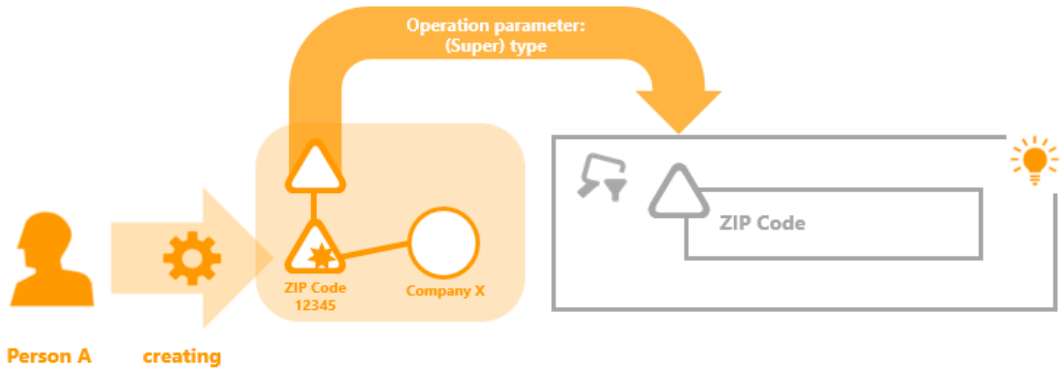
Operationsparameter: Die Benutzung des Operationsparameters "Übergeordnetes Element" hat zur Folge, dass nicht das zu ändernde Attribut "Postleitzahl" an den Suchfilter weitergereicht wird, sondern das Objekt weitergereicht wird, für welches das Attribut definiert wurde. In diesem Fall ist es das Unternehmen X.

Zusätzlich zur Verwendung des übergeordneten Elements ist es möglich, den Operationsparameter "Primäres semantisches Element" zu verwenden.

Der Operationsparameter "Übergeordnetes Element" würde zum Ergebnis führen, dass alle Eigenschaften und das Objekt selbst durch den Filter positiv gewertet würden. Zusätzlich würde der Operationsparameter "Primäres semantisches Element" die Metaeigenschaften des Objektes zulassen, ungeachtet davon, wie viele Eigenschaften sich zwischen dem Objekt und der Metaeigenschaft befinden.

Beispiel 3: (Ober) Typ im Rechtesystem

Das Beispiel zeigt die Zugriffssituation auf der linken Seite und den in dieser Situation angewendeten Suchfilter auf der rechten Seite.



Zugriffssituation: Person A möchte das Attribut "Postleitzahl" für das Objekt "Unternehmen X" erzeugen. Das Attribut soll den Wert "12345" erhalten.

Suchfilter: Der Suchfilter gibt den Attributtyp "Postleitzahl" zurück.

Prüfung der Zugriffsrechte: Wenn das zu erstellende Attribut vom Typ "Postleitzahl" ist, dann fällt das Ergebnis bei Prüfung des Suchfilters positiv aus.

Operationsparameters: Wenn Element erzeugt werden, so ist es nicht möglich einen Suchfilter zu definieren, welcher das erst zu erstellende Element zurückgibt und daher die Zugriffsrechte prüft.

Dies bedeutet, dass ein anderer Operationsparameter für das Zugriffselement gewählt werden muss, wenn Elemente erst noch erzeugt werden.

Der Operationsparameter "(Ober) typ" ist für diese Situation die geeignete Lösung. In diesem Beispiel wird der Attributtyp verwendet, welcher für Postleitzahlen definiert ist.

1.2.5. Operationen

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. Trigger-Baumes 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 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.

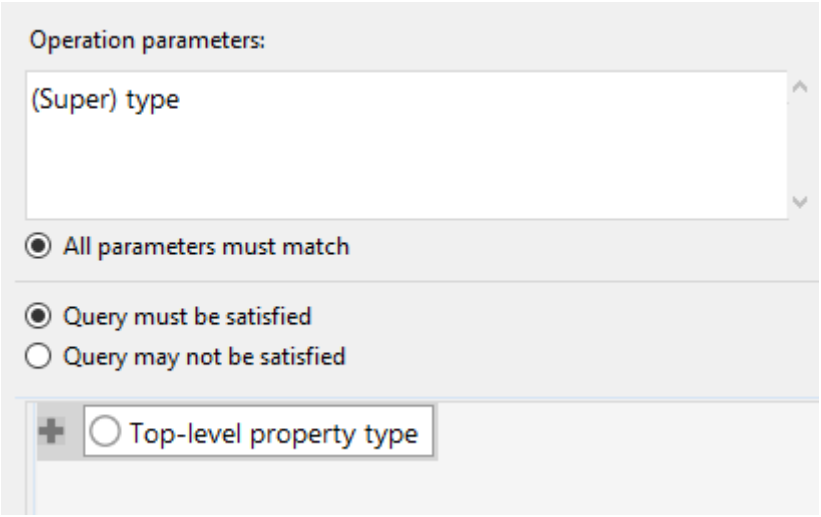
Operationsgruppe	Operation	Operationsparameter
Anzeigen von Objekten	im Grapheditor anzeigen	Zugriffselement
Bearbeiten	Attributwert validieren	Zugriffselement, Eigenschaft, übergeordnetes Element, (zu prüfender Parameter: Attributwert)
Benutzerdefinierte Operation		
Erzeugen	Attribut erzeugen	(Ober)typ, übergeordnetes Element
	Erweiterung erzeugen	(Ober)typ, übergeordnetes Element, Kernobjekt

Operationsgruppe	Operation	Operationsparameter
	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

Operationsgruppe	Operation	Operationsparameter
	Übersetzung entfernen	Zugriffselement, Eigenschaft, übergeordnetes Element
Modifizieren	Attributwert modifizieren	Zugriffselement, Eigenschaft, übergeordnetes Element
	Ordner modifizieren	Ordner
	Schema modifizieren	Zugriffselement, übergeordnetes Element
	Typ wechseln	Zugriffselement, übergeordnetes Element
Werkzeuge verwenden	Export	*wird nicht mehr ausgewertet*
	Import	*wird nicht mehr ausgewertet*
	Script bearbeiten/ausführen	*wird nicht mehr ausgewertet*

Objekt lesen Die Operation *Objekt lesen* deckt das Anzeigen von Objekten auf dem Reiter Objekte bei dem entsprechenden 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.



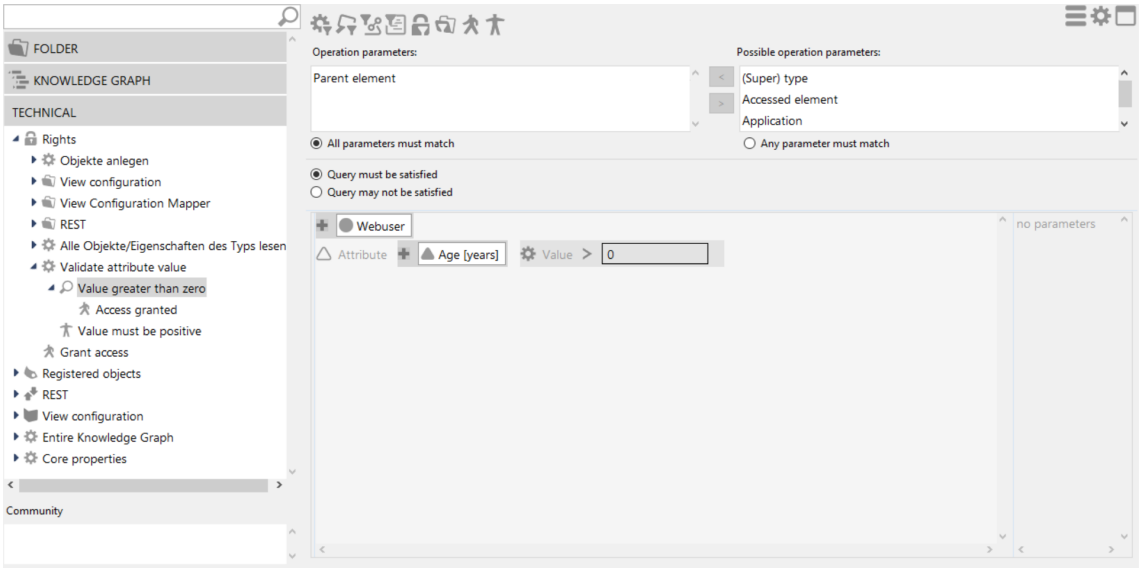
Zur Überprüfung, welche Zwischenergebnisse geprüft werden, kann man diese Information in einer Struktursuche einblenden lassen. Dies geschieht über "Einstellungen > Persönlich > Strukturabfrage > Leserechtprüfungen anzeigen".

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.

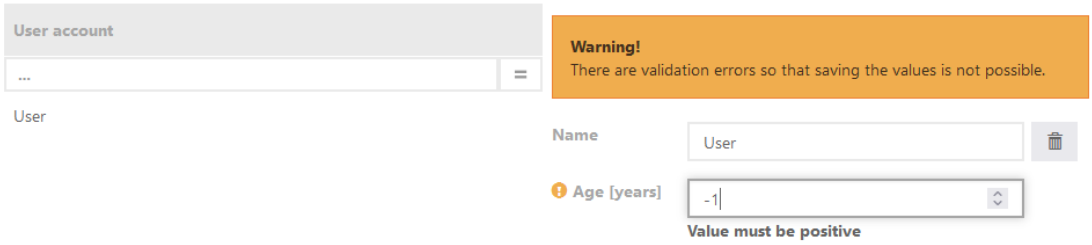
Fallbeispiel einer Konfiguration: Das eingegebene Alter eines Webusers muss größer null sein.

Konfiguration per Strukturabfrage im Rechtesystem: Das Attribut "Alter [Jahre]" des zugegriffenen Parents "Objekte vom Typ 'Webuser'" beinhaltet die Bedingung "Wert > 0". Wenn dies zutrifft, dann darf der Wert abgespeichert werden — dies wird konfiguratorisch durch "Zugriff gewähren" gesteuert, für alle anderen Fälle wird das Abspeichern mit "Zugriff verweigern" verweigert.

HINWEIS	Die Benennung "Wert muss positiv sein" des Stopmarkers wird bei der Auswertung als Fehlermeldung im Web-Frontend mit angezeigt.



Anzeige im Web-Frontend: Bei Eingabe eines falschen Wertes gibt der Validator eine gelbe Warnmeldung aus, mit dem Text des Stopmarkers unterhalb des betreffenden Eingabefelds:



In der Strukturabfrage 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:

Eigenschaft	Wert
attributeValue	zu setzender Wert
property	zu ändernde Eigenschaft (Attribut)
topic	Element der Eigenschaft
user	Nutzer, der den Wert setzt

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.

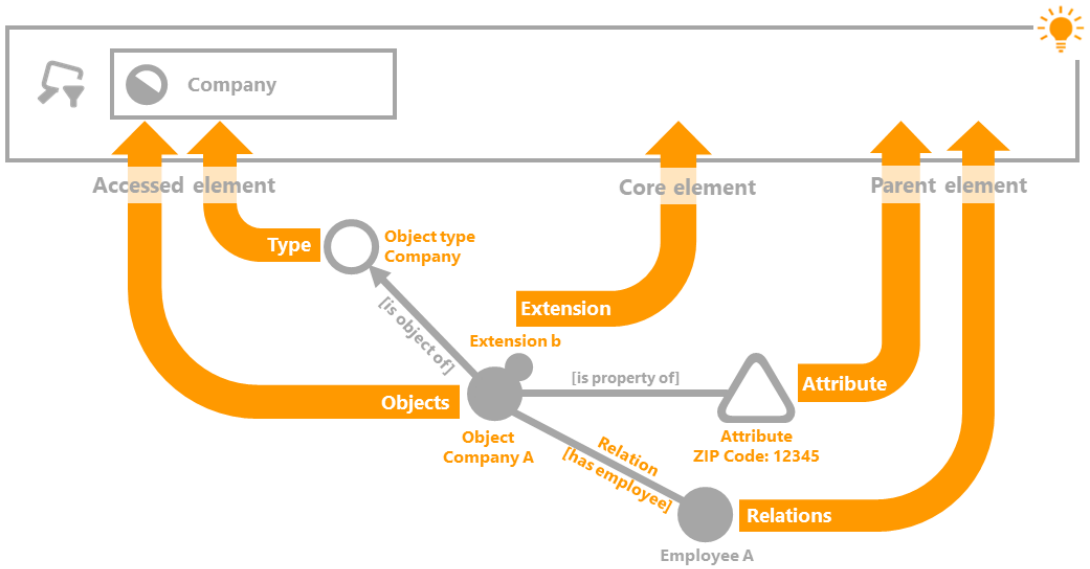
Schema modifizieren Die Operation Schema modifizieren, betrifft Änderungen am Definitionsbereich von Relationen und Änderungen an der Typenhierarchie (*ist Untertyp von* und *ist Obertyp von* Relationen).

1.2.5.1. Beispiel: Die Verwendung von Operationsgruppen im Rechtesystem

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:

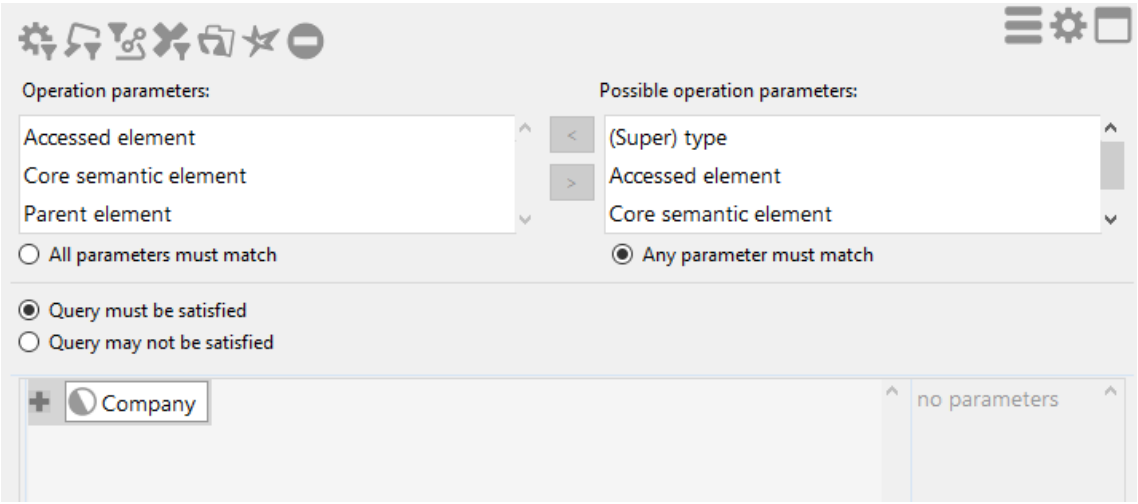
- Das Löschen des Objekttyps Company
- Das Löschen von bestimmten Unternehmens (Objekte von Company)
- Das Löschen von Attributen, welches an einem Unternehmen vorkommt
- Das Löschen von Relationen, die an einem Unternehmen vorkommt (Relationsziel und -quelle)
- Das Löschen von Erweiterungen, die Objekte von Company erweitern
- Das Löschen von Attribut- und Relationstypen, die Objekte oder Untertypen von Company 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 "Company", bei dem die Einstellung Objekte und Untertypen ausgewählt ist. Der Operationsparameter Zugriffselement deckt den Objekttyp "Company" und alle Objekte, die zu diesem Typ gehören, ab. Der Parameter Kernobjekt deckt die Erweiterungsobjekte ab, die zu Unternehmen 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.

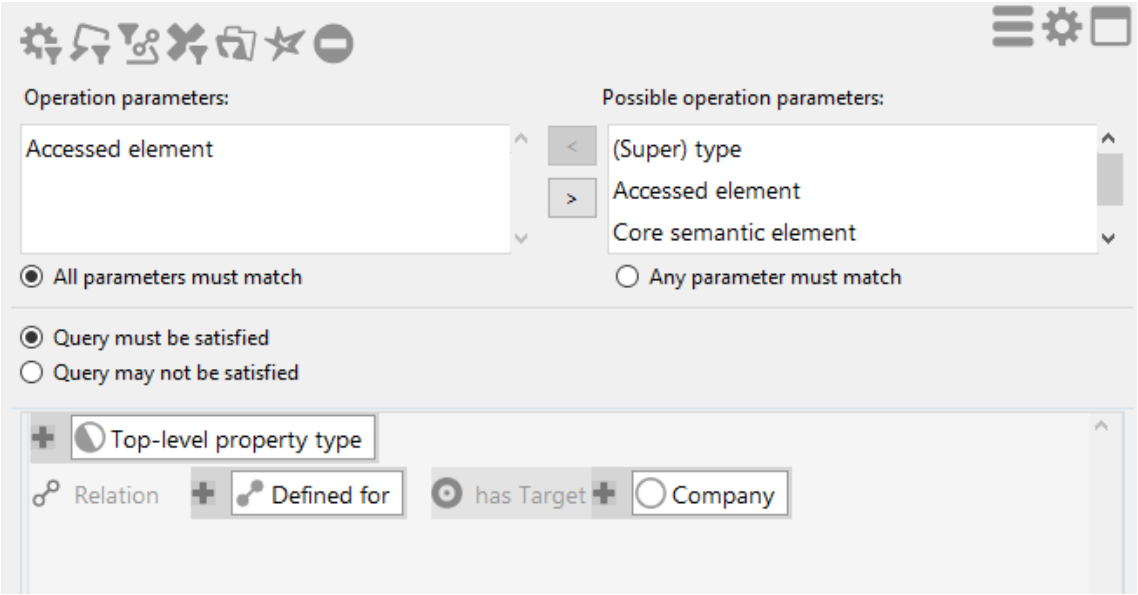


Im Beispiel verwendeter Suchfilter: Kernobjekt, übergeordnetes Element und Zugriffselement wurden als Operationsparameter ausgewählt. Die Einstellungen Ein Parameter muss zutreffen und Suchbedingung muss erfüllt sein werden verwendet.

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:

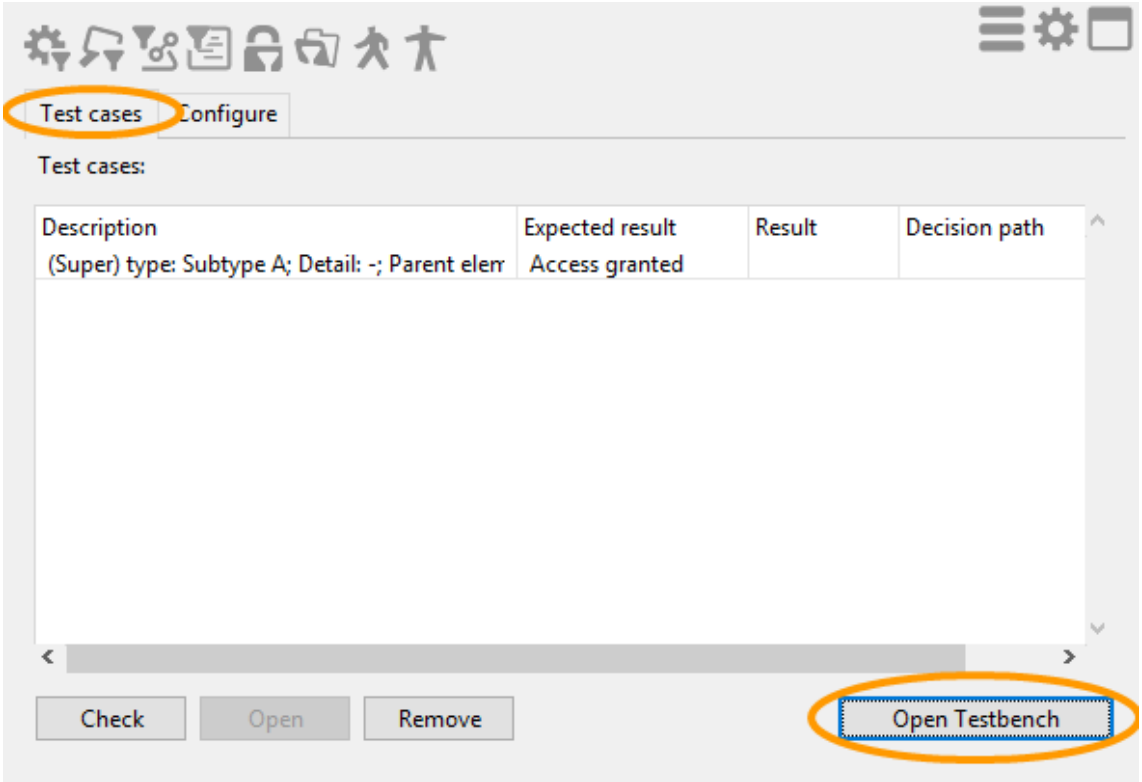


Der Suchfilter erfasst alle Eigenschaftstypen (Attribut- und Relationstypen) die für Objekte bzw. Untertypen von Company definiert sind. In der Suchfilterdefinition wird der Parameter Zugriffselement und die Einstellung Suchbedingung muss erfüllt sein verwendet.

1.2.6. Testumgebung

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.

1.2.6.1. Eine Zugriffssituation testen

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.

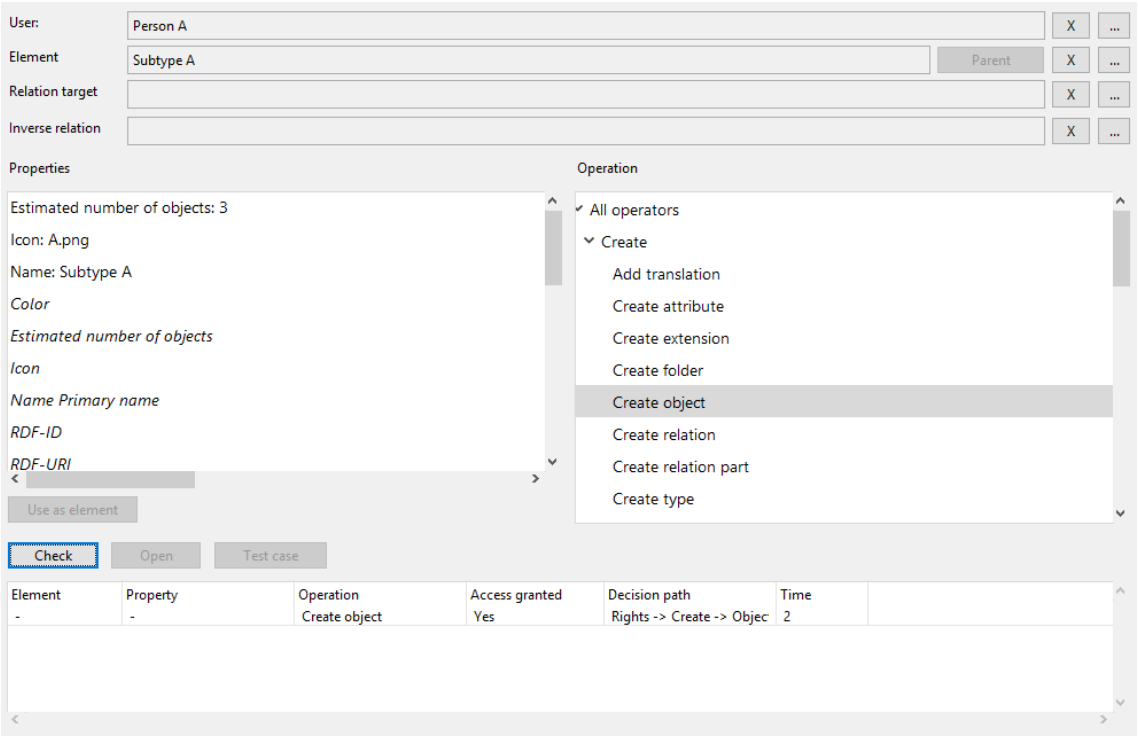
3. Klicken Sie *Testumgebung öffnen* (rechts unten) an, damit sich die Testumgebung in einem neuen Fenster öffnet.

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.

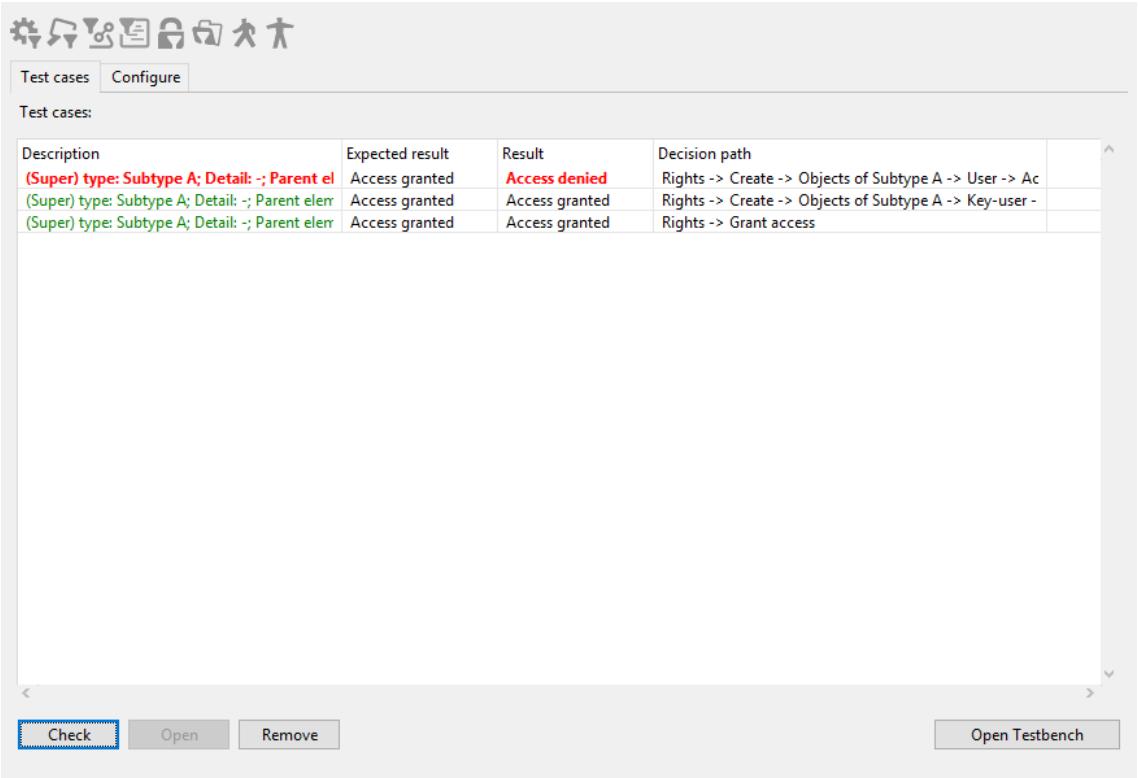
1.2.6.2. Testfälle definieren

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.

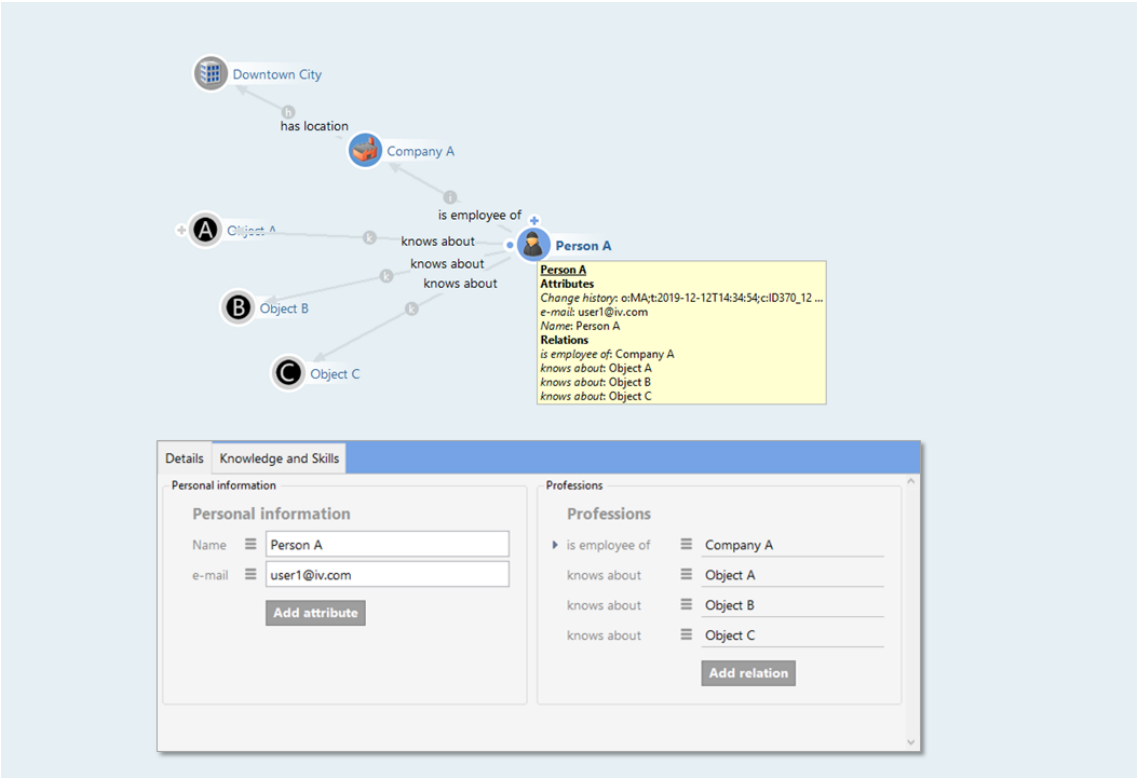
1.3. View-Konfiguration

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 Zusammenstellungen 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 Teil des Screenshot: Nicht konfigurierter Ausschnitt eines Objektes in der Graph-Ansicht mit allen seinen Eigenschaften. Unterer Teil des 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.

1.3.1. Konzept

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.

1.3.1.1. View-Konfiguration

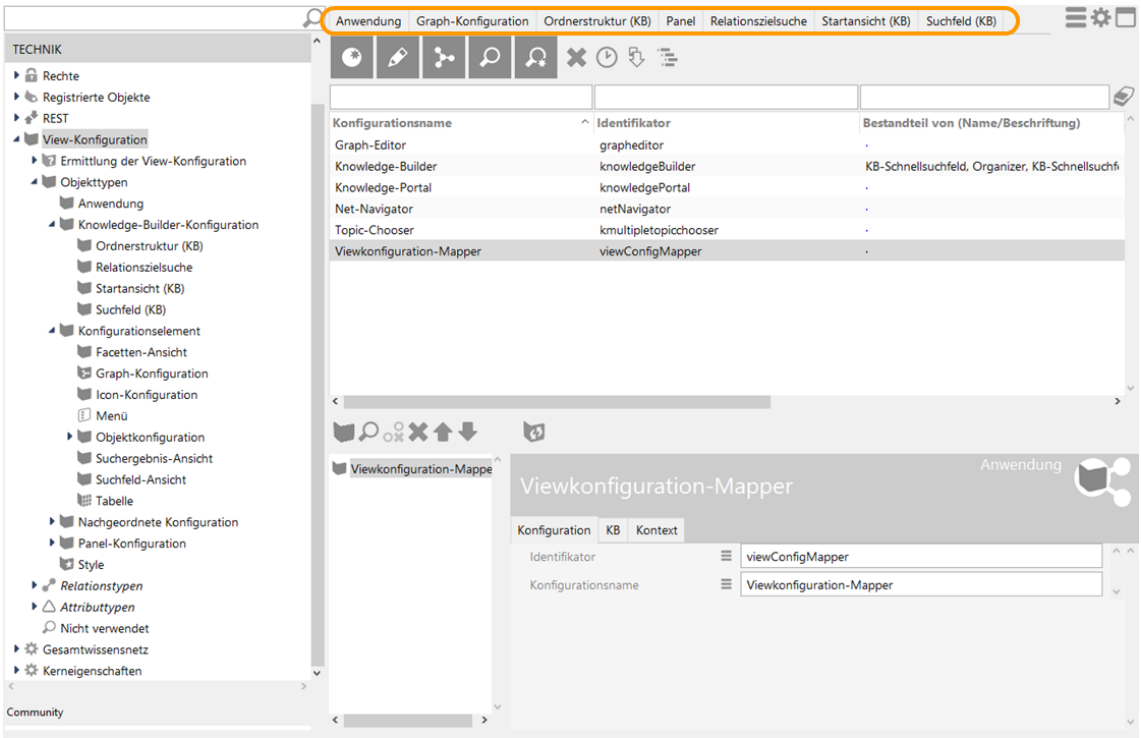
Die View-Konfiguration ist dazu vorgesehen, die Daten des Wissensnetzes für die Anwendungen so aufzubereiten, dass sie entweder im Knowledge-Builder oder mithilfe der Bridge in Form einer Anwendung im Web-Frontend dargestellt werden können.

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.

1.3.1.2. 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 Webseite 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.

1.3.1.3. Erstellung und Aktualisierung von View-Konfigurationen

Erstellung

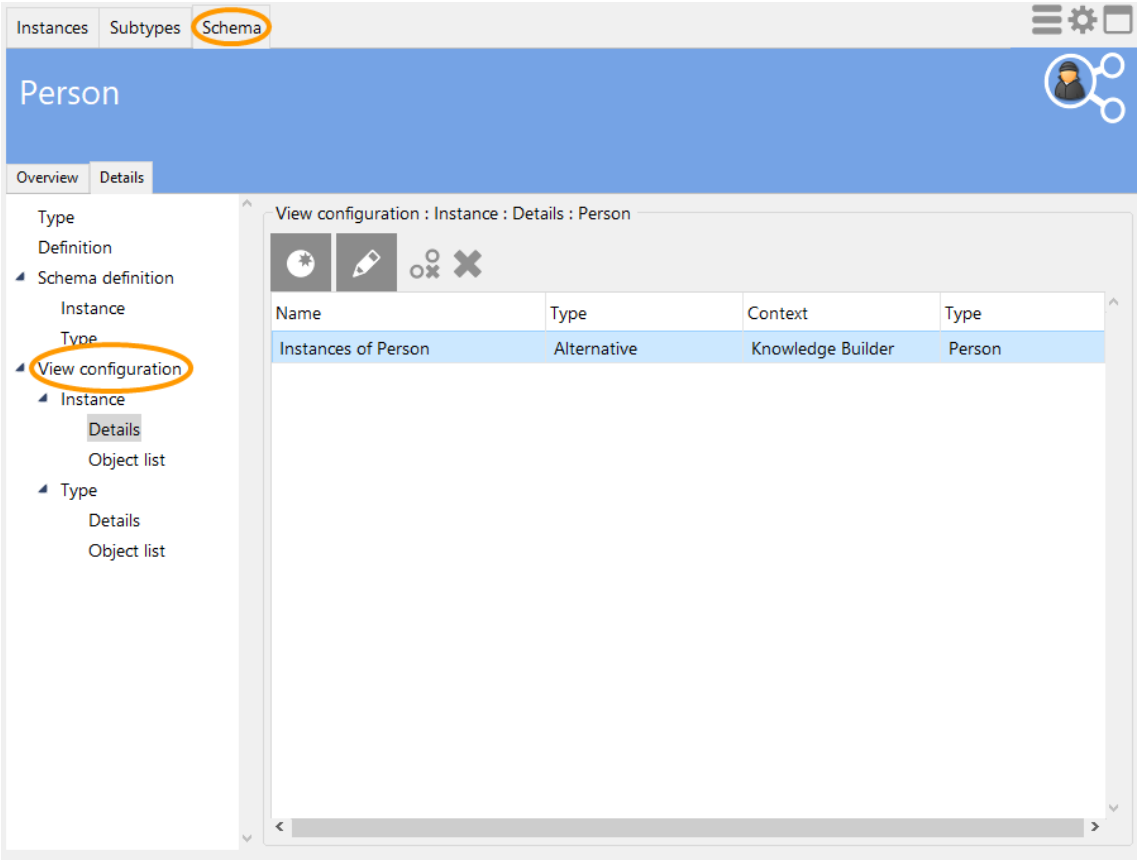
Im Knowledge-Builder gibt es zwei Stellen, an denen sich eine neue View-Konfiguration erstellen lässt:

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 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-orientierte 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.

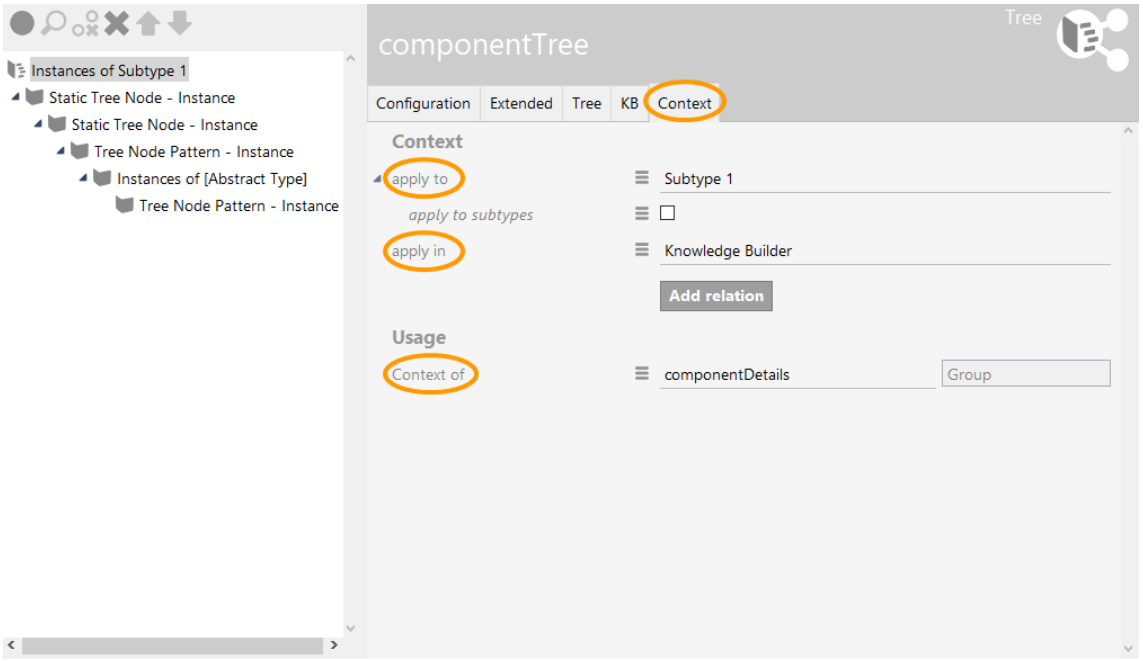
Für das Konfigurieren eines Web-Frontends ist die Panel-Konfiguration unter *TECHNIK > View-Konfiguration > Viewkonfiguration-Mapper* zu verwenden. Näheres hierzu unter Kapitel 3 "ViewConfig-Mapper".

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.

1.3.1.4. 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 eines Layouts ist, dessen 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.

1.3.1.5. Die Gültigkeit von View-Konfigurationen

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.

1.3.2. Menüs

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.

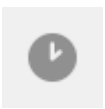
Im Folgenden werden die direkten Einstellungsmöglichkeiten an einem Menü und die bereits existierenden Menüarten und deren Verwendung beschrieben.

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. Die Standardaktionen sind nur für die View-Konfiguration des Knowledge-Builder anwendbar und umfassen die Aktionsarten der Objektlisten:

	Neu
	Anzeigen (Bearbeiten)
	Graphisch darstellen
	Suchen
	Löschen
	Zuletzt verwendete Objekte
	Aktualisieren
	Im Baum anzeigen
	Drucken

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.

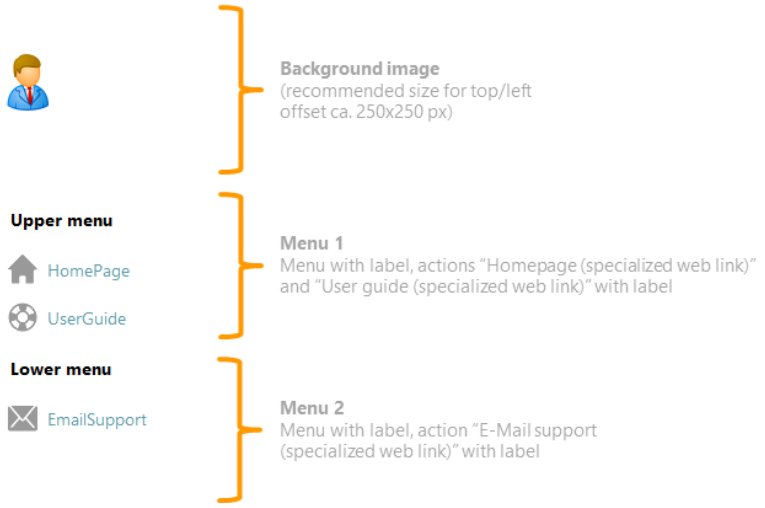
Kontextmenü

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 *$1* 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 Kontextmenü 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" : "Menu (List)",`
	`"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"`

1.3.3. Aktionen

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)

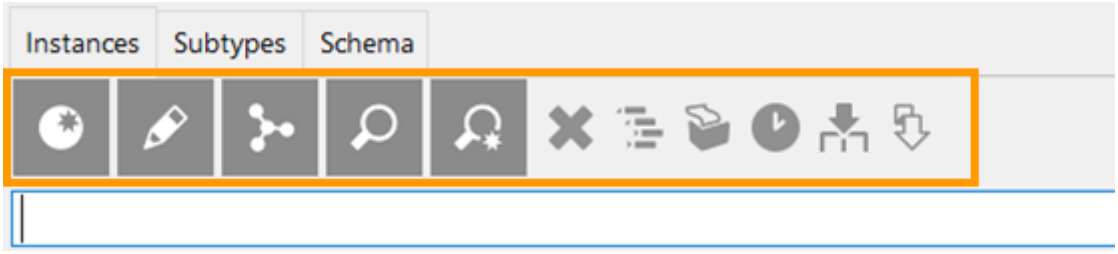
Je nach Aktionsart und Anwendungsfall sind zusätzliche Konfigurationen erforderlich, wie beispielsweise das Anlegen zusätzlicher Panels zur Anzeige der Ergebnisse einer Aktion.

1.3.3.1. Allgemein

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 Button-Größe 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.

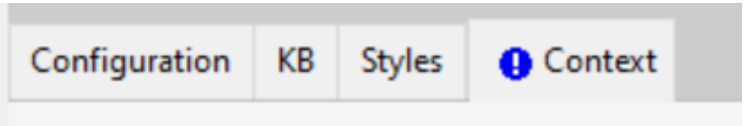
HINWEIS	Aktionen jeglicher Art lassen sich an verschiedensten Stellen anbringen. In den meisten Fällen werden diese auch angezeigt. Ob diese Aktion, in dem Kontext in dem sie gerade eingesetzt wird, ausführbar ist, ist nicht immer gegeben.

Einstellungsmöglichkeiten

Name	Wert
Konfiguration	
Konfigurationsname	Der Konfigurationsname dient zur Identifizierung und Wiederverwendung eines Konfigurationselements.
Beschriftung	Hier lässt sich eine Beschriftung für die Schaltfläche der Aktion festlegen.
Skript für Beschriftung	Die Beschriftung einer Schaltfläche wird über das Skript festgelegt. Diese Option ist nur dann verfügbar, wenn der Eintrag "Beschriftung" nicht belegt ist.
Bookmark path	Der Bookmark-Pfad kann hier ausgewählt oder neu angelegt werden. Der angezeigte Name dient zugleich als Pfadmuster (path pattern). Das Pfadmuster dient zur Pfadmuster-Erzeugung der Bookmarking-Ressource. Für mehr Informationen hierzu siehe Bookmarking und Historie.
Aktionsart	Die Art der Aktion. Die verschiedenen Arten werden weiter unten erklärt. Ein Skript überschreibt die durch die Aktionsart festgelegte Aktion. Bedingt durch die Auswahl des Aktionstyps sind nur bestimmte Skript-Arten verfügbar. **HINWEIS** Wenn der Aktionstyp gewechselt wird, dann werden unter Umständen die Skripte entfernt, welche aufgrund des Aktionstyps nicht mehr anwendbar sind. Wenn das Skript nicht registriert ist, so wird es gelöscht. Ein Dialog informiert in diesem Fall über die Konsequenzen, bevor die Löschung des Skriptes stattfindet.
Skript (benutzerdefiniert)	Das Skript, welches für die Aktion ausgeführt werden soll. Das Skript darf Elemente des Knowledge-Graphen verändern und bestimmt das Aktionsergebnis (ActionResult).Dieses Skript ist verfügbar, wenn einer der folgenden Aktionstypen ausgewählt wurde: • Auswahl • Relationsziel wählen • Skript
Skript (veraltet)	Das Skript das bei dieser Aktion ausgeführt werden soll. Veraltet — bitte "Skript (benutzerdefiniert)" verwenden
Skript (vor der Aktion)	Diese Aktion ist nur verfügbar, wenn der Aktionstyp "Speichern" gewählt wurde.

Name	Wert
Skript (ActionResponse) [VCM]	Ein hier angegebenes Skript führt eine sog. *ActionResponse* nach der Aktion aus. Dieses Skript darf nicht für Standard-VCM-Views verwendet werden. Nicht bei allen Aktionsarten verfügbar.
Skript (nach der Aktion)	Diese Aktion ist nur verfügbar, wenn der Aktionstyp "Speichern" gewählt wurde.
Skript (recall)	.
ausführen durch	Eine View-Rolle kann hier ausgewählt oder definiert werden.
Frage vor der Ausführung [VCM]	Nur für das Web-Frontend. 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. (Ok/Abbrechen/Schließen). 
Skript für Frage vor der Ausführung [VCM]	Der Text des Bestätigungs-Dialogs für die Aktion kann hier über ein Skript ermittelt werden. Wird eine leere Zeichenkette zurückgegeben, erscheint der Dialog nicht. **Achtung:** • Wenn eine leere Zeichenkette zurückgegeben wird, dann erscheint der Dialog nicht. • Wenn im Skript ein Fehler auftaucht, dann erscheint der Dialog auch nicht.
Transaktion	Diese Option wird nur für Editiervorgänge im Web-Frontend benötigt: Mithilfe der Transaktionsart **beginnen** kann ein temporärer Zustand (bzw. ein temporäres Element) erzeugt werden, bis eine andere Aktion die Transaktion mittels der Transaktionsart **beenden** bestätigt. **Beispiel:** Ein Objekt soll in einem Dialogfenster neu angelegt werden. Erst bei Betätigen eines Speichern-Buttons soll das Objekt tatsächlich im Knowledge-Graph erzeugt werden (Aktionsart "**Speichern**" mit Transaktionsart "**beenden**"). Ansonsten sollen bei Abbruch der Transaktion kein Objekt erzeugt oder Änderungen verworfen werden (Schließen des Dialogfensters mit Aktionsart " **Abbrechen**", ohne Angabe einer Transaktionsart).
Darstellung	
Skript (enabled)	Hier kann über ein Skript ermittelt werden, ob die Schaltfläche der Aktion aktiviert und damit ausführbar sein soll.

Name	Wert
Skript (visible)	Hier kann über ein Skript ermittelt werden, ob die Schaltfläche der Aktion angezeigt werden soll.
Icon	Hier lässt sich ein Icon 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. Der Tooltip erscheint, sobald man den Mauszeiger über der Schaltfläche ruhen lässt ("Mouse-over").
Skript für Tooltip	Hier kann über ein Skript der Inhalt des Tooltips der Aktion bestimmt werden, anstatt den Text der Beschriftung zu verwenden.
Nach der Ausführung (Aktion)	
Benachrichtigung [VCM]	Text der in einer Benachrichtigung angezeigt wird, die nach der Aktion eingeblendet wird.
Benachrichtigungstyp [VCM]	Als Metaeigenschaft der Benachrichtigung oder des Skripts für Benachrichtigung kann die Benachrichtigungsart gesetzt werden, um farblich unterschiedliche Hervorhebungen zu setzen: • "Erfolg" (grüne Benachrichtigung) • "Information" (blaue Benachrichtigung) • "Warnung" (gelbe Benachrichtigung) • "Fehler" (farblose Benachrichtigung)
Skript für Benachrichtigung	Der Inhalt der Benachrichtigung kann hier über ein Skript ermittelt werden.
Nach der Ausführung (Panels)	
Ergebnis anzeigen in Panel	Ein Panel in dem das Ergebnis der Aktion angezeigt werden soll.
Aktivierungsmodus [VCM]	Siehe Aktivierung von Panels
Skript für Aktivierung [VCM]	Hier kann mithilfe eines Skriptes bestimmt werden, unter welchen Umständen ein Viewconfig-Element aktiv (= sichtbar) werden soll.
Skript für Zielobjekt [VCM]	Bestimmt, welcher Kontext (welches semantische Element) nach Ausführung der Aktion an die nachfolgende View weitergegeben wird.
Panel schließen [VCM]	Kann nur auf Dialogpanel angewendet werden. Diese Option legt fest, ob das Panel nach der Aktion geschlossen werden soll.
KB	

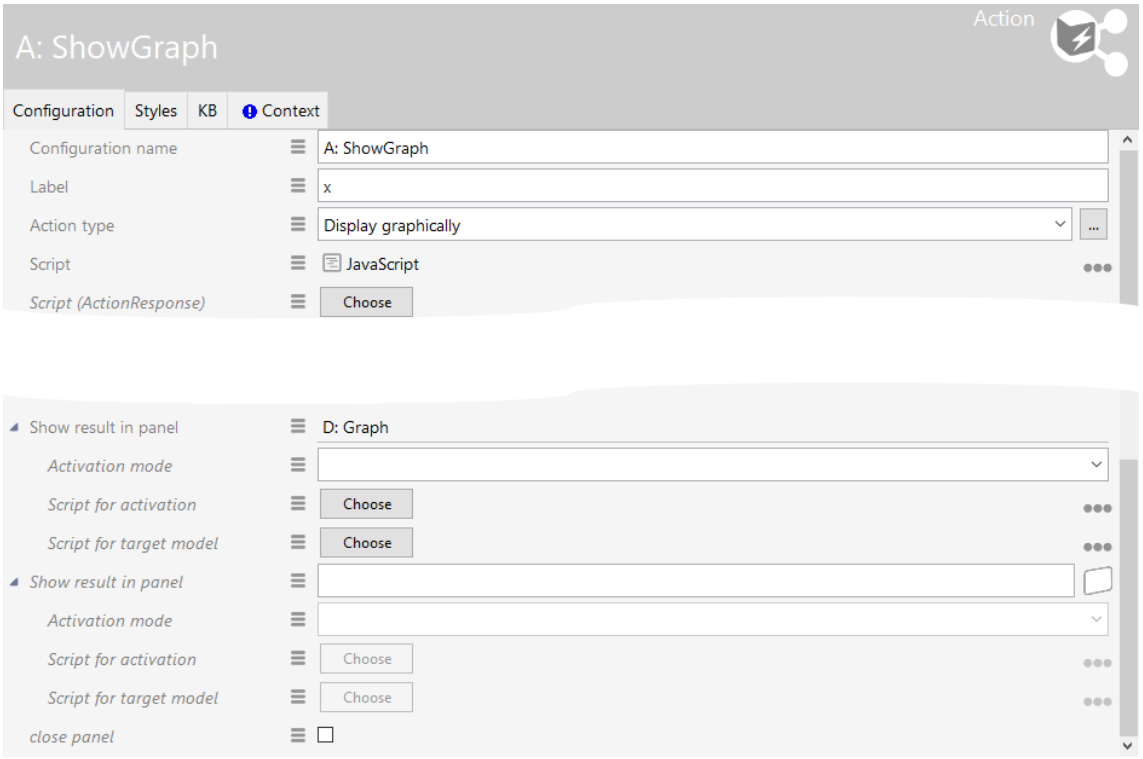
Name	Wert
Aktionsart	Die Aktionsart, welche nur für die Verwendung im Knowledge-Builder verfügbar ist und nicht für das Web-Frontend.
Ursprüngliche Position verwenden	
Styles	
Styles können auf unterschiedliche Arten angewandt werden, um das Erscheinungsbild oder das Verhalten des Buttons zu beeinflussen. Siehe entsprechendes Kapitel.	
Kontext	
Aktion von	Beschreibt, in welchem Menü die Aktion zur Zeit verwendet wird. Eine Aktion kann in mehreren Menüs (wieder)verwendet werden.
Reihenfolge	Beschreibt die Position der Aktion innerhalb des übergeordneten Menüs.
Hinweis	Weist darauf hin, ob die Aktion in mehr als einer Konfiguration verwendet wird. In diesem Fall erscheint ein blaues Symbol mit Ausrufezeichen am Kontext-Reiter: 

1.3.3.2. Universell anwendbare Aktionen

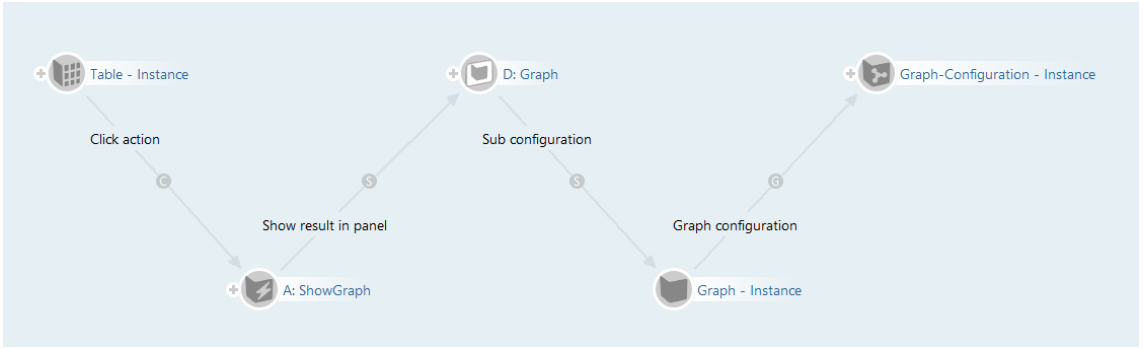
Universell anwendbare Aktionen können sowohl im Knowledge-Builder als auch per Viewconfiguration-Mapper im Web-Frontend angewendet werden. Hierzu zählen die Aktionsarten "Graphisch darstellen", "Löschen" und "Suchen".

1.3.3.2.1. 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:

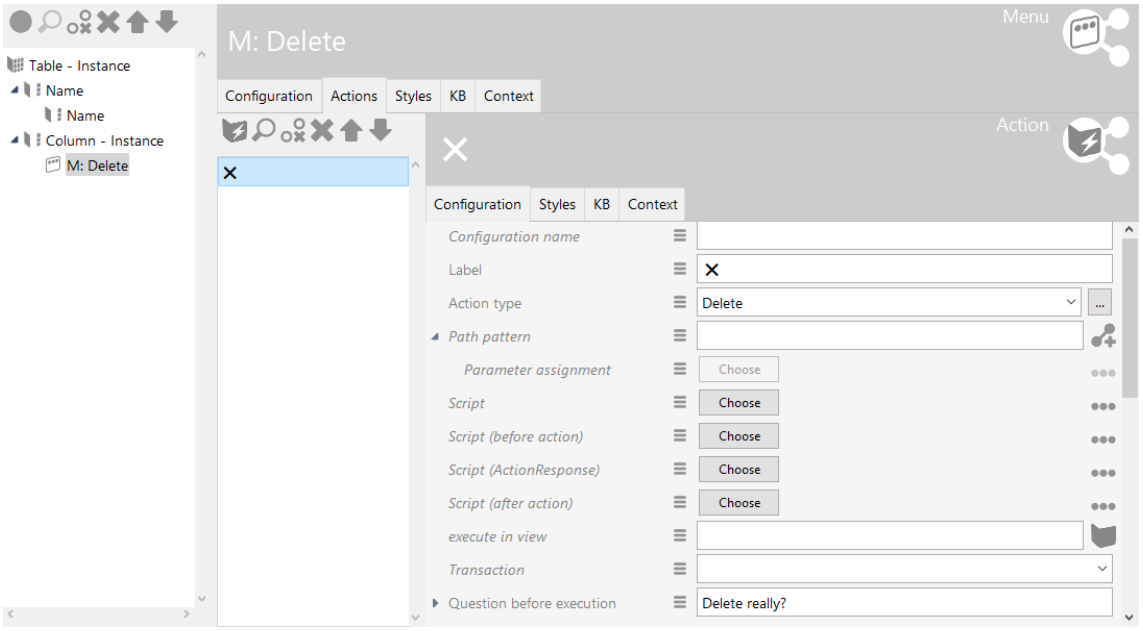


Hierfür muss unter "Ergebnis anzeigen in Panel" ein Panel angegeben werden, das als Sub-Konfiguration ein Graph-Objekt enthält. Das Graph-Objekt wiederum muss für die Definition der darzustellenden Elemente eine Graph-Konfiguration enthalten:

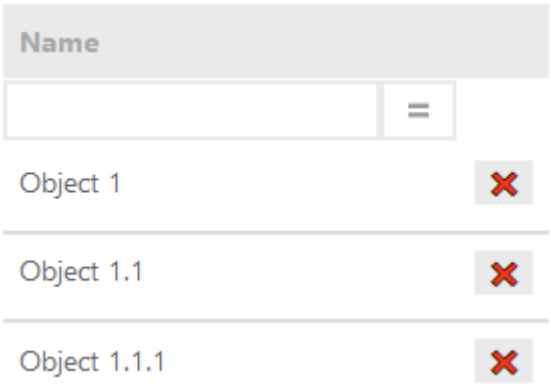


1.3.3.2.2. Aktionsart "Löschen"

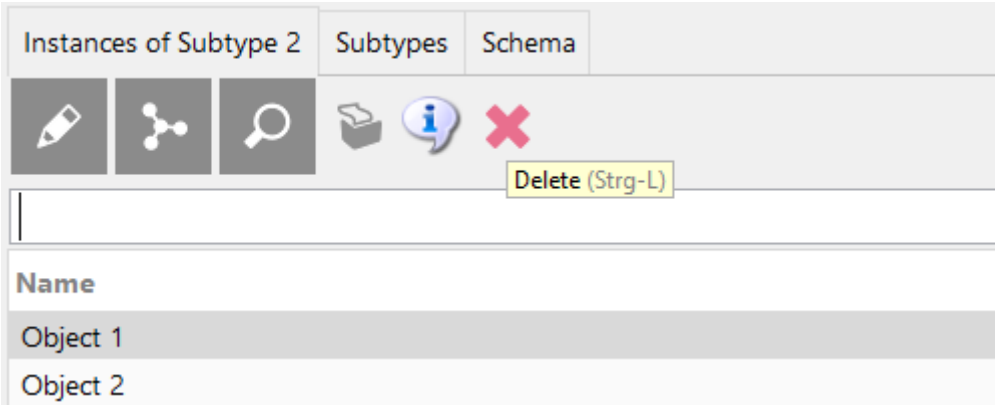
Diese Aktionsart löscht das jeweilige Element.



Bezogen auf die Web-Frontend Konfiguration, bewirkt die Löschen-Aktion ein Löschen des Zugriffselements. Beispielsweise führt das Anlegen einer Löschen-Aktion eines Menüs innerhalb eines zweiten Tabellenspalten-Elements dazu, dass in jeder Reihe eine Schaltfläche angezeigt wird. Wenn auf die Schaltfläche geklickt wird, dann wird das Zugriffselement — in diesem Fall das Zeilenelement — gelöscht.



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

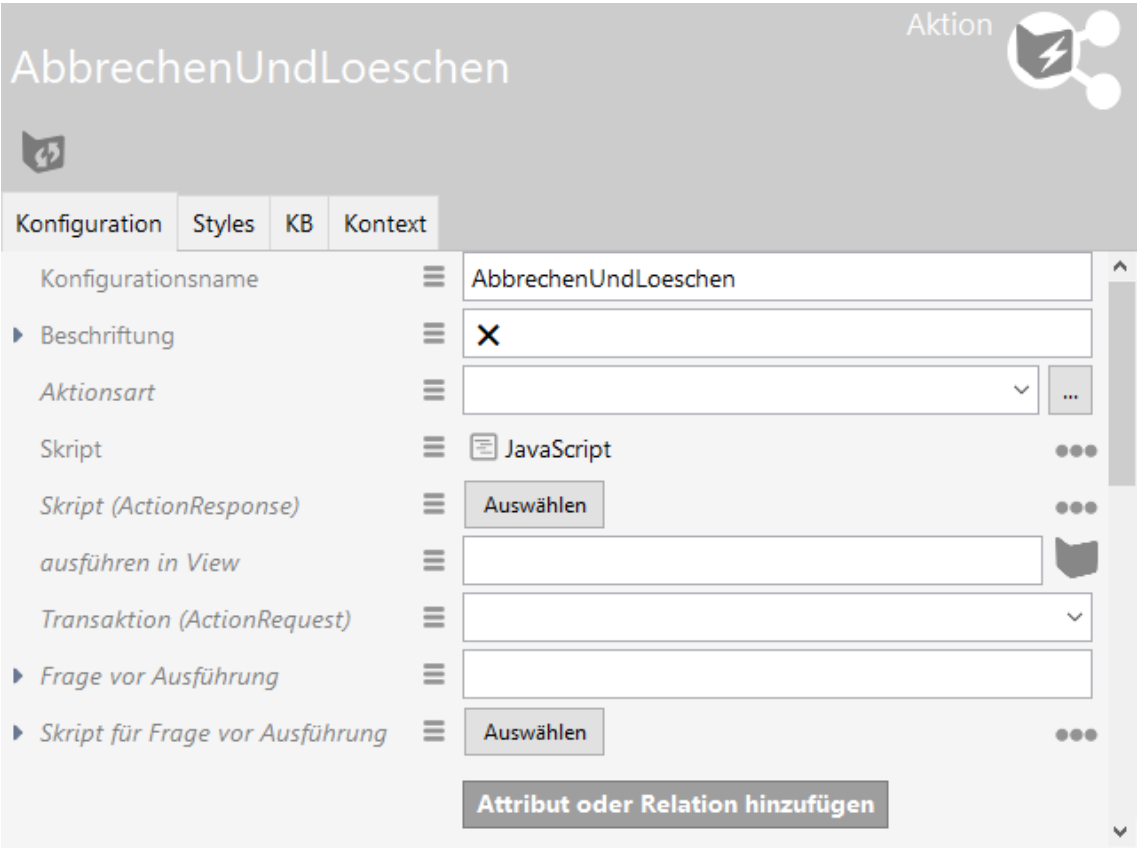


Wie jede Konfiguration im Knowledge-Builder kann auch die Standard-Konfiguration durch eine individualisierte Konfiguration ersetzt werden. In diesem Fall findet die Aktionsart "Löschen" ihre Anwendung.

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:



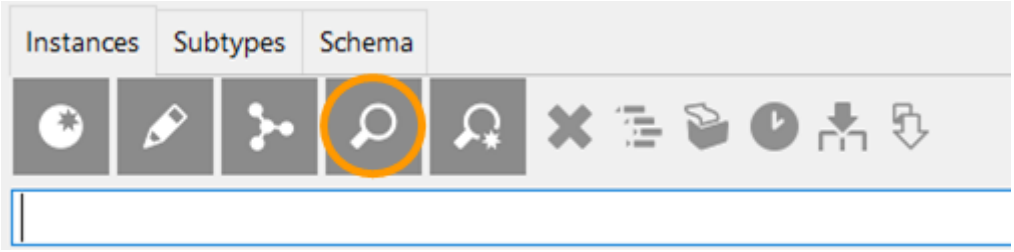
Eine Anwendungsmöglichkeit hierfür ist das Konfigurieren eines Dialog-Panels zum Anlegen neuer Objekte, dessen Abbrechen-Button das temporär erzeugte Objekt wieder löscht.

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

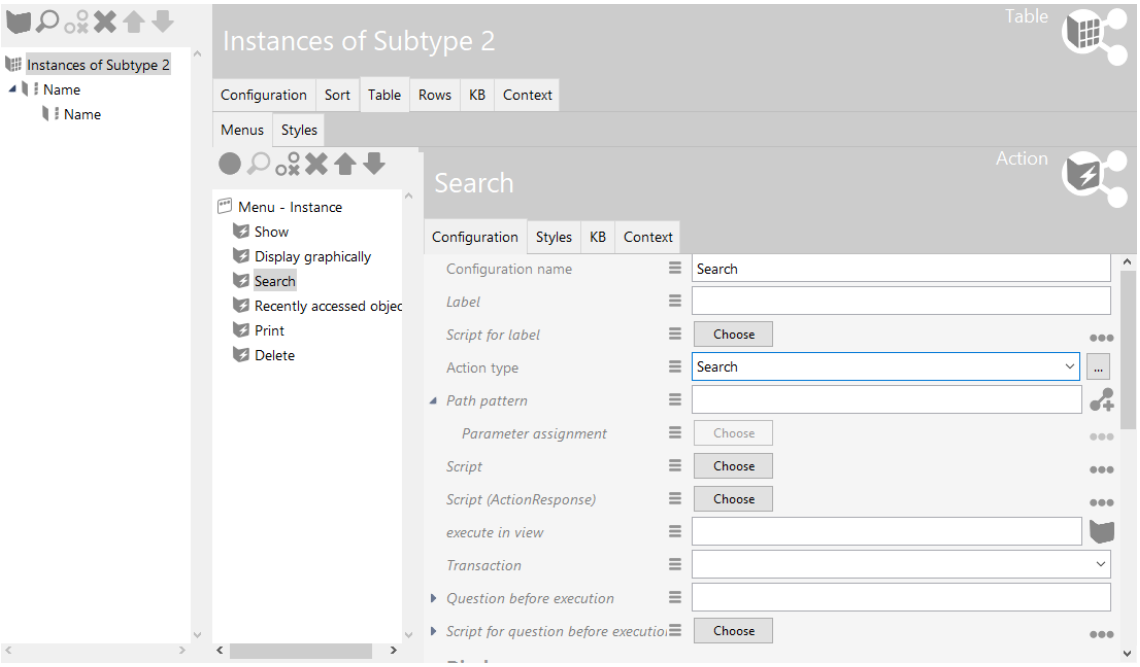
Die Syntax hierzu kann in der i-views spezifischen JavaScript-API Dokumentation nachgeschlagen werden.

1.3.3.2.3. 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.

- Darüber hinaus steht die View "Suche" zur Verfügung, welche bei Bedarf Sucheingabe und Suchergebnis in einem Element vereint. Sofern das Suchfeld nicht an abweichender Stelle stehen soll, emfpiehlt sich diese Ansicht.

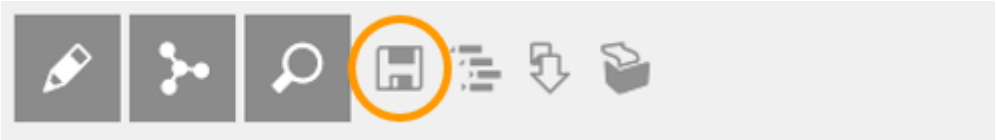
1.3.3.3. Aktionen für den Knowledge-Builder

Diese Aktionsarten können ausschließlich für Konfigurationen im Knowledge-Builder verwendet werden.

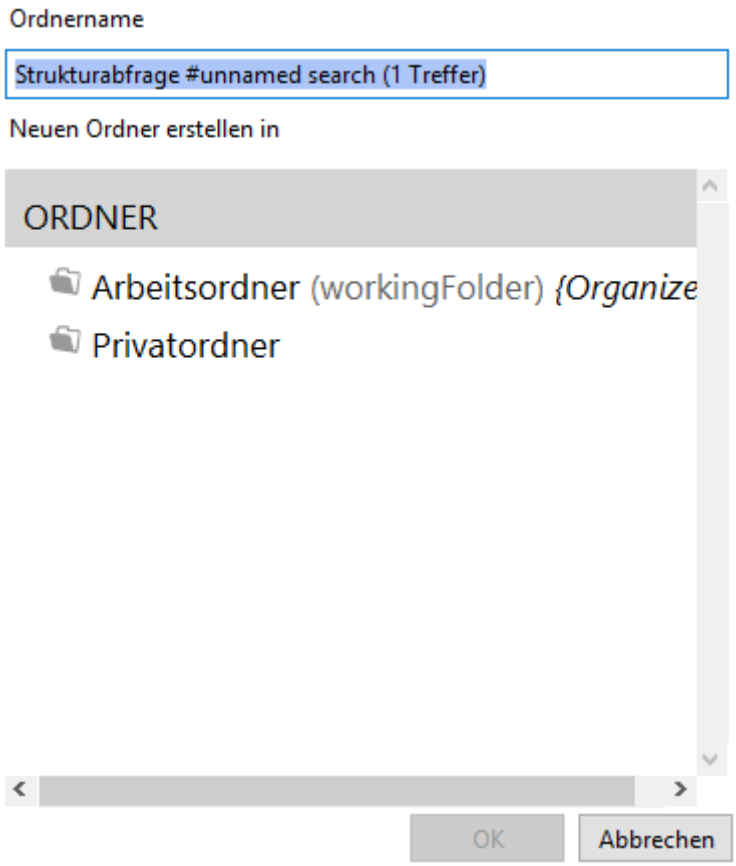
HINWEIS	Die KB-spezifischen Aktionsarten sind seit Version 5.2.2 nur im Reiter "KB" verfügbar. Da diese Aktionsarten bereits standardgemäß für Objektlisten und für die Startseite des Knowledge-Builders verwendet werden, dienen diese Aktionsarten hauptsächlich zur Konfiguration von Menüs mit einer reduzierten Auswahl an Aktionsarten oder zur Vervollständigung individueller Aktionen um die Standard-Aktionsarten.

1.3.3.3.1. 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:

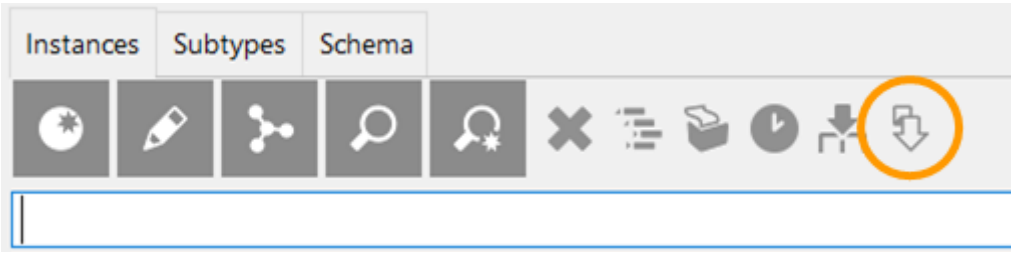


HINWEIS

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.

1.3.3.3.2. Aktionsart "Ansicht aktualisieren"

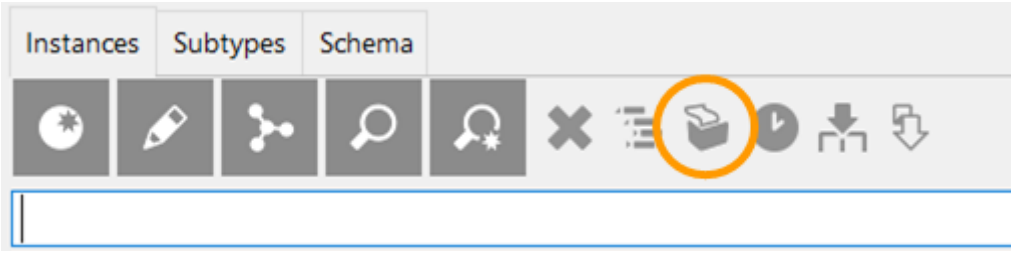
Im KB wird mit dieser Aktion der sichtbare Inhalt von Tabellenzellen neu berechnet. Verfügbar ist diese Option in der Menüleiste von Objektlisten unter dem Button "Aktualisieren" (Shortcut F5).



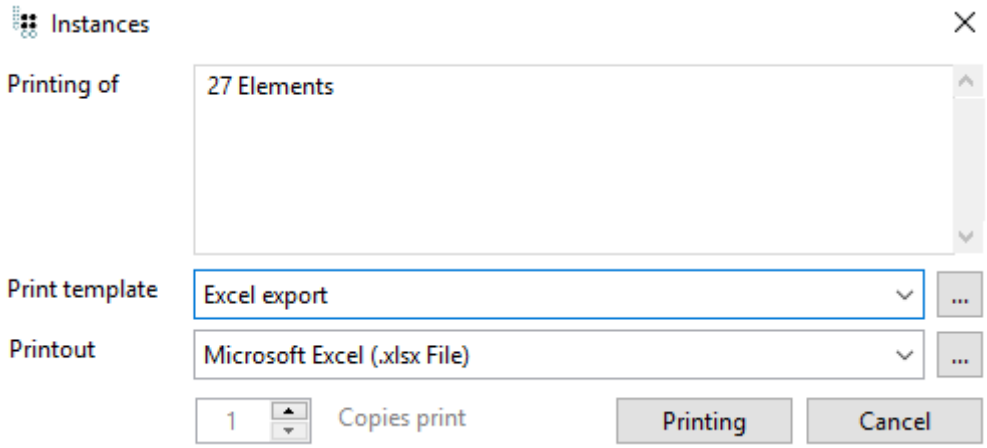
1.3.3.3.3. Aktionsart "Drucken"

Diese Aktion findet in der Menüleiste von Listenansichten Verwendung. Mit der voreingestellten

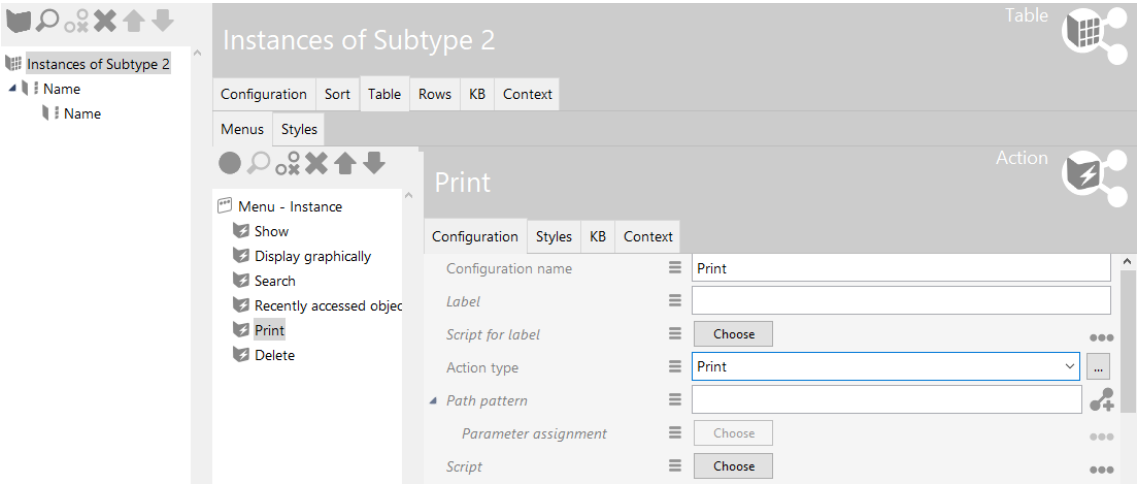
Konfiguration können Objektlisten ausgedruckt oder in Form einer Excel-Tabelle ausgegeben werden, ohne dass dafür ein Export-Mapping angelegt werden muss.



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:



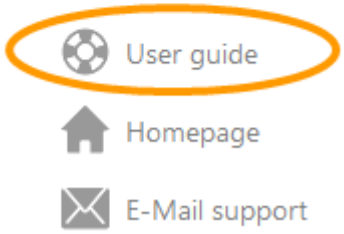
Voraussetzung für die Anwendbarkeit der Aktionsart "Drucken" ist das Vorhandensein der Drucken-Komponente, welche bei Bedarf mithilfe des Admin-Tools nachinstalliert werden kann.

Die Konfiguration der Druckkomponente ist verfügbar im TECHNIK-Teil des Knowledge-Graphen.

Dort können Druckvorlagen mithilfe von Dokumentvorlagen bereitgestellt werden. Für mehr Informationen hierzu siehe Berichte und Drucken

1.3.3.3.4. Aktionsart "Handbuch"

Die Aktionsart "Handbuch" stellt eine vordefinierte Aktion zur Verfügung, welche das i-views Handbuch in einem Browser öffnet.



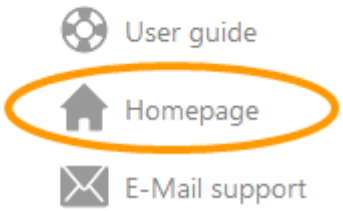
Im Gegensatz zur Aktionsart "Web-Link" ist die Aktionsart "Handbuch", wie die Aktionsart "Homepage", eine Verlinkung zu einer vorkonfigurierten Adresse.

Einstellungsmöglichkeiten

Name	Wert
URL	Voreingestellter Weblink zum i-views Handbuch.

1.3.3.3.5. Aktionsart "Homepage"

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

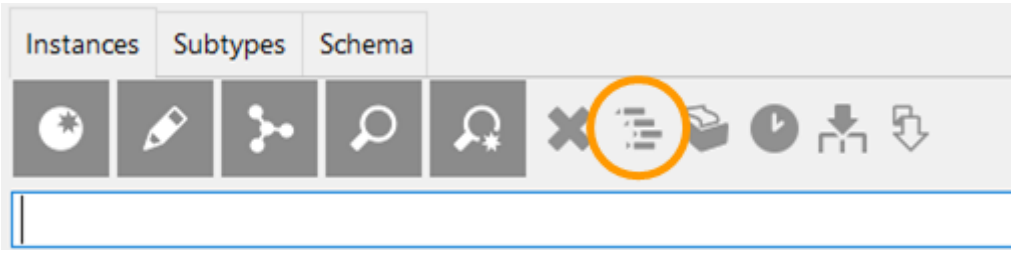


Einstellungsmöglichkeiten

Name	Wert
URL	Link zu einer Webseite

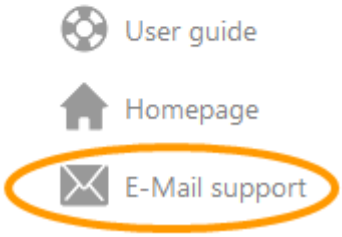
1.3.3.3.6. Aktionsart "Im Baum anzeigen"

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.



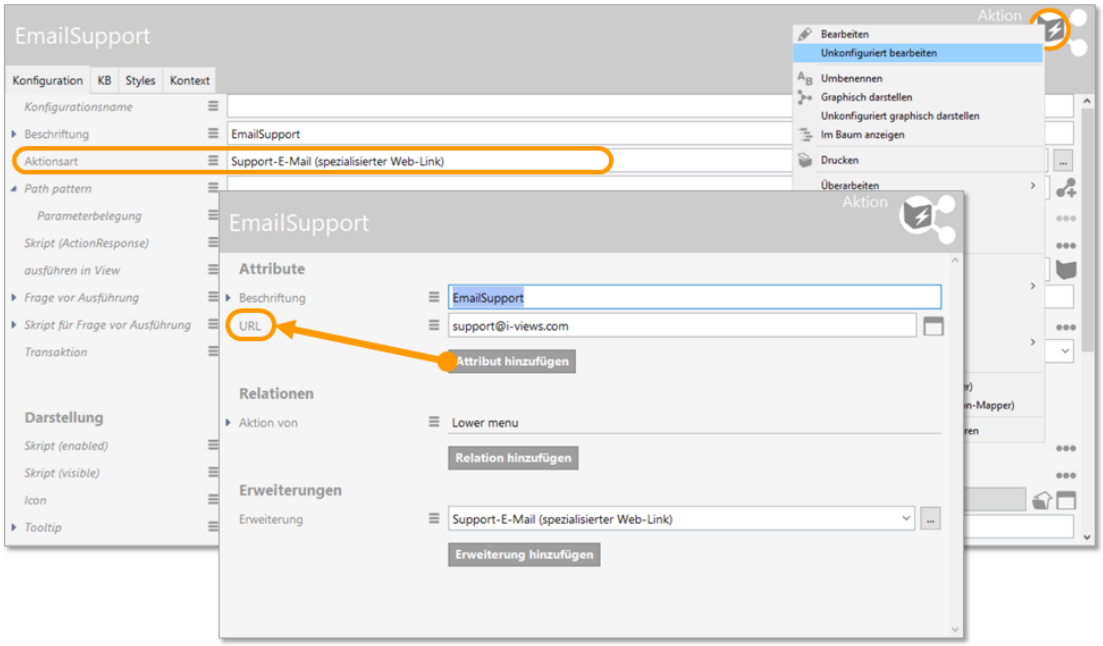
1.3.3.3.7. Aktionsart "Support-Email"

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	Value
URL	E-mail Link

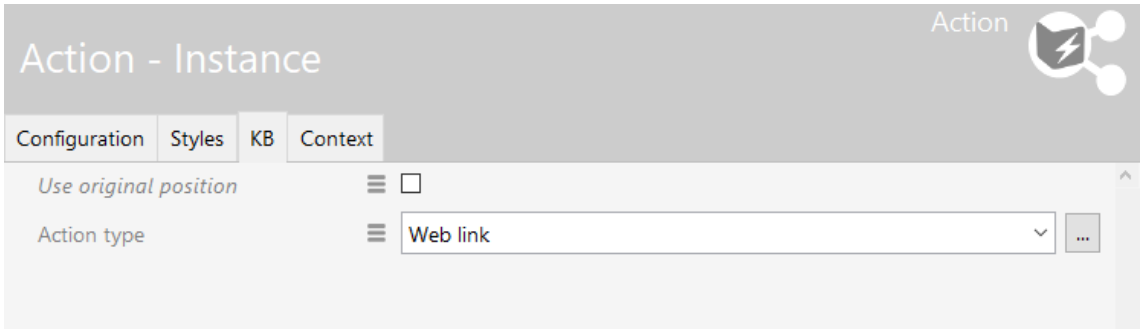


1.3.3.3.8. Aktionsart "Web-Link"

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.

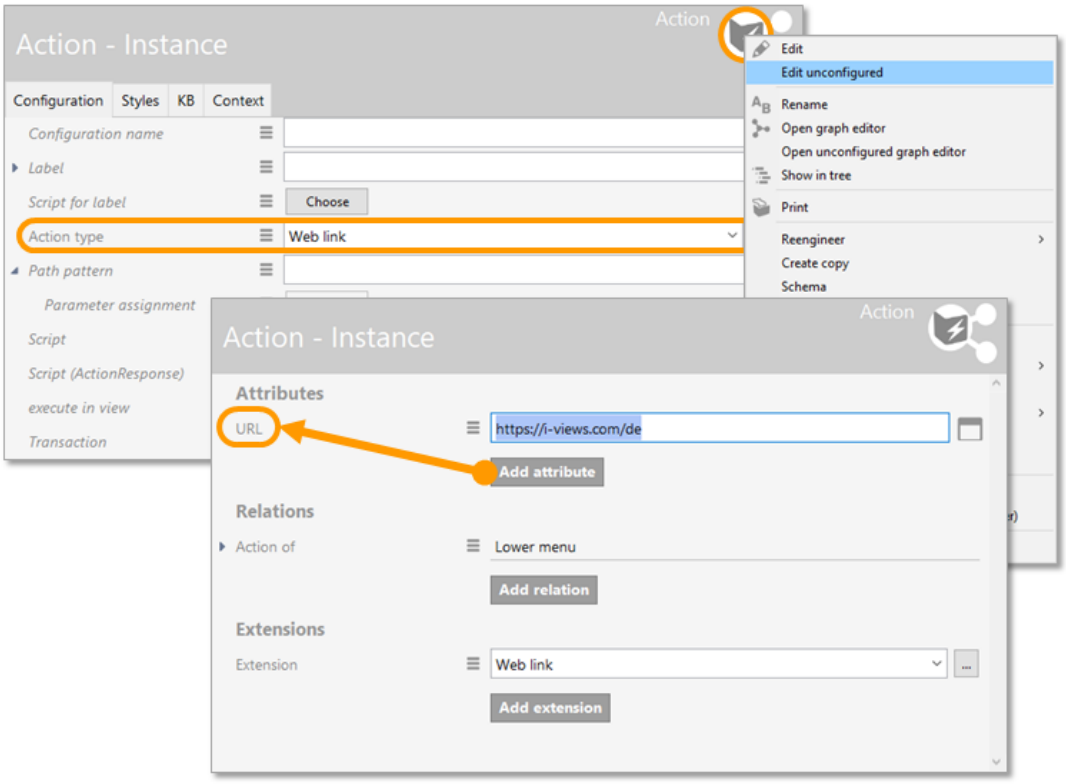
HINWEIS	In neueren KB-Versionen (ab KB 5.2.2) ist die Aktionsart "Web link" nur im Reiter "KB" verfügbar — siehe folgende Abbildung.



Setting options

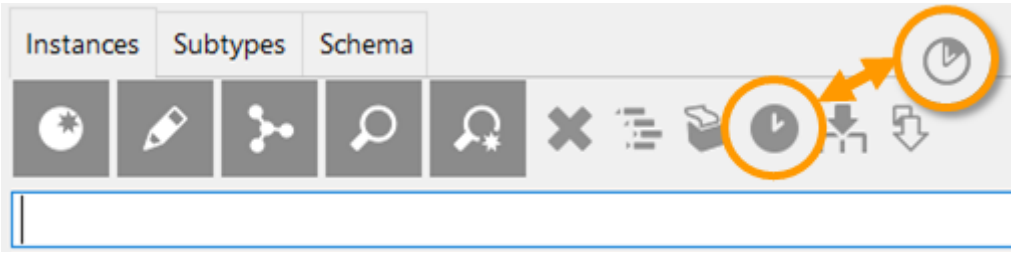
Name	Value
URL	Addresse des the Web-Links.

HINWEIS	Falls das URL-Attribut nicht angezeigt wird, kann dieses durch Öffnen der Aktion in der unkonfigurierten Ansicht hinzugefügt und bearbeitet werden:



1.3.3.3.9. Aktionsart "Zuletzt verwendete Objekte"

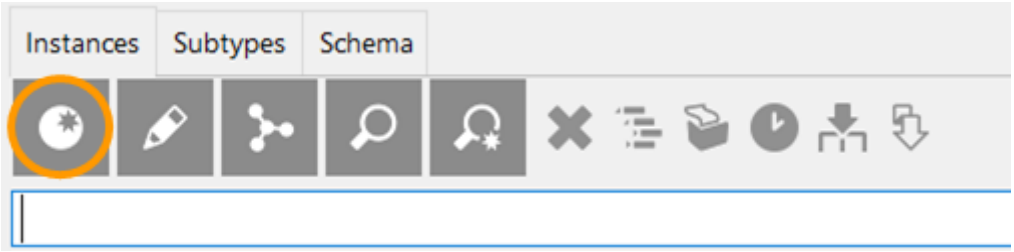
Zeigt die zuletzt verwendeten Objekt (Wissensnetzelemente) in der jeweiligen Tabelle an. Je nach Definition der Tabelle werden die Objekte ggf. gefiltert.



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

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

1.3.3.4. 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.

1.3.3.4.1. Aktionsart "Abbrechen"

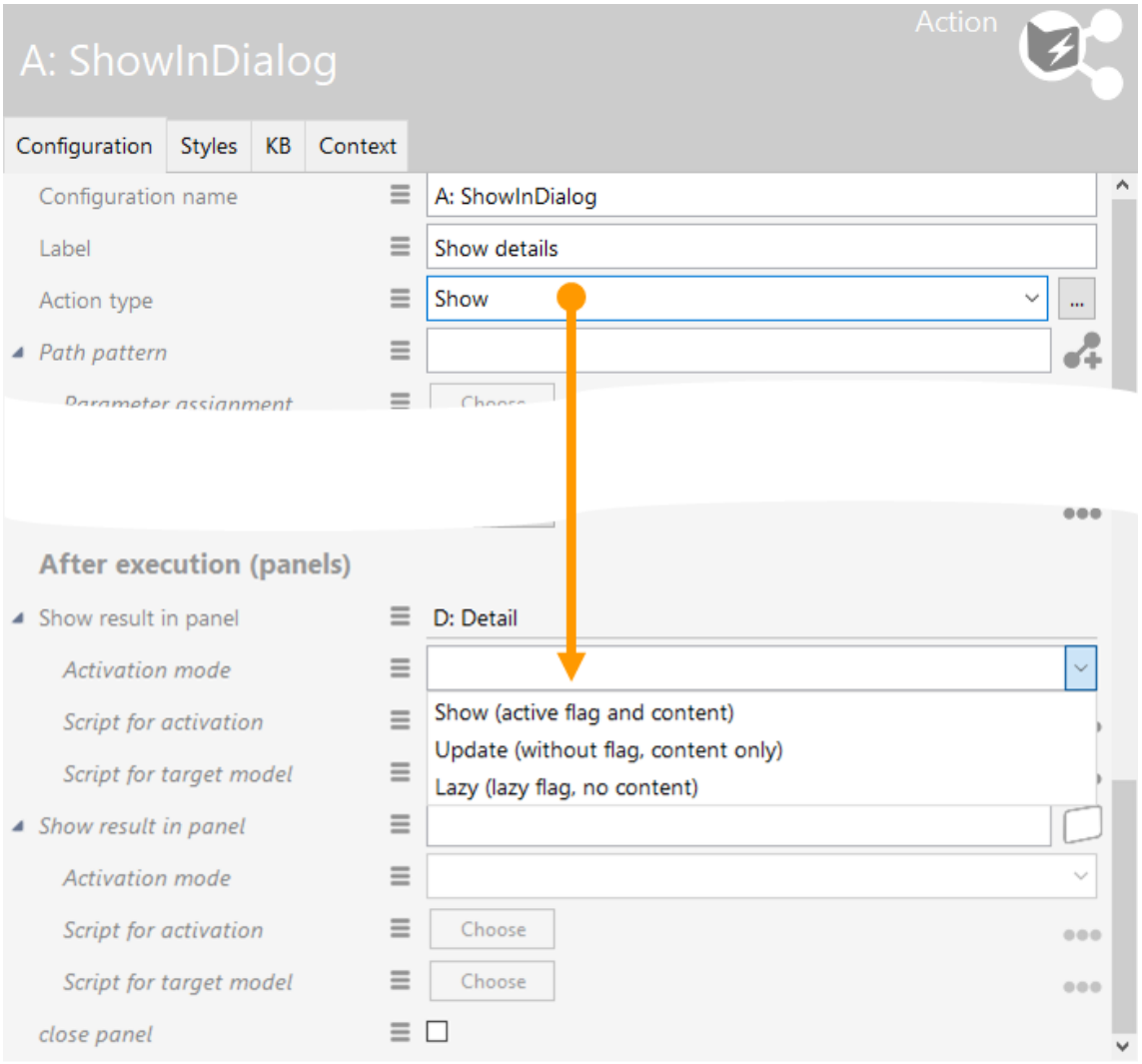
Die Aktionsart "Abbrechen" wird im Web-Frontend dazu verwendet, um eine begonnene Transaktion abzubrechen.

Beispiel: Durch eine Menü-Aktion mit der Option "Transaktion: beginnen" wird innerhalb einer Transaktion ein neues Objekt temporär erzeugt und in einem Dialogfenster angezeigt. Während anschließend eine Aktion mit der Option "Transktion: beenden" die Transaktion vollendet (oftmals in Kombination mit der Aktionsart "Speichern") und das Objekt persistiert, bricht eine Aktion der Aktionsart "Abbrechen" die Transaktion ab und das temporäre Objekt wird verworfen.

1.3.3.4.2. 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:

Aktivierungsmodus	Beschreibung
Standard	Das Zielpanel wird nach Ausführung der Aktion aktiviert (= sichtbar gemacht), unabhängig davon ob es vor der Aktion aktiviert war oder nicht. Dabei können durch die Verknüpfung von Panels durch "beeinflusst"-Relationen auch andere Panels aktiviert und mit Inhalt versorgt werden. Das Aktionsergebnis wird zum neuen angezeigten Modell des Panels und durch Beeinflussung eventuell auch von anderen Panels. Dieser Modus ist beispielweise beim Aufrufen eines Dialog-Panels sinnvoll.Wenn kein Aktivierungsmodus konfiguriert ist, wird dieser Modus als Standard verwendet.

Aktivierungsmodus	Beschreibung
Nur Ansicht aktualisieren	Das Zielpanel wird nur dann aktualisiert, wenn es vor Ausführung der Aktion bereits sichtbar war. Zudem werden keine weiteren Panels durch "beeinflusst"-Relationen aktiviert. Dabei hat das Aktionsergebnis keinen Einfluss auf den Inhalt des Panels: es wird weiterhin dasselbe Modell angezeigt, die Anzeige kann sich aber durch Seiteneffekte der Aktion verändern (zum Beispiel, weil eine Suche nun zu mehr Ergebnissen führt oder ein angezeigtes Objekt neue Eigenschaften erhalten hat).
Modell und Ansicht aktualisieren	Das Zielpanel wird nur dann aktualisiert, wenn es vor Ausführung der Aktion bereits sichtbar war. Zudem werden keine weiteren Panels durch "beeinflusst"-Relationen aktiviert. Das Aktionsergebnis wird dabei zum neuen angezeigten Modell des Panels und ersetzt das bisherige Modell.

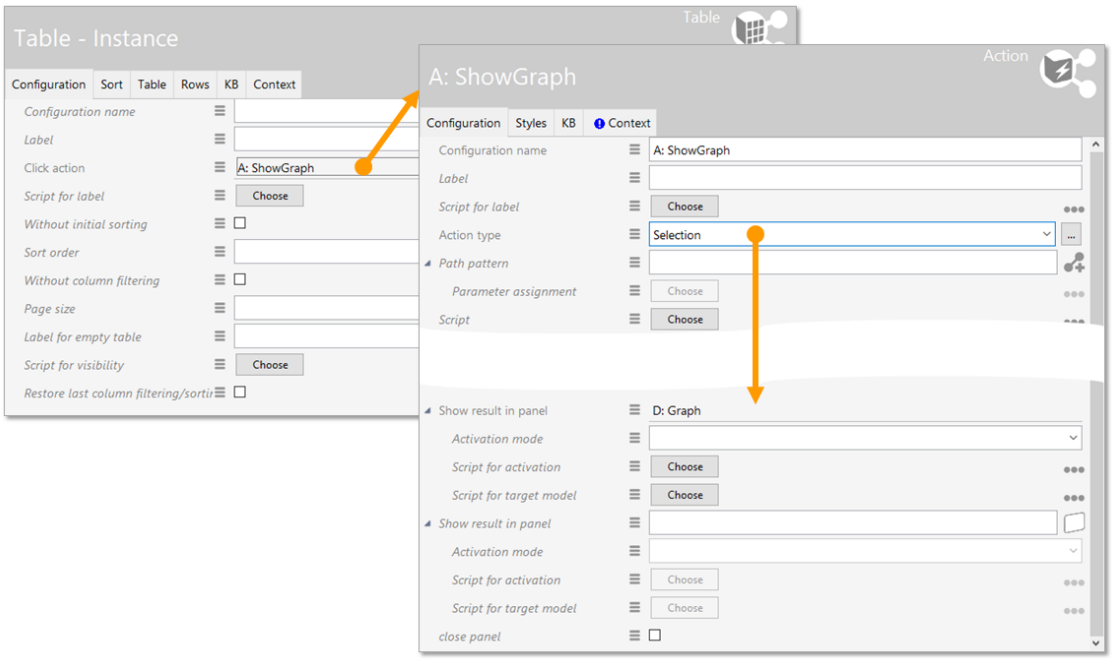
1.3.3.4.3. Aktionsart "Auswahl"

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 bei Verwendung eines Skriptes für die Aktion.

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

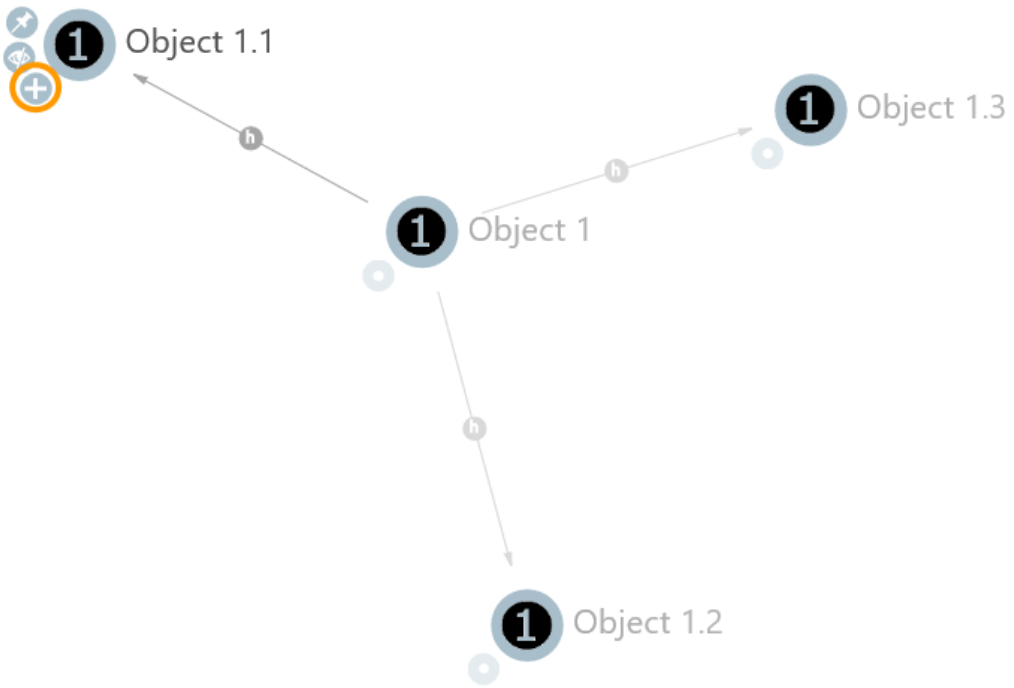


Zu beachten ist, dass in der jeweiligen "Auswahl"-Aktion selbst angegeben ist, auf welches Panel sich die Aktion auswirken soll. Dies wird unter "Ergebnis anzeigen in Panel" angegeben.

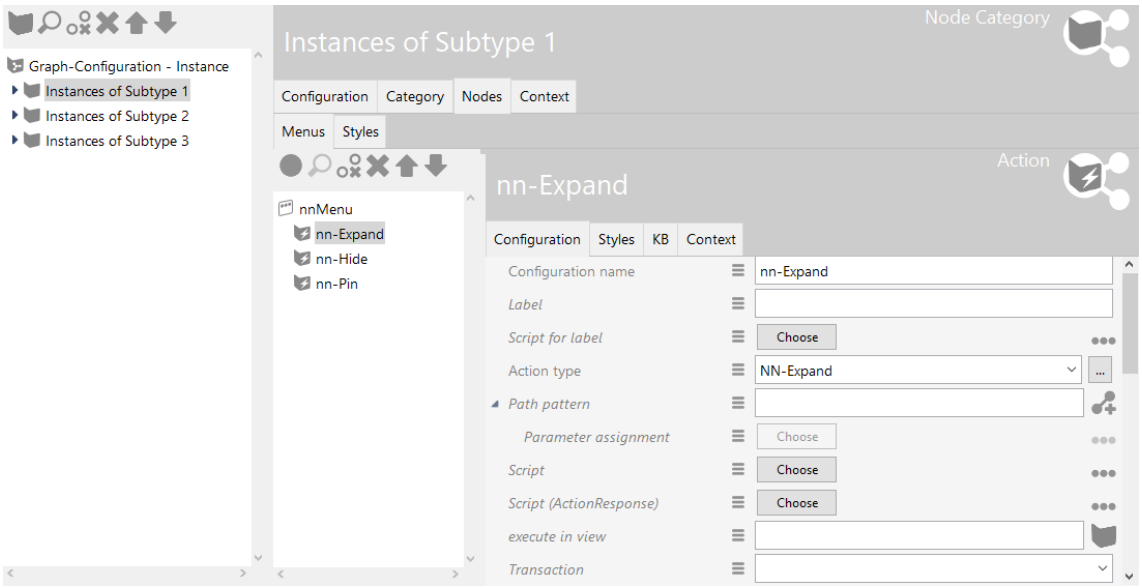
1.3.3.4.4. Aktionsart "NN-Expand"

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. Wenn sich Knoten bereits im Net-Navigator befinden und neue Objekte hinzugefügt werden, dann werden die dazwischen befindlichen Relationen automatisch mit eingeblendet.

Die Darstellung mit einem Plus-Symbol wie im Bild unten ist voreingestellt. Ebenfalls vorkonfiguriert ist das Dialog-Fenster, dass sich nach Klick auf den Plus-Button öffnet, wenn mehrere Relationen zur Auswahl zur Verfügung stehen. In diesem Dialog kann eine Auswahl getroffen werden, welche Knoten angezeigt werden sollen.



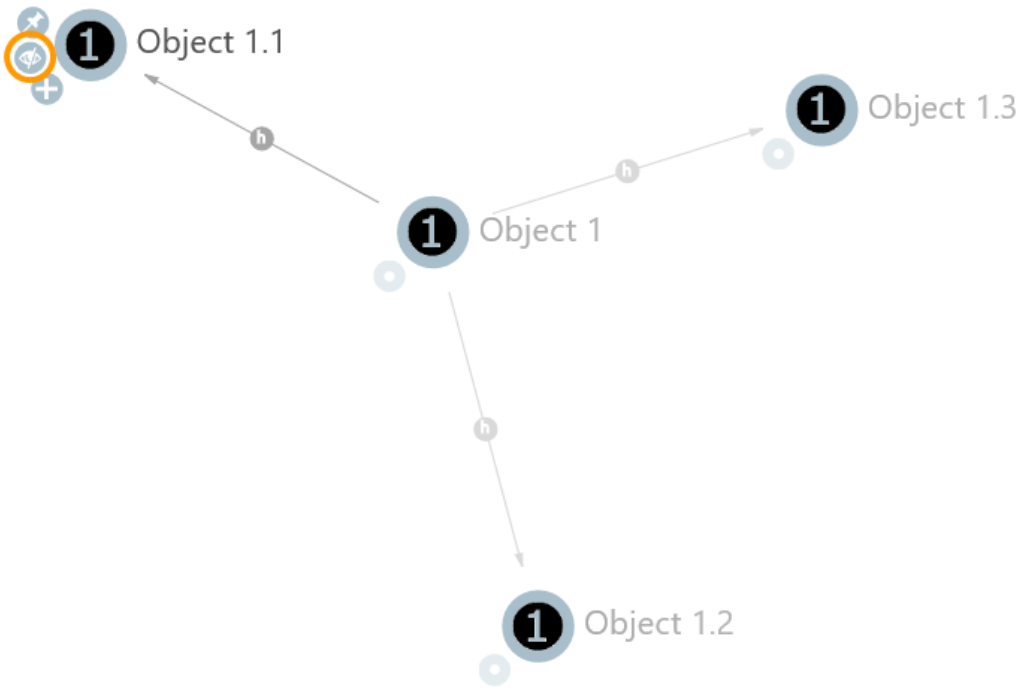
Die Aktion wird in der Graph-Konfiguration an allen Knotenkategorien angebracht, die sie besitzen sollen. Im Reiter "Knoten" wird ein Menü erstellt, das alle NN-Aktionen enthalten kann. In der Aktion selbst muss nur die Aktionsart "NN-Expand" ausgewählt werden, andere Angaben sind optional. Weitere Aktionsarten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



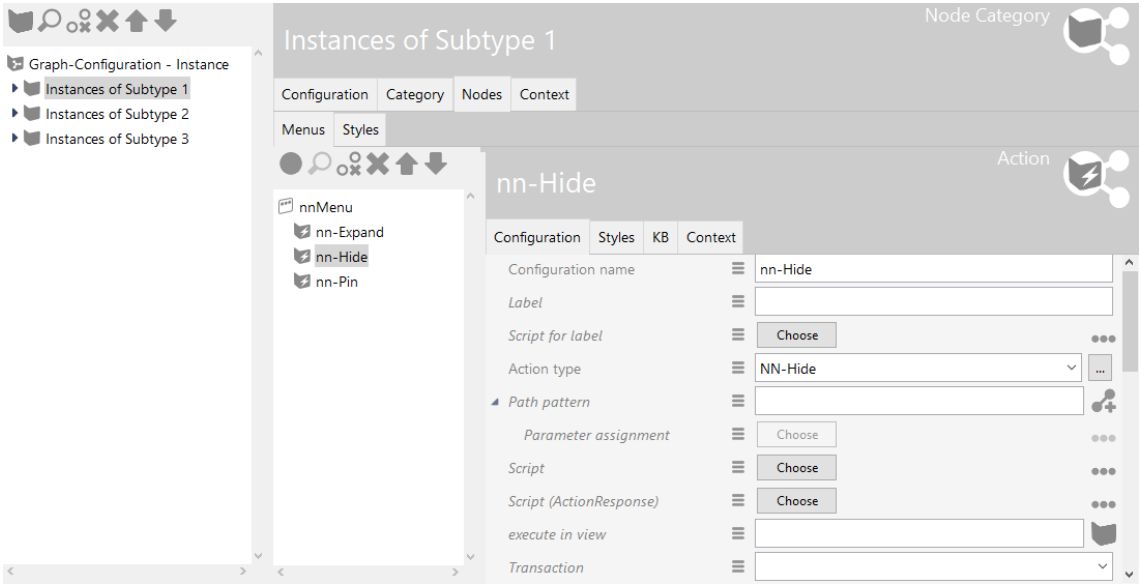
1.3.3.4.5. Aktionsart "NN-Hide"

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 durch die NN-Expand Aktion eines anderen per Relation verbundenen Knotens wieder angezeigt werden.

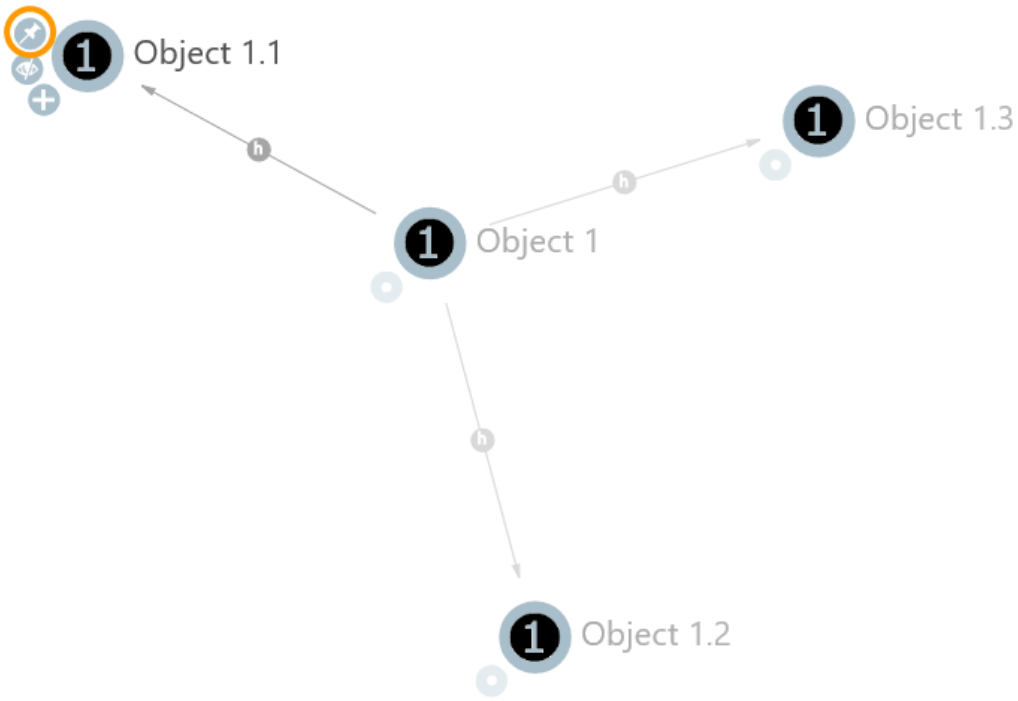


Die NN-Hide-Aktion wird wie die NN-Expand-Aktion konfiguriert, als Aktionsart wird statt "NN-Expand" allerdings "NN-Hide" ausgewählt. Um mehr als eine Aktionsart an einem Knoten zu konfigurieren, müssen mehrere Aktionen an einem Menü angelegt werden.

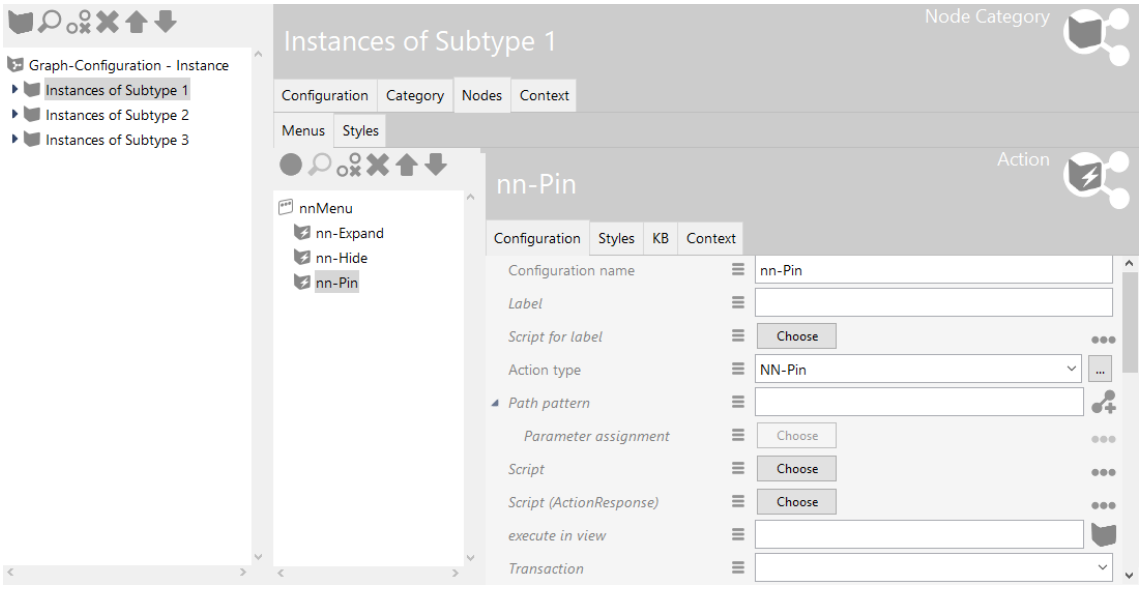


1.3.3.4.6. Aktionsart "NN-Pin"

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 fest-gepinnte 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).



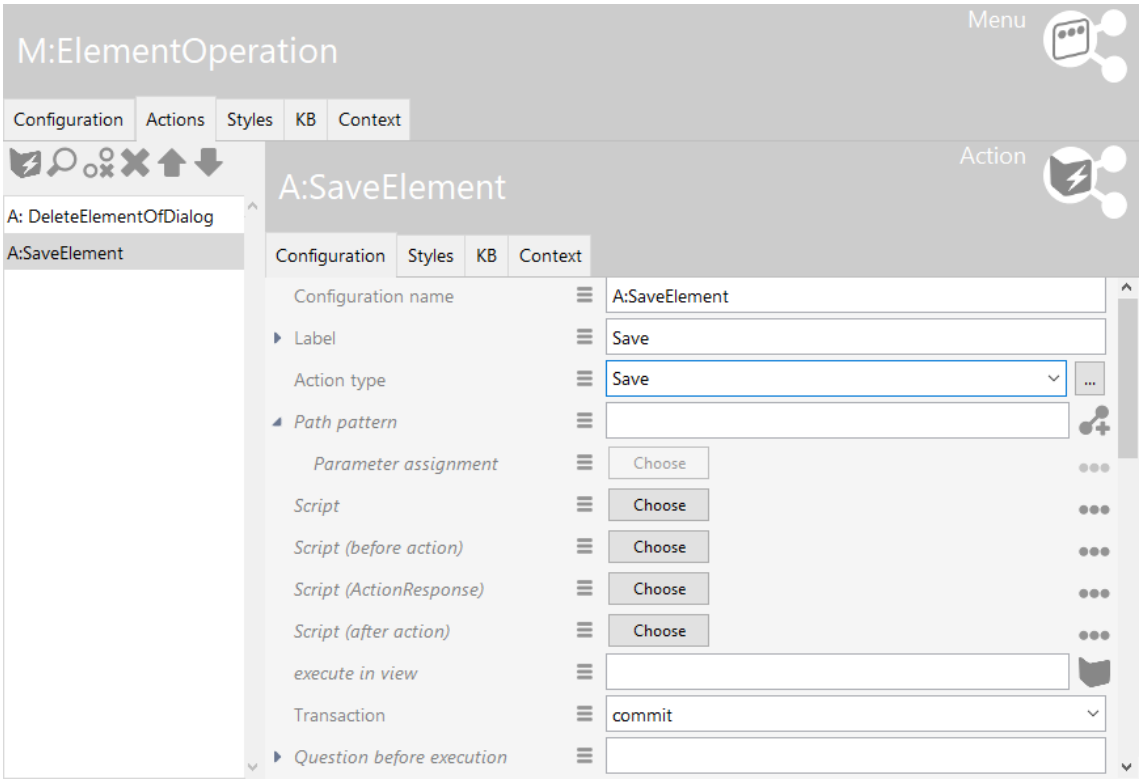
Die Konfiguration der Aktionsart erfolgt wie in der "NN-Expand-Aktion" beschrieben.



1.3.3.4.7. 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 beispielsweise dazu verwendet werden, um die einzelnen Speichern-Buttons mehrerer Edit-Felder in einem Dialog durch einen individualisierten Speichern-Button zu ersetzen.

Achtung: Möchte man bei Klick auf die Speichern-Aktion noch mehr bewirken als nur speichern (z.B. das Anlegen eines Objektes zu dem gerade bearbeiteten Objekt), so muss man anstelle von "Skript" das "Skript (nach der Aktion)" nutzen. Hintergrund ist, dass die Speichern-Aktion ansonsten von dem "Skript" überschrieben wird.

1.3.3.4.8. Aktionsart "Drucken"

Wie auch im Knowledge Builder dient die Drucken-Aktion der Generierung von Dokumenten aus dem angezeigten Modell. Es wird jedoch kein Konfigurationsdialog geöffnet, weshalb die nötigen Einstellungen an der Aktionskonfiguration hinterlegt werden müssen. Voraussetzung für die Nutzung ist das Vorhandensein der Drucken-Komponente, welche mithilfe des Admin-Tools installiert werden kann.

Je nach dem, in welcher Art von View die Drucken-Aktion ausgeführt wird, verhält sie sich leicht unterschiedlich:

- Tabellendruck: Wenn die Aktion durch eine Tabellen- oder Suche-View ausgeführt wird, wird ein Tabellendruck mit dem Inhalt der jeweiligen Tabelle ausgelöst. Wenn die Aktion mit keiner Druckvorlage verknüpft ist, wird ein neues .xlsx-Dokument erzeugt. Mit einer Druckvorlage kann der Tabelleninhalt auch in ein hinterlegtes Dokument eingebettet werden. In beiden Fällen wird die Filterung und Sortierung der angezeigten Tabelle berücksichtigt, es werden jedoch unabhängig von der eingestellten Paginierung alle Tabellenelemente ausgegeben.
- Elementdruck: Wenn die Aktion in einer anderen View ausgeführt wird, wird das Element, welches das aktuelle Modell der jeweiligen View ist, als Basis für das generierte Dokument genutzt. In diesem Fall ist die Konfiguration einer Druckvorlage zwingend erforderlich. Dieser Modus kommt auch bei Druckaktionen zum Tragen, die in Tabellenzeilen konfiguriert sind. Sie beziehen sich dann auf das jeweilige Zeilenelement.

Für eine Drucken-Aktion kann der gewünschte Dateiname sowie das Zielformat konfiguriert werden. Letzteres setzt das Vorhandensein einer passenden Konverter-Konfiguration vom Quellformat der Druckvorlage zum konfigurierten Zielformat voraus. Für mehr Informationen zur Konfiguration von Druckvorlagen siehe Berichte und Drucken.

1.3.3.5. 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.

1.3.3.6. Skripte von Aktionen

1.3.3.6.1. Skript (custom)

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

Argument	Wert
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

Variable	Wert
elements	Alle Elemente der Objektliste
type	Typ der Objektliste

Transaktionen

Bei schreibenden Änderungen ist eine Transaktion erforderlich. Bei Ausführung über den ViewConfigMapper ist das automatisch der Fall.

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());
```

1.3.3.6.2. 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.ActionResponse();

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

    return actionResponse;
}
```

Argumente

Argument	Wert
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
show-element	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.

1.3.3.6.3. 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 booleschen 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) {
        return actionEnabled(element, contexts[index]);
    });
}
```

1.3.3.6.4. 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 booleschen Werten erwartet:

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

1.3.3.6.5. 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.

1.3.3.7. Aktionssequenzen

Nicht selten möchte man Änderungen zusammenfassen, die der Anwender am Wissensnetz durchführt und die sich in mehrere aufeinanderfolgende Aktionen aufteilen.

Beispiel: In einer Aktion wird ein neues Produkt angelegt und in der nächsten Aktion werden die Eigenschaften des Produkts beschrieben. Ein Abbrechen der zweiten Aktion würde ein Produkt ohne Beschreibung im Wissensnetz hinterlassen.

Gewünscht ist ein Verhalten "Alles oder Nichts", das sicherstellt, dass entweder alle zusammengehörigen Aktionen ausgeführt werden oder keine. Weiterhin möchte man sicherstellen, dass andere Anwender die Veränderung am Wissensnetz erst dann sehen, wenn sie abgeschlossen ist. Ein solches Verhalten erzielt man durch Kapselung der Aktionen in einer "Transaktion".

Um eine Sequenz von Aktionen in einer Transaktion zusammenzufassen, markiert man die erste Aktion mit "Transaktion - beginnen" und die abschließende Aktion mit "Transaktion - beenden".

Vorsicht: Die Transaktion wird nur dann begonnen, wenn die erste Aktion auch tatsächlich eine Modifikation man Wissensnetz vornimmt. Wenn in der Aktionssequenz mehrere Objekte erzeugt werden, ist darauf zu achten, dass die Reihenfolge der Erzeugung deterministisch ist. Bei erneuter Ausführung einer Aktion müssen die Objekte also in gleicher Reihenfolge erzeugt werden. Variiert die Menge der erzeugten Objekte je nach aktueller Situation, sollte die Ausgangsmenge vor der Erzeugung stabil sortiert werden (z.B. durch Sortierung nach idString()).

Das Transaktionsende kann auch dynamisch über die Skript-Funktion "setTransactionCommit()" herbeigeführt werden.

Soll die Transaktion abgebrochen werden, kann man dies mittels einer Aktion der Art "Abbrechen" erzielen. Ein Abbruch bedeutet, dass alle bisherigen Veränderungen am Wissensnetz verworfen werden, die innerhalb der Transaktion getätigt wurden. Über die Skript-Funktion "setFailed()" kann ein Abbruch dynamisch herbeigeführt werden.

Da eine Transaktion immer an die Lebensdauer einer Session gekoppelt ist, wird eine Transaktion automatisch abgebrochen, wenn die Session endet, in der die Transaktion gestartet wurde. Öffnet man zu Beginn der Transaktion beispielsweise einen Dialog und wird dieser geschlossen, bevor die Transaktion beendet wurde, dann wird die Transaktion automatisch abgebrochen. Dies gilt nicht für einen Dialog, der während einer bereits laufenden Transaktion geöffnet wird, denn dies erzeugt eine neue Session auf dem Session-Stapel. Auch Dialog-Sequenzen (dem Schließen eines Dialogs folgt direkt das Öffnen des nächsten Dialogs), unterbrechen die Transaktion nicht.

1.3.4. View-Konfigurationselemente

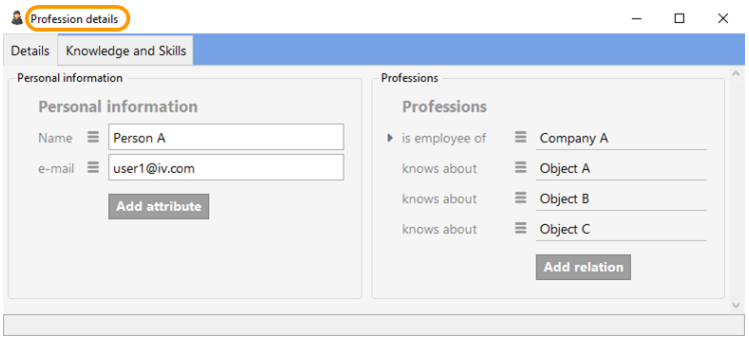
Eine Viewkonfiguration beschreibt, wie Objekte oder Typen dargestellt werden sollen. Im folgenden werden die verschiedenen Elementarten, die der View-Konfiguration zur Verfügung stehen, beschrieben.

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
Layout	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.

1.3.4.1. Alternative

Eine Alternative wird verwendet, um beliebig viele alternative Sichten auf ein Objekt zu

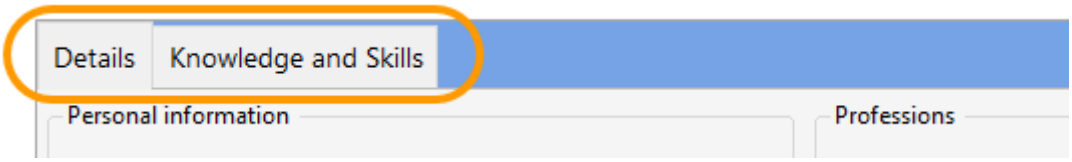
konfigurieren. Zwischen den Ansichten kann mittels Reitern gewechselt werden.

Einstellungsmöglichkeiten

Name	Value
Konfigurationsname	Der Konfigurationsname dient zur Identifizierung und Wiederverwendung der Konfiguration.
Beschriftung	Eine Beschriftung findet nur dann eine Verwendung, wenn diese Konfiguration in einer anderen Konfiguration eingebettet wird, bspw. in einer zusätzlichen *Alternative*.
Skript für Beschriftung	Das Skript für Beschriftung wird zur dynamischen Berechnung der Beschriftung verwendet und es ist nur dann verfügbar, wenn kein Eintrag unter "Beschriftung" vorhanden ist.
Default-Alternative	Die untergeordnete View, welche initial ausgewählt werden soll, kann hier festgelegt werden.
Skript für Default-Alternative	Die ausgewählte View kann auch per Skript bestimmt werden
Zuletzt gewählte Alternative wiederherstellen	Wenn aktiviert, dann bleibt die zuletzt gewählte Alternative sichtbar, auch wenn zwischendurch eine andere View aufgerufen wird.
Skript für Sichtbarkeit	Mit dem Skript für Sichtbarkeit wird dynamisch ermittelt, ob die View sichtbar sein soll oder nicht.
Initialisierungsskript	Über das Skript können Initialwerte bestimmt werden

Anzeige im Knowledge-Builder

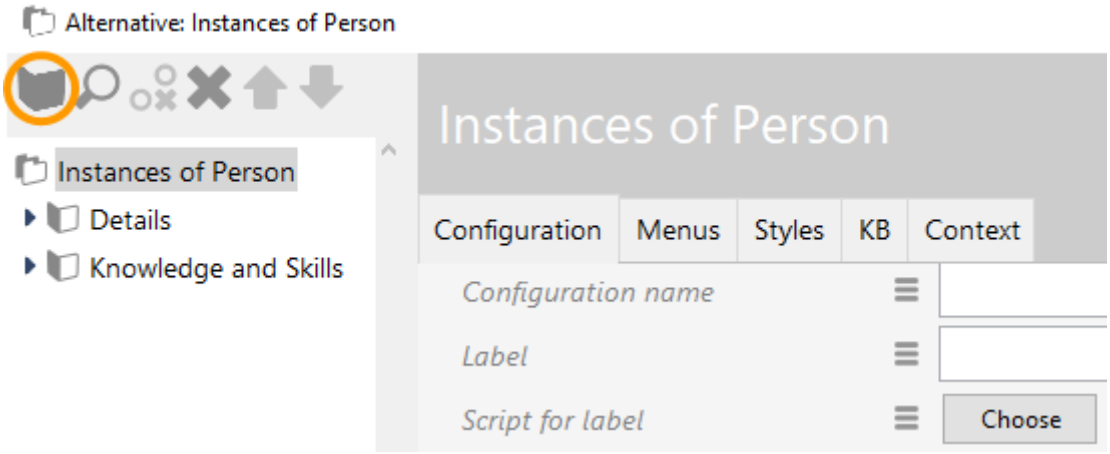
Im Knowledge-Builder werden die konfigurierten Views, welche mit der Alternative verknüpft sind, dem Benutzer in Form von Reitern verfügbar gemacht.



Beispiel einer Alternative im Knowledge-Builder: Die Reiter werden verwendet, um zwischen den Views "Details" und "Knowledge and Skills" zu wechseln.

Konfiguration der Reiter

Nach dem Anlegen einer View des Typs "Alternative" werden weitere Reiter hinzugefügt durch Klick auf die Schaltfläche "Objekte von Elementkonfiguration neu anlegen".



Die Beschriftung der zugeordneten Views einer Alternate dienen zugleich als Beschriftung des Reiters.

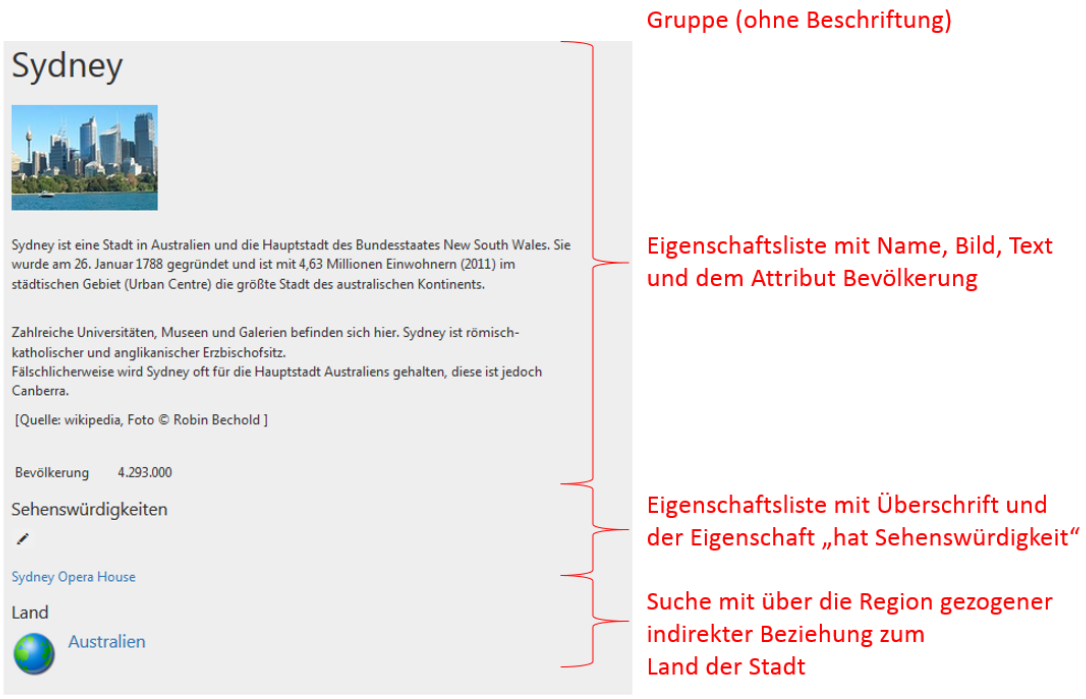
1.3.4.2. Layout

Mithilfe eines Layouts lassen sich verschiedene Unterkonfigurationen in einer Ansicht zusammenfassen. Die Unterelemente werden dann der Reihe nach dargestellt.

Einstellungsmöglichkeiten

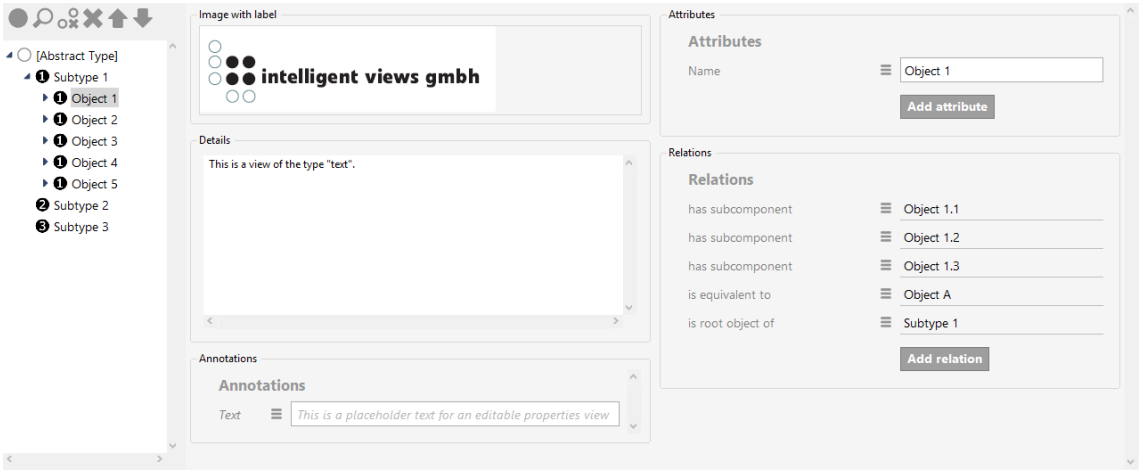
Name	Wert
Konfigurationsname	Der Konfigurationsname wird zur Identifikation und Wiederverwendung der Konfiguration genutzt.
Beschriftung	Eine Beschriftung findet nur ihre Anwendung, wenn diese Konfiguration in einer anderen Konfiguration wie z.B. *Alternative* eingebettet ist.
Ausrichtung	Legt fest, ob die Subviews horizontal oder vertikal aufeinanderfolgen. Das Standardverhalten ist horizontale Ausrichtung.
Skript für Sichtbarkeit	Ein Skript, welches bestimmt, ob das Layout angezeigt wird.

Darstellung in einer Anwendung



Darstellung im Knowledge-Builder

Im Knowledge-Builder wird um ein Layout ein Rahmen gezeichnet. Die Views der Unterkonfigurationen werden dann in diesem Rahmen angezeigt.



Ein Layout mit folgenden Unterkonfigurationen: der Eigenschaftsliste "Bild und Text", der Eigenschaftsliste "Eigenschaften" und der Suche "Ähnliche Sehenswürdigkeiten"

1.3.4.3. Hierarchie

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.

Name	Wert
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 Relationsziels 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

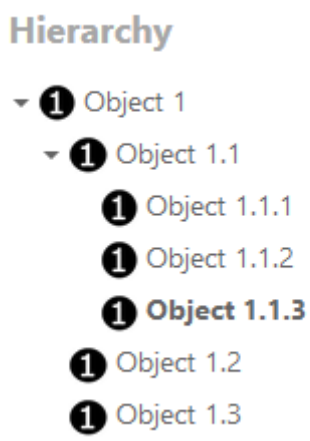
Name	Ermittlung von …
Relation (absteigend)	Unterelementen
Relation (aufsteigend)	Oberelementen
Strukturabfrage (absteigend)	Unterelementen
Strukturabfrage (aufsteigend)	Oberelementen
Skript (absteigend)	Unterelementen
Skript (aufsteigend)	Oberelementen

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.

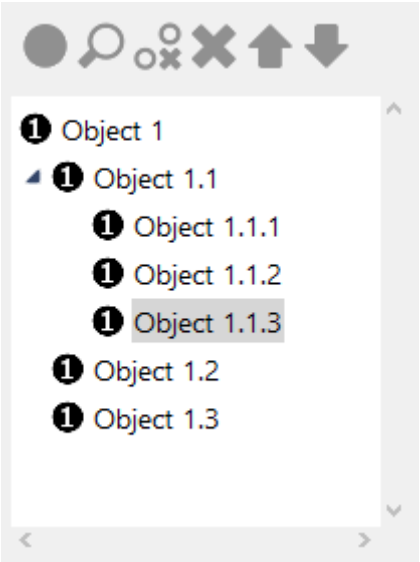
Darstellung in einer Anwendung

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

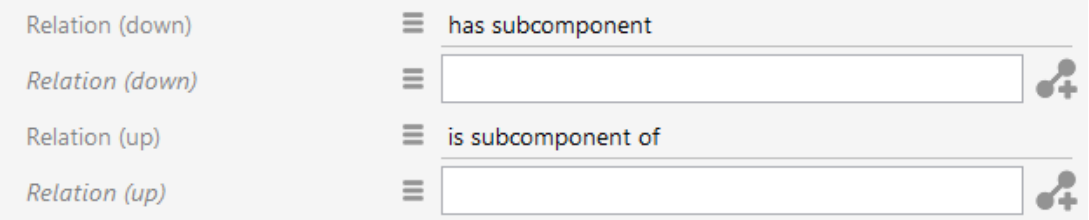
verschiedene Attributtypen angegeben, wird mit den Teilmengen eine Schnittmenge gebildet.

Beispiel — Anwendungsfall

Typischerweise werden Hierarchien verwendet, um Ober-/Unterthema-Relationen oder Teil-von-Relationen darzustellen.

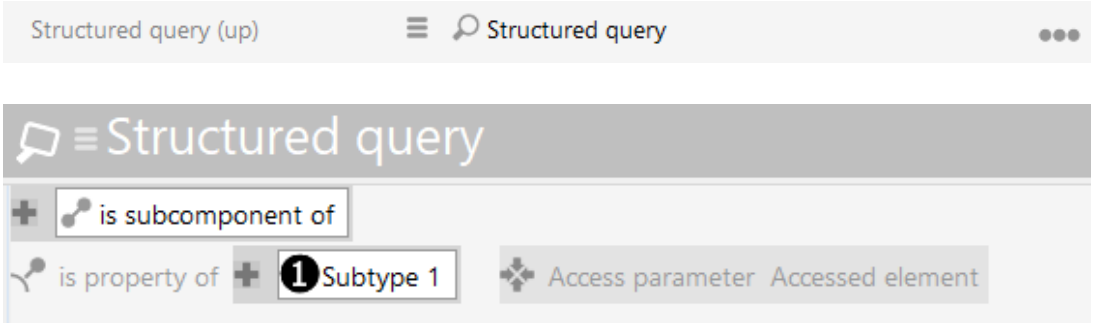
1. Hierarchiebildende Relation

   Die direkteste Variante. Die Relationen, die die Hierarchie bilden, werden eingetragen.



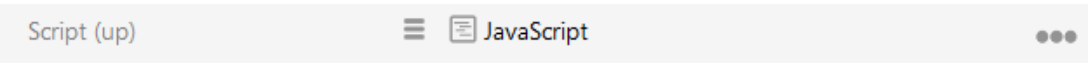
2. Hierarchiebildende Strukturabfrage

   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.



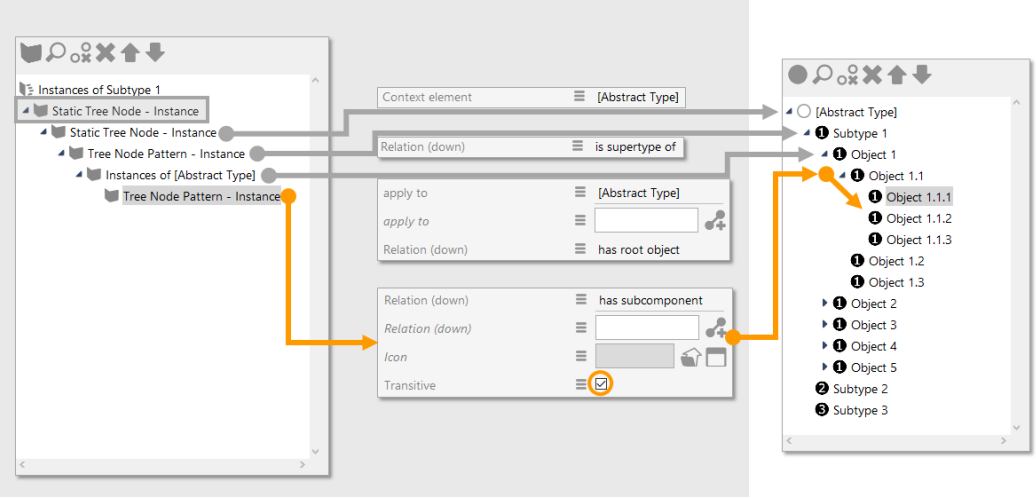
   Skript *hat Oberthema*

```
function relationsOf(element) {
    return element.relations("hatOberthema");
}
function targetsOf (element)  {
    return element.relationTargets("hatOberthema");
```

```
}
```

1.3.4.4. Baum

Ebenso wie die "Hierarchie" dient der "Baum" der Konfiguration einer hierarchischen Baumstruktur. Im Gegensatz zur Hierarchie kann ein Baum auch statische Knoten enthalten. Es ist somit möglich, einen Baum ohne ein Wissensnetz-Ausgangselement zu bilden. Ein weiterer Unterschied besteht darin, dass die Unterknoten eines "Baums" verschiedenartig konfiguriert sein können, während sich alle Knoten einer "Hierarchie" für ein gegebenes Wissensnetzelement gleichartig verhalten.



View configuration for tree view

Tree view in Knowledge Builder

Die Baumkonfiguration kennt grundsätzlich zwei Arten von Knoten:

- **Statischer Hierarchieknoten:** Knoten dieses Typs sind immer vorhanden, sofern eine Verbindung zur Baumwurzel existiert. Über die Relation "Kontextelement" kann der Knoten optional an ein Wissensnetzelement gebunden werden. **Achtung:** Der oberste Knoten des Baums ist immer statisch und immer unsichtbar.
- **Hierarchie-Knotenmuster:** Dieser Typ kann pro Ebene mehrere Knoten ausbilden. Je Relationsziel, welches sich vom Element des übergeordneten Knotens ausgehend erreichen lässt, wird ein Knoten ausgebildet. Über Setzen der Eigenschaft "Transitiv" können mehrere Ebenen ausgebildet werden. Durch das Setzen der Eigenschaft "anwenden auf", kann eingeschränkt werden, auf welche Elementtypen das Knotenmuster anwendbar ist — ansonsten kann das Knotenmuster auf alle Elemente angewendet werden, die im Ziel-Gültigkeitsbereich der konfigurierten Relationen liegen. Alternativ zur Ermittlung über einen Relationstyp können Subknoten über eine Strukturabfrage ermittelt werden. Die Strukturabfrage beginnt mit dem Element des übergeordneten Knotens. Die untergeordneten Knoten werden durch den Teil der Abfrage ermittelt, der mit dem vordefinierten Bezeichner "subnode" gekennzeichnet ist. Möchte man die Option "Transitiv" nutzen, dann ist die entsprechende Relation in der Abfrage mit dem vordefinierten Bezeichner "subnodeRelation"

zu kennzeichnen.

Analog zur "Hierarchie" kann die Sortierung der Baumknoten konfiguriert werden. Diese Konfiguration wirkt allerdings nicht für den Baum global sondern je Knotenkonfiguration für dessen Unterknoten.

Schließlich sind angezeigtes Bild und Beschriftung pro Knotentyp wahlweise direkt oder per Skript konfigurierbar.

Einstellungsoptionen

Name	Wert
Konfigurationsname	Der Konfigurationsname dient zur Identifizierung und Wiederverwendung der Konfiguration.
Beschriftung	Die Beschriftung findet nur dann Anwendung, wenn die Konfiguration in einer anderen Konfiguration, z. B. einer Alternative, eingebettet wird.
Skript für Beschriftung	Das Skript gibt eine Zeichenkette für die Beschriftung aus, anstelle der Verwendung des Beschriftungs-Attributs.
Detailansicht ausblenden	Standardmäßig wird neben einem Baum oder einer Hierarchie eine vorkonfigurierte Detailansicht angezeigt. Durch Setzen der Option wird die Detailansicht ausgeblendet. **HINWEIS** Die Standard-Detailansicht kann durch Konfigurieren einer angepassten Ansicht ersetzt werden.
Verbiete manuelles Sortieren	Standardmäßig können im Knowledge-Builder Elemente innerhalb des Schema-Baumes umgehängt werden. Wenn die Option aktiviert ist, können die Elemente nicht mehr umgehängt werden.
Zuletzt ausgeklappten Knoten wiederherstellen	Wenn aktiviert, bleibt der zuletzt ausgeklappte Knoten für ein- und dasselbe Kontextelement während der gesamten Web-Frontend Sitzung erhalten.
Skript für Sichtbarkeit	Skript, welches einen Booleschen Wert zurückgibt, ob die View angezeigt werden soll oder nicht.
Sortierung	
Absteigend sortieren	Steuert, ob auf- oder absteigend sortiert wird. Ist dieser Parameter nicht gesetzt, wird aufsteigend sortiert.

Name	Wert
Primäres Sortierkriterium	Auswahlmöglichkeit für das Kriterium, nach dem die Unterknoten sortiert werden: • *Position* : Die durch die Metaeigenschaft *Reihenfolge* der involvierten Relationen festgelegte Reihenfolge wird verwendet (Default). • *Wert* : Der Anzeigename des Relationsziels wird verwendet. • *Skript zur Sortierung* : Das im Attribut *Skript zur Sortierung* hinterlegte Skript wird zur Ermittlung des Sortierkriteriums verwendet.
Sekundäres Sortierkriterium	Sortierkriterium für Unterknoten, 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. **Achtung**: bei mehreren unterschiedlichen Unterknoten-Konfigurationen wird das Skript potentiell mit Instanzen unterschiedlicher Typen aufgerufen, und sollte entsprechend allgemein formuliert werden.
KB	
Skript für Fensterstatus	Gibt eine Status-Beschriftung für die Fußzeile des Fensters aus, wenn die Detailansicht in einem neuen Fenster geöffnet ist.
Skript für Fenstertitel	Gibt eine Beschriftung für den Fenstertitel aus, wenn die Detailansicht im Knowledge-Builder in einem neuen Fenster geöffnet ist.
Elemente erzeugen ohne Frage nach Namen	Wenn aktiviert, erlaubt das Menü direkt über dem Baum oder der Hierarchie ein Anlegen neuen Objekte, ohne dass nach einem Namen für das neue Element gefragt wird.

1.3.4.5. Eigenschaften

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.

Name	Wert
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

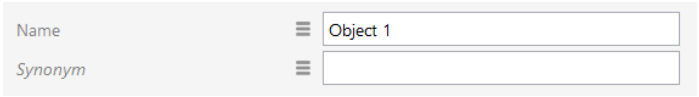
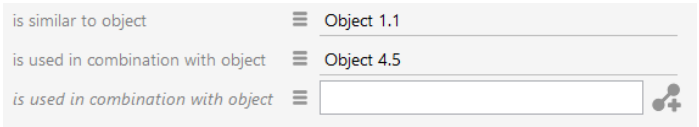
Name	Wert
Absteigend sortieren	Steuert, ob auf- oder absteigend sortiert wird. Ist dieser Parameter nicht gesetzt, wird aufsteigend sortiert.
Primäres Sortierkriterium	Auswahlmöglichkeit für das Kriterium, nach dem die Eigenschaften sortiert werden: • *Position* : Die in der Konfiguration festgelegte 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.

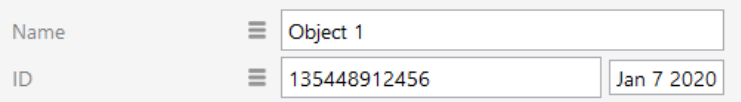
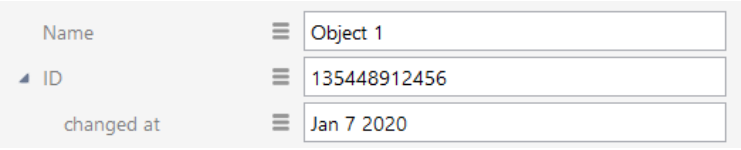
1.3.4.6. Eigenschaft

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
Konfigurationsname	Der Konfigurationsname dient zur Indentifizierung und Wiederverwendung der Konfiguration.
Beschriftung	Zeigt den Namen der Eigenschaft an. Wenn keine Beschriftung eingetragen wurde, wird der Name des Eigenschaftstyps ausgegeben.
Skript für Beschriftung	Die Beschriftung kann hier mithilfe eines Skripts ermittelt werden.
Eigenschaft	Verlinkung zum Eigenschaftstyp, der angezeigt werden soll.
Abfrage für virtuelle Eigenschaften	Alternativ zu "Eigenschaft": Anstatt des Verweises auf einen Eigenschaftstyps kann eine Strukturabfrage zur Ermittlung des Eigenschaftswertes verwendet werden. Dies ist vor allem dann nützlich, wenn die anzuzeigende Eigenschaft sich nicht direkt am Zugriffselement der View befindet.
Skript für virtuelle Eigenschaften	Alternativ zu "Eigenschaft": Verwendung eines Skriptes, das die anzuzeigenden Werte berechnet.Wenn die Meta-Eigenschaft "**Automatisch aktualisieren**" gesetzt ist, wird die Ansicht im KB automatisch aktualisiert, falls sich ein Eingangswert geändert hat, auf dem die Berechnung basiert. **Vorsicht:** Wenn diese Option gesetzt ist, kann dies einen signifikanten Einfluss auf die Anzeige-Performance haben, abhängig vom Skript.
Anzeigeart	Diese Option ist verfügbar in zwei Fällen: 1. Die Eigenschaft ist eine Relation:Auswahloption für die Anzeige der Beschriftung des Relationsziels. Diese Einstellung ist nur dann verfügbar, wenn die Einstellung für die Relationszielansicht auf *Auswahl* oder *Relationsstruktur* gesetzt ist. 2. Die Eigenschaft ist ein Dateiattribut: Auswahloption für die Anzeige des Wertes in einem Dateiattribut. Auswahloptionen: ◦ **Symbol (topicIcon):** Symbol des Relationsziels bzw. Datei als Symbol ◦ **Symbol und Zeichenkette** ◦ **Zeichenkette (Namensattribut):** Names des Relationsziels / Name der Datei

Name	Wert
Einblendungsfilter	Nur relevant in der View für das Editieren von Elementen: Diese Option kann dazu benutzt werden, um zu entscheiden, ob die Konfiguration angezeigt werden soll. Die Abfrage erhält das Zugriffselement der Eigenschaft als Input. Die Eigenschaft wird nur dann zum Editieren angezeigt, wenn die Abfrage ein Ergebnis zurückliefert.
Neue Eigenschaften einblenden	Nur relevant in der View für das Editieren von Elementen.Folgende Optionen sind verfügbar: • **nie:** Die betreffende Eigenschaft wird nur angezeigt, wenn sie bereits gesetzt wurde. Wenn der Eigenschaftswert ersatzlos entfernt wird, dann verschwindet auch die Eingabezeile. Um wieder neue Eigenschaften hinzuzufügen und anzeigen zu lassen, kann dies mittels der Schaltfläche "Attribut oder Relation hinzufügen" erreicht werden. • **wenn noch nicht vorhanden:** Die Eingabezeile für die Eigenschaft wird nur dann angezeigt, wenn die Eigenschaft noch nicht gesetzt wurde. Dies ermöglicht ein schnelles und einfaches Eingeben von Eigenschaften und verhindert, dass das Eingeben der Eigenschaft vergessen wird.  • **immer:** Die Eingabezeile für die Eigenschaft wird immer angezeigt, auch wenn bereits ein Eintrag dazu vorhanden ist. Das Schema muss hierbei die mehrfache Vergabe der Eigenschaft zulassen.  **HINWEIS** Wenn keine dieser Optionen gewählt wurde, gleicht das Standardverhalten der Auswahl "nie".Die zuvor verfügbare Eigenschaft "Einblendung zusätzlicher Eigenschaften" früherer i-views Versionen (5.3 und früher) ist in der Option "immer" untergebracht.

Name	Wert
Konfiguration für eingebettete Eigenschaften	Verweist auf eine View-Konfiguration, welche dazu verwendet wird, die Meta-Eigenschaft zu einer Eigenschaft anzuzeigen. Die Metaeigenschaften werden eingebettet angezeigt, also hinter dem Eigenschaftswert. Der Name des Eigenschaftstyps der Metaeigenschaft wird dabei nicht angezeigt. 
Konfiguration für Metaeigenschaften	Verweist auf eine View-Konfiguration, welche dazu verwendet wird, die Metaeigenschaft zu einer Eigenschaft anzuzeigen. Die Metaeigenschaft wird unterhalb des Eigenschaftswerts angezeigt. Für eine Anzeige im Web-Frontend müssen die Eigenschaften-Konfigurationen der Eigenschaft und der Metaeigenschaft auf "initial ausgeklappt" gesetzt sein. 
Klick-Aktion	Die Aktion, die bei Klick auf die Eigenschaft ausgeführt wird.
Skript für Sichtbarkeit	Skript, das die Bedingungen definiert, unter welchen die Eigenschaft sichtbar ist.
Relationsziel (nur verfügbar für Relationen)	

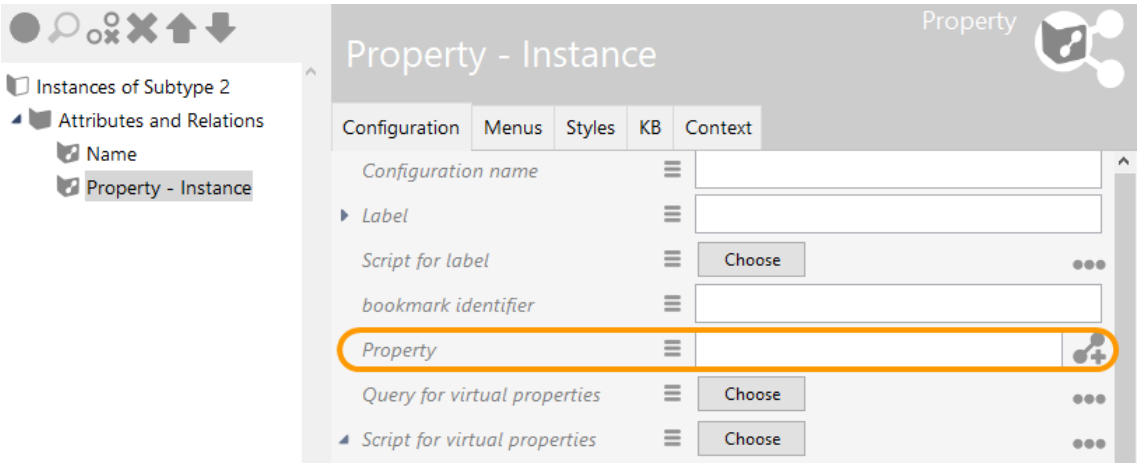
Name	Wert
Relationszielansicht	Wenn eine Relation als Eigenschaft festgelegt ist, bestimmt dieser Parameter die View für die Relationsziele: • **Auswahl:** Alle Relationsziele werden mit einer vorangestellten Checkbox dargestellt. Im Fall existierender Relationen ist die Checkbox angehakt. • **Drop down:** Diese Einstellung ist nützlich, wenn das Relationsziel nur einmal gewählt werden soll. Es wird eine Dropdown-Liste angezeigt, welche alle Relationsziele enthält. • **Relationsstruktur:** Alle Relationsziele werden im linken Bereich der Relationszielansicht aufgelistet, eher in Form einer Hierarchie. Der rechte Bereich der Relationszielansicht zeigt die Detailansicht zur ausgewählten Relation. Diese Ansicht ist dann wirkungsvoll, wenn die Konfiguration direkt einer Top-Level Konfiguration untergeordnet ist. • **Tabelle:** Tabellenansicht der *Relationen*. **HINWEIS** Die Tabellenansicht kann nicht im Knowledge-Builder angewendet werden. Für die Tabellenansicht muss eine Tabellen-View im Eintrag "*Tabelle*" zugewiesen sein. Der Eintrag erscheint erst, wenn die Relationszielansicht "Tabelle" auch ausgewählt wurde. • **Tabelle (Relationsziele):** Tabellenansicht der *Relationsziele*. **HINWEIS** Diese Tabelle kann im Knowledge-Builder angewendet werden.
Tabelle	Nur verfügbar, wenn zuvor im Eintrag "Relationszielansicht" der Wert "*Tabelle*" oder der Wert "*Tabelle (Relationsziele)*" gewählt wurde. Die hier definierte Tabellen-Konfiguration bestimmt, welche Eigenschaften in Tabellenform ausgegeben werden sollen. Um ein Relationsziel anzeigen zu können, muss mindestens das Attribut "Name" in der Tabelle konfiguriert sein. Für die Konfiguration einer Tabelle siehe Kapitel "Tabelle".
Relationszielfilter	Abfrage für die Filterung der anzuzeigenden Relationsziele.
Relationszieltypfilter	Abfrage für die Filterung der anzuzeigenden Relationsziele anhand ihres Typs.

Name	Wert
Skript für Relationszielbezeichner	Skript, welches eine Zeichenkette für die Beschriftung des Relationsziels zurückgibt. Falls nicht verwendet, wird der Primärname des Relationsziels für dessen Beschriftung angezeigt. **Beispiel:** Eine Person gehört zu einer Abteilung mit dem Namen "Abt. IV". Mithilfe eines entsprechenden Skriptes kann die Beschriftung für das Relationsziel umgeändert werden in: "Verwaltung Darmstadt, Abt. IV".
Einblendung des Relationsziels	Nur verfügbar für Relationen.Normalerweise wird nur der Name des Relationsziels angezeigt. Wenn man auf den Namen klickt, wird das Relationsziel in einer anderen Ansicht geöffnet.Wenn jedoch die Option "Einblendung des Relationsziels" aktiviert ist, wird das Relationsziel inkl. aller Eigenschaften direkt in derselben Ansicht angezeigt.
Darstellung	
Tooltip	Tooltip, welcher erscheint, wenn der Mauszeiger über das Relationsziel positioniert wird.
Platzhaltertext	Platzhaltertext, welcher in hellgrauer Schrift angezeigt wird, wenn das betreffende Zeichenkettenattribut noch keinen Attributwert besitzt.
Skript für Platzhaltertext	Skript, welches eine Zeichenkette für den Platzhaltertext zurückgibt, anstatt eines statisch konfigurierten Platzhaltertextes.
Skript für Tooltip	Skript, welches eine Zeichenkette für den Tooltip zurückgibt, anstatt eines statisch konfigurierten Tooltips.
Sortierung	
Skript für Sortierung	Das Skript wird verwendet, um den zu sortierenden Wert zu ermitteln. Siehe nachfolgendes Beispiel.
Absteigend sortieren	Bestimmt, ob die Eigenschaften anhand ihres Namens nach absteigender oder nach aufsteigender Reihenfolge sortiert werden sollen. Wenn diese Option nicht gesetzt ist, erfolgt die Sortierung in *aufsteigender* Reihenfolge.

HINWEIS	Optionen können entweder gesetzt werden, indem ihr Wert festgelegt wird oder, falls verfügbar, durch ein gleichwertiges Skript. Optionswert und Skript können nicht zur selben Zeit verwendet werden.

Konfiguration einer Eigenschaft

Eine Eigenschaft kann nur als Teil einer Liste von Eigenschaften — der Eigenschaften-View — konfiguriert werden. Es ist jedoch möglich, innerhalb einer Eigenschaften-View nur eine Eigenschaft-View zu verwenden.

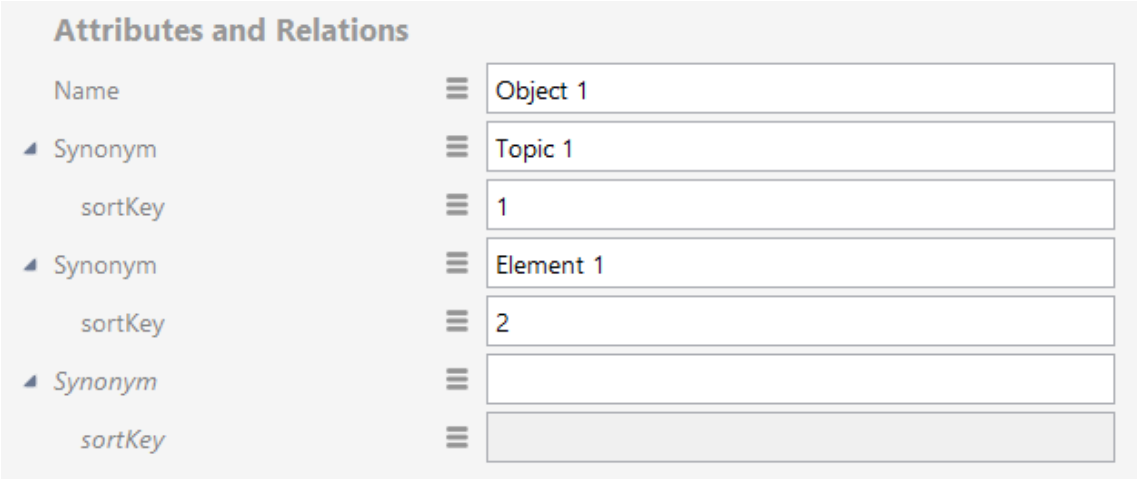


In diesem Beispiel enthält die Eigenschaften-View bereits die Eigenschaft "Name". Eine zweite Eigenschaft wird erzeugt, indem ein Attribut oder eine Relation für den Eintrag "Eigenschaft" (markiert in Orange) gewählt wurde.

Sortierte Darstellung der Eigenschaften eines Objekts

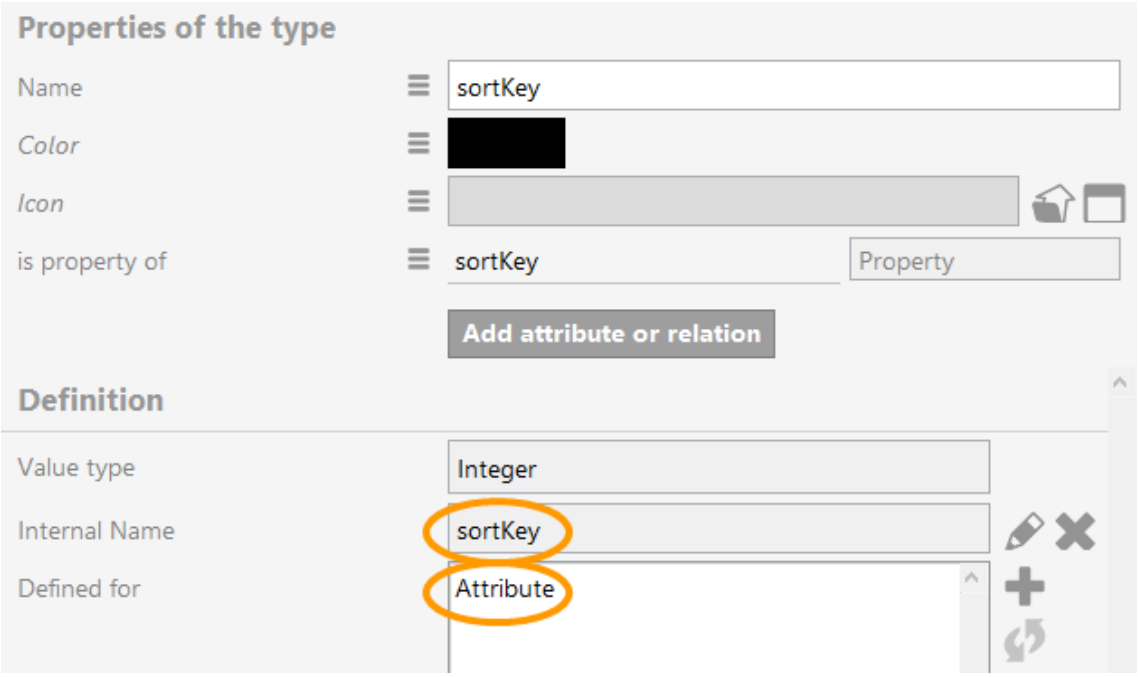
Wenn ein Objekt mehrere Eigenschaften desselben Typs besitzt, werden sie normalerweise in alphabetischer Reihenfolge dargestellt. Falls nichtsdestotrotz die Eigenschaften in einer abweichenden Reihenfolge dargestellt werden sollen, (bspw. um Präferenzen für Synonyme oder Vornamen aufzuzeigen), kann ein dediziertes Metaattribut an jeden Eigenschaftswert angehängt werden.

Das Attribut "sortKey" kann für Editierzwecke mithilfe einer Konfiguration für Metaeigenschaften angezeigt werden:

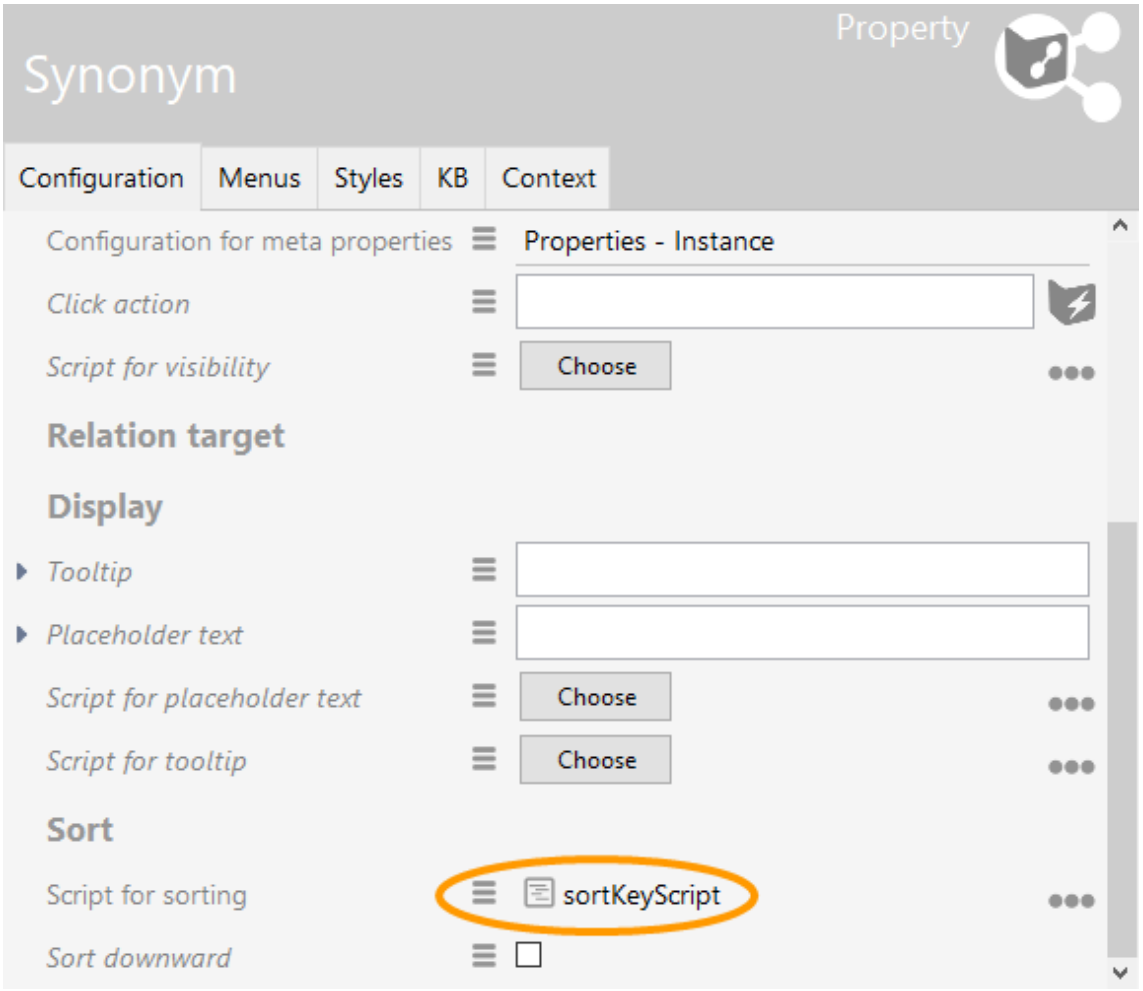


HINWEIS Im Fall des Attributs "Synonym" wurde 2 für den sortKey-Wert eingegeben, wodurch dieser Wert am Ende der Liste angezeigt wird.

Für diesen Zweck muss ein Attributtyp mit dem internen Namen `sortKey` definiert werden, welcher auf jede individuelle Eigenschaft anwendbar sein soll:



Das sortKey Attribut wird dann mithilfe eines Skriptes für Sortierung referenziert, welches an die Eigenschaft-View angehängt wird:



Beispiel eines *Skript für Sortierung*:

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

1.3.4.7. Edit

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
Konfigurationsname	Der Konfigurationsname dient zur Identifizierung und Wiederverwendung einer Konfiguration.
Beschriftung	Eine Beschriftung findet nur ihre Anwendung, wenn diese Konfiguration in einer anderen Konfiguration wie z.B. *Alternative* eingebettet ist.
Skript für Beschriftung	Mittels Skript kann die Beschriftung dynamisch ermittelt werden. Skript für Beschriftung und Beschriftung schließen sich gegenseitig aus.
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	Wenn diese Option gesetzt ist, wird der Speichern-Knopf nicht angezeigt. Stattdessen kann ein angepasster Button mit einer Aktion des Aktionstyps "Speichern" verwendet werden.
Skript für Sichtbarkeit	Skript, das einen Booleschen Wert zurückgibt, ob die View sichtbar sein soll oder nicht.

1.3.4.8. Tabelle

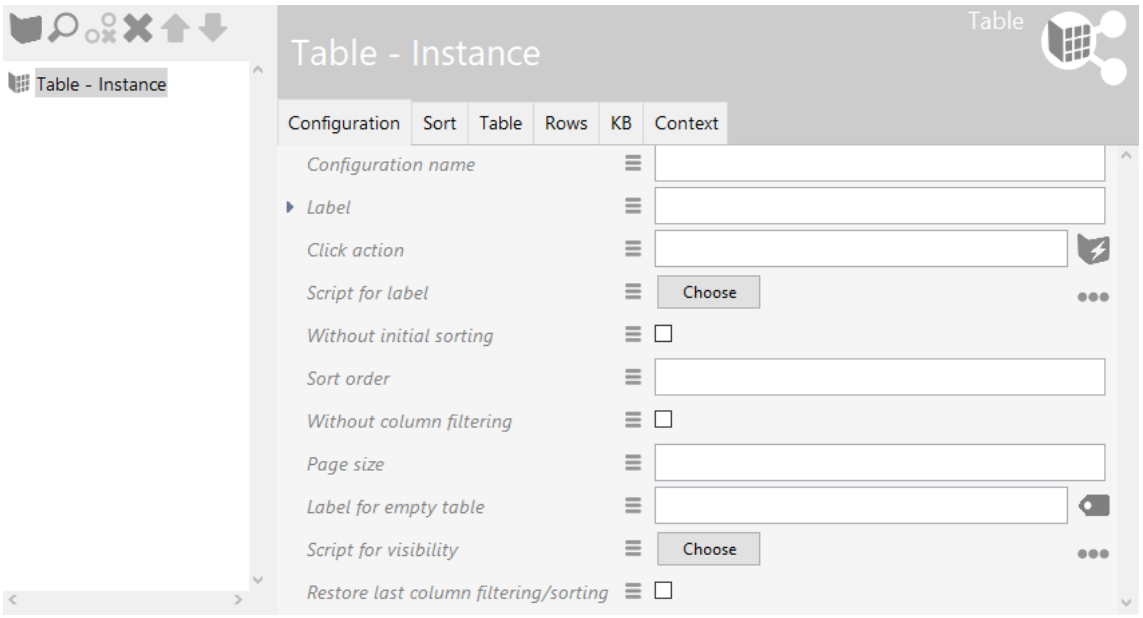
Tabellen können als Unterkonfiguration für die Ergebnisanzeige von Abfragen des Konfigurationstyps "Suche" oder als eigenständige Konfiguration zur Darstellung der Objektlisten im Knowledge-Builder verwendet werden.

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:

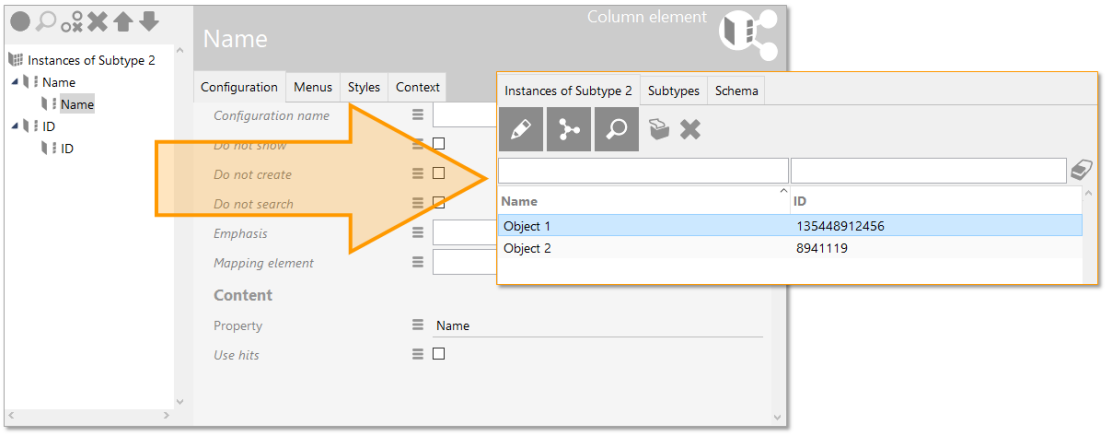
	Neues Unterelement anlegen und verknüpfen.
	Bereits vorhandene mögliche Unterelemente durchsuchen und verknüpfen
	Verknüpfung wieder löschen. Das Unterelement bleibt dabei als Objekt erhalten und kann in anderen Konfigurationen wieder verwendet werden.
	Gewähltes Unterelement komplett löschen. Falls 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.

HINWEIS	Die Verfügbarkeit einer Aktion hängt davon ab, welches Tabellen-

Konfigurationselement in der Hierarchie auf der linken Seite ausgewählt ist.

Beispiel einer einfachen Tabellenkonfiguration

Für eine Objektliste soll zusätzlich eine ID in der Tabelle angezeigt werden. Das Namensattribut sollte bei einer angepassten Konfiguration aus Gründen der Übersichtlichkeit nicht vergessen werden.



Setting options (table)

Name	Wert
Konfiguration	
Konfigurationsname	Der Konfigurationsname dient zur Identifizierung und Wiederverwendung von Konfigurationen.
Beschriftung	Bestimmt eine statische Überschrift für die Tabelle.
Klick-Aktion	Legt fest, welche Aktion ausgeführt werden soll, wenn in eine Tabellenzeile geklickt wird.
Skript für Beschriftung	Dient zur dynamischen Ermittlung der Beschriftung; Rückgabewert ist eine Zeichenkette.
Ohne automatische Sortierung	Beim Laden der Tabellen-Ansicht wird die Sortierung nicht automatisch ausgeführt. Standard-Prozess: Die erste Tabellenspalte wird zur initialen Sortierung verwendet.

Name	Wert
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
Ohne Spaltenfilter (VCM)	Unterdrückt die Anzeige der Spaltenfilter im Web-Frontend. Im Knowledge-Builder werden die Spaltenfilter immer angezeigt.
Anzahl Zeilen (Page size) (VCM)	Legt fest, wie viele Tabellenzeilen (= Suchergebniseinträge) auf einer Seite angezeigt werden sollen. Standard-Wert: 20
Label bei leerer Tabelle (VCM)	Eine Beschriftung, welche anstelle der ursprünglichen Beschriftung angezeigt wird, wenn die Tabelle leer ist.
Skript für Sichtbarkeit (KB)	Skript, welches einen Booleschen Wert ausgibt, ob die Tabelle sichtbar sein soll oder nicht. Zum Beispiel wird im Knowledge-Builder der gesamte Reiter der Tabelle nicht angezeigt, wenn die Sichtbarkeit auf *false* gesetzt ist. Im Web-Frontend hat diese Option keine Auswirkung.
Zuletzt gewählte Sortierung/Spaltenfilterung wiederherstellen (VCM)	Stellt die zuletzt gewählte Filterung oder Sortierung während der Dauer einer Web-Frontend Session wieder her.
Strukturrelation	Wenn diese Tabellenkonfiguration in eine andere Tabellenkonfiguration eingebettet ist, beziehen sich alle Spalten dieser Tabelle auf die Relationsziele der konfigurierten Strukturrelation. Wenn zum Beispiel die äußere Tabelle Personen auflistet und die innere Tabelle "besitzt" als Strukturrelation nutzt, beziehen sich alle Spalten der inneren Tabelle auf die Dinge, die eine Person besitzt. Die konfigurierten Eigenschaften der Relationsziele (z.B. alle Kategorienamen von allen besessenen Dingen) werden in der Tabellenspalte akkumuliert. Wenn ein Spaltenelement der inneren Tabelle seine Werte per Skript oder Abfrage bestimmt, wird das Skript oder die Abfrage einmal für jedes Relationsziel ausgeführt, und die Ergebnisse ebenso in der Spalte akkumuliert.
Sortierung	
Spalte	Spalten-Konfiguration, auf deren Basis die Sortierung angewendet werden soll.

Name	Wert
Sortierpriorität	Ein Ganzzahlwert bestimmt die Abfolge, nach welchen Spaltenwerten die Zeilen der Tabelle zuerst gefiltert werden. **Beispiel:** Wenn eine ID wichtiger für die Sortierung der Objekte ist als der Primärname der Objekte, dann erhält die Spalte für die ID-Werte die Sortierpriorität 1 und die Spalte für den Primärnamen die Sortierpriorität 2. Eine höhere Sortierpriorität überschreibt die Sortierrichtung ("Absteigend sortieren") einer anderen Spalte.
Absteigend sortieren	Legt fest, ob die Werte nach aufsteigender oder absteigender (alphanumerischer) Reihenfolge sortiert werden sollen.
Tabelle	
Reiter "Menüs"	Für den Knowledge-Builder können die Menü-Aktionen oberhalb der Tabelle hier konfiguriert werden. Für mehr Informationen siehe Kapitel "Aktionen für den Knowledge-Builder".
Reiter "Styles" (VCM)	Für das Web-Frontend können unterschiedliche Styles auf die gesamte Tabelle angewendet werden.
Zeilen	
Reiter "Styles"	Wenn eine Tabelle im Knowledge-Builder verwendet wird, können Styles für die Zeichenformatierung verwendet werden.
KB	
Automatische Suche	• Automatische Suche: Die Suche wird gestartet, sobald die Tabelle durch Auswahl sichtbar wird • Keine automatische Suche: Die Suche wird erst gestartet, wenn auf den Suche-Button geklickt wurde. • Automatische Suche bis Grenzwert (System-Einstellungen): Die automatische Suche wird bis zu einer bestimmten Anzahl an Objekten ausgeführt, darüber nicht mehr. Die Anzahl kann in den globalen Einstellungen des KB festgelegt werden unter *Einstellungen > System > Ordner > Automatische Abfrage bis Anzahl Objekte*.
Elemente erzeugen ohne Frage nach Namen	Wenn diese Option aktiviert ist, können neue Elemente durch Klick auf den Button "Neu" angelegt werden, ohne dass ein Dialog nach einem Namen für das Element fragt. Als Hinweis für den fehlenden Namen wird ein Punkt "." angezeigt.
Skript für Fenstertitel	Gibt eine Zeichenkette für die Beschriftung zurück, wenn die Tabelle im KB in einem eigenen Fenster geöffnet wird. Für das Hinzufügen dieses Attributs muss die Tabellen-Konfiguration u. U. unkonfiguriert bearbeitet werden.

Name	Wert
Skript für Fensterstatus	Gibt eine Zeichenkette für die Beschriftung der Fenster-Fußzeile zurück, wenn die Tabelle im KB in einem eigenen Fenster geöffnet wird. Für das Hinzufügen dieses Attributs muss die Tabellen-Konfiguration u. U. unkonfiguriert bearbeitet werden.
Ohne Vererbung	Wenn die Tabelle für Objektlisten verwendet wird, bewirkt das Setzen dieser Option, dass nur die Objekte des aktuell gewählten Typs angezeigt werden und nicht noch zusätzlich die Objekte der Untertypen.
Kontext	
anwenden auf	Schränkt den Kontext der Tabelle auf Objekte eines bestimmten Typs ein.
anwenden auf Untertypen	Schränkt den Kontext auf den Untertyp ein (anstatt auf dessen Objekte)
anwenden in	Anwendungskontext, in welchem die Tabelle angewendet wird. Damit eine Tabelle im Knowledge-Builder angezeigt wird, muss hier die Anwendung "Knowledge-Builder" gewählt werden.
Verwendung	Im Abschnitt "Verwendung" zeigt die Relation "Kontext von" an, für welche View das aktuelle Element als Anwendungskontext verwendet wird. Diese Relation ist die Gegenrichtung der Relation "anwenden in", welche zum jeweiligen anderen Viewconfig-Element verweist.
Tabelle von	Zeigt das übergeordnete Viewkonfigurations-Element an, in dem die Tabelle verwendet wird.

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.

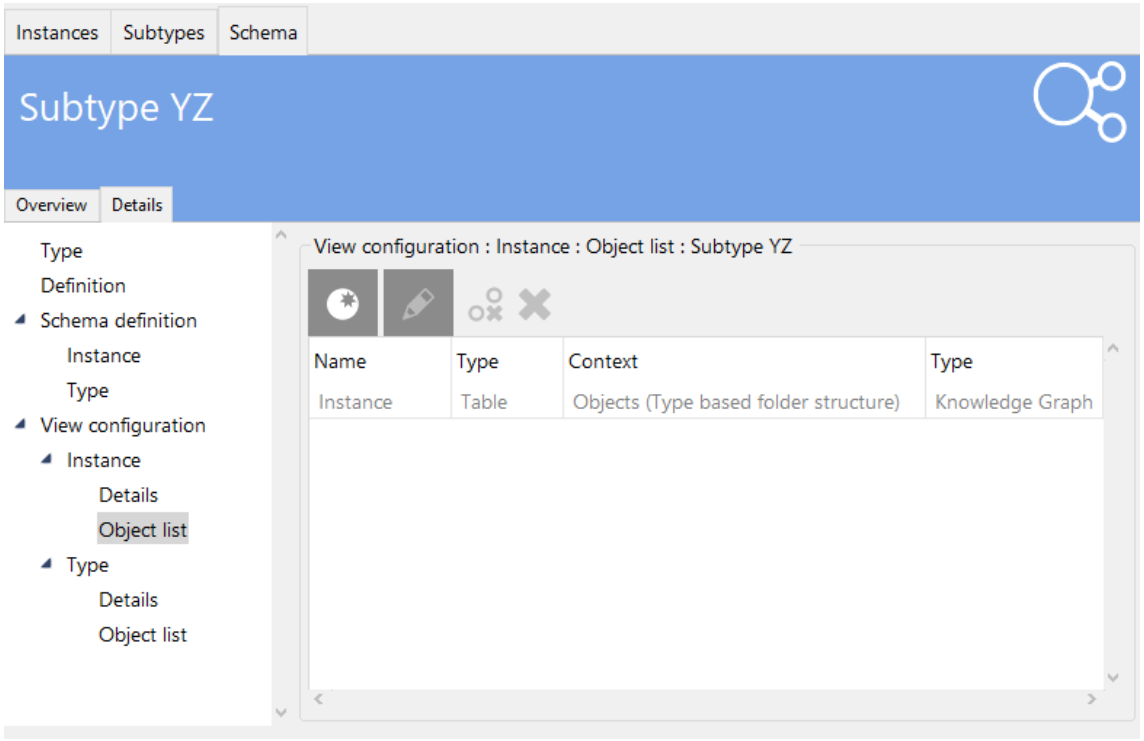
Beispiele:

- 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 / Objektlisten 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 von *Untertyp YZ* gezeigt.



Noch wurde keine Tabellenkonfiguration mit diesem Typ verknüpft. Der ausgegraute Eintrag zeigt, dass eine Standard-Konfiguration in Verwendung ist, welche vom obersten Typ "Knowledge Graph" stammt und vererbt wird. Durch Klicken auf den Knopf *Neu*  wird eine neue, leere Konfiguration erzeugt. Diese kann dann selektiert und wie benötigt bearbeitet werden. Sobald der Anwendungskontext bestimmt wurde (z. B. "anwenden in: Knowledge-Builder"), kann die Konfiguration nach dem Aktualisieren der View-Konfiguration verwendet werden.

1.3.4.8.1. Spaltenkonfiguration

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
Konfiguration	
Konfigurationsname	Der Konfigurationsname dient zur Identifizierung und Wiederverwendung von Konfigurationen.
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.
Skript für Beschriftung	Als Alternative zum statischen Beschriftungstext kann ein Skript verwendet werden, welches eine Zeichenkette zurückgibt.
Bookmark Identifikator	Der Bookmark-Identifikator wird verwendet, um Suchabfrageparameter in Form eines Teilausdrucks der Web-Frontend URL zu repräsentieren. Der Identifikator kann dazu verwendet werden, um Views und Tabellenspaltenfilter abzufragen und um Parameterwerte und die URL in beide Richtungen zu synchronisieren.
Breite der Spalte (%)	Hier wird als Eingabge eine Ganzzahl erwartet, um den prozentualen Anteil der Spaltenbreite im Vergleich zur Tabellenbreite zu bestimmen (für eine Spaltenbreite von 60 % muss der Zahlenwert *60* eingegeben werden).
Standard-Operator	Der Operator, welcher initial in der Suche für einen Suchtext verwendet wird.
Suchtext	Voreingestellter Suchtext für den Spaltenfilter.
Nicht anzeigen	Wenn dieser Wert gesetzt ist, wird die komplette Spalte ausgeblendet. Diese Option wird beispielsweise dazu verwendet, um eine Tabelle nach Qualitätswerten zu sortieren, diese aber nicht anzuzeigen.
Obligatorisch für Abfrage	Wenn dieser Wert gesetzt ist, muss der Spaltenfilter zuerst ausgefüllt sein, damit die Suche ausführbar ist.
Nicht sortierbar	Unterdrückt das Sortieren der Tabelle, wenn auf den Tabellenspalten-Kopf geklickt wird.
Skript für Vorverarbeitung von Eingabefeldern	Um die Eingabe des Textes in den Tabellenfilter vorzuverarbeiten, bevor er als Parameter an die Spaltenelement-Abfrage weitergeleitet wird, kann ein Skript verwendet werden.
Mapping-Element	.
Operators	

Name	Wert
Konfigurationsname	Der Konfigurationsname dient zur Identifizierung und Wiederverwendung von Konfigurationen.
Icon	.
Key	.
Label	.
Modifier	.

Menüs

Für die Spalte kann ein Menü konfiguriert werden, welches im Web-Frontend neben dem Beschriftungstext der Spalte angezeigt wird.

Styles

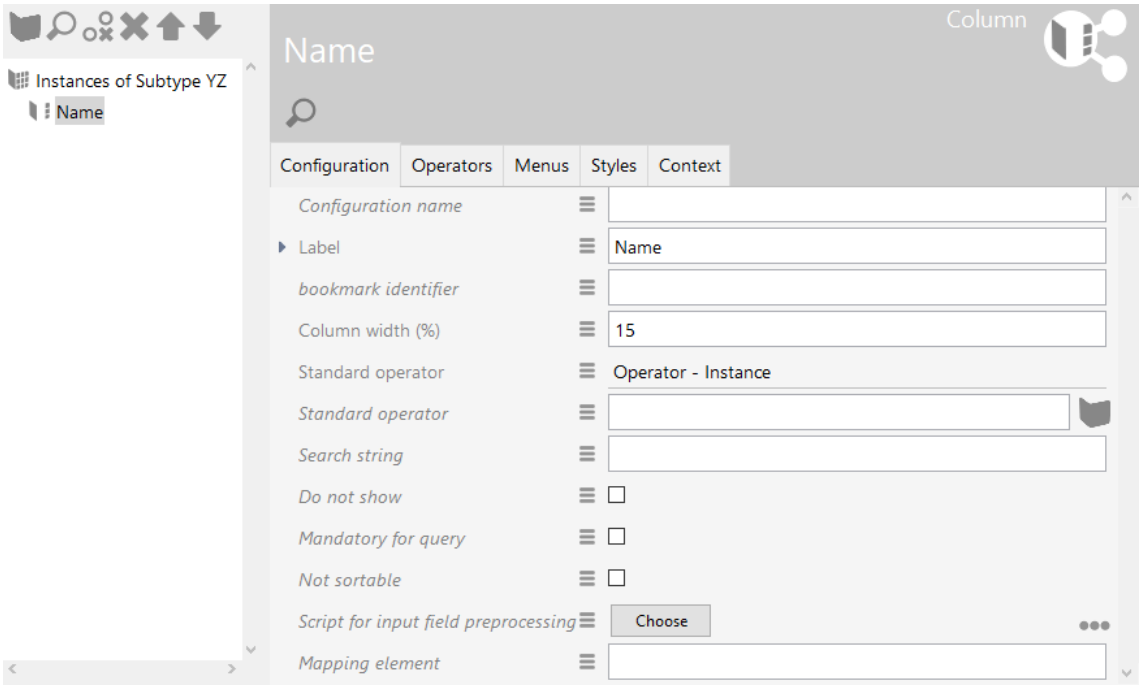
Für Spalten gibt es folgende Style-Einstellungen, welche im Viewconfigmapper angewendet werden können:

hideFilters	Unterdrückt die Anzeige der Spaltenfilter im Web-Frontend.
hideLabel	Unterdrückt die Anzeige der Spaltenbeschriftung im Web-Frontend.

Kontext

Sub-Konfiguration von	Zeigt an, innerhalb welcher Tabellenkonfiguration die Spalte verwendet wird.
Reihenfolge	Spiegelt die Position der Spaltenkonfiguration innerhalb der Tabellenkonfiguration wieder. Wenn mehr als eine Spalte mit derselben Reihenfolge spezifiziert wird, dann werden diese Spalten alphabetisch anhand ihrer Spaltenbeschriftung sortiert.
Sortierte Spalte von	Zeigt an, dass die Spalte zur Sortierung des Tabelleninhaltes verwendet wird.
Sortierpriorität	Spezifiziert die Rangfolge der zur Sortierung eingesetzten Spalte.

Beispiel



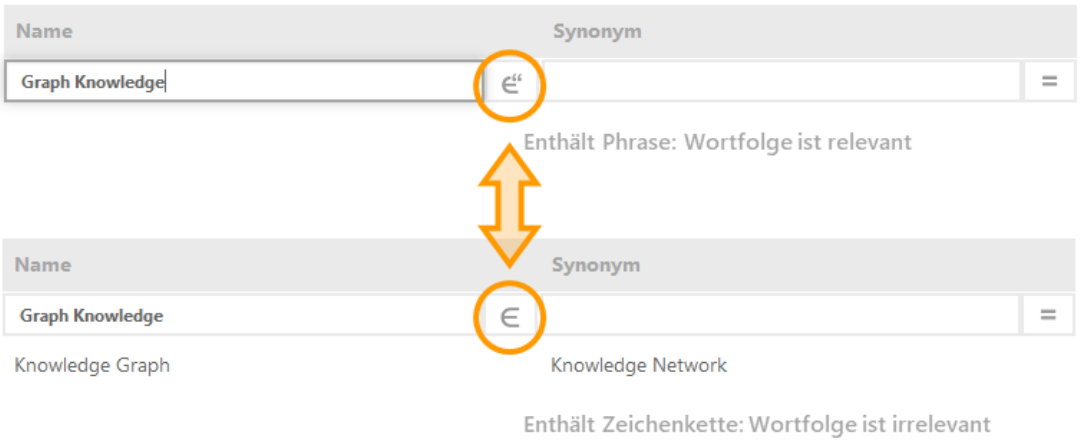
Spaltenkonfiguration für die Spalte "Name"

1.3.4.8.2. Spaltenoperator

Der Spaltenoperator legt fest, welcher Vergleichsoperator in der Tabellen-Ansicht verwendet werden kann, wenn ein Begriff in den Spaltenfilter eingegeben wird. In den meisten Fällen werden Operatoren wie bspw. "Gleich", "Enthält Phrase" oder "Enthält Zeichenkette" benötigt.

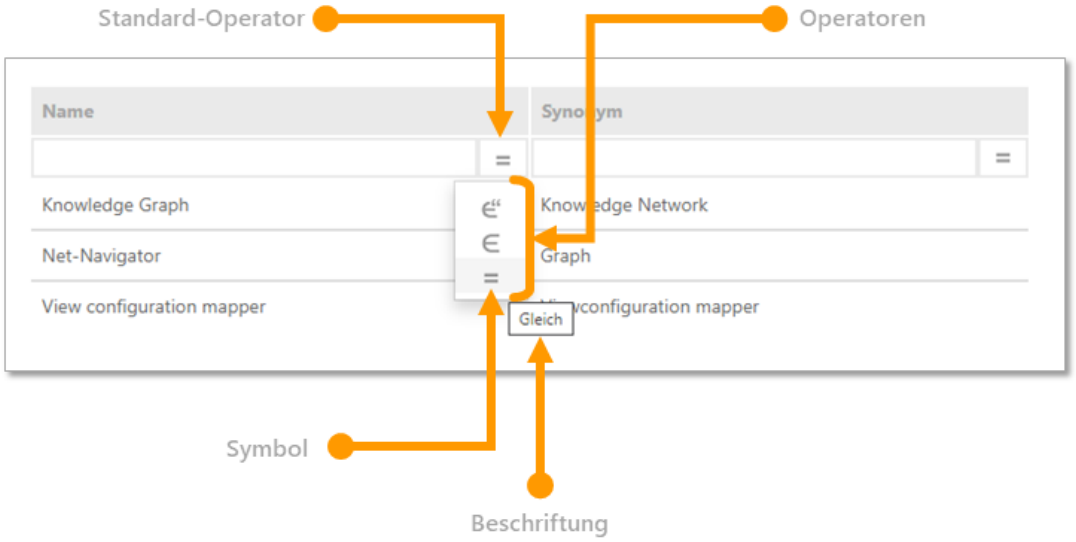
Der Unterschied zwischen "Enthält Phrase" und "Enthält Zeichenkette" ist beispielsweise wie folgt:

- "**Enthält Phrase**": Wenn mehrere Wörter (= Phrase) im Filter eingegeben werden, dann wird nur der Inhalt mit exakt derselben Wortfolge aufgefunden
- "**Enthält Zeichenkette**": Wenn mehrere Wörter im Filter eingegeben werden, dann wird der Inhalt mit den gleichen Worten aufgefunden, jedoch unabhängig von der Wortfolge

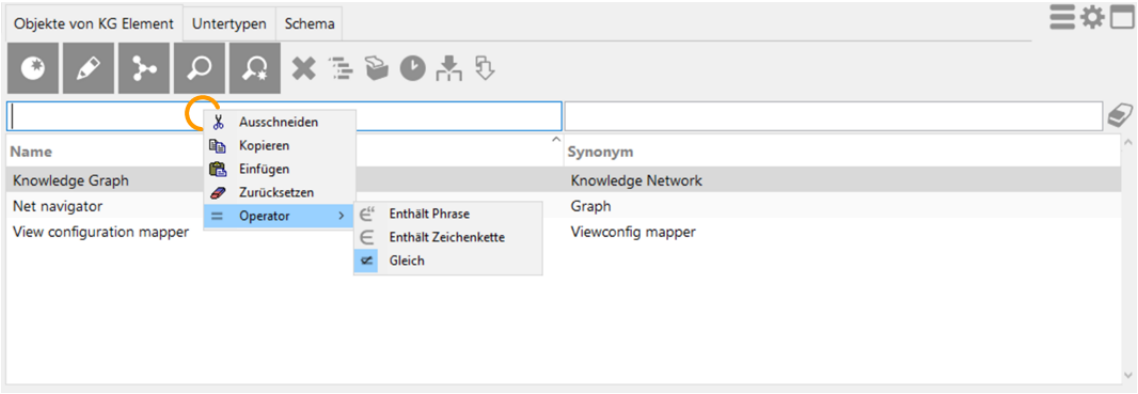


Dies ermöglicht die Verwendung unterschiedlicher Filterverhalten, um Tabellen mit großen Suchergebnismengen auf einen bestimmten Inhalt einzugrenzen.

Die Filteroperatoren sind dann in einer Dropdown-Auswahl der jeweiligen Spalte verfügbar:



Wenn die Tabelle im Knowledge-Builder angewendet wird, dann bietet ein Kontextmenü Einträge zum Anwenden und zum Entfernen von Operatoren:



Anlegen neuer Spaltenoperatoren

Neue Spaltenoperatoren werden wie folgt angelegt:

Voraussetzung: Für das Spaltenelement der Spalte wurde eine Eigenschaft definiert.

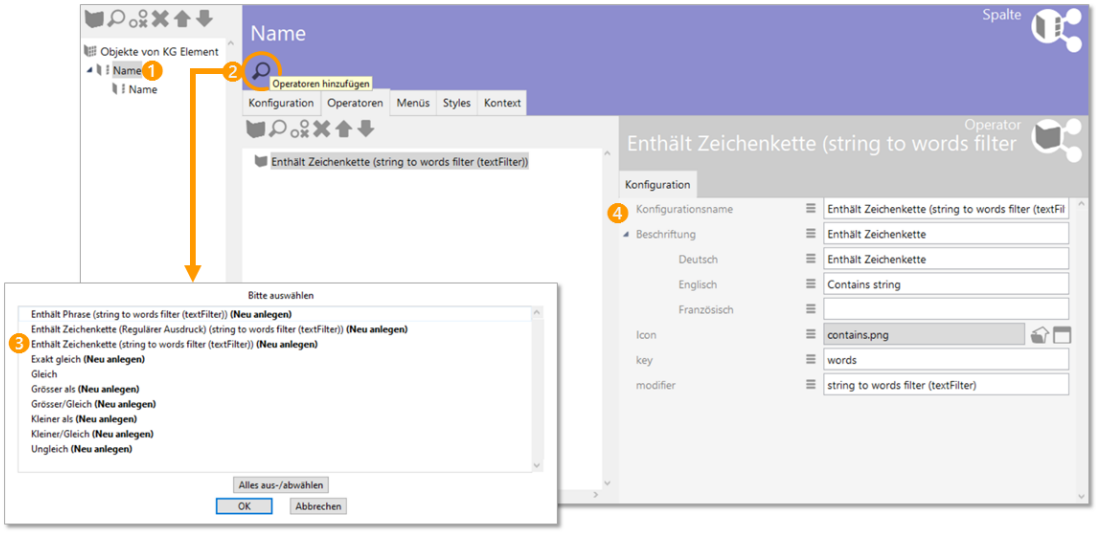
HINWEIS	Da die Anwendbarkeit von Operatoren vom Wertetyp der zu filternden Eigenschaft abhängt, sind die vordefinierten Spaltenoperatoren erst verfügbar, wenn zuvor die Eigenschaft des Spaltenelements definiert wurde.

❶ Nach dem Definieren der Eigenschaft des Spaltenelements die zugehörige Spalte anwählen.

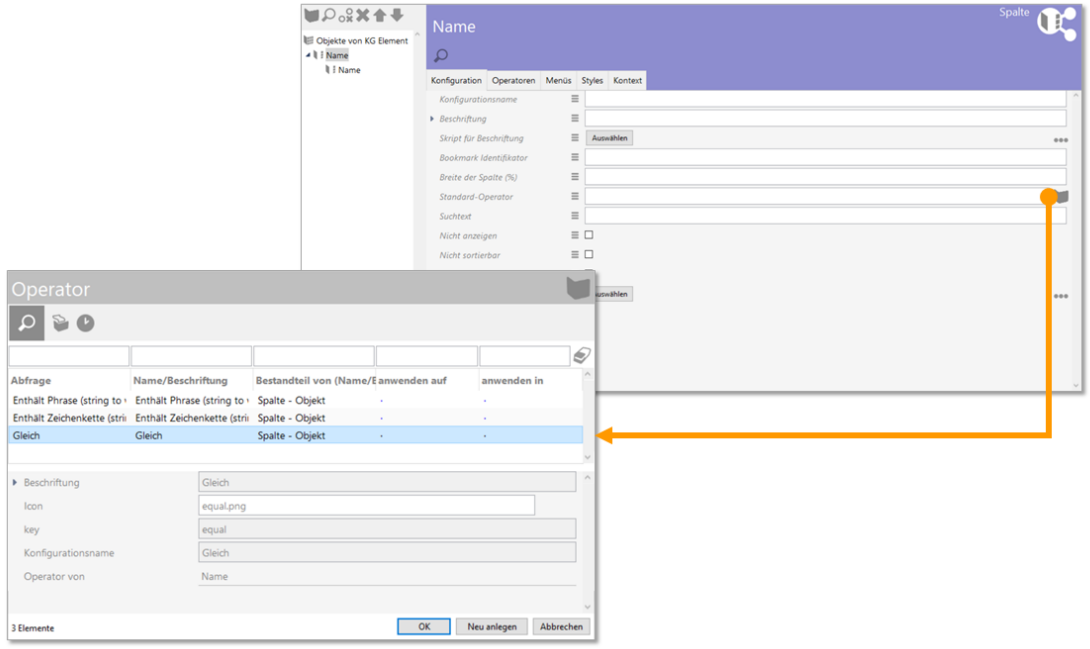
❷ Auf die Such-Schaltfläche klicken: Es wird eine Auswahl an Operator-Vorlagen aufgelistet, welche auf den Wertetyp der Eigenschaft anwendbar ist. Operatoren mit dem Zusatz "Neu anlegen" weisen darauf hin, dass sie noch nicht in Verwendung sind (d. h. aus der Vorlage wurde noch keine Instanz erzeugt).

❸ Benötigten Operator auswählen.

❹ Der Reiter "Operator" enthält die neu erzeugten und zugewiesenen Operatoren. Jeder hier aufgelistete Operator wird im Spaltenfilter der Spalte in der Tabelle verfügbar sein. Bereits erzeugte Operatoren können für andere Tabellenspalten wiederverwendet werden.



❺ Für das Festlegen eines voreingestellten Operators zum Reiter "Konfiguration" wechseln und unter "Standard-Operator" einen der Operatoren auswählen:



HINWEIS

Bei Verwendung von Spaltenoperatoren in Tabellen des Knowledge-Builders wird der Standard-Operator "Gleich" nicht explizit angezeigt. Dieser wird jedoch angewendet, solange kein anderer Operator im Kontextmenü ausgewählt wurde.

Operatoren können auch ohne die Verwendung einer Vorlage definiert werden. Hierfür können folgende Eigenschaften festgelegt werden:

Eigenschaft	Beschreibung	Wertetyp
Konfigurationsname	Der Konfigurationsname dient zur Identifizierung und Wiederverwendung eines Konfigurationselements.	Zeichenkette
Icon	Das Icon wird für die Dropdown-Auswahl im Spaltenfilter benötigt.	Blob (Datei)
	HINWEIS Ohne zusätzliche Plugins können für Konfigurationselemente im Knowledge-Builder keine Vektorgrafiken wie bspw. *.svg verwendet werden.	
key	Der Operator-Key für den Operator. Siehe nachfolgende Tabelle.	Zeichenkette
Beschriftung	Text für den Tooltip, welcher bei Mouse-Over am Symbol mit angezeigt wird.	Zeichenkette
modifier	Filterbezeichner des Index-Zeichenketten-Filters.	Zeichenkette

Operator-Keys

Operator-Name	Beschreibung	Kurzbezeichnung
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	

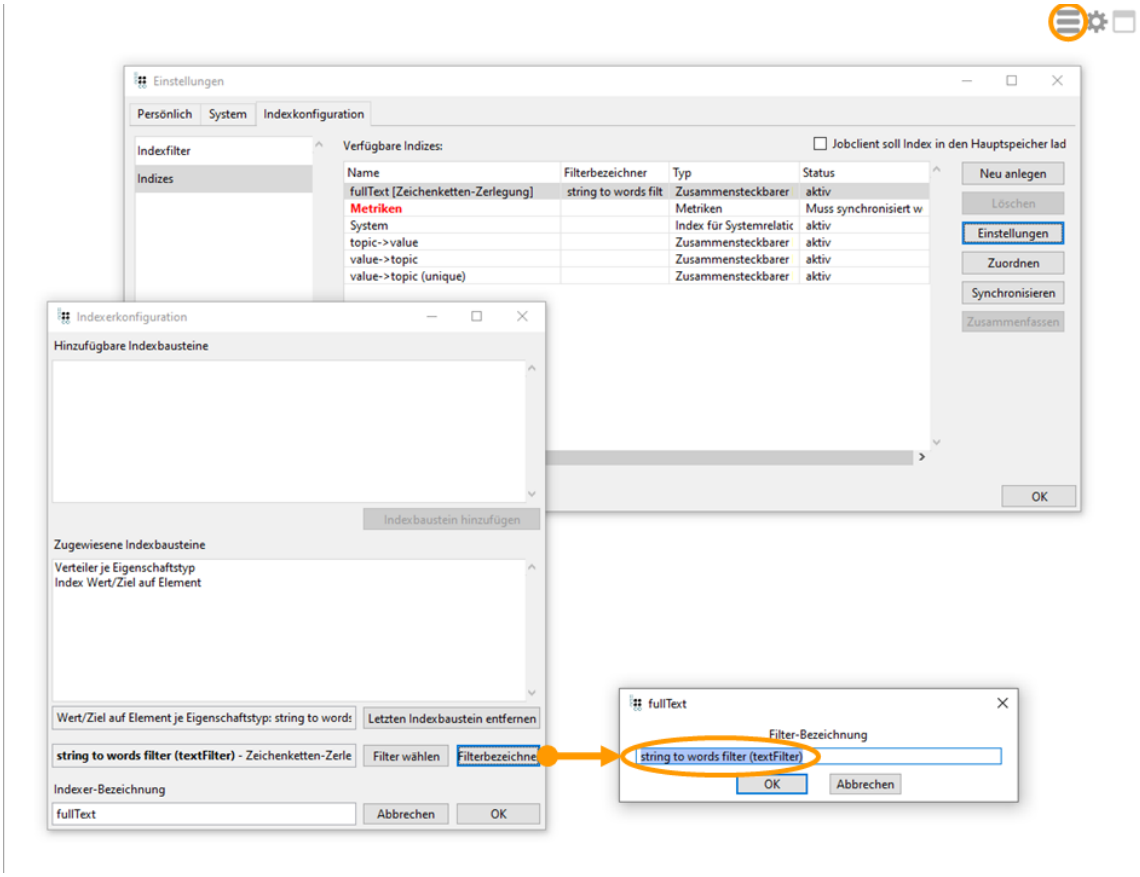
Operator-Name	Beschreibung	Kurzbezeichnung
greaterPresentTime	nach jetzt (Zukunft)	
isCoveredBy	ist enthalten in	
less	Kleiner als	<
lessOrEqual	Kleiner/Gleich	⍰
lessOverlaps	überschneidet von unten	
lessPresentTime	vor jetzt (Vergangenheit)	
notEqual	Ungleich	!=
overlaps	überschneidet	
range	Zwischen	
regexEqual	Regulärer Ausdruck	
regexFulltext	Enthält Zeichenkette (Regulärer Ausdruck)	
unmodifiedEqual	Exakt gleich	
words	Enthält Zeichenkette	

Modifier

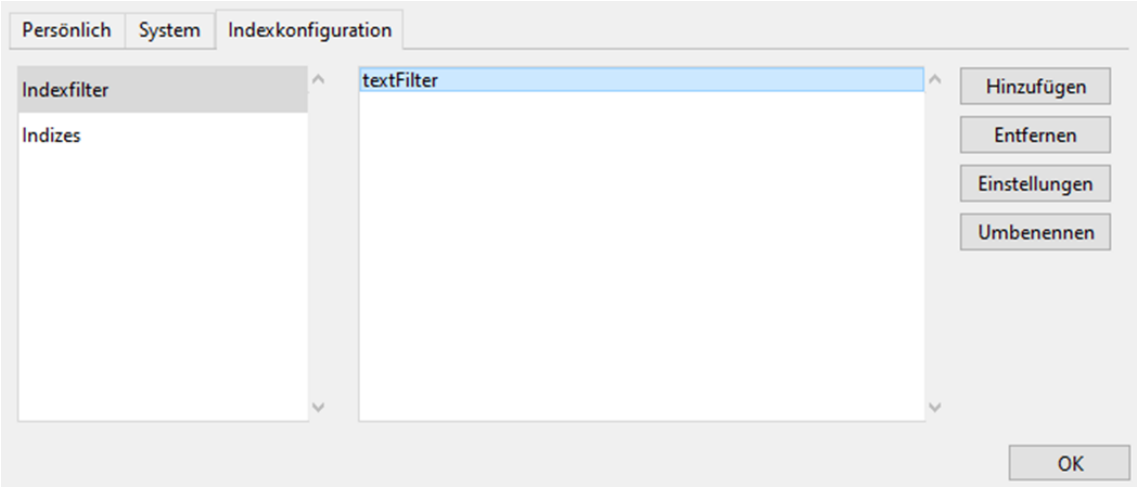
Damit Operatoren wie "Enthält Phrase" überhaupt angewendet werden können, wird bei Verwendung des entsprechenden Operator-Keys "containsPhrase" ein Modifier benötigt. Der Modifier entspricht dem Filterbezeichner des in den Einstellungen konfigurierten Indexfilters.

Der Indexfilter wird in Zusammenhang mit einem Index verwendet. Indizes werden in den globalen Einstellungen des Knowledge-Builders verwaltet: Einstellungen > Indexkonfiguration.

In den Einstellungen des Indexes kann der Filterbezeichner festgelegt und für die Angabe des Modifiers herauskopiert werden:



Neue Indexfilter können in den globalen Einstellungen des Knowledge-Builders angelegt werden: Einstellungen > Indexkonfiguration > Indexfilter



1.3.4.8.3. Spaltenelement

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

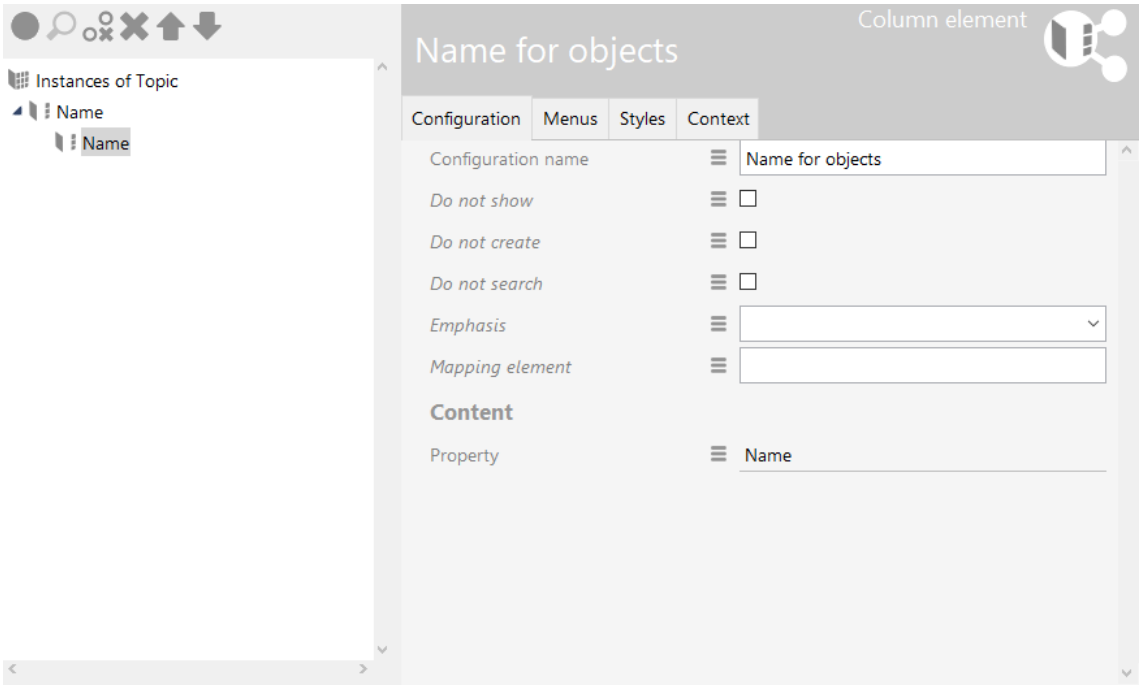
Name	Wert
Konfiguration	
Konfigurationsname	Der Konfigurationsname dient zur Identifizierung und Wiederverwendung von Konfigurationen.
Nicht anzeigen	Über dieses boolesche Attribut kann gesteuert werden, ob Werte der ausgewählten Eigenschaft angezeigt werden sollen. Standardmäßig werden alle Eigenschaften 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*.
Relationszielansicht	Bis auf weiteres ist hier nur die Option "Drop down" verfügbar. Wenn aktiviert, kann im Spaltenfilter die Einschränkung der Suchergebnisse auf ein bestimmtes Relationsziel per Drop-Down Menü ausgewählt werden. Dies ist bei einer überschaubaren Anzahl von Relationszielen zu empfehlen. **HINWEIS** Dieser Parameter ist nur dann verfügbar, wenn als Eigenschaft ein Relationstyp verwendet wird.
Inhalt	Eigenschaften in dieser Gruppe bestimmen den Tabellenzelleninhalt. Die meisten der folgenden Optionen schließen sich gegenseitig aus, angezeigt durch einen *.
Eigenschaft*	Verknüpfung zu einem Eigenschaftstyp, der angezeigt werden soll.
Strukturabfrage-Baustein*	Anstatt einer Eigenschaft kann eine Strukturabfrage dazu verwendet werden, die anzuzeigende Eigenschaft zu ermitteln.
Skript*	Anstatt einer Eigenschaft kann ein Skript verwendet werden, welches den anzuzeigenden Wert bestimmt (eine Eigenschaft, ein Hit, Objekt oder primitiver Wert).

Name	Wert
Mapping element*	.
Namen anzeigen*	Zeigt den Namen des Zeilenelements an, unabhängig davon, welcher Attributtyp als Name definiert ist.
Qualität*	Für das Web-Frontend erlaubt diese Option das Darstellen eines "Fortschrittsbalkens", welcher die Hit-Qualität inklusive eines Prozentwerts anzeigt. **HINWEIS** Diese Option wird anstatt einer Eigenschaft verwendet und erfordert für die Anzeige das Aktiveren der Option "Hits verwenden", da nur Hits einen Qualitätswert transportieren.
Anzahl anzeigen	Statt der über eine der oben genannten Methoden bestimmten Eigenschaften wird nur die Anzahl dieser Eigenschaften angezeigt.
Strukturrelation verwenden	Verändert alle oben genannten Methoden zur Eigenschaftsbestimmung derart, dass sie sich auf die Strukturrelation beziehen, die von dieser eingebetteten Tabelle definiert ist, anstatt auf deren Relationsziele (zu Strukturrelationen siehe Abschnitt zu allgemeiner Tabellenkonfiguration).

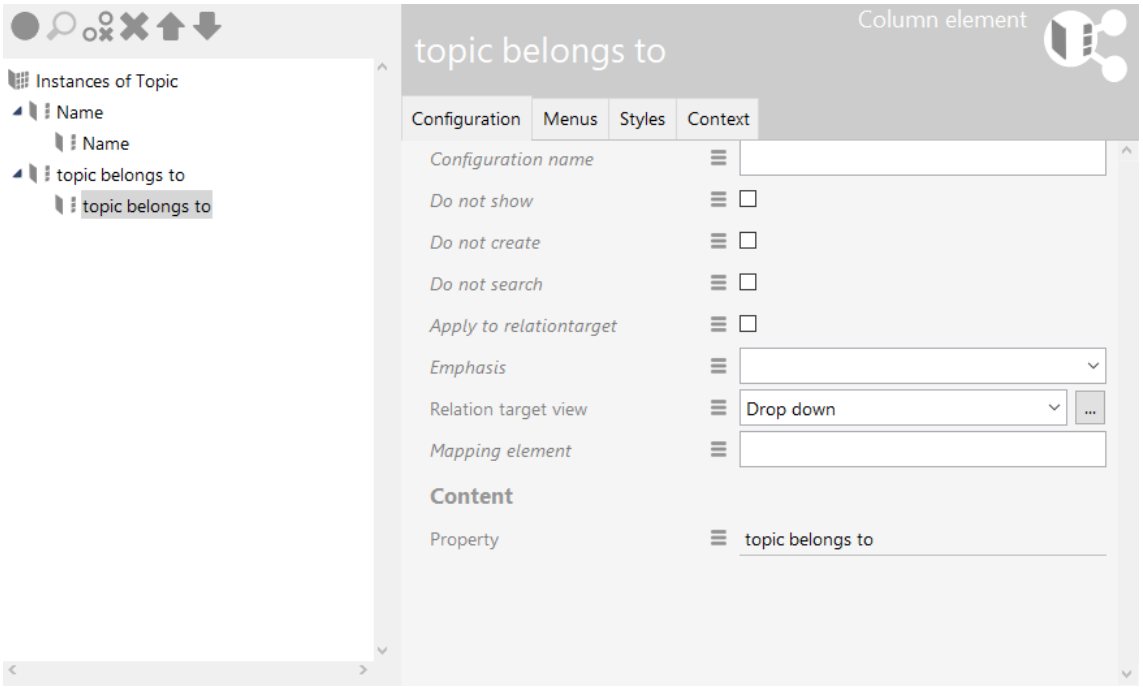
Es ist möglich für eine Spaltenkonfiguration mehrere Spaltenelemente zu definieren. Das ist z.B. dann sinnvoll, wenn mehrere Attribute in der Suche berücksichtigt werden sollen wie beispielsweise das Attribut Name und Synonym, aber nur eines davon angezeigt werden soll.

Beispiel

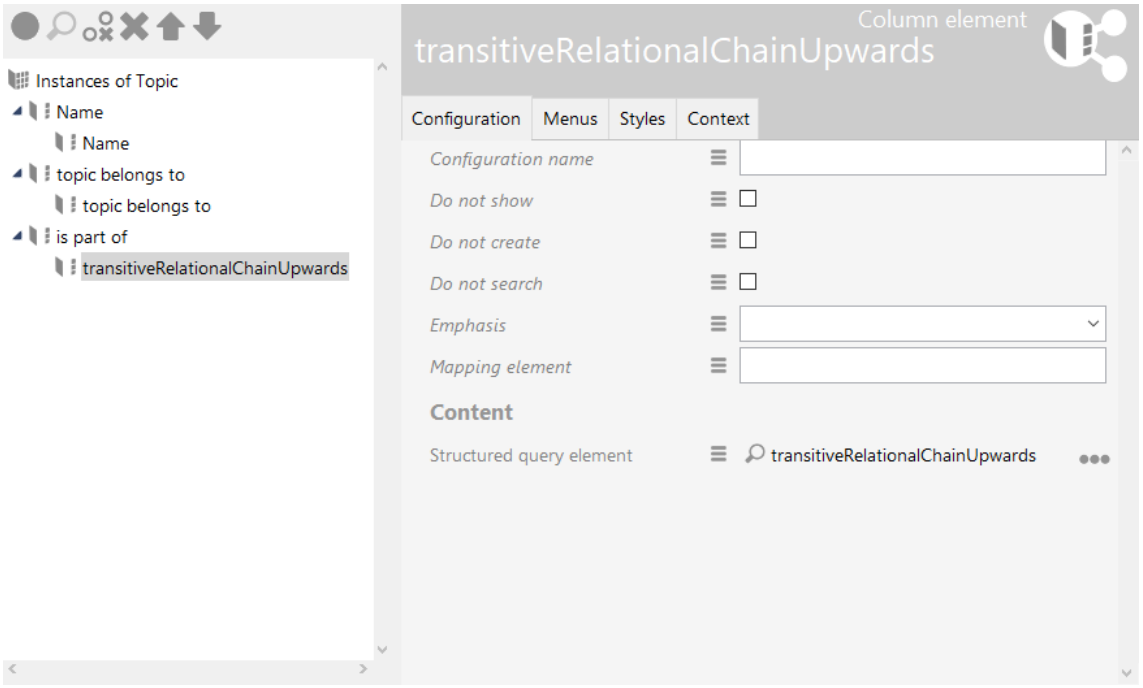
Im ersten Spaltenelement der Spaltenkonfiguration *Name* wurde das Attribut *Name* hinterlegt.



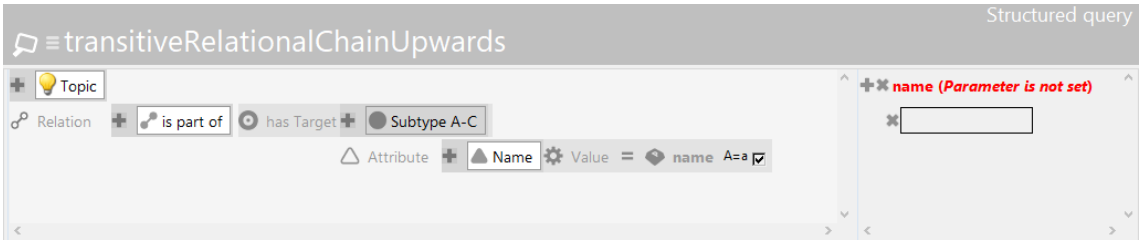
Auf der zweiten Spalte wurde im Spaltenelement die Beziehung *ist Thema von* hinterlegt.



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).

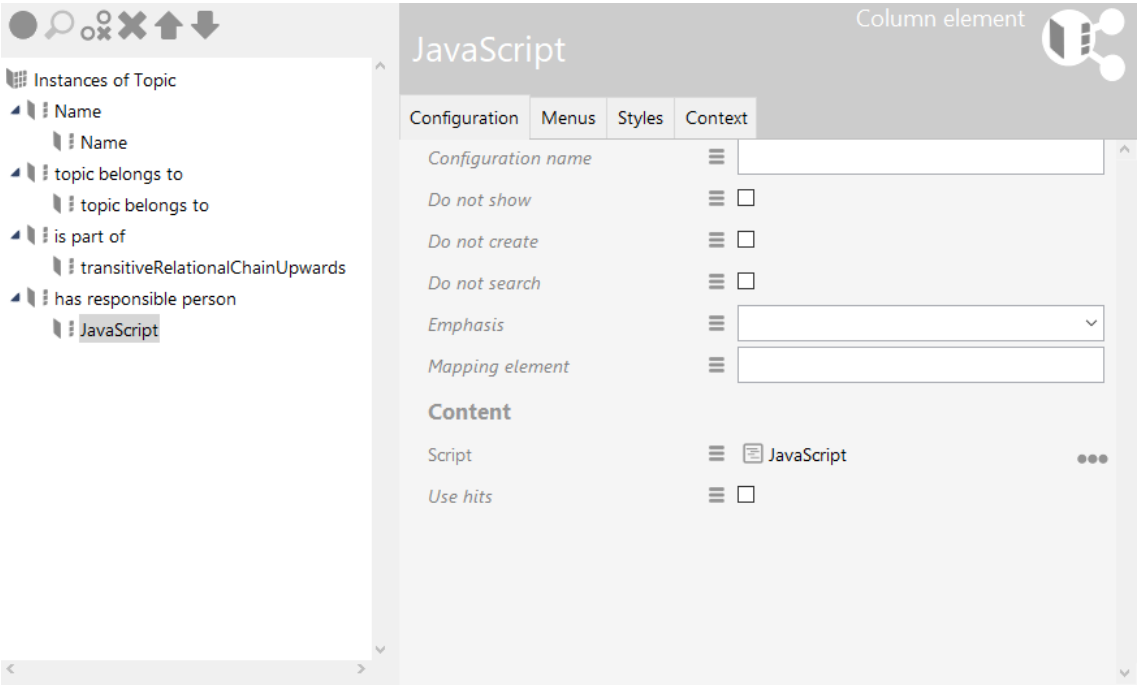
Der in der Spalte gezeigte Wert ist, wenn keine weiteren Anpassungen vorgenommen werden, der Wert des zum Filtern verwendeten Attributes. Wenn sich der angezeigte Wert nicht aus dem zur Filterung verwendeten Attribut ergibt, so gibt es zwei Möglichkeiten:

- 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 column search value "objectListArgument"
 * Note: "elements" may be undefined if no partial query result is
available.
 * Return undefined if the script cannot provide any partial result
itself.
 */
function filter(elements, queryParameters, objectListArgument) {
  return elements;
}
```

```
// Returns cell values rendered as topics for the given element
// For cell values rendered as Hits, use renderHits() instead
function renderElements(element, queryParameters) {
  var result = new Array()
  var firstTargets = element.relationTargets("isTopicOf")
  if ( firstTargets.length == 0 )
     return result;
  else {
     for (var i = 0; i < firstTargets.length; i++) {
         var secondTargets = firstTargets[i].relationTargets(
"hasResponsiblePerson")
         for (var j = 0; j < secondTargets.length; j++)
             result.push(secondTargets[j].name())
         }
     }
  return result.join(", ");
}
```

In 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 beiden 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.

1.3.4.9. Suche

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 eines *Layouts* ausgewählt werden. Eine beliebige Abfrage ist hier obligatorisch. Auch Suchen zur Benutzereingabe können konfiguriert werden; für die View-Konfiguration wird hier anstatt dem Konfigurationselement "Suche" (Objektkonfiguration) das Konfigurationselement "Suchfeld-Ansicht" verwendet.

Beispiele und wichtige Hinweise für die Verwendung von Suchen, Suchfeld-Ansichten, Facetten und Suchergebnis-Ansichten im Web-Frontend finden sich in Kapitel 3 "ViewConfig-Mapper".

Wenn die Suche für ein Web-Frontend mit Facetten konfiguriert werden soll, dann ist folgende Wirkungskette einzuhalten: Suche *oder* Suchfeld-Ansicht > Facetten > Suchergebnis.

Einstellungsmöglichkeiten

Name	Wert
Konfigurationsname	Der Konfigurationsname dient zur Identifizierung und Wiederverwendung von Konfigurationen.
Beschriftung	Eine Beschriftung findet nur ihre Anwendung, wenn diese Konfiguration in einer anderen Konfiguration wie z.B. *Alternative* eingebettet ist.
Skript für Beschriftung	Die Beschriftung lässt sich alternativ auch über ein Skript ermitteln.
Bookmark identifier	Der Bookmark identifier dient zur Repräsentation von Such-Parametern in Form eines Ausdruckes innerhalb der Web-Frontend URL. Der Identifier kann für Suchansichten und Tabellenspalten-Filter verwendet werden und bewirkt eine gegenseitige Synchronisation von Parameterwerten und URL.
Tabelle	Hier wird eine Tabellenkonfiguration angegeben, die zur Darstellung des Suchergebnisses dient.
Skript für Tabellenkonfiguration	Die Tabelle kann auch über ein Skript ermittelt werden.
Suche	Hier wird die Suchabfrage ausgewählt, welche ausgeführt wird, sobald das Konfigurationselement angezeigt wird. Das semantische Object, für welches das View-Konfigurationselement angezeigt wird, steht als Zugriffselement für die Suche zur Verfügung.
Skript für Sichtbarkeit	Skript, das die Sichtbarkeit des Elements durch Rückgabe eines Booleschen Wertes steuert.
Hits verwenden	Wenn deaktiviert: Liefert Treffer in Form von semantischen Elementen zurück. Wenn aktiviert: Liefert Treffer in Form von Hits zurück, damit beispielsweise die Trefferqualität angezeigt werden kann.

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.

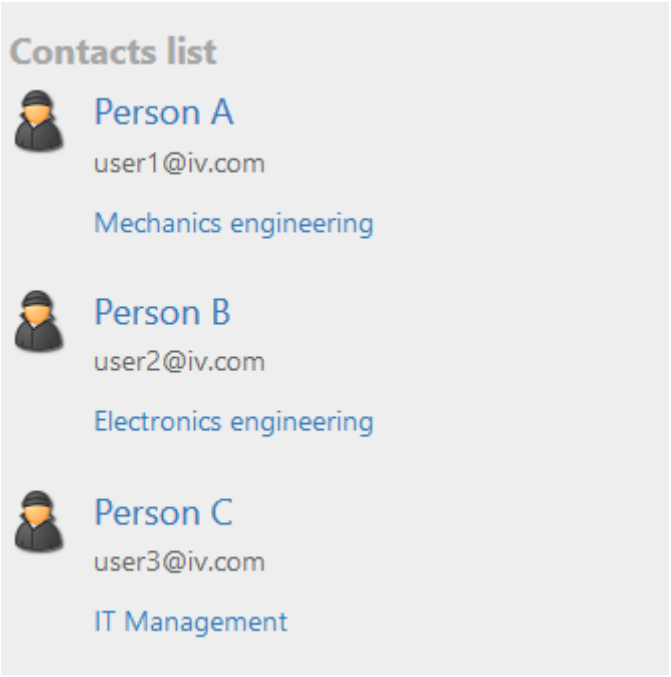
Einstellungsmöglichkeiten für einen Parameternamen

Die folgenden Parameter werden als Meta-Eigenschaften für einen *Parameternamen* gepflegt:

Name	Value
Skript für Wertermittlung	Das Skript *parameterValue* wird zur Ermittlung des Suchwertes für den angegebenen Parameternamen verwendet.
Skript für geparsten Wert	
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.
Value disposition	.
Typ	xsd-typ
Beschriftung	Beim Herausschreiben nach JSON landet dieser Wert in *label*.
Bookmark identifier	.
Tooltip	.
Query for proposed values	.
Script for proposed values	.
Sort Order	.

Darstellung in einer Anwendung

Die Suchergebnisse werden in einer Tabelle ausgegeben. Im Web-Frontend kann für die Tabellen-Darstellung noch ein Render-Mode verwendet werden.



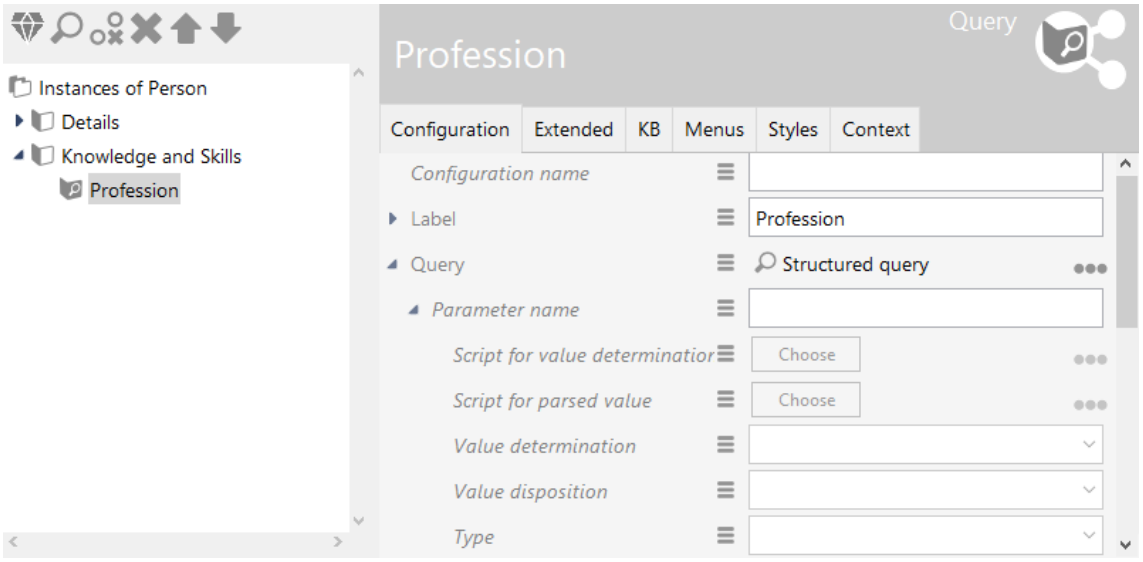
In diesem Beispiel werden die Suchergebnisse im Web-Frontend in Form einer Tabelle mit dem Render-Mode "mediaList" angezeigt. Der "mediaList" Render-Mode konvertiert beispielsweise die Tabelle in eine Ansicht mit Icons und Verlinkungen zum Objekt. Zusätzliche Eigenschaften der Objekte können mithilfe weiterer Spalten-Elemente spezifiziert werden (in diesem Fall sind es die Mailadresse als Attribut und die Spezialisierung als Relationsziel).

Tipp: Anstatt der Verwendung des Konfigurationselements "Suche" können Suchen auch in die separate Konfigurationselemente "Suchfeld-Ansicht" und "Suchergebnis-Ansicht" aufgeteilt werden, sodass eine separate Darstellung von Eingabe und Ausgabe im Web-Frontend ermöglicht wird.

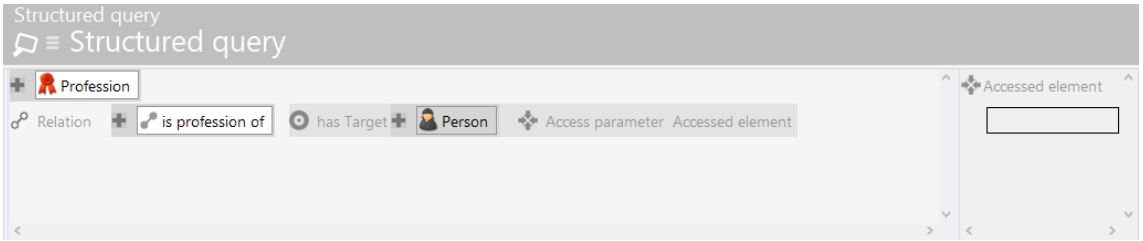
Darstellung im Knowledge-Builder

Die Ergebnisse werden immer in einer Objektliste im Knowledge-Builder angezeigt.

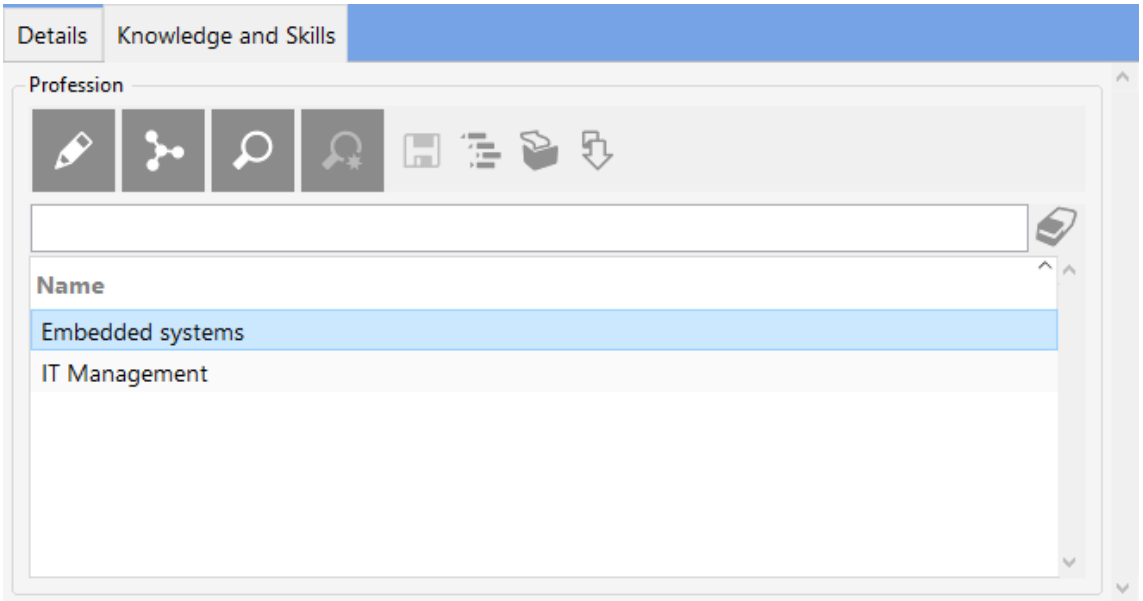
Beispiel:



Die Reiter zu "Details" und "Wissen und Begabungen" ("Knowledge and Skills") werden in der View-Konfiguration definiert. "Spezialisierung" ist ein Konfigurationselement des Typs "Suche". Hierfür kann eine existierende Suche wiederverwendet werden oder es kann eine neue Suche angelegt werden anhand des Felds "Abfrage" ("Query").



Definition der Abfrage



Das Ergebnis der Suche wird im Reiter "Wissen und Begabungen" ("Knowledge and Skills") im Knowledge-Builder für den Typ "Person" angezeigt.

1.3.4.10. 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.

1.3.4.10.1. 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.

Name	Wert
maximale Pfadlänge	Gibt die maximale Pfadlänge eines Knotens zum zuletzt ausgeklappten Knoten an, ab der dieser Knoten ausgeblendet wird. Standardwert: 5Es können beliebig viele Ausklapp-Aktionen getätigt werden, hierbei wird nur die Pfadlänge beachtet.Beispiel:Bei einer *maximale Pfadlänge* von 2, wird der Knoten **C** des Graphs **A** - **B** - **C** beim Aufklappen von **A** ausgeblendet (falls **A** keine direkte Verbindung mit **C** hat) **HINWEIS** Ist nur die maximale Pfadlänge gesetzt, so gilt trotzdem weiterhin der Standardwert 5 für die Schritte bis zur Knotenausblendung.
Schritte bis Knotenausblendung	Die Anzahl der Aktionen (Ausklappen), ab der Knoten ausgeblendet werden. Dabei zählen die Aktionen, die nicht zum Einblenden dieses Knotens geführt haben. Standardwert: 5 Beispiel: Bei 2 *Schritte bis Knotenausblendung* erfolgen diese Aktionen: 1. Im Graph **A** wird **A** ausgeklappt. Hinzu kommt Knoten **B** 2. Im daraus resultierenden Graph **A** - **B** wird **B** ausgeklappt. Hinzu kommt Knoten **C** 3. Im daraus resultierenden Graph **A** - **B-C** wird **C** ausgeklappt. **A** wird jetzt ausgeblendet. **HINWEIS** Sind nur die Schritte bis zur Knotenausblendung gesetzt, so gilt trotzdem weiterhin der Standardwert 5 für die maximale Pfadlänge.

1.3.4.10.2. Knotenkategorie

Einer Graph-Konfiguration werden Knotenkategorien untergeordnet. Den Knoten-Kategorien werden Verknüpfungselemente untergeordnet.

Einstellungsmöglichkeiten

Name	Wert
Konfiguration	
Konfigurationsname	Der Konfigurationsname dient zur Identifizierung und Wiederverwendung von Konfigurationen.

Name	Wert
Beschriftung	Beschriftung, welche für die Legende der Knoten im Web-Frontend verwendet wird. **HINWEIS** Diese Option hat keine Auswirkung bei Anwendung des Graphen im Knowledge-Builder.
Skript für Beschriftung	Skript, welches einen Elementnamen oder eine Zeichenkette für die Beschriftung ausgibt, anstatt das Beschriftungs-Attribut zu verwenden.
An konkreten Typ anpassen	Wenn diese Option aktiviert ist, werden nur die konkreten Untertypen als Legende angezeigt anstatt des übergeordneten Typen.
Abstrakte Typen ausblenden	Diese Option verhindert die Anzeige von abstrakten Typen in der Legende.
In Legende anzeigen	Diese Option ist nur für den Net-Navigator im Web-Frontend verfügbar: • Bei Bedarf: Die Legende am oberen Rand des Graphen wird nur dann angezeigt, wenn der Knoten auch im Graph existiert (angezeigt wird). • Immer: Die Legende wird angezeigt, ungeachtet der Existenz angezeigter Knoten. • Nie: Die Legende wird nie angezeigt, auch wenn der betreffende Knoten im Graph dargestellt wird.
Icon	Icon, welches exklusiv für die Knotenkategorie im Graph angezeigt wird. Wenn kein Icon festgelegt wurde, dann wird das (vererbte) Icon des semantischen Elements des Knowledge-Graphen angezeigt. Wenn auch im Knowledge-Graph kein Icon hinterlegt ist, werden Typen in Form eines eingefärbten Rings und Objekte in Form eines eingefärbten, ausgefüllten Kreises dargestellt.
Skript für Icon	Skript, welches ein Icon für die Knotenkategorie anstatt des Icon-Attributes ausgibt. Rückgabewert ist ein Blob-Attribut oder ein Wert vom Typ $k.Blob
Erweiterungen initial anzeigen	Wenn aktiviert, werden Erweiterungen initial ausgeklappt dargestellt, sobald das Kernobjekt im Graph dargestellt wird.
Farbe	Farbe, welche der Knotenkategorie zugeordnet wird. Dies betrifft die Einfärbung der Knoten-Kreise und der Legende.

Name	Wert
Skript für Farbe	Skript, welches die Farbe für den Knoten ausgibt, anstatt der Verwendung des Farbe-Attributs. Rückgabewert ist ein Hexadezimal-Farbwert.
Kategorie	
Menüs	.
Knoten	
Menüs	Wenn der Graph im Web-Frontend angezeigt wird (Net-Navigator), dann können Aktionen für die Knoten hinzugefügt werden wie folgt: • Satelliten-Menü der Knoten mit Standard-Aktionen zum Ausklappen, Ausblenden und Festpinnen von Knoten. • Aktion, welche ausgeführt wird, wenn auf den Knoten selbst geklickt wird. Für weiterführende Informationen siehe Kapitel *ViewConfig-Mapper > View-Konfigurationselemente > Graph-Konfiguration*.
Kontext	
Anwenden auf	Bestimmt, für welche Objekte oder Typen die Knotenkategorie angewendet wird. Eine Knotenkategorie kann für mehrere Objekte oder mehrere Typen verwendet werden.

1.3.4.10.3. Verknüpfung

Verknüpfungen werden einer Knotenkategorie untergeordnet. Sie repräsentieren die Kanten eines Graphen, also die Relationen des Knowledge-Graphen.

Einstellungsmöglichkeiten

Name	Wert
Konfigurationsname	Der Konfigurationsname dient zur Identifizierung und Wiederverwendung der Konfiguration.
Beschriftung	Das Beschriftungs-Attribut kann nicht für Verknüpfungen verwendet werden. Stattdessen ist hier das Skript für Beschriftung zu verwenden.
Skript für Beschriftung	Skript zur Ausgabe eines Elementnamens oder einer Zeichenkette für die Beschriftung der Verknüpfung.
Farbe	Bestimmt die Farbe der Verknüpfung.

Name	Wert
Abfrage für Verknüpfung	Abfrage, welche das Zielelement der Verknüpfung bestimmt, basierend auf dem Urpsrungselement, welches das übergeordnete Knotenelement ist.
Relation für Verknüpfung	Relationstyp des Knowledge-Graphen, welcher für die Bildung der Verknüpfung verwendet wird. Der Definitionsbereich des Relationstyps muss den Typ des jeweiligen Knotenelements umfassen.
Skript für Verknüpfung	Skript, welches die Verknüpfung definiert. Rückgabewert ist eine Relation am semantischen Element des Knotens.
Initial ausgeklappt	Wenn diese Option aktiviert ist, wird die Verknüpfung automatisch ausgeklappt, sobald das Knotenelement angezeigt wird.
Bevorzugt ausklappen	Wenn ein Knotenelement mehrere Verknüpfungen besitzt welche gleichfalls initial ausgeklappt werden, dann kann diese Option aktiviert werden, um dieser Verknüpfung eine erhöhte Priorität zuzuweisen.

1.3.4.11. Text

Dieses Konfigurationselement gibt einen einfachen Text aus. Dieser wird entweder fest konfiguriert oder über ein Skript ermittelt.

Einstellungsmöglichkeiten

Name	Wert
Konfigurationsname	Der Konfigurationsname dient zur Identifizierung und Wiederverwendung der Konfiguration.
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 anstelle des Beschriftungs-Attributs die Beschriftung ausgibt. Rückgabewert ist eine Zeichenkette.
Text	Text, der ausgegeben werden soll.
Skript für Text	Ein Skript, welches anstelle des Text-Attributs den anzuzeigenden Text zurückliefert.
Skript für Sichtbarkeit	Ein Skript, welches einen Booleschen Wert ausgibt, ob die View angezeigt werden soll oder nicht.

1.3.4.12. Bild

Mit Hilfe dieses Konfigurationselements kann eine statische Grafik eingebunden werden.

Name	Wert
Konfigurationname	Der Konfigurationsname dient zur Identifizierung und Wiederverwendung der Konfiguration.
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. Nicht anwendbar im Knowledge-Builder.
Breite / Höhe	Skaliert die Bilddatei auf die angegebenen Maße.
Skript für Sichtbarkeit	Durch ein Skript lässt sich bestimmen ob die Grafik angezeigt werden soll.

1.3.4.13. Skriptgeneriete View/HTML

Script-generierte View

Diese View wird mithilfe eines Skriptes generiert, welches im Knowledge-Graph hinterlegt ist. Hierbei handelt es sich um ein JavaScript, welches zusammen mit einer individuellen Vorlage verwendet werden kann. Dies erlaubt die Erstellung komplexer Views, welche über den Funktionsumfang der Standard-Viewconfiguration hinausgeht.

Einstellungsmöglichkeiten

Name	Wert
Konfigurationsname	Der Konfigurationsname dient zur Identifizierung und Wiederverwendung der Konfiguration.
Beschriftung	Eine Beschriftung findet nur ihre Anwendung, wenn diese Konfiguration in einer anderen Konfiguration wie z.B. *Alternative* eingebettet ist.
Skript für Beschriftung	Skript zum ermitteln der Beschriftung, welches eine Zeichenkette ausgibt.
Skript	Skript zur Generierung der View.
viewType	Name des Partials.

Name	Wert
Skript für Sichtbarkeit	Skript zum Ermitteln der Sichtbarkeit. Rückgabewert ist ein Boolescher Wert.

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
Konfigurationsname	Der Konfigurationsname dient zur Identifizierung und Wiederverwendung der Konfiguration.
Beschriftung	Eine Beschriftung findet nur ihre Anwendung, wenn diese Konfiguration in einer anderen Konfiguration wie z.B. *Alternative* eingebettet ist.
Skript für Beschriftung	Skript zum ermitteln der Beschriftung.
Skript	Skript zur Generierung einer HTML-Ausgabe.
Skript für Sichtbarkeit	Skript zum Ermitteln der Sichtbarkeit.

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>
```

1.3.5. Knowledge-Builder-Konfiguration

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.

1.3.5.1. Ordnerstruktur

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.

1.3.5.1.1. Die Standard-Ordnerstruktur

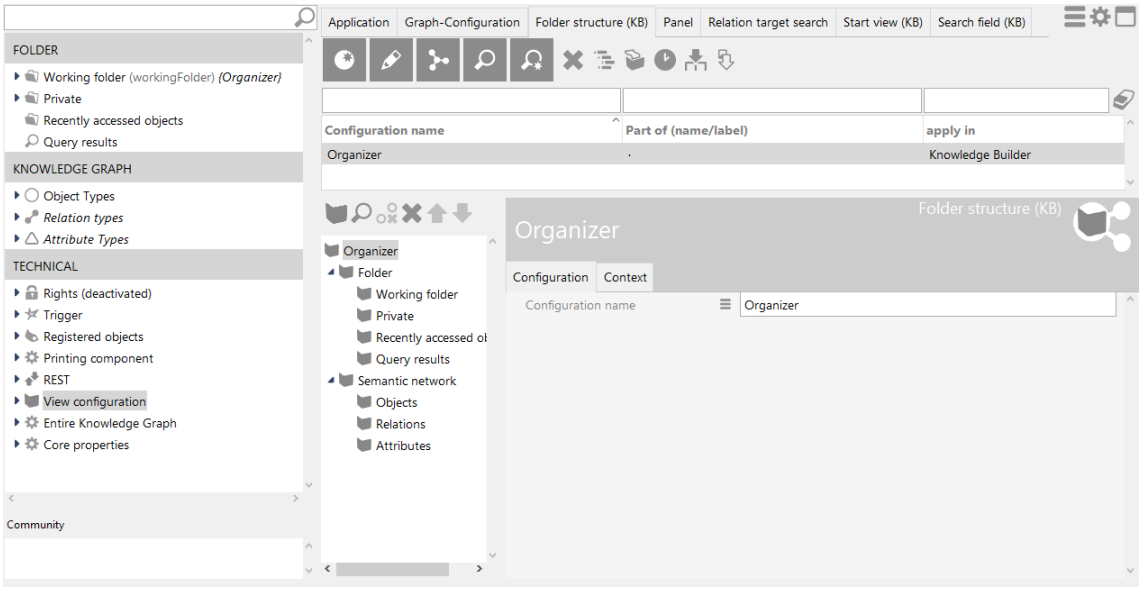
Die Konfiguration der Standard-Ordnerstruktur stellt Ordner zur Verfügung, so dass im semantischen Modell navigiert und Inhalte abgelegt werden können. Für Administratoren werden drei Hauptbereiche zur Verfügung gestellt.

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.

1.3.5.1.2. Konfiguration der Ordnerstruktur

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.

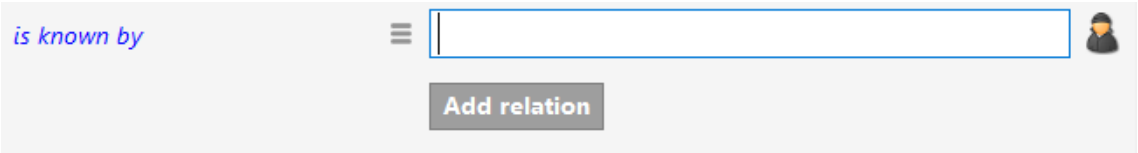
Ordnertyp (obligatorisch)	Parameter	Beschreibung
Attributtypen	Typ	Der angegebene Attributtyp und alle seine Untertypen werden in einem hierarchischen Baum angezeigt.
Privatordner	-	Anzeige des Ordners, den nur der Benutzer selbst sehen darf und der für jeden Benutzer unterschiedlich ist.
Relationstypen	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 Typbasierte Ordnerstruktur dient der Parameter Typ zur Angebe des Attribut-, Relations- oder Objekttyp der und dessen Untertypen in dem Ordner angezeigt werden sollen.

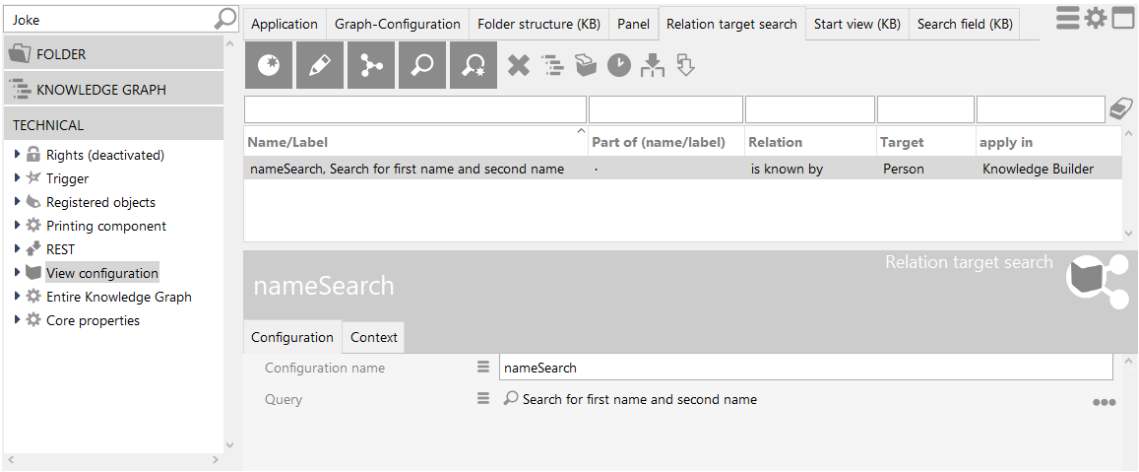
1.3.5.2. Relationszielsuche

Die Konfiguration der Relationszielsuche ermöglicht es, auf die Strategie einzuwirken, mit der mögliche Relationsziele gesucht werden.



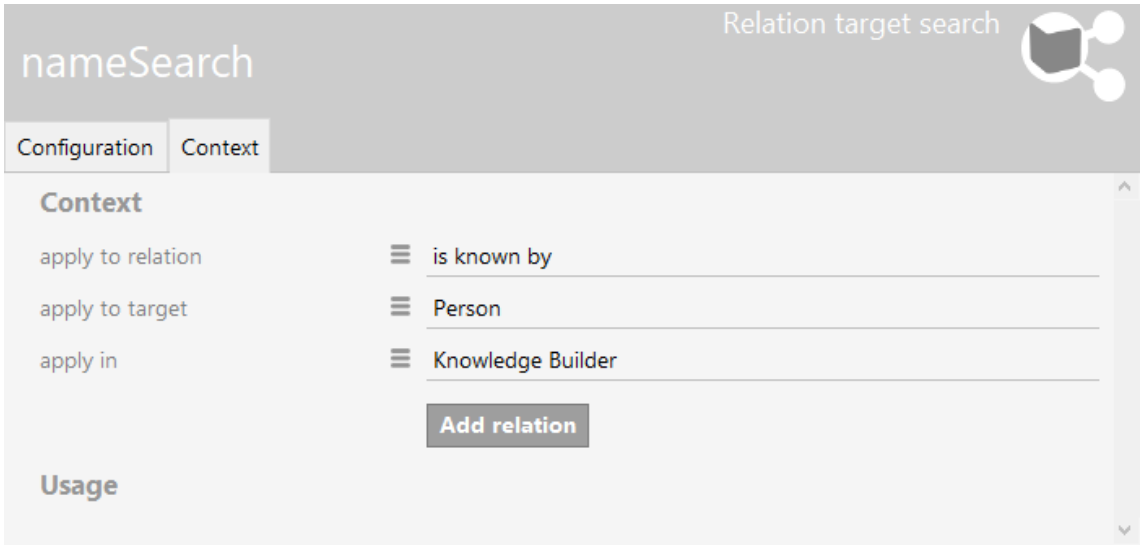
Enthält ein semantisches Modell keine Relationszielsuche, dann wird auf eine Eingabe von "Egon" immer nach einem Objekt mit Namen "Egon" gesucht (d.h. das jeweilig definierte Namensattribut wird verwendet). Durch Angabe einer zuvor definierten Abfrage kann dieses Verhalten verändert werden. Für gewöhnlich werden zu diesem Zweck gewöhnliche Abfragen und nicht etwa Strukturabfragen verwendet.

Beispielsweise könnte man für die Suche nach Personen eine Abfrage definieren, die sowohl den Vornamen als auch den Nachnamen durchsucht. 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 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" können dabei beliebig angewendet werden.

1.3.5.3. Startansicht

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.

Aktionsart	Aktion
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

1.3.5.4. Suchfeld

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.

1.3.5.4.1. Suchfeldkonfiguration für Administratoren

Die "Suchfeld"-Konfiguration legt fest, welche Abfragen vom Administrator im Schnellsuchfeld des Knowledge-Builders zur Verfügung gestellt werden.

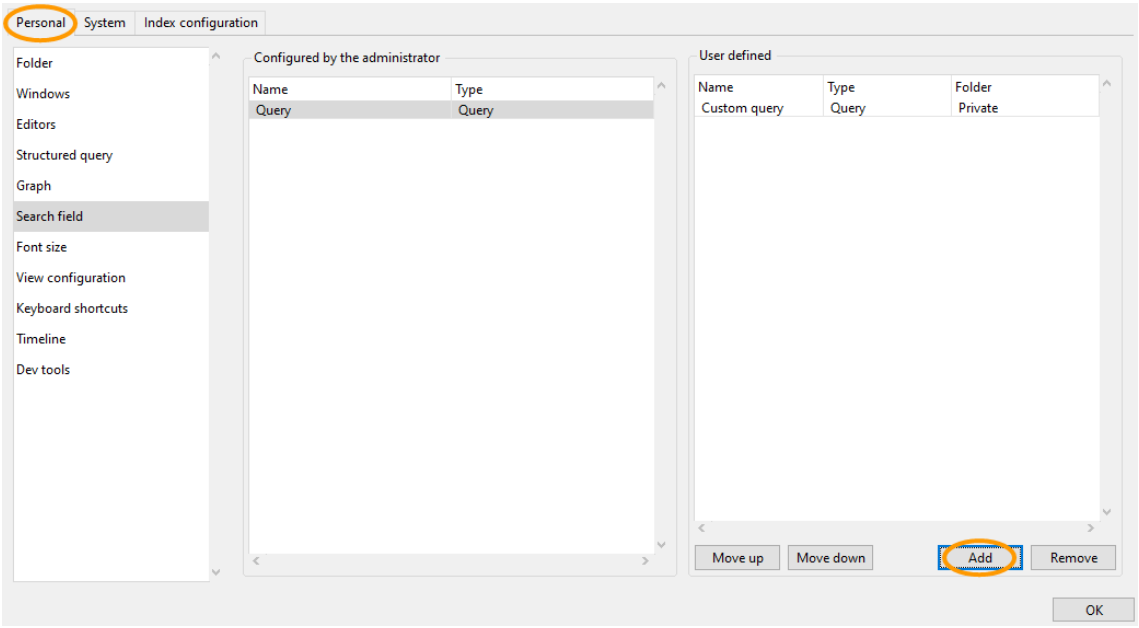
Neu angelegte Netze verfügen über eine Suchfeld-Konfiguration, die für alle Nutzer gleich ist. Der Administrator kann diese Suchfeld-Konfiguration erweitern, um allen Nutzern weitere Abfragen zugänglich zu machen. Zusätzlich kann jeder Nutzer seinem Schnellsuchfeld weitere Abfragen hinzufügen, die dann 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.

1.3.5.4.2. Suchfeldkonfiguration für Anwender

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 *Einstellungen* auch die Operationen *Entfernen* und die Möglichkeit die Reihenfolge zu ändern zur Verfügung.



1.3.6. 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.

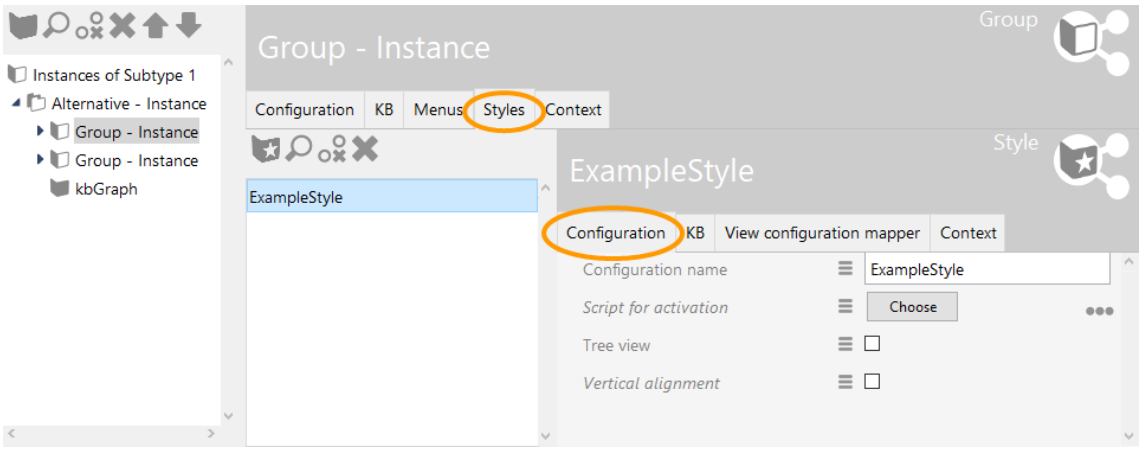
Ein Style-Element kann mit beliebig vielen Style-Eigenschaften befüllt werden. Die Style-Eigenschaften sind auf mehrere Reiter aufgeteilt, die in den folgenden Unterkapiteln beschrieben

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.

1.3.6.1. 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.



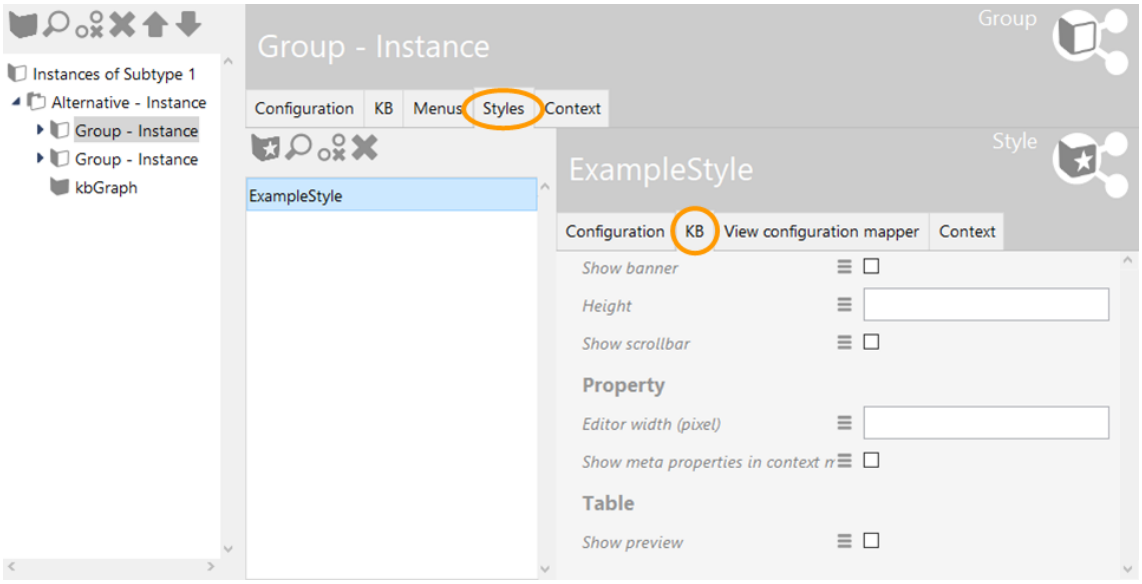
Style-Eigenschaft	Konfigurationstyp	Effekt
Konfigurationsname	Alle	Der Konfigurationsname dient zur Identifizierung und Wiederverwendung der Konfiguration.
Skript für Aktivierung	Alle	Der Style kann über ein Skript abhängig vom aktiven Element aktiviert werden.
Baumansicht	-	*Gruppen werden nicht mehr unterstützt, daher ist diese Option nicht mehr verfügbar.*
Vertikale Anordnung	-	*Gruppen werden nicht mehr unterstützt, daher ist diese Option nicht mehr verfügbar. Stattdessen sollten Layouts mit vertikaler Ausrichtung genutzt werden.*

1.3.6.2. Style-Eigenschaften in Anwendungen

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 Viewconfiguration-Mapper (Kapitel 3) enthalten.

1.3.6.3. Style-Eigenschaften im Knowledge-Builder

Dieses Kapitel beschreibt den Reiter "KB" eines Style-Elements mit den Style-Eigenschaften, welche ausschließliche im Knowledge-Builder verwendet werden können.



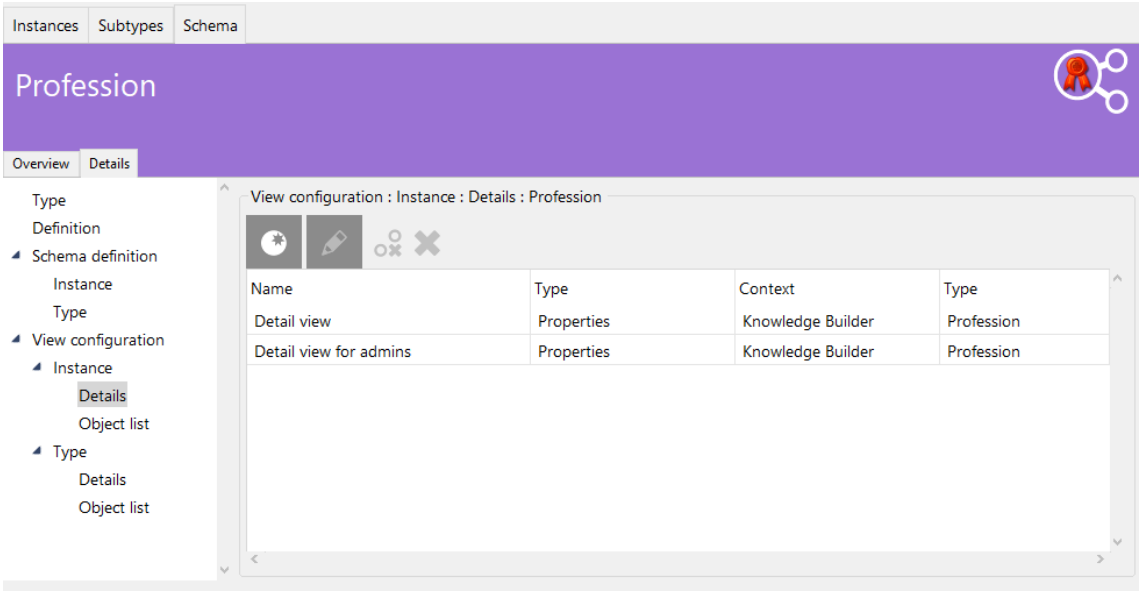
Style-Eigenschaft	Konfigurationstyp	Effekt
Konfigurationsname	Alle	Der Konfigurationsname dient zur Identifizierung und Wiederverwendung der Konfiguration.
Banner anzeigen	Objekt-Konfiguration	Zeigt den Banner mit dem Namen und dem Typ des Objekts an und dem Button für das Kontextmenü zur Bearbeitung. Standard-Einstellung bei neu erstellter Konfiguration ist `false`. 
Höhe	Eigenschaft	Höhe in Zeilen; gilt für Zeichenketten-Attribute (nicht für die "Text"-View).
Scrollbar anzeigen	Objekt-Konfiguration	Wenn aktiviert, wird eine Bildlaufleiste angezeigt, sobald die View größer als die zur Verfügung stehende Anzeigefläche ist. Diese Option ist nützlich, wenn gruppierende Elemente wie bspw. "Eigenschaften" oder "Layout" mehr als eine Sub-Konfiguration enthalten.
Eigenschaft		
Editorbreite (pixel)	Eigenschaft	Breite einer Eigenschaftszeile in Pixel.

Style-Eigenschaft	Konfigurationstyp	Effekt
Meta-Eigenschaften im Kontextmenü einblenden	(Meta-) Eigenschaft (Eigenschaften)	Meta-Eigenschaften werden im Kontextmenü einer Eigenschaft angezeigt. Auf diese Weise können entweder individuelle Metaeigenschaften oder alle Metaeigenschaften in einer Metaeigenschafts-Konfiguration angezeigt werden. **HINWEIS** Der Menüeintrag "*Metaeigenschaften hinzufügen*" bleibt unverändert.
Tabelle		
Vorschau anzeigen	Tabelle	Bestimmt, ob unterhalb der Tabelle ein Detail-Editor angezeigt werden soll.

1.3.7. Detektorsystem zur Ermittlung der View-Konfiguration

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.

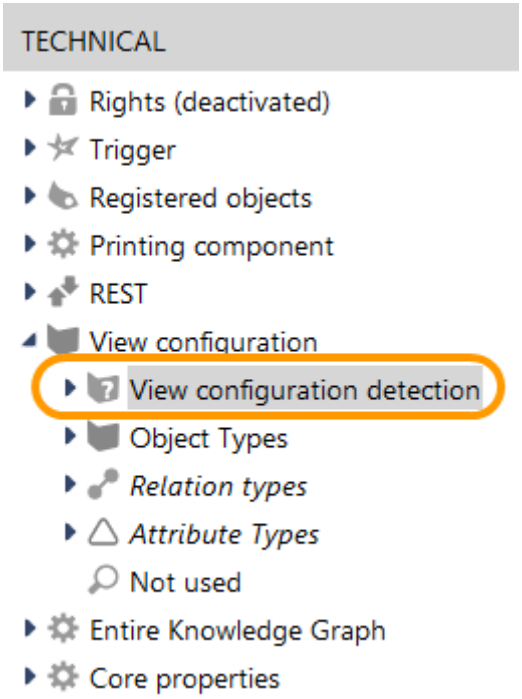
Für Objekte eines Objekttyps können, über Einstellungen in der View-Konfiguration, mehrere Anzeigen erstellt werden. Mit Hilfe des Detektorsystems können diese an Bedingungen geknüpft werden — wie beispielsweise an einen bestimmten Benutzer. Für das hier beschriebene Beispiel wurden für die Objekte eines beliebigen Typs mit Hilfe der View-Konfiguration zwei Ansichten konfiguriert.



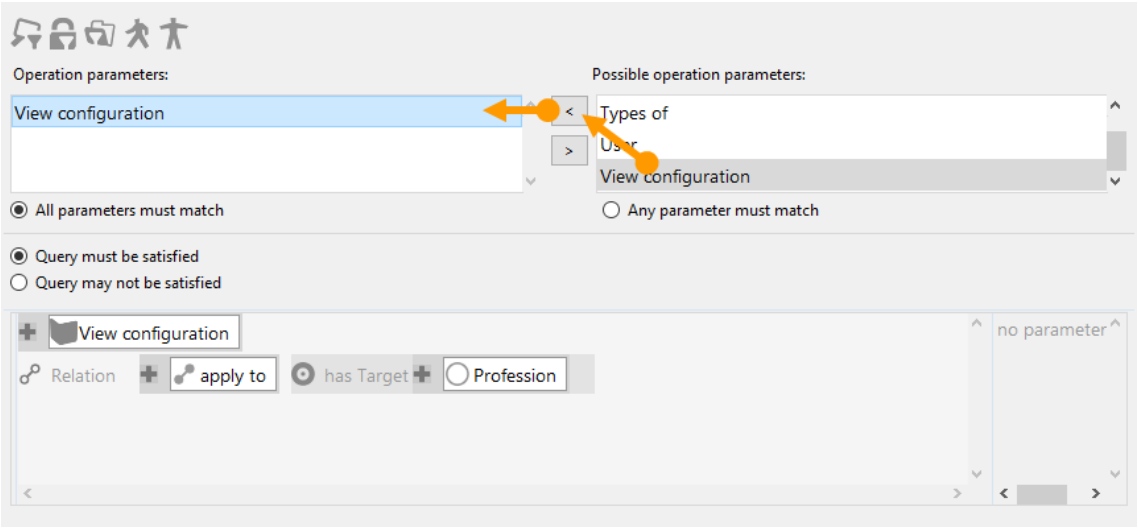
Benutzer, die einen bestimmten Beruf ausüben, auf den sie zugreifen wollen, sollen die Ansicht "AdminView" sehen. Alle Benutzer, die nicht den entsprechenden Beruf haben, auf den sie zugreifen wollen, sollen eine Standard-Ansicht sehen. Die Bedingungen, nach denen die Ansichten benutzt werden sollen, werden im Detektorsystem definiert.

Erstellung einer View-Konfigurations-Ermittlung

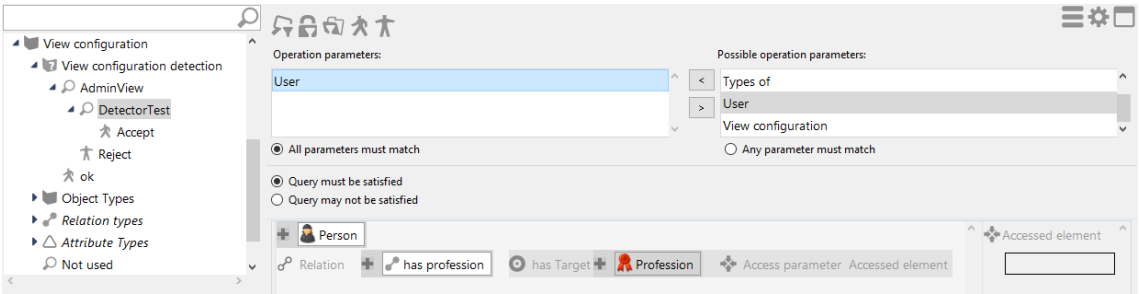
Das Detektorsystem befindet sich in der linken Ordnerhierarchie unter dem Abschnitt "Technik" und ist mit der Bezeichnung "Ermittlung der View-Konfiguration" unter "View-Konfiguration" versehen.



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: "AdminView"), 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:



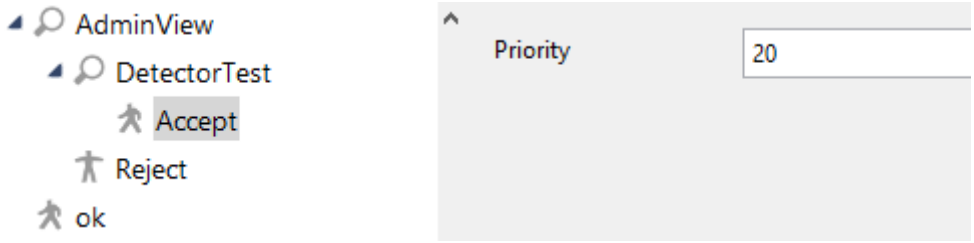
Unter dem Suchfilter, der nach der View-Konfiguration "AdminView" sucht, muss nun ein neuer Suchfilter angelegt werden, der die Bedingung für diese View-Konfiguration beschreibt: Die View-Konfiguration "AdminView" soll nur für Benutzer sichtbar sein, die den entsprechenden Beruf haben, den sie sich gerade ansehen. Der zweite Suchfilter prüft also, ob der aktive Benutzer den richtigen Beruf hat. Über einen Klick auf wird der Menge der Suchergebnisse dann erlaubt, die Konfiguration "AdminView" einzusehen. Die folgende Abbildung zeigt, den Suchfilter nach Benutzern, die denselben Beruf haben, den sie sich gerade ansehen und die Ordnerhierarchie, die auf der linken Seite bisher aufgebaut wurde.



Die Standard-View-Konfiguration wird automatisch für diejenigen Nutzer verwendet, die nicht denselben Beruf haben, den 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.

1.4. JavaScript-API

1.4.1. Einführung

Die JavaScript-API ist eine serverseitige API für semantische Netze. Sie wird u.a. in Triggern, REST-Services und der Viewkonfiguration verwendet.

Mit der API kann man sowohl lesend auf das Wissensnetz zugreifen (Suchen ausführen, Eigenschaften abfragen usw.) als auch Änderungen vornehmen (neue Objekte anlegen, Attribute ändern usw.).

Der Knowledge-Builder stellt hierzu einen eigenen JavaScript-Editor bereit, welcher das Editieren, Ausführen und Debuggen des Codes ermöglicht. Der Editor ist als Ansicht verfügbar, wenn auf das entsprechende JavaScript-Element zugegriffen wird. Registrierte JavaScript-Elemente können aufgefunden werden unter TECHNIK > Registrierte Objekte > Skripte. Ein neues JavaScript kann innerhalb von Konfigurationselementen angelegt werden oder innerhalb des Arbeitsordners oder des Privatordners im Knowledge-Builder.

HINWEIS	Das Auskommentieren von Referenzen auf Abfragen oder sonstige Elemente des Knowledge-Graphen im JavaScript Code führt dazu, dass auch beim bislang referenzierten Element unter "Verwendungen" der Hinweis auf die Verwendung im JavaScript nicht mehr angezeigt wird.

1.4.1.1. API-Referenz

Die API-Referenz ist unter folgender Adresse erreichbar:

https://documentation.i-views.com/6.1/javascript-api/index.html

1.4.1.2. Der Namespace $k

Most objects are defined in the namespace $k. The namespace object itself has a few useful functions, e.g.

```
$k.rootType()
```

which returns the root type of the Knowledge Graph, or

```
$k.user()
```

which returns the current user.

1.4.1.3. Registry

Ein weiteres wichtiges Objekt ist das Registry-Objekt $k.Registry. Es erlaubt den Zugriff auf registrierte Ordnerelemente und Objekttypen.

Beispiele:

```
$k.Registry.type("Article")
```

gibt den Typ mit dem internen Namen "Article" zurück.

```
$k.Registry.query("articles")
```

gibt die Abfrage mit dem Schlüssel "articles" zurück.

Das Registry-Objekt ist ein Singleton, ähnlich wie das Math-Objekt von JavaScript.

1.4.1.4. Mit semantischen Elementen arbeiten

Auf Wissensnetzelemente greift man normalerweise über die Registry oder Abfragen zu.

```
// Personen-Typ anhand seines internen Namen ermitteln
const personType = $k.Registry.type("Person");

// Suche "articles" mit dem Suchparameter "tag" = "Vocal"
const sailingArticles = $k.Registry.query("articles").findElements({tag:
"Vocal"});
```

Auf die Eigenschaften eines Elements kann man über ihren internen Namen zugreifen:

```
// Wert des Attributs "familyName"
const familyName = person.attributeValue("familyName");
// Ziel der Relation "bornIn"
const birthplace = person.relationTarget("bornIn");
```

Die Funktion name() liefert den Wert des Namensattributs:

```
const name = birthplace.name();
```

Bei übersetzten Attributen kann die Sprache als 2- oder 3-stelliger ISO 639 Sprachcode oder als Locale mit Sprache und Territorium angegeben werden. Ohne Sprachangabe wird die aktuelle

Sprache der Umgebung verwendet.

```
const englishTitle = book.attributeValue("title", "eng");
const swedishTitle = book.attributeValue("title", "sv_SE");
const currentTitle = book.attributeValue("title");
```

1.4.1.5. Transaktionen

Zum Anlegen, Ändern oder Löschen von Wissensnetzelementen wird eine Transaktion benötigt. Falls die Transaktionssteuerung durch das Script kontrolliert wird, kann man einen Block in eine Transaktion kapseln:

```
$k.transaction(() =>
    $k.Registry.type("Article").createInstance()
);
```

Man kann konfigurieren, ob ein Script die Transaktionssteuerung übernimmt, oder ob das gesamte Script in einer Transaktion ausgeführt werden soll. Ausgenommen sind davon Trigger-Skripte, die immer als Teil der schreibenden Transaktion ausgeführt werden.

Bei Nebenläufigkeitskonflikten wird eine Transaktion vom Server zurückgewiesen. In diesem Fall wird eine optionale Callback-Funktion aufgerufen, die man als Argument an $k.transaction() übergibt:

```
$k.transaction(
    () => $k.Registry.type("Article").createInstance(),
    () => 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 Skript-Funktion 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(() =>
    $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.

HINWEIS	Eine Fehlerbehandlungsfunktion bei Fehlschlag der optimistischen Transaktion steht nicht zur Verfügung. Sofern eine äußere Transaktion existiert, wird bei Fehlschlag deren Fehlerbehandlung ausgeführt.

1.4.1.6. Elemente modifizieren

1.4.1.6.1. Elemente anlegen

```
// Neues Objekt vom Typ "Person" erzeugen
const person = $k.Registry.type("Person").createInstance();

// Einen neuen Typ erzeugen
const blogType = $k.Registry.type("CommunicationChannel").createSubtype();
blogType.setName("Blog");
```

1.4.1.6.2. Attribute hinzufügen und ändern

Attributwerte können mit setAttributeValue() gesetzt werden. Es wird entweder der Wert des bestehenden Attributs geändert oder ein neues Attribut angelegt, falls noch kein Attribut vorhanden ist. Sollten bereits mehrere Attribute vorhanden sein wird eine Exception geworfen.

```
person.setAttributeValue("familyName", "Sinatra");
person.setAttributeValue("firstName", "Frank");
// Überschreibe den Attributwert "Frank" mit "Francis"
person.setAttributeValue("firstName", "Francis");
```

Mit createAttribute() können mehrere Attribute desselben Typs hinzugefügt werden, da bestehende Attribute nicht überschrieben werden:

```
// Zwei Attribute anlegen
person.createAttribute("nickName", "Ol' Blue Eyes");
person.createAttribute("nickName", "The Voice");
```

Die Attributwerte werden je nach Attributart durch unterschiedliche Objekttypen repräsentiert, wovon manche native JavaScript-Objekte sind, während andere zum $k-Namespace gehören:

Art des Attributs	Objekttyp
Auswahl	$k.Choice
Boolesch	boolean

Art des Attributs	Objekttyp
Datei	$k.Blob
Datum	$k.Date
Datum und Uhrzeit	$k.DateTime
Farbwert	string (Hex-Wert)
Flexible Zeit	$k.FlexDateTime
Fließkommazahl	number
Ganzzahl	number
Geographische Position	$k.GeoPosition
Gruppe	(kein Wert)
Internet-Verknüpfung (URL)	string
Intervall	$k.Interval
Zeichenkette	string
Zeit	$k.Time

1.4.1.6.3. Relationen hinzufügen

Mit `createRelation()` kann eine Relation zwischen zwei Elementen angelegt werden:

```
const places = $k.Registry.query("places").findElements({name: "Hoboken
"});
if (places.length == 1)
    person.createRelation("bornIn", places[0]);
```

1.4.1.6.4. Elemente löschen

Elemente können mit der Funktion remove() gelöscht werden:

```
person.remove();
```

Dabei werden auch alle Eigenschaften des Elements gelöscht.

1.4.2. Beispiele

1.4.2.1. Abfragen

Per API kann man registrierte Abfragen ausführen. Die Abfragen werden durch Objekte der Klasse $k.Query repräsentiert. Strukturabfragen werden durch die Unterklasse $k.StructuredQuery

repräsentiert.

Suche nach Elementen: Abfrage "articles" mit dem Parameter tag = "Soccer" ausführen

```
const articles = $k.Registry.query("articles").findElements({ tag: "
Soccer" });
for (let a of articles) {
  $k.out.print(articles[a].name() + "\n")
}
```

Suche nach Hits: Abfrage "mainSearch" mit dem Suchwert "Baseball" ausführen.

```
const hits = $k.Registry.query("mainSearch").findHits("Baseball");
for (let hit of hits) {
  $k.out.print(`${hit.element().name()} (${Math.round(hit.quality() *
100)}%)\n`)
}
```

Ein Hit kapselt ein Element und fügt einen Qualitätswert (zwischen 0 und 1) sowie weitere Metainformationen hinzu.

Suchergebnis in JSON umwandeln:

```
const elements = $k.Registry.query("articles").findElements({ tag:
"Snooker" })
const json = elements.map(element => ({
  name: element.name(),
  id: element.idString(),
  type: element.type().name()
}))
$k.out.print(JSON.stringify(json, undefined, "\t"))
```

1.4.2.2. Zur Laufzeit generierte Abfragen

Die Javascript-API erlaubt es auch, Abfragen dynamisch zu generieren. Hier einige Beispiele aus einem Filmnetz:

1.4.2.2.1. Suche nach Filmen mit Jahr + Name

```
const query = new $k.StructuredQuery("imdb_film")
query.addAttributeValue("imdb_film_year", "year")
query.addAttributeValue("name", "name")
```

```
query.findElements({ year: "1958", name: "Vert*" })
```

Dem Konstruktor wird die Domain übergeben. Bei internen Namen wird automatisch nach Objekten dieses Types gesucht. Mehr Möglichkeiten bietet die Funktion setDomains()

1.4.2.2.2. Jahr + Anzahl Regisseure >= 3

```
const query = new $k.StructuredQuery("imdb_film")
query.addAttributeValue("imdb_film_year", "year")
query.addCardinality("imdb_film_regisseur", 3, ">=")
query.findElements({year: "1958"})
```

1.4.2.2.3. Jahr + Name des Regisseurs

```
const query = new $k.StructuredQuery("imdb_film")
query.addAttributeValue("imdb_film_year", "year", ">=")
const directorQuery = query.addRelationTarget("imdb_film_regisseur"
).targetQuery()
directorQuery.addAttributeValue("name", "director")
query.findElements({ year: "1950", director: "Hitchcock, Alfred" })
```

1.4.2.2.4. Alternativen (Oder-Bedingungen)

```
const query = new $k.StructuredQuery("imdb_film")
query.addAttributeValue("imdb_film_year", "year")
const alternatives = query.addAlternativeGroup()
alternatives.addAlternative().addAttributeValue("name", "name")
alternatives.addAlternative().addAttributeValue("imdb_film_alternativeTite
l", "name")
query.findElements({ year: "1958", name: "Vert*" })
```

1.4.2.2.5. Operatoren

Operator-Name	Kurzbezeichnung	Beschreibung
containsPhrase		Enthält Phrase
covers		Enthält
distance		Abstand
equal	==	Gleich
equalBy		Entspricht

Operator-Name	Kurzbezeichnung	Beschreibung
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

1.4.2.3. Elemente anlegen und ändern

1.4.2.3.1. Eine Person anlegen

```
// Get the person type by its internal name
const personType = $k.Registry.type("Person");
// Create a new instance
const person = personType.createInstance();
// Set attribute values
person.setAttributeValue("familyName", "Norris");
```

```
person.setAttributeValue("firstName", "Chuck");
```

1.4.2.3.2. Den vollständigen Namen einer Person setzen

```
const familyName = person.attributeValue("familyName");
const firstName = person.attributeValue("firstName");
if (familyName && firstName) {
  const fullName = familyName + ", " + firstName;
  person.setAttributeValue("fullName", fullName);
}
```

1.4.2.3.3. Attributwerte setzen

```
// Boolean attribute
element.setAttributeValue("hasKeycard", true);

// Choice attribute
// - internal name
element.setAttributeValue("status", "confirmed");
// - choice object
const choiceRange = $k.Registry.attributeType("status").valueRange();
const choice = choiceRange.choiceInternalNamed("confirmed");
element.setAttributeValue("status", choice);

// Color attribute
element.setAttributeValue("hairColor", "723F10");

// Date / Time / DateAndTime attribute
element.setAttributeValue("dateOfBirth", new Date(1984, 5, 4));
element.setAttributeValue("lastModification", new Date());
element.setAttributeValue("teatime", new Date(0, 0, 0, 15, 30, 0));

// FlexTime attribute
// - $k.FlexTime (allows imprecise values)
element.setAttributeValue("start", new $k.FlexTime(1984, 6));
// - Date (missing values are set to default values)
element.setAttributeValue("start", new Date(1984, 5, 3));

// Number (integer / float) attribute
element.setAttributeValue("weight", 73);

// Interval
element.setAttributeValue("interval", new $k.Interval(2, 4));
```

```
// String attribute
// - untranslated
element.setAttributeValue("familyName", "Norris");
// - translated (language is an ISO 639-1 or 639-2b code)
element.setAttributeValue("welcomeMessage", "Welcome", "en");
element.setAttributeValue("welcomeMessage", "Bienvenue", "fre");
```

1.4.2.3.4. Neues Attribut anlegen

```
person.createAttribute("nickName", "Ground Chuck");
```

1.4.2.3.5. Neue Relation anlegen

```
const places = $k.Registry.query("places").findElements({name: "Oklahoma
"});
if (places.length == 1)
    person.createRelation("bornIn", places[0]);
```

1.4.2.3.6. Ein Element samt Eigenschaften löschen

```
person.remove()
```

1.4.2.3.7. Eine Zeichenkette in einen Attributwert konvertieren

Der Wertebereich (ValueRange) eines Attributtyps kennt die erlaubten Werte und kann die Zeichenkette einlesen. Bei ungültigen Eingaben wird ein Fehler geworfen.

```
const statusRange = $k.Registry.type("status").valueRange();
const statusConfirmed = statusRange.parse("Confirmed", "eng");
```

1.4.2.3.8. Änderungs-Metadaten ändern

```
element.setAttributeValue("lastChangeDate", new $k.Date());
const userInstance = $k.user().instance();
// Ensure that a single relation to the user instance exists
if (element.relationTarget("lastChangedBy") !== userInstance) {
  const relations = element.relations("lastChangedBy");
  for (let relation of relations)
    relation.remove();
  element.createRelation("lastChangedBy", userInstance);
```

```
}
```

1.4.2.4. Datum und Uhrzeit

Wenn man ein JavaScript-Date als Attributwert setzt, wird der Wert in der lokalen Zeitzone gespeichert. Die Attribute selbst speichern keine Zeitzone, nur Datum/Uhrzeit.

```
const task = $k.Registry.type("Task").createInstance()
task.setAttributeValue("dateOfCreation", new Date())
```

Wenn man dieses Script zum Zeitpunkt 20.6.2023 12:58 MEZ ausführt, wird "20.6.2023 12:58" als Attributwert gesetzt.

Um einen Attributwert unabhängig von der lokalen Zeitzone zu speichern, kann man die $k.DateTime-API verwenden. Dieses hat eine mit Date verwandte API, kann aber zusätzlich mit toUTC() die Zeitzone des Werts wandeln:

```
const task = $k.Registry.type("Task").createInstance()
task.setAttributeValue("dateOfCreation", new $k.DateTime().toUTC())
```

Dieses Script setzt zum selben Zeitpunkt "20.6.2023 10:58" als Attributwert.

Da das Attribut keine Zeitzone speichert, ist die Darstellung bei Clients unabhängig von der lokalen Zeitzone.

Für eine Darstellung in der lokalen Zeitzone kann $k.DateTime mit `toUTCDate()` den in UTC gespeicherten Attributwert in eine Date in der lokalen Zeitzone umwandeln.

```
task.attributeValue("dateOfCreation").toUTCDate()
```

HINWEIS

- `toUTC()` ist nicht beim ECMAScript-Date definiert, nur bei $k.DateTime und $k.Time
- `toUTCDate()` ist leider leicht mit `toUTC()` zu verwechseln.
- `toUTCDate()` liefert ein ECMAScript-Date, `toUTC()` ein $k.DateTime / $k.Time-Objekt

Wenn man zu einem Datum/Uhrzeit-Wert nur das Datum oder nur die Uhrzeit ausgeben möchte, kann man dazu $k.Date und $k.Time verwenden:

```
// Anlegezeitpunkt in lokaler Uhrzeit darstellen
```

```
new $k.Time(task.attributeValue("dateOfCreation").toUTCDate())
```

HINWEIS	Im Gegensatz zur ECMAScript-API ist $k.Date nur das Datum ohne Uhrzeit. $k.DateTime hat Datum + Uhrzeit.

1.4.2.5. Sessions

Um ein semantisches Element oder einen spezifischen Wert einem nachgelagerten View in einer Webanwendung zur Verfügung zu stellen, kann eine Session-Variable verwendet werden.

Das Ablegen in der Session-Variable erfolgt mit:

```
$k.Session.current().setVariable("nameOfVariable", elementOrValue)
```

Das Auslesen der Session-Variable erfolgt mit:

```
$k.Session.current().getVariable("nameOfVariable")
```

1.4.2.6. REST

Ein REST-Script muss eine Funktion respond() definieren, die als Argumente die HTTP-Anfrage, die geparsten Anfrage-Parameter und eine leere HTTP-Antwort entgegennimmt. Das Script füllt dann Header und Inhalt der Antwort.

```
function respond(request, parameters, response) {
  response.setText("REST example");
}
```

1.4.2.6.1. Einen Blob herunterladen

```
function respond(request, parameters, response) {
  const name = parameters["name"];
  if (name) {
    const images = $k.Registry.query("rest.image").findElements({"name":
name});
    if (images.length == 1) {
      // Set the contents and content type (if known) from the image blob.
      response.setContents(images[0].value());
      // Show the image instead of asking to download the file
      response.setContentDisposition("inline");
    } else {
```

```
      response.setCode($k.HttpResponse.BAD_REQUEST);
      response.setText(images.length + " images found");
    }
  } else {
    response.setCode($k.HttpResponse.BAD_REQUEST);
    response.setText("Name not specified");
  }
}
```

1.4.2.6.2. Neues Objekt mit einem hochgeladenen Blob anlegen

```
function respond(request, parameters, response) {
  const formData = request.formData();
  const name = formData.name;
  const picture = formData.picture;
  if (name && picture) {
    const city = $k.Registry.type("City").createInstance();
    city.setAttributeValue("image", picture);
    city.setName(name);
    response.setText("Created city " + name);
  } else {
    response.setCode($k.HttpResponse.BAD_REQUEST);
    response.setText("Parameters missing");
  }
}
```

1.4.2.7. XML

1.4.2.7.1. Suchergebnisse in XML transformieren

```
function respond(request, parameters, response) {
  const name = parameters["name"];
  if (name) {
    // Find points of interest
    const elements = $k.Registry.query("rest.poi").findElements({name:
name});
    // Write XML
    const document = new $k.TextDocument();
    const writer = document.xmlWriter();
    writer.startElement("result");
    for (let element of elements) {
      writer.startElement("poi");
      writer.attribute("name", element.name());
      writer.endElement();
```

```
    }
    writer.endElement();
    response.setContents(document);
    response.setContentType("application/xml");
  } else {
    response.setCode($k.HttpResponse.BAD_REQUEST);
    response.setContents("Name not specified");
  }
}
```

XML-Ausgabe

```
<result>
  <poi name="Plaza Mayor"/>
  <poi name="Plaza de la Villa"/>
  <poi name="Puerta de Europa"/>
</result>
```

1.4.2.7.2. Qualifizierte Namen verwenden

```
const document = new $k.TextDocument();
const writer = $k.out.xmlWriter();
writer.setPrefix("k", "http://www.i-views.de/kinfinity");
writer.startElement("root", "k");
writer.attribute("hidden", "true", "k");
writer.startElement("child", "k").endElement();
writer.endElement();
```

XML-Ausgabe

```
<k:root xmlns:k="http://www.i-views.de/kinfinity" k:hidden="true">
  <k:child/>
</k:root>
```

1.4.2.7.3. Standard-Namespace definieren

```
const document = new $k.TextDocument();
const writer = $k.out.xmlWriter();
writer.startElement("root");
writer.defaultNamespace("http://www.i-views.de/kinfinity");
writer.startElement("child").endElement();
```

```
writer.endElement();
```

XML-Ausgabe

```
<root xmlns="http://www.i-views.de/kinfinity">
  <child/>
</root>
```

1.4.2.8. HTTP-Client

In einem Script können auch HTTP-Requests abgeschickt werden.

1.4.2.8.1. Bild über HTTP laden und als Blob im Wissensnetz speichern

```
const http = new $k.HttpConnection();
const imageUrl = "http://upload.wikimedia.org/wikipedia/commons/e/e7/2007-
07-06_GreatBriain_Portree.jpg";
const imageResponse = http.request(new $k.HttpRequest(imageUrl));
if (imageResponse && imageResponse.code() == $K.HttpResponse.OK ) {
  const portree = $k.Registry.type("City").createInstance();
  portree.setAttributeValue("image", imageResponse);
  portree.setName("Portree");
}
```

1.4.2.8.2. Wetterberichte aller Städte aktualisieren

```
const instances = $k.Registry.type("City").instances();
const http = new $k.HttpConnection();
for (let instance of instances) {
  const city = instance;
  const weatherUrl = "http://api.openweathermap.org/data/2.5/weather";
  const weatherRequest = new $k.HttpRequest(weatherUrl);
  weatherRequest.setQueryData({q: city.name()});
  try {
    const weatherResponse = http.request(weatherRequest);
    if (weatherResponse.code() == $k.HttpResponse.OK) {
      const json = JSON.parse(weatherResponse.text());
      const weather = json.weather[0].description;
      city.setAttributeValue("weather", weather);
    }
  } catch (e) {
  }
```

```
}
```

1.4.2.8.3. Basic-Authentifizierung

Im folgenden Beispiel wird Username + Passwort zur Basic-Authentifizierung aus einer verschlüsselten Zeichenkette entnommen. Solle Zeichenketten können im Admin-Tool unter Systemkonfiguration > Zugangsberechtigung mit "Name/Passwort verschlüsseln" erstellt werden und sind nur diesem Netz gültig.

```
const http = new $k.HttpConnection()
const account =
"GH1Z4FXWrCdEoiDSlCVMZJQ6QaBZ4rfAcJdDliUHn8ep00ZKmUR+f8nvAFEObB1pjrQId0Bn9
rjmaSZJtz4X6RSAGONfHRxlWG62V3itUPeHzqs7DE90/jG+cv/rVKNrxcdFGRja6cjnHOTK4LG
jZiuUV313GsC1EDr8GEctfeo="
http.authenticateFromEncrypedAccount(account)
const request = new $k.HttpRequest("http://example.org/restricted")
const response = http.request(request)
```

1.4.2.8.4. Ein JSON-Objekt per POST senden

```
const http = new $k.HttpConnection();
const objectToPost = [{foo: "bar"}, "baz"];
const destinationURL = "http://upload-via-post.domain.com";
const postRequest = new $k.HttpRequest(destinationURL, "POST");
postRequest.setText(JSON.stringify(objectToPost))
postRequest.setHeaderField("Content-Type", "application/json");
const response = http.request(postRequest);
```

1.4.2.8.5. Einen Blob per PUT senden

```
const blob = $k.Registry.elementAtValue("isbn", "978-0544003415"
).attributeValue("coverPicture")
const http = new $k.HttpConnection()
const request = new $k.HttpRequest("http://mybookservice/cover/978-
0544003415", "PUT")
request.setContents(blob)
const response = http.request(request)
```

Der Content-Type wird vom Blob übernommen.

1.4.2.8.6. Zwei Blobs per POST als multipart/form-data senden:

```
const book = $k.Registry.elementAtValue("isbn", "978-0544003415")
const pdfBlob = book.attributeValue("pdf")
const previewBlob = book.attributeValue("preview")
const http = new $k.HttpConnection()
const request = new $k.HttpRequest("http://mybookservice/ebooks/978-
0544003415", "POST")
request.setContentType("multipart/form-data")
const pdfPart = new $k.NetEntity()
pdfPart.setContentDisposition('form-data; name="ebook"')
pdfPart.setContents(pdfBlob)
request.attach(pdfPart)
const previewPart = new $k.NetEntity()
previewPart.setContentDisposition('form-data; name="preview"')
previewPart.setContents(previewBlob)
request.attach(previewPart)
const response = http.request(request)
```

Der Dateiname der beiden Form-Daten wird vom Blob übernommem. Wenn ein anderer Dateiname gewünscht ist, kann man diesen mit setFilename(string) setzen.

1.4.2.8.7. Ein URL-kodiertes Formular per POST senden

```
const data = { name: "Gandalf", occupation: "Wizard" }
const http = new $k.HttpConnection()
const request = new $k.HttpRequest("http://mybookservice/user", "POST")
request.setFormData(data)
const response = http.request(request)
```

Die Daten werden mit dem Content-Type application/x-www-form-urlencoded verschickt.

Um zu verhindern, dass Skripte Requests zu beliebigen Hosts schicken, kann man in der Konfigurationsdatei einer Anwendung eine Whitelist definieren:

```
[script]
allowedOutgoingDomains=*.i-views.de,*.intelligent-
views.com,ivinternal:8080
```

Die durch Komma getrennten Zeichenketten werden mit der Domain der URL verglichen, "*" als Wildcard ist dabei erlaubt. Optional kann auch ein Port angegeben werden. Falls Domain oder Port nicht passen wird bei der Ausführung des Requests ein URIError geworfen. Ohne Angabe des Port ist jeder Port gültig.

1.4.2.9. E-Mails versenden

E-Mails können mit dem $k.MailMessage-Objekt versendet werden.

Dazu kann im Netz ein SMTP-Server konfiguriert werden (Einstellungen > System > SMTP), oder man kann über ein $k.SmtpConnection-Objekt einen SMTP-Server im Script angeben.

Versand einer Mail über einen im Netz konfigurierten SMTP-Server:

```
const mail = new $k.MailMessage();
mail.setSubject("Hello from " + $k.volume());
mail.setText("This is a test mail");
mail.setSender("kinfinity@example.org");
mail.setReceiver("developers@example.org");
mail.setUserName("kinf");
mail.send();
```

Das Benutzerkonto "kinf" wird für die Authentifizierung verwendet. Das Passwort wird in den SMTP-Einstellungen hinterlegt.

1.4.2.9.1. Versand einer Mail über $k.SmtpConnection

In diesem Beispiel wird Username + Passwort zur Authentifizierung aus einer verschlüsselten Zeichenkette entnommen. Solche Zeichenketten können im Admin-Tool unter Systemkonfiguration > Zugangsberechtigung mit "Name/Passwort verschlüsseln" erstellt werden und sind nur diesem Netz gültig.

```
const mail = new $k.MailMessage()
mail.setSubject("Hello from " + $k.volume())
mail.setText("This is a test mail")
mail.setSender("kinfinity@example.org")
mail.setReceiver("developers@example.org")
const smtp = new $k.SmtpConnection()
smtp.setHost("mailgateway.local", 22)
smtp.authenticateFromEncrypedAccount("Qi3Eky7itkf2NckwgcKemiZvNGGoXcbo43O2
/nZ5RvoRvv7AukUM0LIVUw1WJ+uMDgzxw7JA5gtYyLgNg7fHaC4wJCQIgnIfXVPSW6u391NmUq
nZkcuc0nl4u2nPymAmcqzoUDJRSHrMVy1qEsbxXtfhsJzh7e4EDKIAeJ75BxE=")
smtp.send(mail)
```

1.4.2.9.2. Eine E-Mail mit Attachment versenden

```
const mail = new $k.MailMessage()
mail.setSubject("Daily report")
mail.setText("Here is the daily report")
```

```
const attachment = new $k.NetEntity()
attachment.setContentType("text/html")
attachment.setText("<html><body><h1>Daily report</h1>No problems
found</body</html>")
attachment.setContents(report)
mail.attach(attachment)
mail.setSender("kinfinity@example.org")
mail.setReceiver("developers@example.org")
mail.setUserName("kinf")
mail.send()
```

1.4.2.10. Abbildungen von Datenquellen

Per API kann man registrierte Abbildungen von Datenquellen ausführen. Die Abbildungen werden durch Objekte der Klasse $k. Mapping repräsentiert. Abbildungen zur Laufzeit zu generieren ist derzeit nicht möglich.

Einen Export mit einer registrierten Abbildung mit dem Registierungsschlüssel `products` durchführen:

```
const mapping = $k.Registry.mapping("products")
mapping.runExport()
```

Bei Datei-basierten Datenquellen verwendet die API standardmäßig die konfigurierten Ein-/Ausgabedateien. Alternativ kann von/in eine $k.NetEntity im-/exportiert werden:

```
const mapping = $k.Registry.mapping("products")
const productsEntity = new $k.NetEntity()
mapping.setParameter("netEntity", productsEntity)
mapping.runExport()
```

Dadurch können die Inhalte per HTTP oder E-Mail transportiert werden. Derzeit werden die Inhalte der NetEntity im Hauptspeicher abgelegt, für große Datenmengen ist diese Methode deshalb nicht geeignet.

1.4.2.11. ZIP-Dateien

Zip-Dateien können sowohl gelesen als auch erstellt werden. Sowohl die Zip-Datei selbst als auch die enthaltenen Dateien werden über $k.NetEntity-Objekte repräsentiert. Es können aber auch Blobs als Inhalt hinzugefügt werden.

1.4.2.11.1. Eine Zip-Datei in einem REST-Request als Antwort liefern

```
function respond(request, parameters, response) {
  const zip = new $k.Zip("avatars.zip")
  $k.Registry.type("account").allInstances().forEach(account =>
    zip.addEntry(account.attributeValue("avatar"))
  )
  response.setContents(zip)
}
```

1.4.2.11.2. Den Inhalt einer als Body eines POST-Requests geschickten Zip-Datei auslesen

Dazu wird der Konstruktor mit einer $k.NetEntity aufgerufen.

```
function respond(request, parameters, response) {
  if (request.contentType() !== "application/zip") {
    response.setCodeBadRequest().setText("Zip expected")
    return
  }
  const zip = new $k.Zip(request)
  zip.filenames().forEach(filename => {
    const entityInZip = zip.entry(filename)
    const account = $k.Registry.type("upload").createInstance()
    account.setAttributeValue("file", entityInZip)
  })
}
```

1.4.2.12. Mustache-Templates

Die folgende Funktion erzeugt ein Dokument mit Hilfe der Mustache Template-Bibliothek. Es erwartet folgendes Schema:

- ein Zeichenketten-Attribut (interner Name "template.id"), um das Template zu identifizieren
- ein Dateiattribut (interner Name "template.file") mit dem Template, z.B. ein HTML-Dokument
- Eine Relation zu einem MediaType-Objekt (interner Name "template.contentType")

Die Abfrage "rest.articles" gibt alle Element zurück die dargestellt werden sollen. Die Mustache-Bibliothek ist unter "mustache.js" registriert.

```
function respond(request, parameters, response) {
  // Get template
  const templateId = parameters["templateId"];
  const templateElement = $k.Registry.elementAtValue("template.id",
```

```
templateId);
  const templateText = templateElement.attributeValue("template.file"
).text("utf-8");

  // Find elements
  const elements = $k.Registry.query("rest.articles").findElements
(parameters);

  // Prepare template parameters
  const elementsData = elements.map(element => ({
    name: element.name(),
    id: element.idNumber(),
    type: element.type().name()
  }))
  const templateParameters = {
    elements: elementsData
  };

  // Render with Mustache
  const output = $k.Mustache.render(templateText, templateParameters);

  // Return the rendered document
  response.setText(output);
  response.setContentType(templateElement.relationTarget(
"template.contentType").name());
}
```

1.4.2.13. Java Native Interface

Mit Hilfe von JNI (Java Native Interface) können Java-Bibliotheken eingebunden werden.

WARNUNG

JNI ist ein experimentelles Feature und hat einige Einschränkungen:

- JNI kann nicht in Triggern verwendet werden
- Es ist nicht möglich, Klassen zu definieren (beispielsweise für Callbacks)
- Generics werden nicht unterstützt
- JNI erlaubt den Zugriff auf Systemressourcen (z.B. Dateien)
- JNI muss in den Konfigurationsdateien aller Anwendungen eingerichtet werden, die diese Skripte verwenden. Der Klassenpfad kann nicht zur Laufzeit geändert werden.

1.4.2.13.1. Konfiguration

```
[JNI]
classPath=tika\tika-app-1.5.jar
libraryPath=C:\Program Files\Java\jre7\bin\server\jvm.dll
```

1.4.2.13.2. Beispiel

Mithilfe der Funktion `$jni.use()` wird eine Liste von Klassen importiert. Für jede Klasse wird ein gleichnamiges Funktionsobjekt angelegt, das mit new instanziiert werden kann. Außerdem werden alle statischen Eigenschaften an dieses Funktionsobjekt übertragen. Der Namensraum der Java-Klassen kann optional auch weggelassen werden.

```
// Import the StringBuilder class, without namespace
$jni.use(["java.lang.StringBuilder"], false);
// Create a new instance
const builder = new StringBuilder();
// Javascript primitives and Strings are automatically converted
builder.append("Welcome to ");
builder.append($k.volume());
// toJS() converts Java objects to Javascript objects
$k.out.print(builder.toString().toJS());
```

1.4.2.13.3. Text/Metadaten-Extraction mit Apache Tika

```
$jni.use([
    "java.io.ByteArrayInputStream",
    "java.io.BufferedInputStream",
    "java.io.StringWriter",
    "org.apache.tika.parser.AutoDetectParser",
    "org.apache.tika.metadata.Metadata",
    "org.apache.tika.parser.ParseContext",
    "org.apache.tika.sax.BodyContentHandler"
  ], false);
// Get a blob
const blob = $k.Registry.elementAtValue("uuid", "f36db9ef-35b1-48c1-9f23-
1e10288fddf6").attributeValue("ebook");
// Blobs have to be explicitely converted to Java byte arrays
const bufferedInputStream = new BufferedInputStream(new
ByteArrayInputStream($jni.toJava(blob)));
// Parse the blob
try {
  const parser = new AutoDetectParser();
  const writer = new StringWriter();
  const metaData = new Metadata();
```

```
  parser.parse(bufferedInputStream, new BodyContentHandler(writer),
metaData, new ParseContext());
  const string = writer.toString().toJS();
  // Print extracted metadata
  const metaNames = metaData.names().toJS().sort((a, b) => a.
localeCompare(b));
  for (let name of metaNames)
    $k.out.print(`${name} = ${metaData.get(name)}`).cr();
  // Print extracted text (first 100 chars)
  $k.out.cr().cr().print(`${string.substring(1, 100)} [...]\n\n(${string
.length} chars)`);
} catch (e)  {
  $k.out.print("Extraction failed: " + e.toString());
} finally {
  bufferedInputStream.close();
}
```

1.4.2.14. XML parsen

Die experimentelle DOMParser-API bietet eine Teilmenge der Web API-Funktionalität zum Parsen von XML-Inhalten.

1.4.2.14.1. XML als DOM einlesen

```
const xml = "<rootNode><node1>Some text</node1><node2>More
text</node2></rootNode>"
const dom = new $dom.DOMParser().parseFromString(xml)
$k.out.print(dom.firstChild.children[0].nodeName)
```

1.4.2.14.2. Knoten per XPath adressieren

```
const xml = "<rootNode><node1>Some text</node1><node2>More
text</node2></rootNode>"
const dom = new $dom.DOMParser().parseFromString(xml)
$k.out.print(dom.evaluate("//node2").stringValue)
```

1.4.3. Module

1.4.3.1. Module definieren

Ein Modul wird mit der Funktion define() definiert. Als Argument übergibt man entweder ein Modulobjekt oder eine Funktion, die ein Modulobjekt zurückgibt. Ein Script sollte nur ein Modul definieren.

Beispiel: Modul mit einer Funktion jsonify()

```
$k.define({
    /*
     * Ein Array von JSON-Objekten aus elements erzeugen
     */
    jsonify: function(elements) {
        return elements.map(element => {
            name: element.name(),
            id: element.idString(),
            type: element.type().name()
        });
    }
});
```

Module können auch von anderen Modulen abhängig sein. Das folgende Skript definiert ein Modul, das ein anderes ("rest.common") verwendet.

```
$k.define(["rest.common"], function(common) {
    return {
        stringify: function(elements) {
            return JSON.stringify(common.jsonify(elements), undefined, "
\t")
        }
    }
});
```

1.4.3.2. Module verwenden

Ein Modul kann entweder mit require() oder module() verwendet werden.

require() erwartet einen Array von Modulnamen und eine Callback-Funktion. Die Callback-Funktion wird mit den Modulen als Argumente ausgeführt. require() gibt den Rückgabewert der Callback-Funktion zurück.

```
const elements = $k.Registry.query("rest.poi").findElements({name: "
Madrid"});
const json = $k.require(["rest.common"], function(common) {
     return common.jsonify(elements);
});
$k.out.print(JSON.stringify(json, undefined, "\t"));
```

module() erwartet den Namen eines Moduls und gibt das Modulobjekt zurück.

```
const json = $k.module("rest.common").jsonify(elements);
$k.out.print(JSON.stringify(json, undefined, "\t"));
```

module() kann auch mit Skripten verwendet werden, die keine Module definieren. Das Script wird ausgeführt und alle deklarierten Funktionen instanziiert. Diese Funktionen können anschließend aufgerufen werden.

1.4.3.3. 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");
```

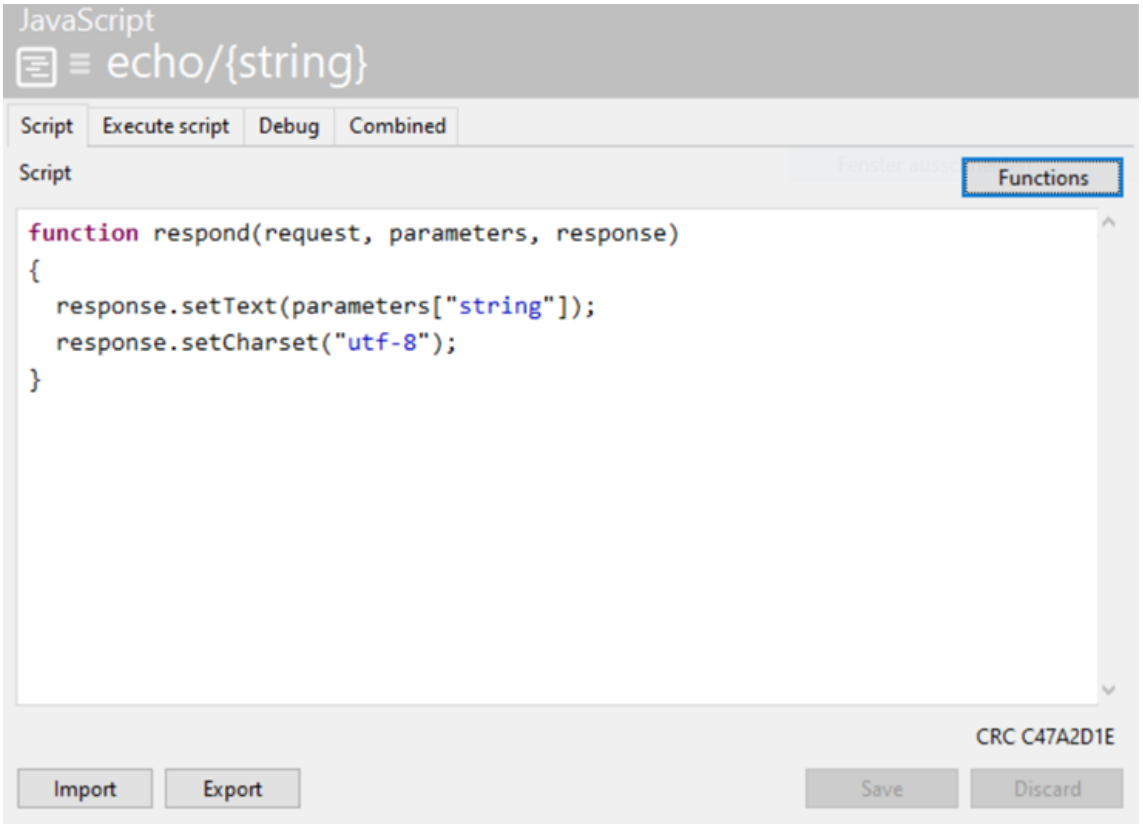
Nun kann man underscore.js als "lib.underscore" registrieren und das dort definierte Modul "underscore" verwenden.

1.4.4. Editor und Debugger

Der Editor enthält vier durch Reiter getrennte Abschnitte:

1.4.4.1. Skript

Funktionen: Importieren, Editieren und Exportieren von Skripten



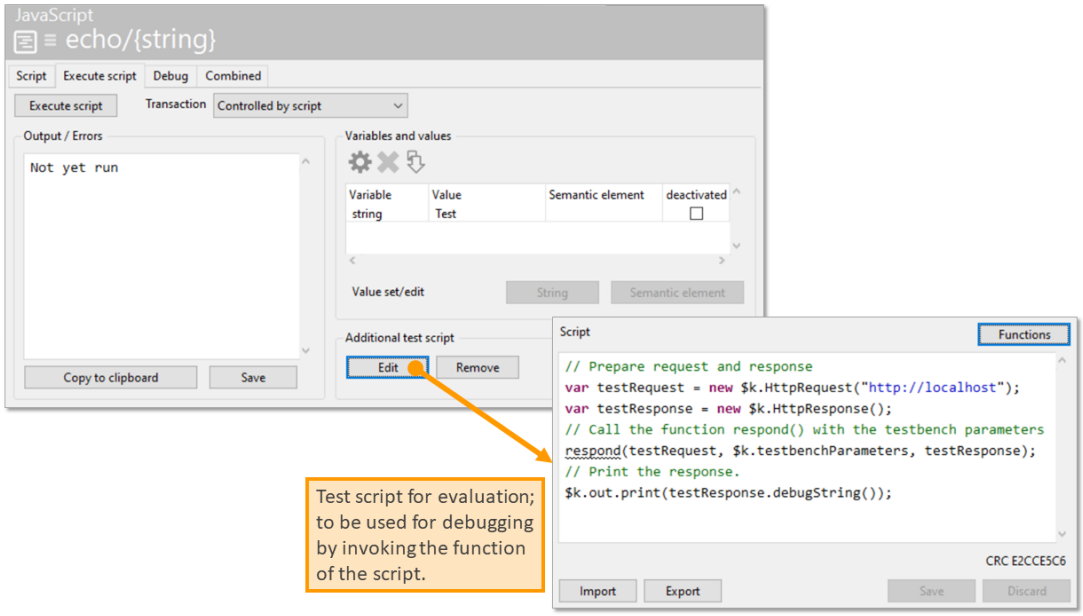
Funktion	Beschreibung
Import/Export	Ermöglicht das Importieren/Exportieren von *.js-Dateien vom/ins Dateisystem des PCs.
Funktionen	Listet alle im Code verwendeten und benannten Funktionen auf. Bei Auswahl einer Funktion springt der Editor an die Stelle, an der sich der Funktionsaufruf befindet.
Speichern	Speichert die Änderungen im Code (Shortcut: Strg + S).
Verwerfen	Verwirft alle Änderungen seit dem Zeitpunkt des letzten Abspeicherns.

1.4.4.2. Skript ausführen

Funktionen: Ausführen des Skripts, Anzeige des Outputs (Variablen und deren Werte), Anlegen eines Test-Skriptes für das Debugging.

Das Skript in der folgenden Abbildung ist ein Beispiel für das Testen einer REST-Anfrage ("restlet") im Knowledge-Builder. Es wird definiert im Skript-Editor des Reiters "Skript ausführen", dort bei "Additional test script".

Breakpoints zum Debuggen können im Reiter "Debug" gesetzt werden.

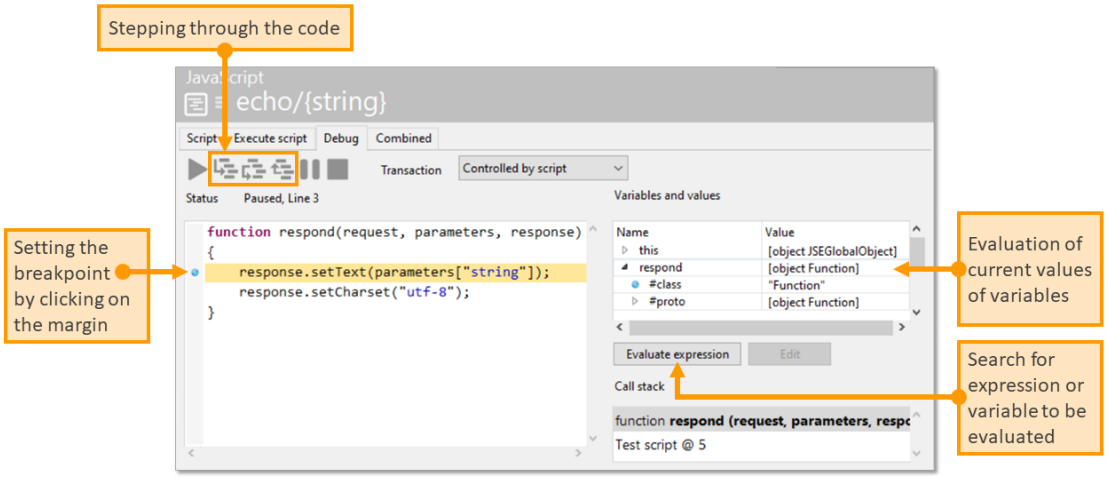


Funktion	Beschreibung
Skript ausführen	Das Skript wird in einem Durchgang ausgeführt.
Transaktion	Schreibende Vorgänge (bspw. Erzeugen oder Löschen von Objekten) innerhalb eines Skriptes erfordern eine Transaktion. Wenn das Skript innerhalb einer Aktion des Web-Frontends ausgeführt wird, dann geschieht dies automatisch innerhalb einer Transaktion. Zum Ausführen des gleichen Skripts ohne Web-Frontend oder zum Debuggen muss die Transaktion gesondert hervorgerufen werden durch eine der folgenden Optionen: • **Durch Skript gesteuert:** In diesem Fall muss im Skript ein Code enthalten sein, der die schreibenden Funktionen in einer Transaktion kapselt. Für das Anlegen einer Transaktion siehe hierzu die i-views JavaScript-API-Referenz. • **Nur lesen:** Erlaubt das Ausführen/Debuggen des Skriptes, solange keine schreibenden Transaktionen ausgeführt werden. • **Lesen und Schreiben:** Die Transaktionen werden, wo benötigt, im Hintergrund automatisch ausgeführt.
In Zwischenablage kopieren	Kopiert die Ausgabe in die Zwischenablage.
Speichern	Speichert die Ausgabe auf dem Dateisystem des PCs ab.

HINWEIS Die Konfiguration für Variablen und das zusätzliche Testskript wurde in Version 5.7 entfernt.

1.4.4.3. Debug

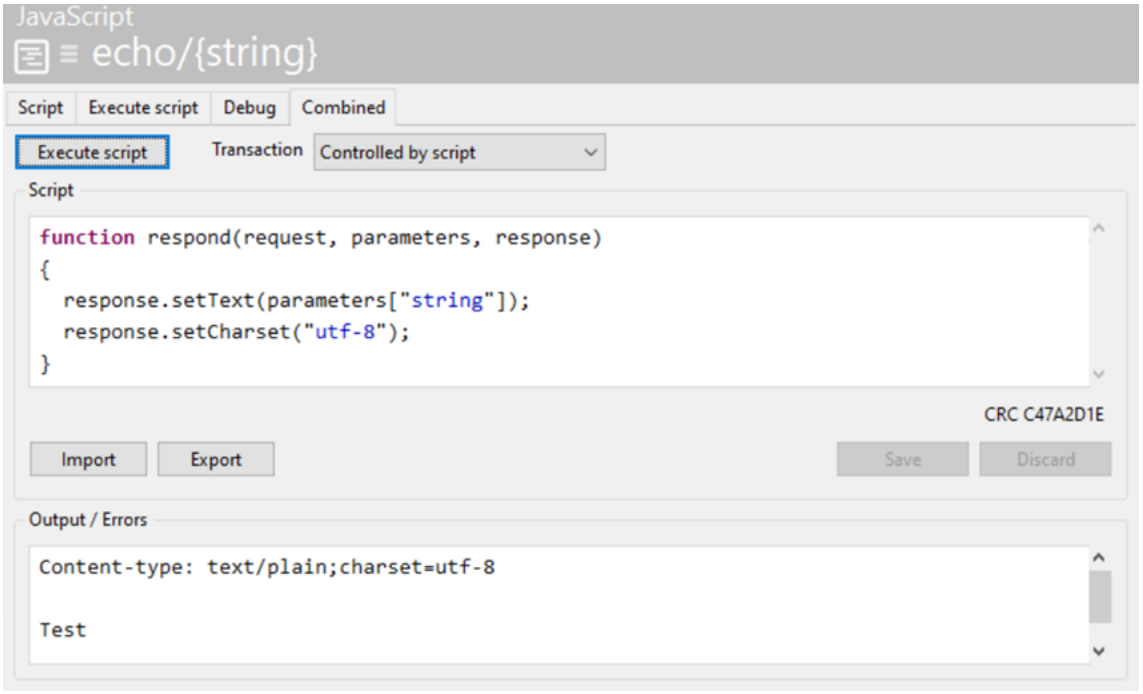
Funktionen: Setzen von Breakpoints, schrittweises Abrufen des Codes, Auswerten von Ausdrücken zwischen den Abarbeitungsschritten



Funktion	Beschreibung
Starten/Weiterlaufen (F4)	Startet die Ausführung des Skriptes, wenn keine Breakpoints gesetzt sind oder das (schrittweise) Debuggen des Skripts, wenn mindestens ein Breakpoint gesetzt wurde. Achtung: Wenn der Breakpoint auf eine auskommentierte Zeile gesetzt ist, dann wird der Breakpoint ignoriert.
Einzelschritt (F5)	Führt nur den nächsten logischen Schritt aus.
Einzelschritt (kompletter Block) (F6)	Führt den aktuellen Block komplett aus.
Aus Kontext zurückkehren (F7)	Bewirkt das Ausführen des referenzierten Codes und kehrt zum ursprünglich aufgerufenen Code zurück.
Anhalten (F9)	Unterbricht (pausiert) das Ausführen des Codes. Beim Debugging des Codes fährt der Debugger trotzdem bis zum nächsten Breakpoint fort.
Beenden (F10)	Beendet das Ausführen bzw. Debuggen des Skriptes.
Ausdruck auswerten	Dient zur Auswertung eines Variablenwertes, nachdem der Debugger an einem Breakpoint im Skript angelangt ist.
Bearbeiten	Wenn eine Variable selektiert ist, die auf ein Knowledge-Graph-Element verweist, öffnet dieser Button ein Bearbeitungsfenster auf dem Element.

1.4.4.4. Kombiniert

Funktionen: Kombiniert die Skript-Ausführung und den Output in einer Ansicht



1.4.5. API-Erweiterungen

1.4.5.1. Zusätzliche Funktionen

Die API kann erweitert werden, indem eigene Funktionen bei Prototyp-Objekten hinzufügt. Im folgenden Beispiel werden einige Prototypen erweitert, um Schemainformation mit printSchema() auszugeben.

```
// Print the schema of the instances and subtypes of a type
$k.Type.prototype.printSchema = function () {
  this.typesDomain().printSchema("Type schema of \"" + this.name() + "\"
");
  this.instancesDomain().printSchema("Instance schema of \"" + this.name()
+ "\"");
  this.subtypes().forEach(subtype => subtype.printSchema());
}

// Print information about a property type
$k.PropertyType.prototype.logPropertySchema = function () {
  $k.out.print("\t" + this.name() + "\n");
}

// Attribute types print their type
$k.AttributeType.prototype.logPropertySchema = function () {
  $k.out.print("\t" + this.name() + " (Attribute of type " + this
.valueRange().type() + ")\n");
```

```
}

// Relation types print their target domains
$k.RelationType.prototype.logPropertySchema = function () {
  $k.out.print("\t" + this.name());
  const inverse = this.inverseRelationType();
  if (inverse) {
    const inverseDomains = inverse.domains();
    if (inverseDomains.length > 0) {
      $k.out.print(" (Relation to ");
      let separate = false;
      inverseDomains.forEach(function (inverseDomain) {
        if (separate)
          $k.out.print(", ");
        else
          separate = true;
        $k.out.print("\"" + inverseDomain.type().name() + "\"");
      });
      $k.out.print(")");
    }
  }
  $k.out.cr();
}

// Print all properties defined for a domain
$k.Domain.prototype.printSchema = function (label) {
  const definedProperties = this.definedProperties();
  if (definedProperties.length > 0) {
    $k.out.print(label + "\n");
    definedProperties.sort((p1, p2) => p1.name().localeCompare(p2.
name()));
    definedProperties.forEach(propertyType => propertyType
.logPropertySchema());
  }
}

// Print the entire schema
$k.rootType().printSchema();
```

1.4.5.2. Eigene Prototypen definieren

Der Prototyp eines Wissensnetzelements ist normalerweise vordefiniert (Instance, Relation usw.). Es ist aber auch möglich, eigene Prototypen zu definieren und Objekten von bestimmten Typen mit der Funktion mapInstances(internalName, prototype) zuzuordnen.

Beispiel: Ein Warenkorb-Prototyp

```
// Define a Basket prototype with a function totalPrice()
function Basket() { }

Basket.prototype.totalPrice = function() {
  return this.relationTargets("contains").reduce(
    (sum, item) => sum + item.attributeValue("price"),
    0
  );
}

// Set the prototype of instances of the basket type
$k.mapInstances("Basket", Basket);

// Print the total price of all baskets
const baskets =  $k.Registry.type("Basket").instances();
for (let basket of baskets)
  $k.out.print(basket.totalPrice() + "\n");
```

Für die Verwendung in anderen Skripten muss zunächst das Modul geladen werden:

```
$k.module("myBasketSkript");
const basket = $k.Registry().elementWithID("ID_123");
$k.out.print(basket.totalPrice() + "\n");
```

1.4.6. Status der JavaScript-Engine in i-views

Die JavaScript-Engine ist weitestgehend auf den Stand von ECMAScript 14. Dies ermöglicht u.a. die Verwendung von:

- ECMAScript-Modulen
- Klassen
- Map und Set
- Arrow-Funktionen

Die Modul-Syntax, die Import/Export-Deklarationen ermöglicht, muss beim Skript explizit aktiviert werden (analog zur Dateiendung `.mjs` in Node.js). Dabei ist zu beachten, dass diese zwei Modul-Arten unterschiedlich definiert und verwendet werden:

- Bei ECMAScript-Modulen mit `$k.module` bzw. `$k.require`
- klassisch per `$k.define` und `module.exports`.

Nicht unterstützt werden folgende Objekte:

- Strukturierte Arrays
  - Typed Arrays (z.B. Int8Array)
  - ArrayBuffer
  - SharedArrayBuffer
  - DataView
  - Atomics
- WeakRef / WeakSet / WeakMap
- Proxy
- Internationalization API

Die Implementierung weicht vom Standard in folgenden Punkten ab:

- Die Reihenfolge der Evaluierung von Promises entspricht nicht vollständig der Spezifikation. Dies betrifft auch asynchrone Generatoren.
- Subklassen von folgenden Standard-Klassen führen bei Funktionsaufrufen zu Fehlern: Array, BigInt, Boolean, Date, Number, RegExp, String, Symbol

1.5. REST-Services

Das REST Interface can verwendet werden, um Lese- und Schreibzugriffe auf den Knowledge Graph zu konfigurieren. Dafür müssen *Ressourcen* und *Services* definiert werden. Ressourcen beschreiben das Verhalten des Interface, wenn auf diese zugegriffen wird. Das Verhalten einer Ressource wird von einem Skript gesteuert. Zusätzlich können auch vordefinierte Ressourcen benutzt werden. Services fassen mehrere Ressourcen zusammen.

Ein Zugriff wird über eine HTTP-Anfrage eingeleitet. Diese sind wie folgt strukturiert:

```
https://<hostname>:<port>/[<service-pfad>||<service-id>]/<ressource-pfad-
und-parameter>
```

1.5.1. Konfiguration

Die REST-Komponente muss im Knowledge Graph hinzugefügt werden. Sie definiert das benötigte Schema, welches im Bereich "Technik" unter "REST" im Knowledge Builder zu finden ist.

Das REST Interface wird üblicherweise vom Bridge Service bereitgestellt. Es reagiert auf HTTP-Anfragen unter Verwendung der REST-Konfiguration im Knowledge Graph. Das Interface ist bereits in der Testversion des Knowledge Builders enthalten, sodass kein Bridge Service benötigt wird.

Änderungen an der Konfiguration im Knowledge Graph werden nicht auf aktuell laufende Interfaces angewendet. Dies passiert nur dann, wenn im Menü des Knowledge Builders unter "Administrator" > "Update REST interface" ausgeführt wird.

Der Bridge Service benötigt eine passende Konfigurationsdatei (bridge.ini). Der Name des Servers (host), der Knowledge Graph (volume) und die REST Service ID werden hier angegeben. Die Zeile "services" kann ganz ausgelassen werden, damit alle existierenden Service-Objekte automatisch aktiviert werden.

```
[Default]
host=localhost
loglevel=10

[KHTTPRestBridge]
volume=demo
port=8086
services=core,extra
```

1.5.2. 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.

1.5.3. Ressourcen

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

1.5.3.1. Methoden

Eine Resource ist mit einer oder mehreren *Methoden* verknüpft. Dies definert sowohl die Antwort auf die Anfrage als auch die unterstützten Ein- und Ausgabetypen (Inhaltstypen). Die Methoden und Typen der HTTP-Anfrage werden verwendet, um eine entsprechend konfigurierte Methode auszuwählen.

In der Strukturansicht werden die Methoden als Unterlemente von Ressourcen angezeigt. Diese können hier erstellt und gelöscht werden.

Eigenschaft	Beschreibung
HTTP method	Unterstütze HTTP-Methoden (GET, POST, PUT, DELETE). Mehrere Einträge sind möglich.
Input media type	Nur POST/PUT: Erwarteter Inhaltstyp des Inhalts der Anfrage.
Output media type	Inhaltstyp der Antwort. Wenn die Anfrage einen erwarteten Inhaltstyp via `Accept` spezifiziert, muss der Ausgabetyp damit übereinstimmen.
Script	Ein registriertes Skript für die Definition der Antwort (nur relevant bei Skript-Ressourcen)
Transaktion	Transaktionssteuerung (nur relevant bei Skript-Ressourcen)

Die Transaktionssteuerung ist für Schreibzugriffe auf den Knowledge Graph relevant, da diese nur in einer Transaktion ausgeführt werden können.

Transaktionssteuerung	Beschreibung
Automatisch	Nur für GET Lesezugriffe; Für POST/PUT/DELETE wird das Skript in einer Transaktion ausgeführt. Dies ist die Standardeinstellung.
Durch Skript gesteuert	Keine Transaktion; das Skript übernimmt die Kontrolle.
Lesen	Nur für Lesezugriffe; das Skript kann keine Transaktion starten.
Schreiben	Das Skript wird in einer Transaktion ausgeführt.

1.5.3.2. 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) {
  const albums = $k.Registry.query("albums").findElements(parameters);
  const albumData = albums.map(album => ({
    name: album.name(),
    id: album.idString()
  });
  response.setText(JSON.stringify(albumData, undefined, "\t"));
  response.setContentType("application/json");
}
```

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.

1.5.3.3. Built-In Ressourcen

Built-In Ressourcen 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).

Rest resource ID	Methode	Beschreibung
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.
ObjectListPrintTemplateResourceWithFilename	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.

1.5.3.3.1. 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 Multipart 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.

1.5.3.3.2. Topic icon

Mit dem angegebenen Pfad kann eine Bilddatei für ein bestimmtes semantisches Element geladen werden. Hat ein Element keine eigene Bilddatei, dann wird die Bilddatei des Typs vererbt. Der optionale Parameter `size` wird verwendet, um die passende Bildgröße auszuwählen, wenn mehrere Größen eines Bildes im Knowledge Graph hinterlegt sind.

```
http://{server:port}/baseService/topicIcon/{topicID}?size=10
```

1.5.3.3.3. 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.

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",
        "project A, project B"
      ]
    },
    {
      "topicID": "ID987_123456789",
      "row": [
        "MF",
        "Musterfrau",
        "Maxine",
        "222",
        "m.musterfrau@email.net",
        "10",
        "8",
        "2000-01-01",
        "project X, project Y, project Z"
      ]
    }
  ],
  "columnDescriptions": [
    {
      "label": "Login",
      "type": "string",
      "columnId": "1"
    },
    {
      "label": "Last name",
      "type": "string",
      "columnId": "2"
    },
    {
      "label": "First name",
      "type": "string",
      "columnId": "3"
    },
    {
      "label": "Telephone extension",
      "type": "string",
      "columnId": "4"
    },
    {
      "label": "email",
      "type": "string",
      "columnId": "5"
    },
    {
```

```
      "label": "Availability",
      "type": "number",
      "columnId": "6"
    },
    {
      "label": "Expenditure",
      "type": "string",
      "columnId": "7"
    },
    {
      "label": "created on",
      "type": "dateTime",
      "columnId": "8"
    },
    {
      "label": "Project",
      "type": "string",
      "columnId": "9"
    }
  ]
}
```

1.5.3.3.4. Object list print template

The following path can be used to fill an object list in a "print template for list" and download the result:

```
http://{server:port}/baseService/{conceptLocator}/objectList/printTemplate
/
```

```
{templateLocator}/{filename}
```

The service functions exactly the same way as retrieving an object list, however, as an additional parameter, features a reference to the individual of the type "print template for list" in the Knowledge Graph.

`templateLocator` must have one of the formats described under *General*

The optional path parameter `filename` is not evaluated, and is used to improve browser performance.

The header field `Accept` is used to control the output format into which conversion occurs. If there is no header field, or the value is `*/*`, no conversion occurs. Accept with multiple values is not

supported and will result in an error message.

The optional query parameter `targetMimeType` is used to overwrite the value of the `Accept` header field. This is necessary when the user would like to call the request from a browser, and has no influence on the header fields.

1.5.3.3.5. 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.

1.5.3.3.6. 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"
```

1.5.3.4. 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:{bridge-port}/test/icons/bullet.png
```

1.5.3.5. Ressourcen-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:

Eigenschaft	Wert
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.

1.5.4. Authentifizierung

Jeder Ressource kann ein Authentifizierungsobjekt zugewiesen werden, um den Zugriff einzuschränken und die Anfrage an ein Benutzerobjekt zu binden.

Authentifizierungen werden als Objekte vom Typ *Authentication* im Bereich *REST* definiert. Ressourcen wird ein Authentifizierungsobjekt mit der Relation *Authentication* zugewiesen.

Die Art der Authentifizierung wird durch das Attribut *Authentication type* des Authentifizierungsobjekts definiert. Einige Konfigurationswerte sind nur für bestimmte Authentifizierungstypen verfügbar.

Authentication name

Beliebiger Name zur Unterscheidung von Konfigurationen

Cache duration

Die Benutzersuche wird für diesen Betrag (in Sekunden) zwischengespeichert. Siehe Login-Konfiguration.

Trusted login

Anmeldeinformationen (z. B. das Kennwort) werden bei Aktivierung nicht überprüft

1.5.4.1. Authentifizierungsverfahren

No authentication

Wie der Name schon sagt, wird keine Authentifizierung durchgeführt und somit ist standardmäßig kein Benutzer aktiviert. Wenn eine REST-Ressource, die diese Authentifizierung verwendet, eine Fallback-Benutzerinstanz definiert, wird die Anfrage an dieses Objekt gebunden.

Basic

Authentifiziert einen Benutzer mit dem durch RFC 7617 definierten Basic-Verfahren. Erfordert die Identifizierung eines Benutzerobjekts mit einem passenden Kennwort. Siehe Login-Konfiguration.

Bearer

Verwendet JSON-Webtoken, die durch RFC 7519 definiert sind, um Anfragen zu authentifizieren.

Benutzer werden durch den Subject Claim identifiziert. Der Wert dieses Claims ist die Element-ID des Benutzerobjekts, es sei denn, in der Authentifizierungskonfiguration wird ein *Token subject attribute* angegeben, das den Wert eines Attributs der Benutzerobjekte angibt. Die Login-Konfiguration wird hier nicht verwendet.

Standardmäßig kann das Token als Header-, Cookie- oder Abfrageparameter übergeben werden. Dies kann über die Attribute *Allow cookie*, *Allow authorization header* und *Allow query parameter* angepasst werden.

Das Attribut *Allow cookie* kann so eingestellt werden, dass zusätzliche Cookie-Parameter definiert werden, die als Meta-Attribut angegeben werden können. Der Wert wird in der Antwort dem Header *Set-Cookie* hinzugefügt.

Das Attribut *Cookie/parameter name* definiert den Namen des Cookie-/Abfrageparameters.

Token expiry interval und *Token renew interval* definieren die Lebensdauer neuer Token.

Negotiate

Verwendet Windows Negotiate, definiert durch RFC 4559, um Authentifizierungsanfragen zu verarbeiten. Dieser Authentifizierungstyp wird nur unter Windows unterstützt.

Es wird nicht empfohlen, dieses Authentifizierungsschema zu verwenden. Negotiate authentifiziert Verbindungen und nicht Anforderungen, was für den Lastenausgleich nicht geeignet ist.

Scripted

Dieser Authentifizierungstyp verwendet ein Skript mit vier benutzerdefinierten Funktionen, um die Benutzerauthentifizierung zu behandeln. Dies ist nützlich, um die Authentifizierung an externe zentrale Authentifizierungsanbieter wie OpenID Connect auszulagern oder um benutzerdefinierte Authentifizierungsschemas zu implementieren.

HINWEIS	Während alle Skriptfunktionen über sinnvolle Standardimplementierungen verfügen, die jeden unbefugten Zugriff verhindern, ist es aufgrund unvorsichtiger Implementierungen leicht möglich, Teile des REST-Dienstes allgemein verfügbar zu machen.

Jede Funktion erhält das Anfrageobjekt, ein Objekt mit den Abfrageparametern der Anfrage und ein Antwortobjekt als Eingabe. Die Hauptauthentifizierungsfunktion, die bei jeder Anfrage aufgerufen wird, heißt **authenticate**:

```
function authenticate(request, parameters, response) {
    const encryptedToken = request.cookies().access_token
    if (encryptedToken) {
        const token = $k.JWT.parse(encryptedToken)
        try {
            token.verify()
            return $k.Registry.elementWithID(token.payload().sub)
        } catch (e) {}
    }
    response.setCode(302)
    response.setHeaderField("Location",
"http://auth.example.com?return_url=" + request.url())
}
```

In diesem Beispiel erwartet die Authentifizierungsroutine ein verschlüsseltes JWT-Token mit dem Namen **access_token**, das als Cookie mit der Anfrage gesendet wird. Wenn das Token vorhanden ist und erfolgreich überprüft wurde, kann ein Benutzerobjekt aus der Nutzlast des Tokens abgeleitet werden. Die Anfrage wird dann unter diesem Benutzer ausgeführt. Andernfalls wird eine Umleitung (HTTP 302) zu einem externen Authentifizierungsanbieter ausgelöst. Es wird implizit davon ausgegangen, dass der Authentifizierungsanbieter das access_token-Cookie nach erfolgreicher Authentifizierung so setzt, dass die nächste Anfrage an diesen Endpunkt erfolgreich ist.

Wenn die Authentifizierungsfunktion keine Benutzerinstanz oder `null` zurückgibt, wird die Authentifizierung als gescheitert angesehen und die Anfrage wird nicht weiter ausgeführt. Die Funktion kann `null` zurückgeben, wenn die Authentifizierung erfolgreich ist, aber kein Benutzer abgeleitet werden kann. Die Anfrage wird dann ohne aktiven Benutzer weiter ausgeführt.

Die Funktionen **login**, **logout** und **renew** können für komplexere Authentifizierungsschemata implementiert werden, die das Abmelden von Benutzern usw. unterstützen. Sie werden aufgerufen, wenn der jeweilige Built-in-Request (`accessToken/login`, `accessToken/logout`, `accessToken/renew`) aufgerufen wird. Diese werden auch für den Authentifizierungsfluss des View-Config-Mappers verwendet und müssen daher implementiert werden, um eine reibungslose Interaktion mit diesem zu gewährleisten.

1.5.4.2. Login-Konfiguration

Die Basic-Authentifizierung sucht nach Benutzerobjekten mit einer Login-Suche. Die Login-Suche kann durch Erstellen einer Abfrage mit dem Registrierungsschlüssel *login* definiert werden. Wenn es sich bei der Abfrage um eine strukturelle Abfrage handelt, wird die Identität an jeden definierten Abfrageparameter übergeben.

Wenn keine Login-Suche konfiguriert ist, wird das Benutzerobjekt durch Suche nach dem Wert des Attributs *login* der Benutzerobjekte ermittelt. Dieses Attribut wird automatisch im Schema definiert, wenn der Benutzertyp in den Zugriffsrechteeinstellungen konfiguriert ist.

Der Typ der Benutzerobjekte wird durch die Zugriffsrechteeinstellungen definiert. Die gefundenen Objekte müssen dem angegebenen Typ entsprechen, andernfalls schlägt die Authentifizierung fehl.

Wenn für die Identität nicht oder mehr als ein Benutzerobjekt gefunden wird, schlägt die Authentifizierung fehl.

Der Hashwert des angegebenen Kennworts wird mit dem Hashwert verglichen, der im entsprechenden Attribut des Benutzerobjekts gespeichert ist. Die Authentifizierung schlägt fehl, wenn sie nicht übereinstimmen. Die Kennwortprüfung wird übersprungen, wenn in der Authentifizierungskonfiguration *Trusted login* aktiviert ist.

Die Anforderung wird dann an den Benutzer gebunden, z.B. werden Zugriffsrechte für diesen Benutzer überprüft.

1.5.5. 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
```

1.5.6. OpenAPI-Dokumentation

i-views bietet die Möglichkeit, OpenAPI-3.0-Dokumentation für konfigurierte Services zu generieren. Dazu können Service- und Ressourcen-Konfigurationen mit ergänzenden Dokumentationsdaten angereichert werden.

1.5.6.1. Konfiguration

1.5.6.1.1. Service

Eigenschaft	Beschreibung	Abbildung auf OpenAPI 3.0
Service Description	Freitextbeschreibung des Services, unterstützt GitHub Flavored Markdown	`info.description`
Service Version	Versionsangabe, die als Semantic Version interpretiert wird.	`info.version`
Service ID		`info.title`
OpenAPI components	Skript, welches wiederverwendbare OpenAPI-3.0-Komponenten als JSONObjekt erzeugt.	`components`

1.5.6.1.2. Resource

Eigenschaft	Beschreibung	Abbildung auf OpenAPI 3.0
Resource Description	Freitextbeschreibung der Resource, unterstützt GitHub Flavored Markdown	`paths.{path}.description`

1.5.6.1.3. Method

Die angegebenen Abbildungen auf OpenAPI-Elemente sind jeweils als relativ zu paths.{path}.{method} zu betrachten.

Eigenschaft	Beschreibung	Abbildung auf OpenAPI 3.0
Method description	Freitextbeschreibung der Ressource, unterstützt GitHub Flavored Markdown	`.description`
Request Body	Siehe Abschnitt *Request Body*	`.requestBody`
Response	Siehe Abschnitt *Response*	`.responses.{code}`

1.5.6.1.4. Parameter

Die angegebenen Abbildungen auf OpenAPI-Elemente sind jeweils als relativ zu paths.{path}.{method}.parameters.{index} zu betrachten.

Eigenschaft	Beschreibung	Abbildung auf OpenAPI 3.0
Parameter Description	Freitextbeschreibung des Parameters, unterstützt GitHub Flavored Markdown	`.description`
Parameter name		`.name`

Eigenschaft	Beschreibung	Abbildung auf OpenAPI 3.0
Repeating	Bei Path-Parametern darf der Haken NICHT gesetzt sein.	`.explode: true`
		`.schema: {"type": "array"}`
Required	Bei Path-Parametern MUSS der Haken gesetzt sein.	`.required`
Style		`.in`
Type		`.schema`

1.5.6.1.5. Request Body

Die angegebenen Abbildungen auf OpenAPI-Elemente sind jeweils als relativ zu paths.{path}.{method}.requestBody zu betrachten.

Eigenschaft	Beschreibung	Abbildung auf OpenAPI 3.0
Request Body Description	Freitextbeschreibung des Bodies, unterstützt GitHub Flavored Markdown	`.description`
Required		`.required`
Media type	Ersetzt den *Input media type*, der in i-views 5.3 noch an der Method hinterlegt werden konnte, da jetzt mehrere mögliche Anfrageformate beschrieben werden können. Siehe Abschnitt *Media Type*.	`.content.{mediaType}`

1.5.6.1.6. Response

Für eine valide OpenAPI-Dokumentation muss für jeden Request mindestens eine mögliche Response dokumentiert sein. Die angegebenen Abbildungen beziehen sich jeweils auf das Response-Objekt.

Eigenschaft	Beschreibung	Abbildung auf OpenAPI 3.0
Response Code	HTTP-Status-Code der Response	Key
Response Description	Freitextbeschreibung der Response, unterstützt GitHub Flavored Markdown	`.description`
Media type	Ersetzt den *Output media type*, der in i-views 5.3 noch an der Method hinterlegt werden konnte, da jetzt mehrere mögliche Responseformate beschrieben werden können. Siehe Abschnitt *Media Type*.	`.content.{mediaType}`

1.5.6.1.7. Media Type

Ab OpenAPI 3.0 kann über die Angabe mehrerer Media types dokumentiert werden, dass ein Request mehrere mögliche Ein- bzw. Ausgabeformate anbietet.

Eigenschaft	Beschreibung	Abbildung auf OpenAPI 3.0
Media Type Name	Der MIME-String, der den Media Type definiert.	Key
OpenAPI schema	Skript, welches ein JSON-Schema-Objekt erzeugt, das das Format der Struktur mit diesem Media Type beschreibt.	`.schema`

1.5.6.1.8. JSON-Schema-Definitionen

An verschiedenen Stellen können Skripte angebracht werden, die JSON-Schema zur weiteren Beschreibung von Ein- und Ausgaben erzeugen. Unterstützt wird ein Subset des JSON-Schema-Standards, welches sich der OpenAPI-Spezifikation entnehmen lässt.

Beispielskript *OpenAPI components* :

```
function openAPIComponents() {
    return {
        "schemas": {
            "Example": {
                "properties": {
                    "id": { "type": "integer" },
                    "name": { "type": "string" }
                }
            }
        }
    }
```

```
}
```

Beispielskript *OpenAPI schema* mit Referenz auf obige Definition:

```
function swaggerJSONSchema() {
    return {
        "$ref": "#/components/schemas/Example"
    }
}
```

1.5.6.2. Generierung der API-Dokumentation

1.5.6.2.1. Manuelle Generierung im KB

Zur manuellen Generierung einer .json-Datei mit der OpenAPI-Dokumentation mit dem Knowledge-Builder gibt es an der Service-Konfiguration den Button *Als OpenAPI 3.0 exportieren* über der Liste der Services.

1.5.6.2.2. CLI

Der gleiche Export wird auch über das Command Line Interface angeboten:

```
bridge.exe -exportBuiltInRequestAPI {filename} {serviceID}
```

1.5.6.2.3. Als REST-API-Endpoint

In i-views 5.4 gibt es eine BuiltIn-Resource *APIResource*, die die API-Dokumentation zur Verfügung stellt. Diese kann über den Button zum Anlegen einer neuen Resource dem entsprechenden Service hinzugefügt werden und ist fortan unter /api bzw. dem konfigurierten Pfad verfügbar.

1.6. Berichte und Drucken

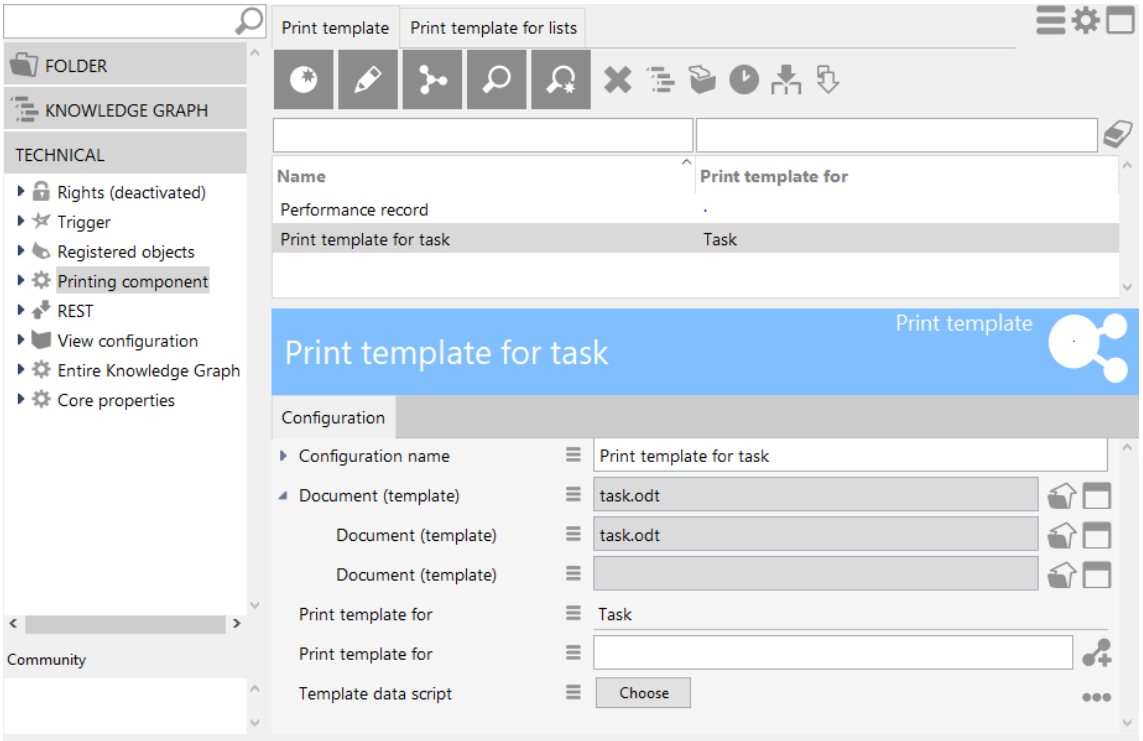
You can use the printing component to use document templates (ODT/DOCX/XLSX/RTF files) with KPath expressions on objects or object lists and then use them to generate an adapted output file, which can be either printed or stored.

The adding of the printing component via the Admin tool creates configuration schemas for objects ("print template") and lists ("print template for lists") in the Knowledge Graph. The existence of this component is prerequisite for the print function being available in Knowledge Builder or via the REST interface.

1.6.1. Druckvorlagen erstellen

In Knowledge Builder, print templates are created in the "Technical > Printing component" area. Each print template object contains a print template document (ODT, DOCX, RTF) and a relation that specifies to which objects the print template is to be applied.

The following example shows an ODT print template for objects of the "Task" type.



The following chapters explain how print template documents are created.

1.6.1.1. RTF-Vorlagen erstellen

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 gmbhJulius-Reiber-Str. 1764293 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.

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:

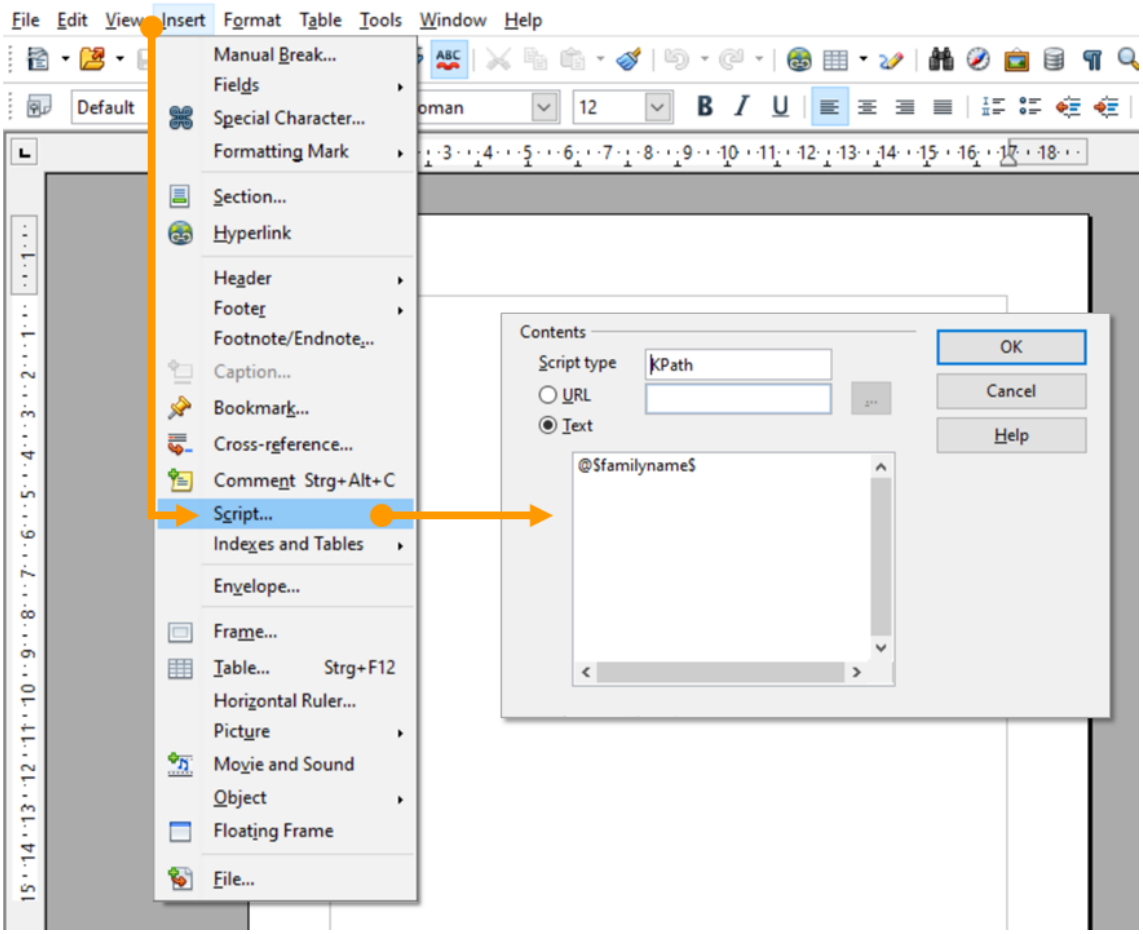
Spalte1	Spalte2
Meier	Peter
Schulze	Helmut

1.6.1.2. ODT-Dokumente (OpenOffice) erstellen

Printing using the ODT format (Open Document Text, open standard) has many advantages compared to the RTF format:

- The embedded script instructions are not part of the text, and are instead filed in special script elements. This ensures that the formatting is not destroyed by lengthy scripts.
- The ODT format supports a large set of format instructions (comparable with MS Word) that RTF cannot process.
- As a format, RTF does not have a uniform standard (MS Word can, for example, "do more" than the standard).
- Editing of the RTF templates is highly fragile. MS Word, above all, tends to "supplement" the templates with control elements (for example, the cursor position current during the most recent editing), preventing the scripts from being reliably identified.

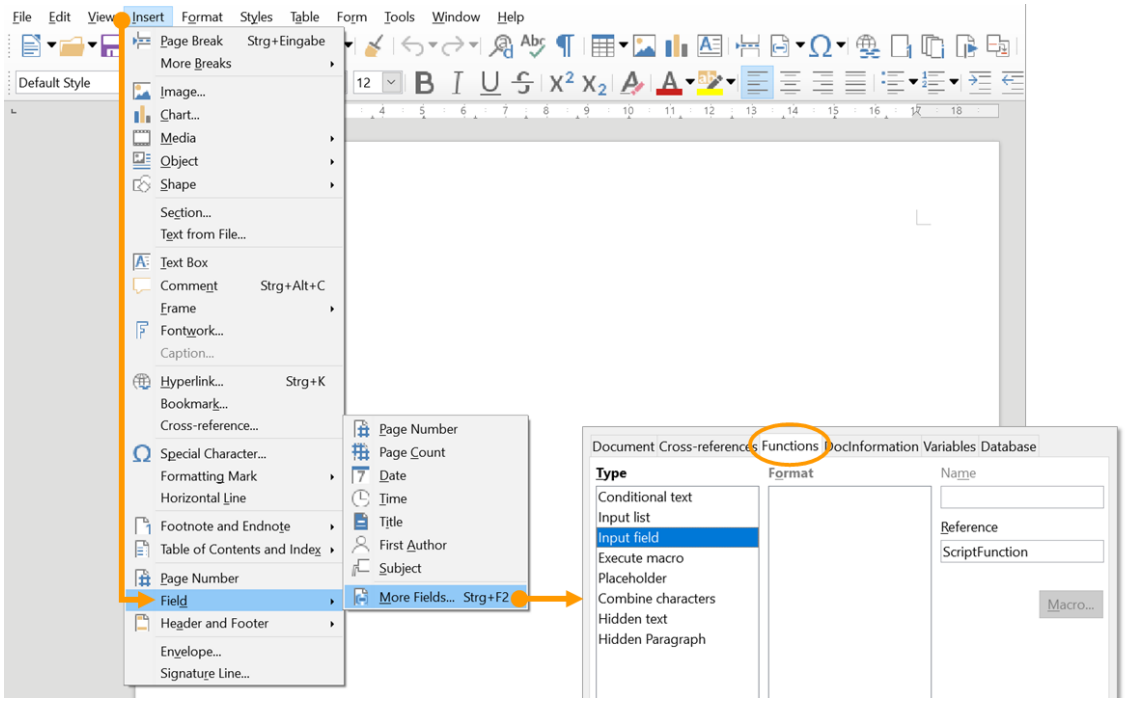
ODT templates can be created using OpenOffice or LibreOffice. They are created the same way as RTF templates are created, with the only difference being that the path/script instructions are saved in script elements, as the following diagram shows.



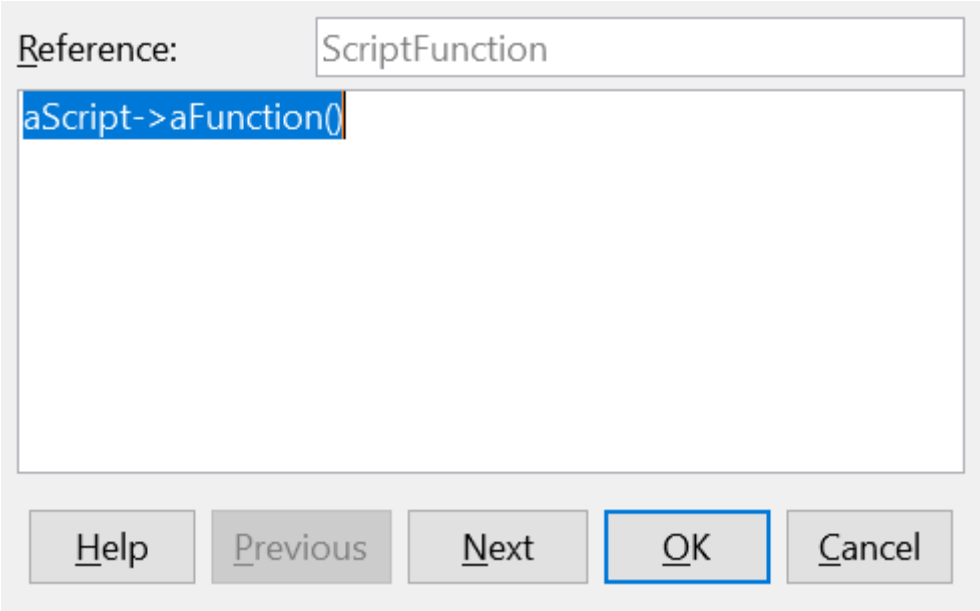
The script field can no longer be integrated in LibreOffice 5. As an alternative to this, the "Input field" can be used:

Insert > Field command > Other field commands (alternative keyboard shortcut Ctrl+F2)

The input field is found there on the "Functions" tab.



"Note" is equivalent to the previous "Script type"; after clicking on insert, another window opens in which the script can be entered.



Available script types

There are the following script types:

- **KPath** : analogous to **KPATH_EXPAND**
- **KScript** : analogous to **KSCRIPT_EXPAND**
- **KPathRows** : analogous to **KPATH_ROWS**

- **KPathImage** : for embedding images
- **ScriptFunction** : Calls a function of a registered script. A string with the following format is expected as text:

```
ScriptID->Functionname()
```

The function call is automatically expanded by two arguments: the semantic element and the variables determined by the environment

An example of a script that was called:

```
function headerLabel(element, variables)
{
   return element.name().toLocaleUpperCase();
}
```

- **ScriptRowsFunction** : Analogous to ScriptFunction. Table rows are generated for the returned objects, analogous to KPathRows.
- **ScriptImageFunction:** for adding bitmap images
- **ScriptSVGImageFunction:** for adding SVG drawings * **DataPath:** The "script for generating JSON contents" must be set on the print template. The corresponding key can now be used to access the values of the JSON object.

Example of generating the JSON object:

```
function templateData(element)
{
    return {
            name: element.name(),
            idNumber: element.idNumber(),
            someData: { idString: element.idString() }
    }
}
```

To access the value idString, for example,

```
someData.idString
```

must be set as text. * **DataRowsPath:** In table rows or sections (Libre Office only), DataRowsPath can be used to transform an array of objects in the templateData JSON to a table or sequence of sections in the printed document. Each object in the array is transformed into a new row with

identical formatting as the row the DataRowsPath element is placed in. This allows having lists of variable length in the printed document. DataPath and DataConditionPath elements in the same table row or section as a DataRowsPath element are interpreted relative to the path of the DataRowsPath element.

```
function templateData(element) {
    return {
        rowData: [
            { name: "Element 1", someValue: 123 },
            { name: "Element 2" }
        ]
    }
}
```

- **DataConditionPath:** Like DataRowsPath elements, DataConditionPath can be placed in table rows or sections. Unlike DataRowsPath elements, DataConditionPath can reference anything in the templateData JSON, not only arrays of objects. When the referenced property in the templateData JSON is a JavaScript falsy value (false, undefined, null, 0 or an empty String) or an empty Array, the table row or section the DataConditionPath element is placed in is removed from the printed document.

File attributes or URLs can be used for embedding images. When URLs are used, an attempt is made to load an image from the address specified.

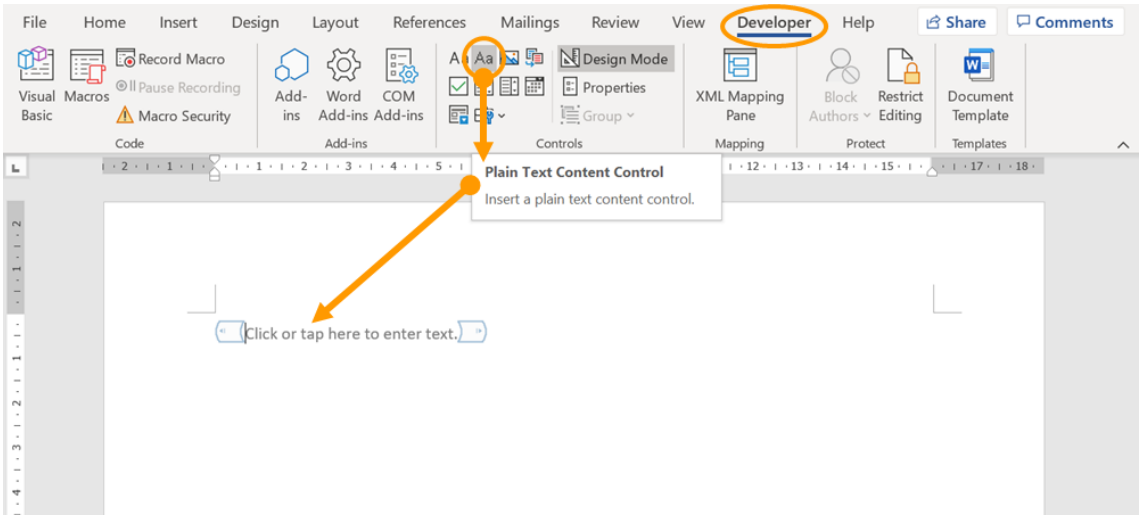
Embedded images are always sourced in their original size (at 96d dpi). If another size should appear in the printout, a frame with the required dimensions (absolute dimensions in cm must be used!) must be built around the script element. The resulting embedded image is then fit into the frame so that the frame dimension is not exceeded while retaining the image aspect ratios.

1.6.1.3. DOCX-Dokumente (Microsoft Word) erstellen

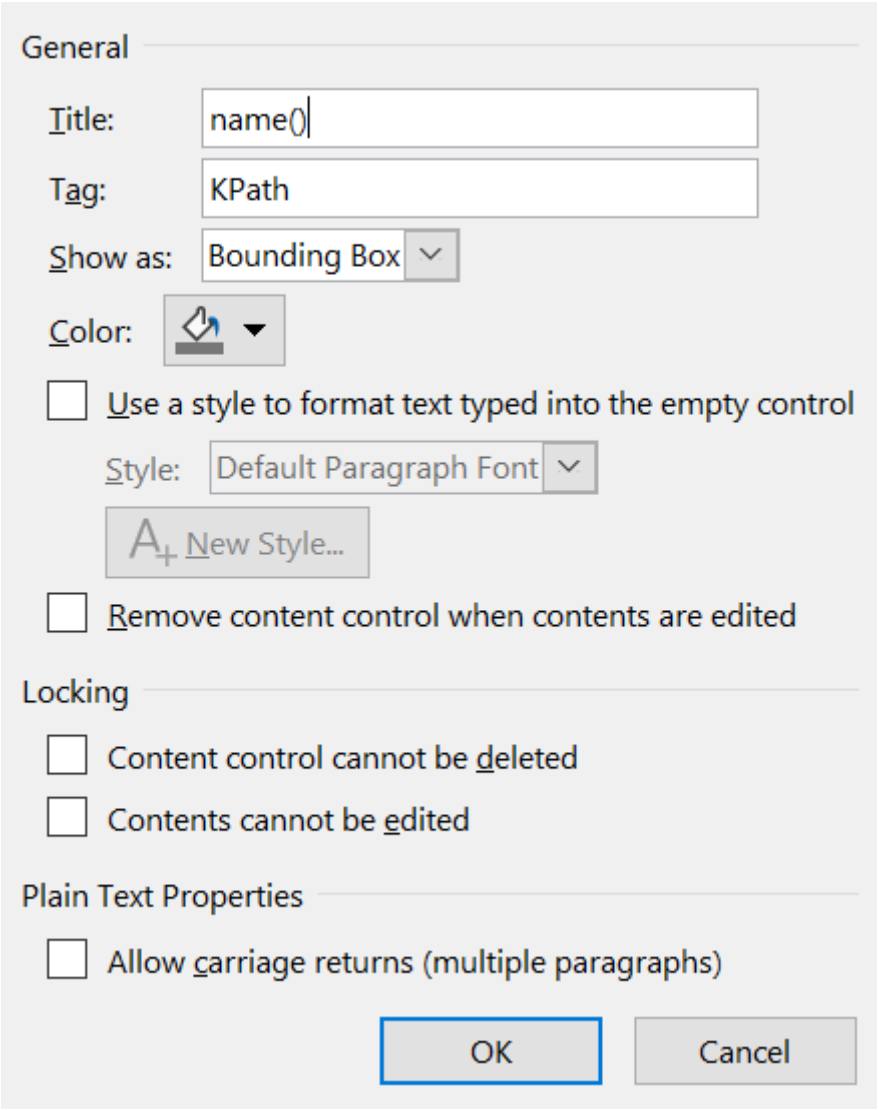
DOCX templates can be created using Microsoft Word 2007 or higher.

They are created the same way as RTF templates are created, with the only difference being that the path/script instructions are saved in text content control elements.

To insert the control elements, it is first necessary to activate the developer tools in Word. To do so, go to the Office menu, open the **Word options**, go to the **Popular commands** category and activate the option **Show Developer tab in the ribbon**. Now go to the **Developer tools** tab and activate **Design mode**.



To add KScript/KPath expressions, insert a **Text-only content control element**. The text of the control element is replaced by the calculated text. Go to the properties of the control element (via the context menu on the closing bracket) and specify the KScript or KPath under **Title**. If you leave the title empty, the text of the control element will be used instead. Enter the script type under **Tag**. The available script types are all the types available in ODT, with the exception of KPathImage **.**

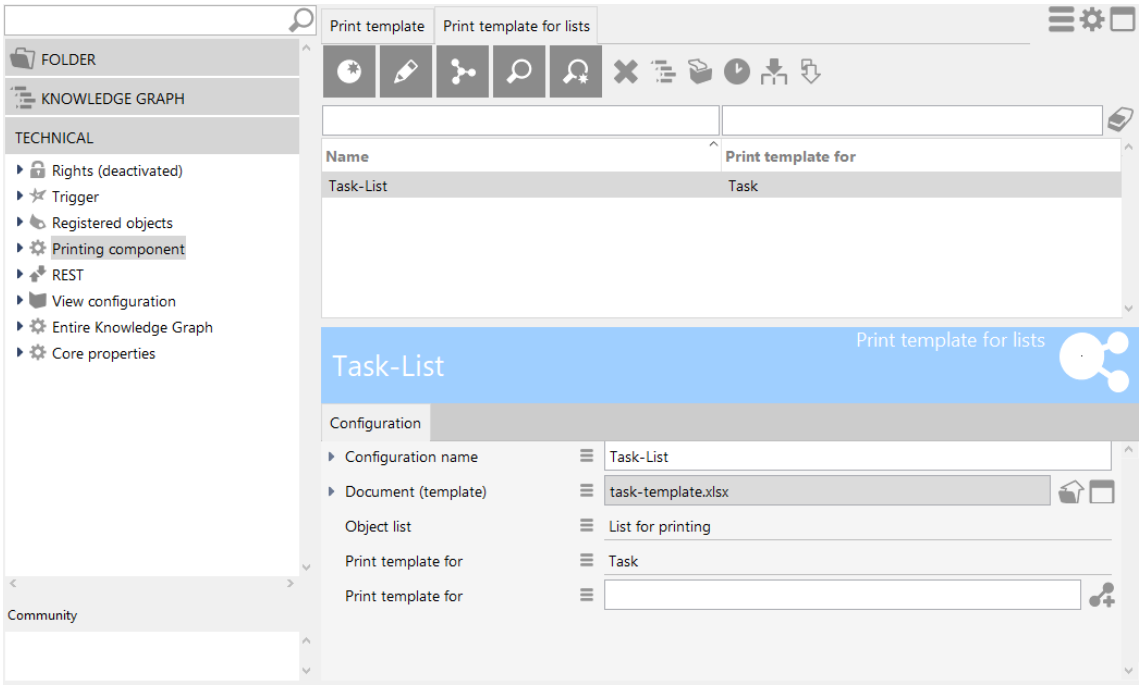


1.6.2. Druckvorlagen für Listen erstellen

Print templates for lists are saved in the "TECHNOLOGY/Print components" area in the Knowledge Builder. Each "Print template for lists" object contains a print template document (XLSX) and a relation that specifies to which objects the print template is to be applied. Optionally, an object list can be specified that should be used for generating the output. This allows the format of the list that the user sees on the screen, and the format of the list that was output, to be different.

When the attribute "Document (print template)" was not created, then when a document is generated, an Excel file is generated that contains one spreadsheet with the data in the object list and the column headings from the object list configuration, i.e. an Excel file does not necessarily have to be specified as the print template.

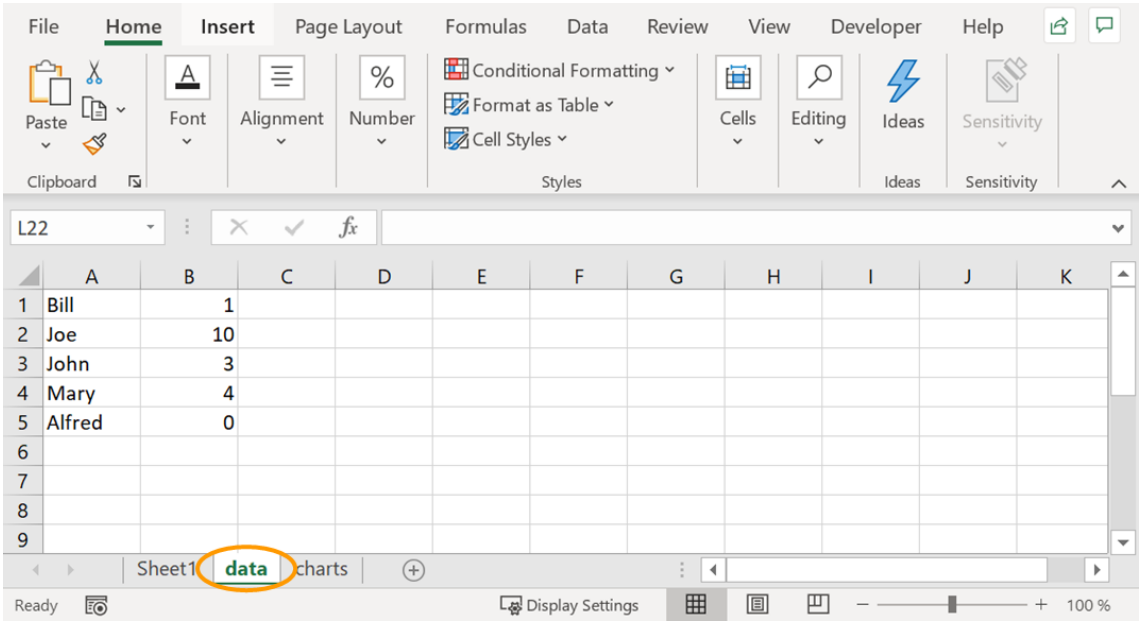
The following example shows a print template for lists with objects of the "Task" type.



XLSX templates can be created using Microsoft Excel 2007 or higher. These templates only function with object lists.

Creating the Excel file

A standard Excel file is used as a template, and must include an additional spreadsheet called "data". This spreadsheet is subsequently filled with the object list data, and this without headings and beginning with cell A1.



The other spreadsheets can reference data from the "data" sheet in formulas. i-views ensures that all formulas are calculated again as soon as the completed Excel file is next opened using Excel.

1.6.3. Dokumentenkonvertierung mit OpenOffice/LibreOffice

The output format of the print operation corresponds to the template used. If you would like to receive a different output format, you have to set up a converter.

To do so, you need an installation of LibreOffice or OpenOffice Version 4.0 or above on the computer that is to perform the conversion. This is usually located in the same place as the bridge or Job-Client that also executes the print operation.

In the configuration file (bridge.ini, jobclient.ini, etc.) you also have to specify the path to the "soffice" program which is part of the LibreOffice/OpenOffice installation and located in the "program" subdirectory there. This must be specified as an absolute path; relative paths (..\LibreOffice\etc.) are not possible here.

```
[file-format-conversion]
sofficePath="C:\Program Files (x86)\LibreOffice 4.0\program\soffice.exe"
```

Conversion service

If you do not want to keep a LibreOffice/OpenOffice installation on all workstations or server installations from which formats are to be converted, an appropriately converted REST bridge can perform the conversion.

To do so, the .ini file of the REST bridge must have the following format:

```
[Default]
host=localhost

[KHTTPRestBridge]
port=3040
volume=cardAdmin
services=jodService

[file-format-conversion]
sofficePath="C:\Program Files (x86)\LibreOffice 4.0\program\soffice.exe"
```

In the Admin tool, you enter the address at which the conversion service can be reached under system configuration/components/conversion service.

Example:

```
http://localhost:3040/jodService/jodconverter/service
```

Document formats

To ensure output formats are available, appropriately configured objects of the "Converter document format" type must be available in the Knowledge Graph.

The important thing is that not all formats can be converted into all formats. The most important ones are:

Name	Extension	Mime type
Portable Document Format	pdf	application/pdf
OpenDocument Text	odt	application/vnd.oasis.opendocument.text
Microsoft Word	doc	application/msword

1.7. Entwicklungsunterstützung

1.7.1. Dev-Tools

Es stehen verschiedene Tools zur Verfügung um die Entwicklung zu erleichtern.

- i-views-Plugin: Bietet Unterstützung für Jetbrains Produkte. Dazu gehört die Synchronisierung von Quelldateien, KJavascript- und KPath-Unterstützung

Mehr Informationen hierüber finden Sie im Benutzerhandbuch des i-views Jetbrains Plugin.

1.7.2. 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
```

1.8. KB-Plugins und Komponenten

1.8.1. Einheiten-Komponente

Die Einheiten-Komponente dient der adäquaten Ausgabe von Einheitenwerten — bestehend aus dem numerischen Wert und dem angehängten Einheitensymbol. Für unterschiedliche Dezimalpräfixe können mehrere Einträge von Einheiten mit relativen Faktoren für ein- und dieselbe Größenart definiert werden. Die Ausgabe mit Zahlenwert und Einheit erfolgt sowohl im Knowledge-Builder als auch im Web-Frontend per View Konfiguration Mapper. Ein Export könnte die Informationen zudem nutzen, um ein Attribut in einer anderen von einem Zielsystem gewünschten Maßeinheit auszugeben (=Maßeinheiten-Umrechnung).

Nach Installation der Komponente über das Admin-Tool finden sich die Maßeinheiten im Technik-Bereich. Dort können Objekte vom Typ Größenart und Maßeinheit angelegt und Konfiguriert werden.

Beispiele:

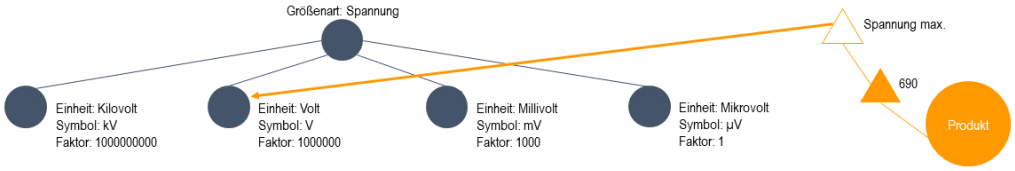
- Größenarten: Länge, Spannung, Temperatur
- Maßeinheiten: Meter, Inch, Millivolt, Grad Kelvin

Eine Größenart ist über die Relation "gemessen in" mit den ihr angehörigen Maßeinheiten verbunden. Eine Maßeinheit kann dabei nur genau einer Größenart angehören. Über die Relation "Basiseinheit" von kann eine Maßeinheit einem Attribut zugeordnet werden. Werte dieses Attributs werden dann intern immer in dieser Einheit gespeichert.

Das Einheiten-Paket "ETIM" bringt die Standard-Einheiten aus der ETIM Klassifikation mit, die um eigene Einheiten ergänzt werden kann.

Konfiguration:

- **"Einheitensymbol":** Diese Zeichenkette wird dem Wert nachgestellt, einem Wert von "1" wird die Einzahl-Variante verwendet.
  - **Beispiele:** "2,5 cm", "4 Minuten", "1 Minute".
- **"Faktor":** Faktor des Wertes in Relation zum Basiswert. Die Maßeinheit mit dem Faktor "1" repräsentiert den Basiswert, in dem Attributwerte gespeichert werden.
  - **Beispiel:** "Einheit (Entfernung)" hat die Maßeinheit "mm" mit dem Faktor "1", die Maßeinheit "cm" mit dem Faktor "10" und die Maßeinheit "m" mit dem Faktor "1000". Der abgespeicherte "Rohwert" im jeweiligen Attribut entspricht also dem Wert in "mm".
- **"Bruchteil":** Bruchteil des Wertes in Relation zum Basiswert. Durch die Verwendung von Faktor und Bruchteil wird eine höhere Genauigkeit in der Umrechnung erzielt.



HINWEIS

Wenn Attributwerte per Skript als virtuelle Eigenschaft ausgegeben werden, so gibt value() den Attributwert selbst aus, während valueString() den Attributwert mitsamt Einheit entsprechend der Einstellungen des Einheiten-Plugins ausgibt.

Für mehr Informationen zum Plugin kontaktieren Sie bitte empolis intelligent views: support@i-views.com.

1.8.2. Benutzerdefinierte Komponenten

Benutzerdefinierte Komponenten sind Bündel von semantischen Elementen, Abfragen, Skripten und anderen Elementen. Diese können zu anderen Knowledge-Graphen übertragen werden. Übliche Anwendungsfälle sind:

- Definition einer Komponente als Basis für spezialisierte Komponenten
- Entwicklung von Komponenten und deren Übertragung zu Integrations- und Produktions-Systemen

Eine Komponente ist ein Objekt, welches folgende Bestandteile aufweist:

- Name
- Version
- URI, welche als Basis für RDF-URIs verwendet wird
- Zeichenketten-Präfix, welcher als Basis für Konfigurationsnamen verwendet wird
- Optionale Regeln, die definieren, welche Objekte Teil der Komponente sind

Um diese Kapitel zu vereinfachen und abzukürzen wird im Weiteren alles, was einer Komponente zugewiesen werden kann als Element bezeichnet. Dies beinhaltet:

- Knowledge-Graph-Element-Typen
- Knowledge-Graph-Elemente
- Relationstypen
- Attributtypen
- REST Elemente
- View-Konfigurations-Elemente

- Datenquellen
- Abbildungen von Datenquellen
- Abfragen
- Sammlungen von semantischen Elementen
- Ordner
- Skripte
- Trigger
- Zugriffsrechteparameter

WARNUNG	Konkrete Eigenschaftsobjekte sollen nicht zu Komponenten zugewiesen werden. Sie werden beim Transfer mit ihren Objekten übertragen.

HINWEIS	Ordner und Sammlungen von semantischen Elementen kennen nach dem Export nur ihre Elemente und Unterordner, sie wissen aber nicht, wovon sie selbst ein Unterordner sind. Deshalb sind sie nach dem Import nicht im Ordner-Bereich auffindbar, sondern unter ***TECHNIK > Registrierte Objekte > Ordner /Sammlungen von semantischen Elementen***. Das bedeutet auch, dass nur der höchste Ordner einer Hierarchie wieder im Ordner-Bereich eingehängt werden muss, da er seine Unterordner kennt.

Wenn Ordner oder Sammlungen von semantischen Elementen wiederum Elemente enthalten, die weder zur Komponente gehören noch im Zielgraph vorhanden sind, dann wird dies beim Import zu Warnungen führen, da diese Elemente nicht gefunden werden können.

HINWEIS	Wenn eine exportierte Eigenschaft einen Index verwendet, so wird dieser Index ebenfalls exportiert. Wenn es im Zielgraph keinen Index mit selbem Namen und Konfiguration gibt, wird dieser angelegt, andernfalls werden die neuen Elemente dem bestehenden Index zugeordnet. Wenn der Import zu mehreren Indizes mit derselben Konfiguration führt, können diese in den Knowledge-Builder-Einstellungen unter ***Indexkonfiguration > Indizes*** zusammengefasst werden.

1.8.2.1. Konfiguration

Um Zugriff auf benutzerdefinierte Komponenten zu haben, müssen Sie zuerst im Admin Tool die Software Komponente ***Benutzerdefinierte Komponenten*** hinzufügen. Komponenten werden im Knowledge Builder verwaltet unter: ***Technik > Benutzerdefinierte Komponenten***

Hier werden alle bestehenden Komponenten aufgelistet und es können neue angelegt werden.

HINWEIS	Die Tabelle hat noch weitere Spalten für die ***Zuweisungsart*** und die ***Handhabung überschüssiger Elemente***, welche standardmäßig ausgeblendet sind und über ***Spalten auswählen*** in den Tabelleneinstellungen angezeigt

werden können. Die Tabelleneinstellungen müssen erst in den Knowledge-Builder-Einstellungen unter ***Persönlich > Editoren > Einstellungen für Tabellenspalten anzeigen*** aktiviert werden. Nach dem Neuladen der Tabelle sollten die Einstellungen nun oben rechts angezeigt werden.

Eine Komponente ist ein Objekt, welches aus folgenden Konfigurationen besteht:

Konfigurationswert	Beschreibung
Name	Der Name der Komponente.
Beschreibung	Ein kurzer Text der den Nutzen oder Inhalt der Komponente beschreiben sollte.
Präfix	Eine kurze Zeichenkette, die zur Identifikation von Elementen der Komponente genutzt wird.
Basis-URI	Eine URI, die zur Identifikation von Elementen der Komponente genutzt wird. **HINWEIS** Im Kapitel Präfix und Basis-URI wählen wird näher darauf eingegeangen, wie eine gültige Basis-URI aussieht.
Elemente anhand Präfix/URI/Relation auswählen	Wenn angehakt, werden Elemente dieser Komponente anhand der oben genannten Basis-URI und des Präfixes identifiziert.
Abhängigkeiten einschließen	Wenn angehakt, werden beim Exportieren dieser Komponente alle Elemente, von denen diese Komponente abhängt, ebenfalls exportiert, egal ob sie zu dieser Komponente zugewiesen sind oder nicht.

Konfigurationswert	Beschreibung
Handhabung überschüssige Elemente	Gibt an wie mit Elementen, welche im Zielgraph zu der Komponente gehören, aber nicht in der exportierten Datei vorhanden sind, umgegangen wird. **Beibehalten** Die Elemente werden nicht verändert **In den Papierkorb legen** Die Elemente werden aus der Komponente entfernt und im Papierkorb im Benutzerdefinierte Komponenten Bereich abgelegt **Skript ausführen** Die Elemente werden an ein vor dem Export definiertes Skript geleitet **Löschen** Die Elemente werden gelöscht **HINWEIS** Beim Import werden die Einstellungen und das Skript von der importierten Komponente verwendet, egal was in der gleichen Komponente im Zielgraph ausgewählt ist.
Skript zur Verarbeitung überschüssiger Elemente	Das Java-Skript welches beim Import mit den überschüssigen Elementen aufgerufen wird, wenn die ***Skript ausführen*** Option bei der Handhabung überschüssiger Elemente ausgewählt ist.
Zusätzliche Übersetzungen behalten	Wenn angehakt, werden beim Import dieser Komponente keine zusätzlich konfigurierten Übersetzungen für Attributtypen im Zielgraph überschrieben.
Schreibgeschützt	Wenn angehakt, kann nichts mehr verändert werden, was mit dieser Komponente zu tun hat, außer das Komponentenobjekt selbst. Es können auch keine Elemente zur Komponente hinzugefügt oder entfernt werden. Es ist jedoch noch möglich, Relationen von und zu Elementen der Komponente zu ziehen. Ebenfalls ist es möglich, aber nicht empfohlen, manuell die Basis-URI oder den Präfix zu einem Element hinzuzufügen, um es zu einem Teil der Komponente zu machen. Dadurch können Elemente allerdings nur zugewiesen werden, da sie ab dann schreibgeschützt sind.
Schreibschutz nach Import deaktivieren	Wenn angehakt, wird das `Schreibgeschützt` Attribut dieser Komponente, nachdem sie in einen Graph importiert wurde, ausgeschaltet.

Konfigurationswert	Beschreibung
Attribut zur Identifikation beim Transfer	Bestimmt ein vom Nutzer definiertes Attribut, welches beim Transfer der Komponente zum Identifizieren von Elementen genutzt wird. **HINWEIS** Das Attribut braucht einen Eindeutigkeitsindex und sollte ein Zeichenkettenattribut sein.
Zuweisungsart	Bestimmt mit welchen Mitteln Elemente dieser Komponente zugewiesen werden: **Relation** Es wird eine Einwegrelation von dem Element zur Komponente gezogen. **Relation (Internen Namen anpassen)** Es wird eine Einwegrelation von dem Element zur Komponente gezogen, aber es wird zusätzlich noch der interne Name des Elements angepasst wenn möglich. **Präfix/Basis-URI** Es werden der konfigurierte Präfix und die Basis-URI verwendet, um Namen, interne Namen und RDF-URIs zu kennzeichnen.
Benötigte Komponente	Deklariert andere Komponenten als notwendig damit diese funktioniert.
Überschreibt Komponente	Wenn angekreuzt, werden Elemente, die sowohl dieser als auch der benötigten Komponente angehören, nur dieser zugeordnet.
Version	Die Version der Komponente besteht aus Major Version, Minor Version und Patch. **HINWEIS** Major und minor können positive, ganze Zahlen bis zu 1152921504606846975 sein. Patch hat keine solche Einschränkungen und kann eine beliebig große, ganze Zahl sein, auch eine negative.

Es gibt noch weitere Möglichkeiten, um Elemente zu einer Komponente zuzuweisen, auf welche näher im Kapitel Zusätzliche Auswahl und Konfiguration von spezifischen Elementen eingegangen wird.

1.8.2.2. Ein minimales Beispiel

Navigieren Sie zum Benutzerdefinierte Komponenten Bereich und erstellen Sie eine neue Komponente. Dabei wird nach einem Namen, der frei wählbar ist und jederzeit geändert werden kann und drei weiteren Werten gefragt:

- **Präfix:** Eine Zeichenkette, welche zur Identifikation von Elementen mit einem Konfigurationsnamen genutzt wird. Diese wird auch benutzt, um Konfigurationsnamen beim Erstellen von Elementen vorzuschlagen. Zum Beispiel `accounting`.
- **Basis-URI:** Diese URI wird als Basis zur Erstellung der RDF-URIs von Elementen der Komponente verwendet. Sie sollte mit dem Präfix enden, z.B. `http://example.org/accounting`.
- **Handhabung überschüssiger Elemente:** Hiermit wird bestimmt, was beim Import einer Komponente mit Elementen passieren soll, welche im Zielgraph zu der importierten Komponente gehören aber im Import nicht vorhanden sind.

Nun können Elemente zu der Komponente zugewiesen werden. Dazu muss zuerst zu dem gewünschten Element navigiert werden und sein Kontextmenü geöffnet werden. Dort sollte das ***Benutzerdefinierte Komponenten*** Untermenü zu finden sein, welches drei Möglichkeiten bietet, um das Element zuzuweisen:

1. ***Element zuweisen*** ist die direktere Variante, welche einfach alle passenden Komponenten vorschlägt und weist das Element nach Bestätigung zu.
2. Alternativ kann auch ***Zuweisungswerkzeug öffnen*** gewählt werden, welches einen Überblick darüber verschafft, welche anderen Elemente mit dem gewähten zusammenhängen. Dort können dann alle nötigen Elemente zugewiesen werden.
3. Zuletzt gibt es noch die Option ***Ebenfalls zu x zuweisen*** verwendet werden, um das Element zur zuletzt zugewiesenen Komponente zuzuweisen.

Nun sollte das zugewiesene Element seine Zugehörigkeit auf der rechten Seite des Banner-Bereichs anzeigen. Außerdem wurden je nach Element RDF-URI und interner Name oder Registrierungsschlüssel angepasst.

1.8.2.3. Präfix und Basis-URI wählen

Obwohl es keine technischen Beschränkungen gibt, wie ein Präfix oder eine Basis-URI aussehen muss, gibt es ein paar Regeln, um die Nutzung von benutzerdefinierten Komponenten zu vereinfachen:

- Nur alpha-numerische Zeichen und Punkte verwenden.
- Keinen Punkt an das Ende des Präfixes setzen, da dies automatisch als Trennzeichen verwendet wird.
- Kein `#` an das Ende der Basis-URI setzen, da dies automatisch als Trennzeichen verwendet wird.
- Keine generischen Namen nutzen, die mit anderen bereits vorhanden Komponenten verwechselt werden können, wie z.B. `view-config` oder `rest`.

- Der Präfix sollte es offensichtlich machen zu welcher Komponente er gehört.
- Der Präfix sollte als letzter Teil der Basis-URI genutzt werden, wie im Beispiel des vorhergehenden Kapitels gezeigt.

HINWEIS

Eine URI (unique resource identifier) wird hauptsächlich genutzt, um Ressourcen prezise im Internet zu finden und ist dementsprechend eine Internet-Adresse. Da die Basis-URI ein Namensraum ist, welcher dieses Konzept benutzt, sollte sie mit `http://` oder `https://` anfangen, wonach eine Domäne kommt, welche ihr Unternehmen oder Projekt repräsentiert. Danach sollte eine weiterer `/` gefolgt von dem Präfix der Komponente kommen. Daraus ensteht dann zum Beispiel so etwas wie: `http://example.org/accounting` für das Projekt Beispiele zur Verfügung zu stellen und die Komponente für accounting Elemente.
Da die URI lediglich zum Identifizieren und Zuweisen von Elementen im Graph verwendet wird, muss sie nicht tatsächlich zu einem Ergenbnis führen, wenn sie in einen Web-Browser eingegeben wird.

1.8.2.4. Ändern von Präfix und Basis-URI

Das manuelle Anpassen von Präfix und Basis-URI wird nicht von den ausgewählten Elementen übernommen und führt dazu, dass diese nicht mehr ausgewählt sind.

Damit die Änderungen auch von den ausgewählten Elementen übernommen werden, muss ein bestimmter Dialog verwendet werden. Unter ***TECHNIK > Benutzerdefinierte Komponenten*** muss die gewünschte Komponente und unten links das spezifische Unterobjekt ausgewählt werden. Anschließend muss das Bearbeiten-Symbol über dem Banner-Bereich angeklickt werden.

Figure 1. Der Knopf um Präfix oder Basis-URI einer Komponente zu ändern.

In dem Dialog können die aktuellen Werte geändert oder neue hinzugefügt, jedoch keine bestehenden Werte gelöscht werden. Die neuen Werte werden dann direkt von den ausgewählten Elementen übernommen.

Das Ändern des Komponentenobjekts selbst beeinflusst nur Elemente, die exklusiv von diesem Objekt ausgewählt sind. Elemente, die zusätzlich noch von einem Unterobjekt ausgewählt sind, werden nicht angepasst, da die Unterelemente die Komponentenauswahl aufgrund ihrer zusätzlichen Optionen überschreiben.

WARNUNG	Sobald Präfix und Basis-URI der Komponente und ihrer Unterobjekte gleich sind, wählen alle Objekte alle Elemente aus und es ist unmöglich, diese automatisch wieder zu trennen.

HINWEIS	Sollte es während dem Überschreiben der Elemente zu einem Problem kommen, bleiben zwar die bereits angepassten Elemente erhalten aber die Komponente behält noch ihre alten Werte. Das sorgt zwar dafür, dass diese Elemente vorübergehend nicht mehr Teil der Komponete sind, jedoch wird es dadurch sehr einfach den Prozess neu zu starten, nachdem das Problem behoben wurde, um die restlichen Elemente anzupassen.

1.8.2.5. Zuweisung von Elementen

Wie ein Element einer Komponente zugewiesen wird, kommt darauf an, was bei der Zuweisungsart der Komponente eingestelt ist.

Steht die Zuweisungsart auf ***Relation***, werden semantische Elemente mithilfe einer Einwegrelation zu der Komponente zugewiesen.

Für diese Zuweisungsart gibt es auch die Variation, dass, zusätzlich zur Relation, interne Namen mit dem Präfix angepasst werden, um Verwirrung durch keine oder falsche Präfixe zu vermeiden.

Wenn die Zuweisungsart auf ***Präfix/Basis-URI*** steht, wird die Zugehörigkeit von Elementen zu einer Komponente über einige der identifizierenden Attribute der Elemente angegeben:

- RDF-URI fängt mit der Basis-URI der Komponente an
- Interner Name fängt mit dem Präfix der Komponente an
- Registrierungsschlüssel fängt mit dem Präfix der Komponente an
- Name fängt mit dem Präfix der Komponente an

Der Name eines Elements wird nur von den benutzerdefinierten Komponenten genutzt, wenn das Element Benennungs-Regeln folgt, wie zum Beispiel View-Konfigurations-Elemente.

HINWEIS	Nur eines dieser Attribute muss zur Komponente passen, um die Zugehörigkeit zu erkennen, aber zum Export benötigen Elemente eine RDF-URI. Sollte ein Element beim Export noch keine RDF-URI haben, so wird diese automatisch aus der Basis-URI der Komponente und dem Namen des Elements erstellt.

Diese Attribute werden automatisch geändert, sobald ein Element durch eins der zur Verfügung gestellten Werkzeuge zugewiesen wird, können aber auch manuell angepasst werden.

Der Name der Komponente, welche einem Element zugewiesen ist, wird auf der rechten Seite im Banner des Elements angezeigt. Dies kann auch in den Knowledge-Builder-Einstellungen unter ***Editoren*** deaktiviert werden.

Außerdem ist es möglich, sich alle Elemente einer Komponente anzeigen zu lassen, indem Sie auf den -Knopf in der Detailansicht der entsprechenden Komponente klicken.

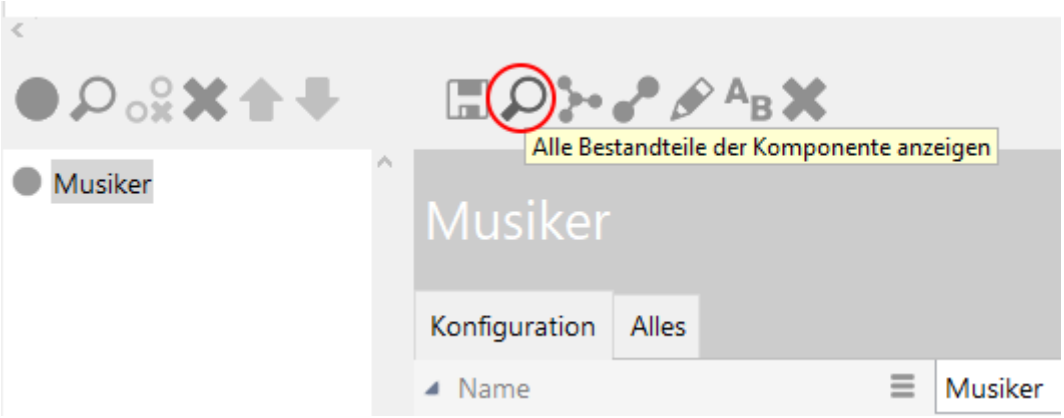


Figure 2. Der Knopf um alle Bestandteile einer Komponente anzuzeigen.

1.8.2.5.1. Elemente zuweisen

Für die meisten Elemente wird beim Erstellen ein Drop-down Feld angezeigt, welches mögliche Komponenten enthält. Wenn Sie eine Komponente auswählen, wird der zugehörige Präfix direkt in das Namensfeld des neuen Elements eingetragen. Das UI versucht dabei, eine passende Komponente anhand des Kontexts zu ermitteln. Wenn z.B. ein Untertyp eines Typs erstellt wird, welcher einer Komponente angehört, so wird diese Komponente automatisch für den neuen Untertyp ausgewählt.

Unter bestimmten Bedingungen werden einige Elemente jedoch auch automatisch zugewisen:

- Beim Erstellen einer Abbildung mit bereits vorhandener, aber unzugwiesener und ungenutzter Datenquelle wird auch die Datenquelle der ausgewählten Komponente zugewiesen.
- Beim Registrieren einer unregistrierten Datenquelle aus der Abbildungs-Ansicht, bekommt die Quelle den Präfix der Abbildung, solange diese einer Komponente zugewiesen ist und als einziges diese Quelle verwendet.
- Beim Verknüpfen einer unbenutzten und unzugewiesenen Datenquelle mit einer Abbildung in einer Komponente, wird auch die Quelle dieser Komponente zugewiesen.
- Beim Anlegen einer Erweiterung an einem Objekt in einer Komponente wird die Erweiterung auch direkt der Komponente zugewiesen.

Wenn beim Erstellen eines Elements kein interner Name oder Registrierungsschlüssel gesetzt, jedoch eine Komponente ausgewählt wird, so wird der Registrierungsschlüssel automatisch durch den Präfix der Komponente und den Namen des Elements zusammengesetzt. Interne Namen werden standardmäßig nicht neu erstellt, sondern nur bereits bestehende bekommen den Präfix hinzugefügt. Dies kann in den System-Einstellungen angepasst werden.

Nach ihrer Erstellung können einzelne oder mehrere Elemente einfach mittels ihres Kontextmenüs unter ***Benutzerdefinierte Komponenten*** über die Optionen ***Element zuweisen***, ***Ebenfalls zu x zuweisen***, wobei x die zuletzt zugewiesene Komponente ist, oder mit dem ***Zuweisungswerkzeug***

einer Komponente zugewiesen werden.

Beim Zuweisen eines registrierbaren Elements, welches weder einen Namen, noch einen Registrierungsschlüssel besitzt, wird ein Registrierungsschlüssel aus dem Kontext des Elements erstellt. Damit der Schlüssel korrekt auf Eindeutigkeit geprüft werden kann, sollte das Element, welches dieses Element verwendet bereits zugewiesen sein.
Auch wenn das nicht der Fall ist, können niemals zwei identische Registrierungsschlüssel vergeben werden. Das Zuweisen kann allerdings fehlschlagen.

HINWEIS	Das genaue verhalten beim Zuweisen von Elementen kann, wie im Kapitel System-Einstellungen beschrieben, angepasst werden.

Über das Kontextmenü kann außerdem noch die aktuelle Komponente einzelner Elemente geöffnet werden, damit Sie nicht zum ***Benutzerdefinierte Komponenten*** Bereich navigieren müssen.

HINWEIS	Beim Zuweisen einer Relation werden automatisch beide Richtungen der Relation zugewiesen. Die Logik der benutzerdefieierten Komponenten behandelt die zwei Hälften einer Relation immer wie ein einziges Element.

Ist die Zuweisungsart `Präfix/Basis-URI` ausgewählt und ein Element hat keine zur Basis-URI passende RDF-URI, wenn es zu einer Komponente zugewiesen wird, erhält das Element einen RDF-URI-Alias mit der Basis-URI.
Dieser Alias wird allerdings beim Transfer der Komponente nicht übernommen und ist daher nur für Komponenten geeignet, welche niemals exportiert werden.
Um dieses Problem zu umgehen, können Sie einfach eine relationsbasierte Zuweisungsart verwenden.

HINWEIS	Wenn eine relationsbasierte Zuweisungsart verwendet wird und Sie die Zuweisung eines Elements ändern, wird die RDF-URI nicht angepasst. Das kann dazu führen, dass die RDF-URI eines Elements mit der Basis-URI einer Komponente beginnt, der es nicht zugewiesen ist. Auch hier gilt: Die Zuweisung per Relation ist immer das stärkste Zuweisungsmerkmal, welches alle anderen überschreibt. Ist die Zuweisungsart lediglich auf `Relation` gesetzt, sodass der interne Name ebenfalls nicht angepasst wird, kann es auch bei diesem vorkommen, dass er mit dem Präfix einer anderen Komponente startet.

Zusätzlich gibt es im Untermenü zwei Optionen, welche exklusiv für View-Konfigurations-Elemente verfügbar sind. Damit können Elemente aus einer Komponente durch andere, externe Elemente ersetzt werden, wodurch die ersetzenden Elemente bei zukünftigen Updates nicht überschrieben werden.

WARNUNG	Dieses Feature sollte nur als letzter Ausweg benutzt werden, da es aufgrund der vielen möglichen Vernetzungen eines ersetzten Elements potenziell sehr fehleranfällig ist.

1.8.2.5.2. Das Zuweisungswerkzeug

Das Zuweisungswerkzeug kann genutzt werden, um einen Überblick über die Elemente, die Sie zuweisen möchten, deren Beziehungen und Schichtenverletzungen zu erhalten.

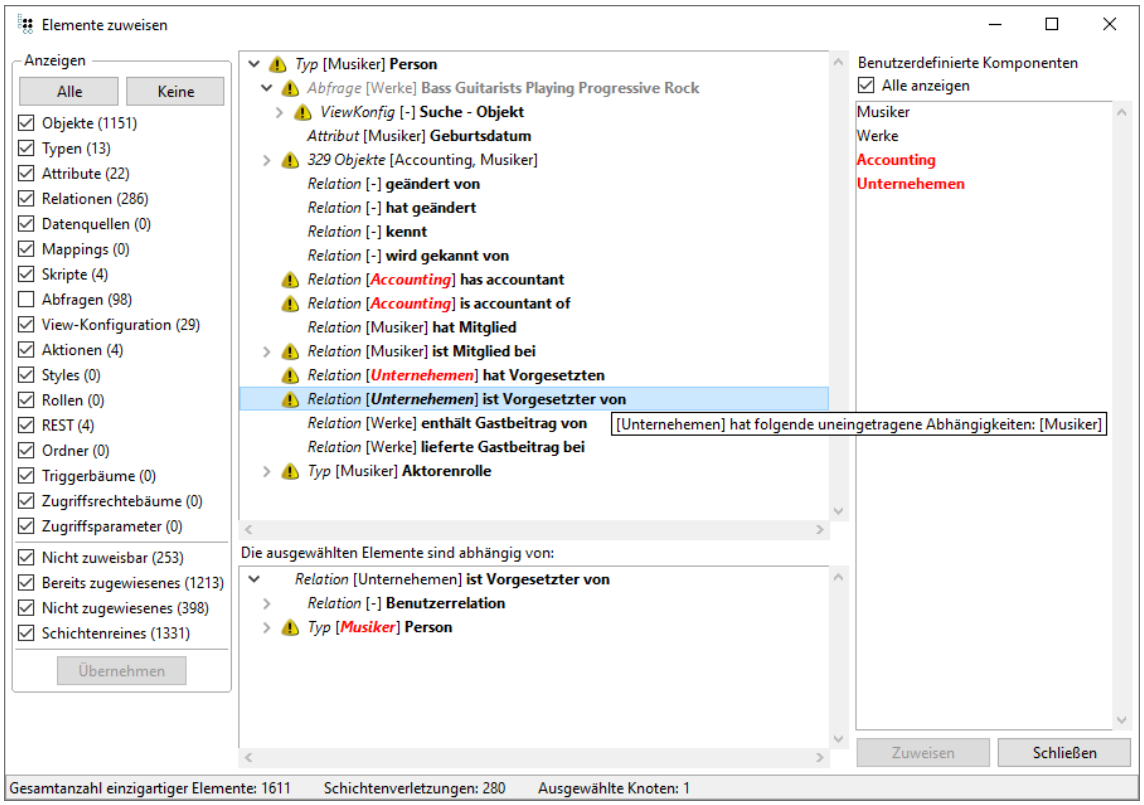


Figure 3. Das Zuweisungswerkzeug.

Auf der linken Seite sind Filter für die obere Baum-Ansicht zu finden, welche bestimmte Arten von Elementen ausblenden können. Im zweiten Bereich können Sie Elemente nach dem Status ihrer Zuweisung filtern. Jeder Filter zeigt an, wie viele einzigartige Elemente seiner Art sich im oberen Baum befinden. Wenn ein Filter ausgeschaltet wird, werden alle dazugehörigen Elemente ausgeblendet, außer wenn sich unter diesen Elementen noch andere befinden, die nicht ausgeblendet werden sollen. Solche Elemente werden lediglich ausgegraut, wie im obigen Bild zu sehen ist. Im oberen Bereich der Filter, der sich auf die Art der Elemente bezieht, können alle Filter mithilfe der oberen Knöpfe gleichzeitig an oder aus geschaltet werden. Wenn die Hauptansicht sehr viele Elemente enthält, werden die Filter nicht automatisch angewendet. Stattdessen erscheint ein ***Übernehmen*** Knopf, der alle geänderten Filter auf einmal anwendet, um Wartezeiten zu verringern, wenn Sie mehrere Filter ändern möchten.

Oben in der Mitte befindet sich eine Baum-Ansicht mit den Elementen, für die das Werkzeug geöffnet wurde und allen Elementen, die irgendwie von ihnen abhängen. Dies gilt nicht für View-Konfigurations-Elemente, da es bei ihnen mehr Sinn macht sich für die Anzeige nicht strikt an Abhängiges zu halten, sondern stattdessen auch Dinge wie z.B. verwendete Skripte anzuzeigen, obwohl diese als Abhängigkeit gelten.

Jeder Knoten zeigt den Typ seines Elements, welcher Komponente das Element zugewiesen ist und wie es heißt. Wenn ein Element aufgrund seiner zugewiesenen Komponente oder weil es mehreren

Komponenten zugewiesen ist, eine Schichtenverletzung verursacht, wird die Zugehörigkeit in fett gedruckten, roten Buchstaben markiert und es erscheint ein Warnsymbol vor dem Knoten. Wird der Mauszeiger über ein solches Element gehalten, erscheint ein Tooltip zur Erklärung der Ursache der Schichtenverletzung. Elemente deren Unterknoten eine Schichtenverletzung verursachen haben ebenfalls ein Warnsymbol, um darauf hinzuweisen.

HINWEIS	Diese Tooltips dienen nur als Erklärung des aktuellen Standes und sollen keine Aufforderung dazu sein, unsinnige Abhängigkeiten zwischen Komponenten zu erstellen, um Schichtenverletzungen loszuwerden.

Für bessere Übersicht werden Objekte von Typen in speziellen Knoten gebündelt, welche die Anzahl der Objekte und bis zu 5 Komponneten, denen die verschiedenen Objekte zugewiesen sind, anzeigen. Wenn in den Optionen eingestellt ist, dass Objekte ignoriert werden sollen, gibt es diese Knoten im Zuweisungswerkzeug nicht, da keine Objekte die nur über ihren Typen gefunden wurden angezeigt werden.

In der unteren Baum-Ansicht befindet alle Elemente, von denen die im Baum markierten Elemente abhängen. Wenn ein oben ausgewähltes Element eine Schichtenverletzung verursacht, wird die Ursache hier markiert.

Beispiel: Im obigen Bild können Sie sehen, dass die Relation "ist Vorgesetzter von" eine Schichtenverletzung verursacht. Das liegt daran, dass die Relation zur Komponente "Unternehmen" gehört, wie aber in der Liste der Abhängigkeiten zu sehen ist, vom Typ "Person" abhängt. Der Typ Person gehört allerdings zur Komponente "Musiker". Dies bedeutet, dass die Komponente "Unternehmen" von der Komponente "Musiker" abhängt. Das sollte aber nicht der Fall sein und ist daher auch nicht konfiguriert. Somit wird hier eine Schichtenverletzung verursacht. Wie rechts vorgeschlagen, sind die einzigen Komponenten, welche dieser Relation momentan ohne Probleme zugewiesen werden können, "Musiker" und "Werke".

Auf der rechten Seite befindet sich die Komponentenauswahl. Standardmäßig werden hier nur Komponenten angezeigt, denen die im Baum ausgewählten Elementen zugewiesen werden können, ohne Schichtenverletzungen zu erzeugen. Um alle Komponenten zu sehen, kann ***Alle anzeigen*** ausgewählt werden. Die dadurch zusätzlich angezeigten Komponenten werden rot markiert. Ein Doppelklick auf eine Komponente öffnet ein Fenster, um diese zu bearbeiten.

In der Fußleiste können Sie die Anzahl der einzigartigen Elemente im oberen Baum, die Anzahl der Schichtenverletzungen dieser und die Anzahl der momentan ausgewählten Knoten sehen. Einzigartige Elemente bedeutet, dass wenn es im Baum zwei Typen mit derselben Relation gibt, diese Relation unter beiden Typen angezeigt, aber nur als ein einzigartiges Element gezählt wird.

Das Kontextmenü von Elementen in den Baum-Ansichten enthält die Optionen, ein neues Zuweisungswerkzeug für die markierten Elemente zu öffnen und einzelne Elemente zu bearbeiten, indem ein neues Fenster mit dem ausgewählten Element geöffnet wird. Außerdem gibt es noch die Option alle Unterknoten der ausgewählten Elemente aufzuklappen und zu ebenfalls auszuwählen.

HINWEIS

Sollte es zyklische Abhängigkeiten geben, werden alle Unterelemente genau einmal markiert, auch wenn nicht alle Knoten ausgeklappt werden können. Dementsprechend können Sie diese Option für alle Wurzelknoten nutzen und haben immer jedes einzigartige Element im Baum einmal ausgewählt.

Sobald eine Komponente und mindestens ein Element aus dem oberen Baum ausgewählt sind, können Sie den ***Zuweisen***-Knopf unten rechts drücken. Dieser zeigt dann nochmal eine Liste aller Elemente an, die nun zugewisen werden. Nach erneuter Bestätigung startet der Zuweisungsprozess und am Ende werden alle Elemente angezeigt, bei denen die Zuweisung scheiterte zusammen mit dem jeweiligen Grund.

In beiden Listen können die Elemente mittels Doppelklick geöffnet und bearbeitet werden.

HINWEIS

Wenn ein Zuweisungswerkzeug offen ist, während ein Element einer Komponente zugewiesen wird, aktualisiert sich das Zuweisungswerkzeug automatisch, solange das Element dort vorhanden ist. Dies funktioniert jedoch nicht, wenn das Element durch manuelles Anpassen seiner Attribute zugewiesen wird.

Wenn die Komponenten sich verändern, während ein Zuweisungswerkzeug geöffnet ist, kann es mit `F5` neu geladen werden. Dadurch wird die Liste der Komponenten, deren Abhängigkeiten und alle Schichtenverletzungen neu berechnet. Wenn zusätzlich die `Strg` Taste gedrückt gehalten wird, werden auch die Zuweisungen aller Elemente aktualisiert.

Falls Sie nur eine kleine Übersicht über die direkten Abhängigkeiten eines Elements haben möchten, siehe Abhängigkeiten im Graph-Editor anzeigen.

1.8.2.5.3. Zuweisen mit RegEx-Suche

Ein weiterer Weg, Elemente einer Komponente zuzuweisen, ist einen regulären Ausdruck zu formulieren und alle dazu passenden Elemente der ausgewählten Komponente zuzuweisen. Diese Funktionalität finden Sie unter ***TECHNIK > Benutzerdefinierte Komponenten*** in der Detailansicht der gewünschten Komponente.

Figure 4. Der Knopf um das Werkzeug zur Zuweisung per RegEx-Suche zu öffnen.

Als Erstes kann hier ausgewählt werden, ob die Elemente nach Attributen, die den Präfix benutzen oder ihrer RDF-URI durchsucht werden sollen. Dann kann der reguläre Ausdruck formuliert werden,

mit dem nach Elementen gesucht wird. Dieser Ausdruck muss mindestens eine Gruppe beinhalten, welche dann, durch den im unteren Textfeld eingetragenen Text, ersetzt wird. Dort steht standardmäßig erstmal der Präfix der Komponente mit einem Punkt dahinter.

HINWEIS	Wenn die definierte Gruppe nicht am Anfang des gefundenen Wertes steht, wird das Element nicht der Komponete zugewiesen, da der Präfix/die Basis-URI dann auch nicht am Anfang des Wertes stehen werden.

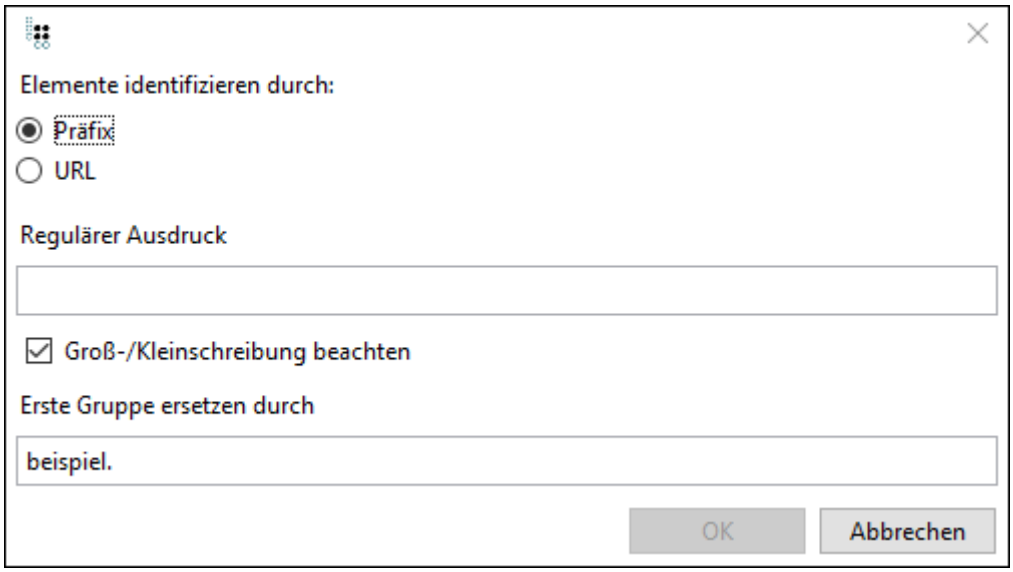


Figure 5. Das Werkzeug zur Zuweisung per RegEx-Suche.

Beispiel: Der reguläre Ausdruck ^(alterPräfix\.).* findet alles, was mit `alterPräfix.` anfängt.

1.8.2.5.4. Komponenten-Vorschläge

Die Liste der möglichen Komponenten zeigt nur Komponenten an, die für das aktuelle Element keine Schichtenverletzungen erzeugen. Dazu gibt es ein paar Regeln:

- Schreibgeschützte Komponenten werden niemals vorgeschlagen.
- Nur Komponenten, die abhängig von allen Komponenten der direktesten Elemente sind, welche für dieses Element benötigt werden (z.B. der Typ eines Objekts; eine Releation, die von einer Abfrage benutzt wird) und eine Komponente haben, werden vorgeschlagen.
- Nur Komponenten, von denen alle Komponenten der direktesten Elemente, welche das Element benötigen (z.B. ein Objekt eines Typs; eine Abfrage, die eine Relation verwendet) und eine Komponente haben, abhängen, werden vorgeschlagen.
- Wenn es keine Einschränkungen gibt, werden alle nicht-schreibgeschützten Komponenten vorgeschlagen.

1.8.2.5.5. Elemente entfernen

Um ein Element aus seiner aktuellen Komponente zu entfernen, können Sie im Kontextmenü des Elements ***Benutzerdefinierte Komponenten > Element entferen*** wählen.

Beim Entfernen eines Elements aus einer Komponente wird ebenfalls die RDF-URI gelöscht, der Registrierungsschlüssel deregistriert und der Präfix aus dem normalen und internen Namen entfernt. Außerdem wird das Element in einem Papierkorb-Ordner unter ***TECHNIK > Benutzerdefinierte Komponenten > Entfernte Elemente*** abgelegt.

HINWEIS	Beim Entfernen oder neu Zuweisen eines Elements bleiben der Name und der interne Name gleich und nur der Präfix wird entfernt bzw. angepasst. Wenn also ein Element umbenannt werden soll und auch für den internen Namen und die URI der neue Name verwendet werden soll, müssen diese entweder manuell angepasst werden oder es müssen erst beide gelöscht und dann das Element neu zugewiesen werden, da sie sonst noch den alten Namen verwenden.

1.8.2.5.6. Der Papierkorb-Ordner

Der Papierkorb-Ordner ist unter ***TECHNIK > Benutzerdefinierte Komponenten > Entfernte Elemente*** zu finden. Er hat zwei Unterordner für semantische Elemente und registierte Objekte.

Elemente werden in den folgenden zwei Situationen in ihrem jeweiligen Ordner abgelegt:

1. Die Zuweisung zu ihrer Komponente wird entfernt
2. Sie sind überschüssige Elemente nach einem Import und die importierte Komponente hat für die Handhabung überschüssiger Elemente ***In den Papierkorb legen*** ausgewählt

Es gibt drei Arten ein Element aus seinem Papierkorb-Ordner zu entfernen:

1. Das Element wird einer Komponente zugewiesen
2. Das Element wird wiederhergestellt (Rechtsklick auf das Element oder seinen Ordner und die *Wiederherstellen* Option wählen)
3. Das Element wird entfernt (Rechtsklick auf das Element oder seinen Ordner und eine passende *Entfernen* Option auswählen)

Das Wiederherstellen eines Elements setzt internen Namen und RDF-URI oder seinen Registrierungsschlüssel auf die Werte, die sie hatten, bevor das Element in den Papierkorb gelegt wurde.

Über das Kontextmenü von Elementen in einem Papierkorb-Ordner können Sie sich anzeigen lassen, wovon sie verwendet wurden bevor ihre Zuweisung zu einer Komponente entfernt wurde.

HINWEIS	Wenn die Komponente ihren Präfix oder ihre Basis-URI ändert bevor das Element wiederhergeatellt wird, beziehen sich diese Änderungen nicht auf das wiederhergestellte Element. Dies wird trotzdem genau die Werte haben,

welche es hatte, bevor es in den Papierkorb gelegt wurde.

WARNUNG

Gefahr von Datenverlust!
Im Gegensatz zu Schema-Elementen wie Objekt- oder Eigenschaftstypen führt das Entfernen einer Zuweisung von registrierten Objektinstanzen zur Deregistrierung der jeweiligen Objektinstanz. Die Deregistrierung einer Objektinstanz führt zur Löschung der Objektinstanz, sofern sie sich nicht in mindestens einem weiteren Ordner befindet.

Dementsprechend werden registrierte Objektinstanzen beim Entfernen aus dem Papierkorb meistens endgültig gelöscht.

Die Wiederherstellung eines unbeabsichtigt gelöschten Elements ist nicht möglich. In diesem Fall muss das Element neu erstellt werden.
Existierende Referenzen auf den Registrierungsschlüssel eines gelöschten Elements funktionieren nicht mehr und müssen manuell repariert werden.

Zur Vermeidung unbeabsichtigter Löschungen von Elementen ist darauf zu achten, dass noch benötigte Elemente registriert oder in mindestens einem weiteren Ordner hinterlegt sind.

1.8.2.5.7. Schichtenverletzungen finden

Neben dem Zuweisungswerkzeug gibt es eine weitere Methode, um schnell alle Schichtenverletzungen zu finden.

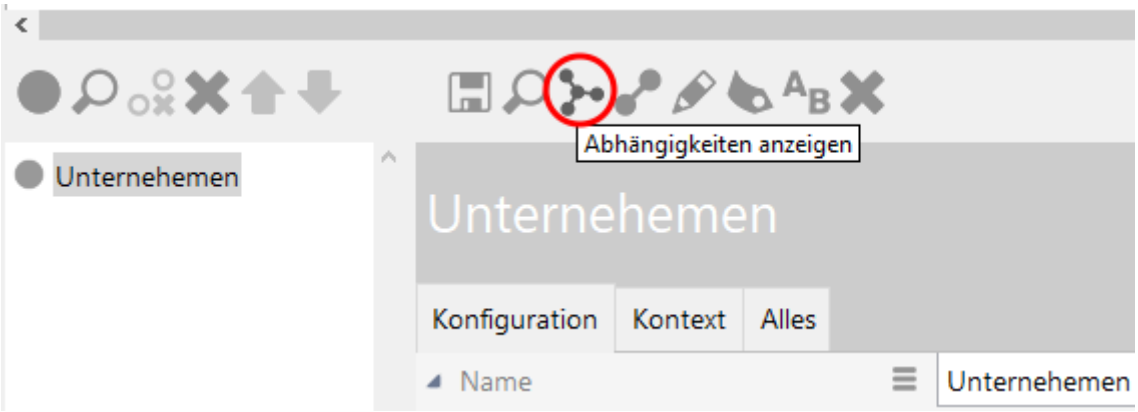


Figure 6. Der Knopf um das Werkzeug zum Finden von Schichtenverletzungen zu öffnen.

Wenn Sie unter ***TECHNIK > Benutzerdefinierte Komponenten*** eine Komponente auswählen, können Sie dort den -Knopf drücken und bekommen eine Liste aller anderen Komponenten, von denen diese abhängt.

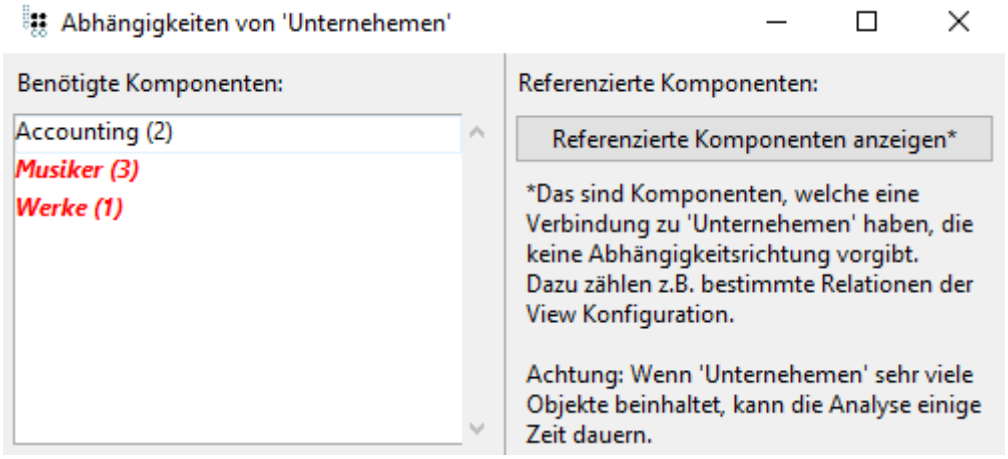


Figure 7. Das Werkzeug zum Finden von Schichtenverletzungen.

Jede Komponente zeigt in Klammern an, wie viele Abhängigkeiten sie es zu ihr gibt.
Hervorgehobene Komponenten, wurden nicht als benötigte Komponente angegeben, enthalten aber dennoch Elemente, die von der gewählten Komponente benötigt werden, weshalb sie eine Schichtenverletzung darstellen.

Wenn Sie rechts ***Referenzierte Komponenten anzeigen**** anklicken, geht eine weitere Liste auf. In dieser werden Komponenten angezeigt, welche eine Verbindung zu der analysierten haben, die keine Abhängigkeitsrichtung vorgibt.

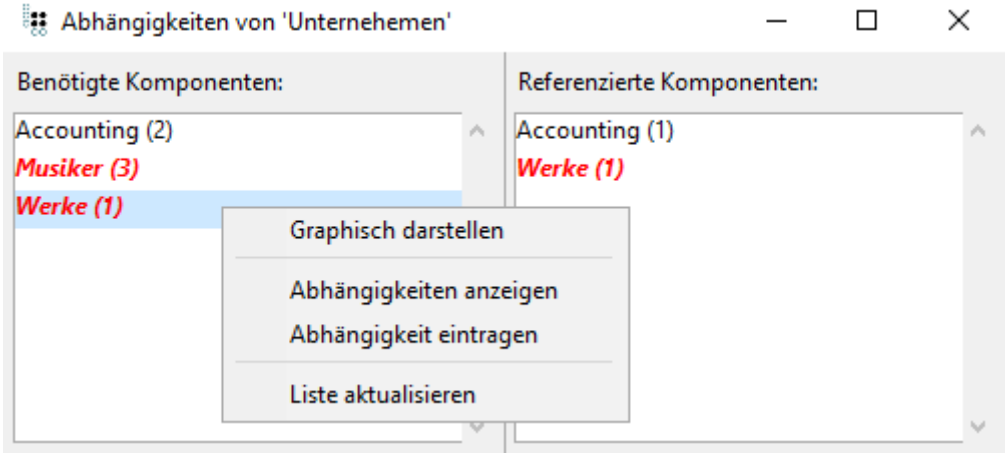


Figure 8. Das Werkzeug zum Finden von Schichtenverletzungen, nachdem die ungerichteten Referenzen ausgeklappt wurden.

HINWEIS	Wenn die zu analysierende Komponente sehr viele Objekte enthält, kann die Analyse für die rechte Liste einige Zeit dauern.

Beide Listen haben das Selbe Kontextmenü mit den folgenden Optionen:

Graphisch darstellen

Öffnet den Graph-Editor für alle Elemente, welche die ausgewählte Komponente mit der analysierten verbinden. Dafür kann auch ein Doppelklick auf eine Komponente verwendet

werden. Das Fenster bleibt dabei offen. Abhängigkeiten anzeigen: Öffnet dieses Fenster für die ausgewählte Komponente.

Abhängigkeit eintragen

Zieht die *Benötigt Komponente* Relation von der analysierten Komponente zu der ausgehwählten.

Liste aktualisieren

Aktualisiert die ausgewählte Liste, indem die Komponente erneut analysiert wird. Dabei wird nur der Teil neu analysiert, der für die jeweilige Liste relevant ist. Durch Drücken der F5 Taste werden beide Listen aktualisiert. Falls nur die linke Liste angezeigt wird, wird auch nur diese aktualisiert.

HINWEIS	Das Aktualisieren einer Liste dauert genauso lange, wie ihr erstes Öffnen.

Durch Drücken der F5 Taste, werden alle angezeigten Listen neu geladen.

1.8.2.5.8. Ändern der Zuweisungsart

Es gibt zwei Methoden, um die Zuweisungsart einer Komponente zu ändern:

1. Sie können einfach direkt an der Komponente die Zuweisungsart ändern. Alle zukünftig zugewiesenen Elemente verwenden dann die neue Zuweisungsart. Elemente, welche bereits zugewiesen waren, werden jedoch nicht angepasst.
2. Über den -Knopf kann ein Dialog zur Auswahl der neuen Zuweisungsart geöffnet werden. Hierbei werden auch alle zur Komponente zugewiesenen Elemente angepasst. Zusätzlich bietet der Dialog die Möglichkeit, alle Zuweisungsmerkmale früherer Zuweisungsarten zu entfernen.

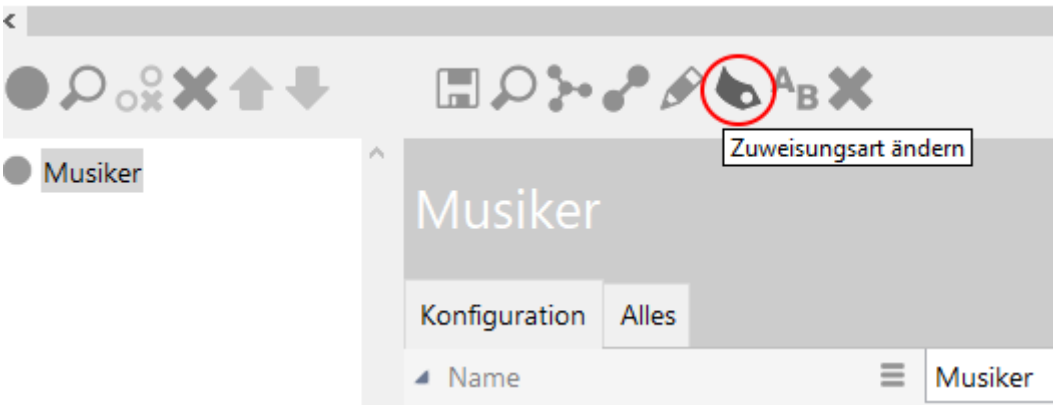


Figure 9. Der Knopf um die Zuweisungsart einer Komponente zu ändern.

1.8.2.6. Abhängigkeiten im Graph-Editor anzeigen

Über das Kontextmenü eines Elements unter ***Benutzerdefinierte Komponenten > Abhängigkeiten anzeigen***, kann der Graph-Editor mit dem ausgewählten Element und allen direkten Abhägigkeiten geöfffnet werden. Dieser Graph-Editor hat dann eine spezifische Konfiguration für

benutzerdefinierte Komponenten.

Hier werden Knoten nicht wie üblich mit Relationen verbunden, sondern die Verbindungen bedeuten, dass die Knoten voneinander Abhängen. Die Art der Verbindung sagt aus, in welche Richtung die Abhängigkeit geht.

Außerdem haben Sie hier die Möglichkeit, über einen Knopf der sich oben links an den Knoten befindet, ihre direkten Abhängigkeiten einzublenden.

Die Legende ist ebenfalls geändert und zeigt nicht die Typen der Elemente an, sondern welcher Komponente sie angehören. Komponentenobjekte selbst werden mit einer eigenen Kategorie gekennzeichnet, um Verwirrung zu vermeiden. Jeder Knoten hat dieselbe Farbe, wie seine Kategorie. Diese Farben werden automatisch zugewiesen, können aber auch manuell konfiguriert werden, indem Sie für eine Komponente das Farb-Attribut vergeben. Die Farben der Kategorien für unzugewiesene Elemente und Komponentenobjekte selbst können in den persönlichen Einstellungen definiert werden. Dort kann auch eingestellt werden, dass die Knoten, sofern möglich, das Icon ihrer Komponente verwenden.

Wie gewohnt können dem Graph-Editor neue Knoten hinzugefügt werden, auch registrierte Objekte.
Es können auch Lesezeichen angelegt werden, welche die registrierten Objekte speichern. Wenn Sie sich diese Lesezeichen im Knowledge Builder ansehen, werden allerdings keine registrierten Objekte angezeigt.

HINWEIS	Graph-Editoren mit dieser Konfiguration können nicht für kooperative Arbeit geöffnet werden.

1.8.2.7. Rechte und Trigger

Möchten Sie Komponenten-spezifische Regeln für Zugriffsrechte oder Trigger formulieren, dann können diese direkt an der jeweiligen Komponente definiert werden. Dazu dienen die beiden Attribute ***Zugriffsrechte*** und ***Trigger*** am Komponentendefinitionsobjekt.

An dieser Stelle definierte Regeln werden automatisch in den Rechte- bzw. Trigger-Entscheidungsbaum eingefügt. Benutzerdefinierte Komponenten-spezifische Regeln werden nach den Systemkomponenten-spezifischen Regeln und vor den allgemeinen vom Benutzer definierten Regeln eingefügt. Innerhalb der benutzerdefinierten Komponenten-spezifischen Regeln wird die Reihenfolge über die Komponentenabhängigkeit geregelt.

HINWEIS	Um bereits bestehende Konfigurationen für Rechte und Trigger einer Komponente zuzuweisen, können diese einfach mittels Drag-and-drop unter eine neue Konfiguration gehängt werden, welche zu einer Komponente gehört.
WARNUNG	Es ist darauf zu achten, dass alle registrierten Objekte, die von Komponenten-spezifischen Regeln verwendet werden, entweder direkt derselben oder einer der Komponenten in der Abhängigkeitskette

zugeordnet sind.

1.8.2.8. Zusätzliche Auswahl und Konfiguration von spezifischen Elementen

Standardmäßig werden Elemente anhand der Relation oder des Präfixes und der URI der Komponente ausgewählt. Diese kann jedoch durch die Option ***Elemente anhand des Komponenten-Hauptobjektes auswählen*** abgeschaltet werden.

HINWEIS	Nachdem dies abgeschaltet ist sorgen die ***Zuweisen*** Funktionen immer noch dafür, dass dem Element Präfix und Basis-URI hinzugefügt werden, jedoch ohne, dass das Element als Teil der Komponente erkannt wird.

Danach können Elemente der Komponente nur noch mithilfe folgender Unterobjekte zugewiesen werden:

- Auswahl von semantischen Elementen
- Auswahl von registrierten Objekten

Diese Unterobjekte können auf der linken Seite der Detailansicht von Komponenten erstellt werden. Sie haben Optionen, um Elemente separat von der Komponente selbst auszuwählen und zu konfigurieren. Die Konfigurationen werden ausschließlich auf Elemente angewendet, welche von den jeweiligen Objekten direkt ausgewählt werden.

Außer diesen Konfigurationen gibt es keine Unterschiede zwischen Elementen, die von der Komponente selbst oder von diesen Unterobjekten ausgewählt werden. Sie gehören alle zur Komponente und werden als eine Menge dargestellt und transferiert.

1.8.2.8.1. Auswahl von semantischen Elementen

Dieses Unterobjekt kann Elemente mittels Basis-URI, Präfix oder einer Abfrage identifizieren und der Komponente zuweisen. Von einer Abfrage ausgewählte Elemente gehören zur Komponente, ohne dabei durch Präfix oder Basis-URI verändert werden zu müssen.

Konfigurationswert	Beschreibung
Abhängigkeiten einschließen	Wenn angehakt, werden Elemente, die von ausgewählten Elementen benötigt werden, auch ausgewählt. Dies ist auf bestimmte bereits vorhandene Abhängigkeiten beschränkt. Wenn z.B. eine View-Konfigurations-Tabelle ausgewählt wird, dann werden auch ihre Tabellenspalten ausgewählt.
Alle Objekte von Komponenten-Typen auswählen	Wenn angehakt, werden alle Elemente von Typen, die von diesem Objekt ausgewählt werden, ebenfalls ausgewählt.
Präfix	Optionales Präfix, das genutzt werden kann, um Elemente für dieses Objekt zu identifizieren. Wenn kein Präfix angegeben ist, dann wird das Präfix des Komponentenobjekts verwendet.

Konfigurationswert	Beschreibung
Anhand des internen Namens auswählen	Wenn angehakt, werden Elemente, welche den in diesem Objekt angegebenen Präfix verwenden, zur Komponente hinzugefügt. Überschreibt die Einstellung in der Komponente selbst.
Basis-URI	Optionale Basis-URI, die genutzt werden kann, um Elemente für dieses Objekt zu identifizieren. Wenn keine Basis-URI angegeben ist, dann wird die Basis-URI des Komponentenobjekts verwendet.
Anhand der RDF-URI auswählen	Wenn angehakt, werden Elemente, welche die in diesem Objekt angegebene Basis-URI verwenden, zur Komponente hinzugefügt. Überschreibt die Einstellung in der Komponente selbst.
Abfrage für semantische Elemente	Eine Abfrage, deren Ergebnisse Teil der Komponente werden.

Sollte die Auswahl per internen Namen oder RDF-URI eingeschaltet sein, ohne dass jeweils Präfix oder Basis-URI gesetzt sind, werden die Werte des Komponentenobjekts selbst verwendet. Dadurch werden alle extra Optionen dieses Objekts auf alle Elemente angewendet, die vom Komponentenobjekt selbst ausgewählt sind, da sie dann auch von diesem Objekt ausgewählt werden.

1.8.2.8.2. Auswahl von registrierten Objekten

Dieses Unterobjekt kann Elemente mittels Basis-URI auswählen und einschränken, in welchen Registratur-Typen nach solchen Elementen gesucht wird.

Konfigurationswert	Beschreibung
Abhängigkeiten einschließen	Wenn angehakt, dann werden Elemente, die von ausgewählten Elementen benutzt werden, ebenfalls ausgewählt. **Beispiele:** Skripte, die Abfragen referenzieren; Abfragen mit Abfragemakros
Präfix	Optionales Präfix, der genutzt werden kann, um Elemente für dieses Objekt zu identifizieren. Wenn kein Präfix angegeben ist, dann wird das Präfix des Komponentenobjekts verwendet.
Umfasst Registry	Schränkt ein, in welchen Registratur-Typen nach Elementen mit passendem Präfix gesucht werden soll.

Sollte kein Präfix gesetzt sein, wird Wert des Komponentenobjekts selbst verwendet. Dadurch werden alle extra Optionen dieses Objekts auf alle Elemente angewendet, die vom Komponentenobjekt selbst ausgewählt sind, da sie dann auch von diesem Objekt ausgewählt werden.

1.8.2.9. Abhängigkeiten zwischen Elementen selbst hinzufügen

Mit dem Abhängigkeits-Definier-Werkzeug können Relationen als Abhängigkeitsrelationen markiert werden. Solche Relationen vermitteln den benutzerdefinierten Komponenten, dass ihre Quellen von ihren jeweiligen Zielen abhängen und werden zum Beispiel benutzt, um das Zuweisungswerkzeug zu bauen oder mögliche Komponenten für Elemente zu finden.

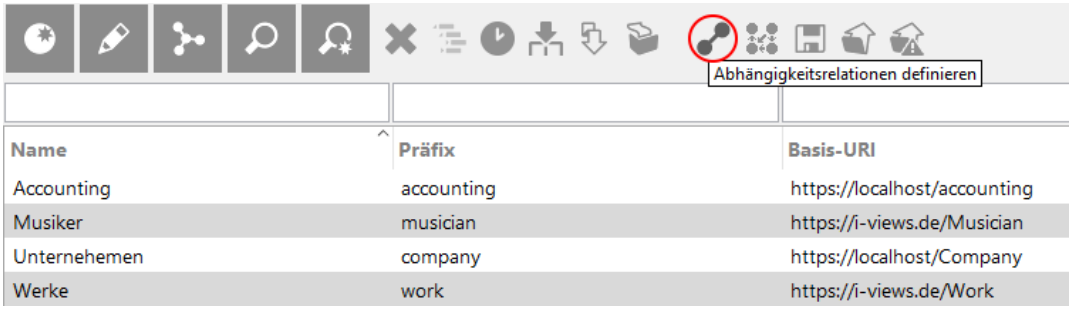


Figure 10. Der Knopf zum Öffnen des Werzeugs um bestimmte Relationen als Abhängigkeiten zu definieren.

Das Werkzeug kann unter ***TECHNIK > Benutzerdefinierte Komponenten*** durch Klicken auf das Relations-Symbol der jeweiligen Komponente geöffnet werden.

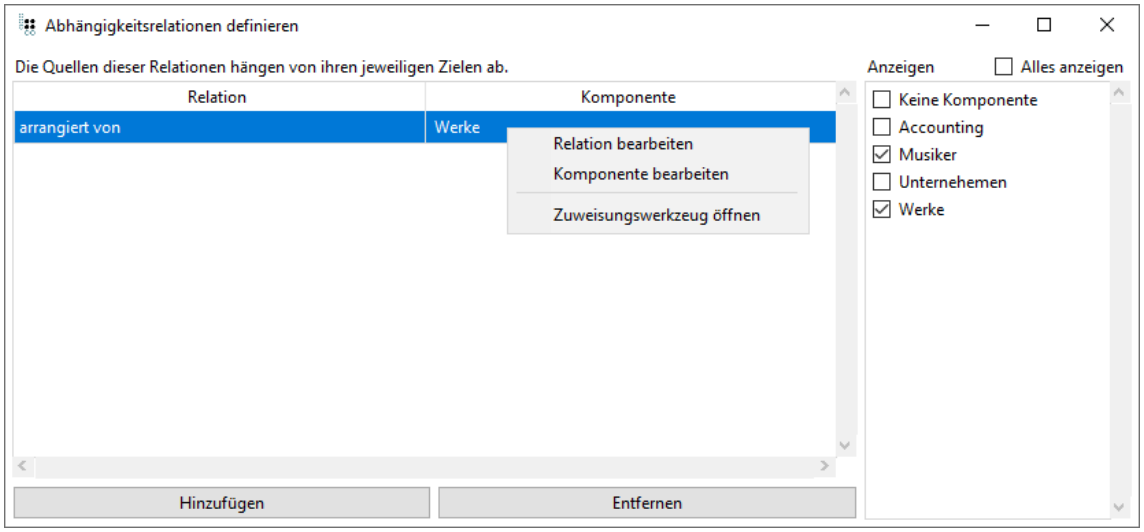


Figure 11. Das Werkzeug um bestimmte Relationen als Abhängigkeiten zu definieren.

Links ist eine Tabelle mit allen Relationen, die als Abhängigkeitsrelationen markiert wurden, zusammen mit der Komponente, der sie zugewiesen sind. Die Tabelle ist alphabetisch nach Komponenten und dann dem Namen der Relation sortiert.

Wenn die Tabelle eine Relation und ihre inverse Relation enthält, werden beide in fetten, roten Buchstaben hervorgehoben. Außerdem werden, wenn eine Relation zu mehreren Komponenten zugewiesen ist, diese gleichemaßen hervorgehoben.

Das Kontextmenü von Tabellenzeilen erlaubt es neue Fenster zum Bearbeiten der Relation oder ihrer Komponente oder ein Zuweisungwerkzeug für die Relation zu öffnen. Ein Doppelklick öffnet ebenfalls ein Fenster zum Bearbeiten der Relation.

Auf der rechten Seite befinden sich Filter für alle Komponenten im Graph, sowie ein Filter für Relationen, die noch keiner Komponente zugewiesen sind. Oben können Sie auch eine Box anhaken, um unabhängig von den Filtern alle Relationen anzuzeigen. Zu Beginn ist nur der Filter der Komponente angehakt, welche beim Öffnen des Werkzeugs ausgewählt war.

Unter der Tabelle befinden sich zwei Knöpfe, um neue Relationen als Abhängigkeitsrelationen zu markieren oder bestehende zu entfernen. Wenn Sie eine neue Relation hinzufügen, wird automatisch der passende Filter für die Komponenten dieser Relation angehakt.

1.8.2.10. Ungültige Zuweisungen entfernen

Gibt es Elemente, welche keiner benutzerdefinierten Komponente zugewiesen werden sollten, die dennoch eine Zuweisung haben, können diese Zuweisungen hiermit entfernt werden.

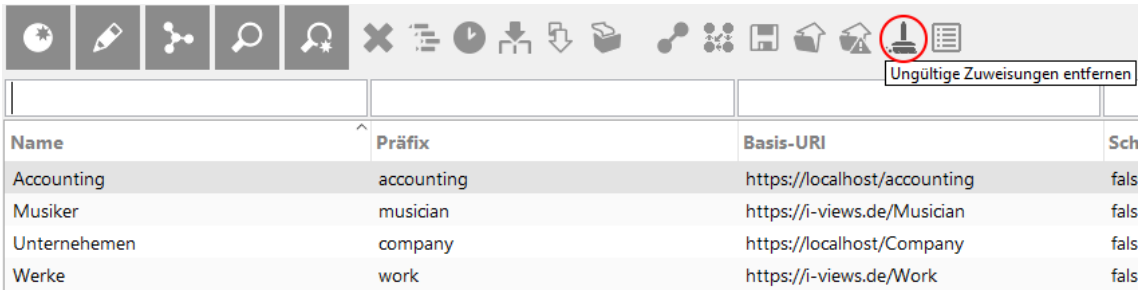


Figure 12. Der Knopf zum Bereinigen aller Zuweisungen.

Ein Dialog wird fragen, ob die Präfixe von validen Zuweisungen ebenfalls angepasst werden sollen, falls der Präfix einer benutzerdefinierten Komponente dem einer Systemkomponente gleicht. Wenn das passiert, wird an den Präfix der aller Elemente der betroffenen benutzerdefinierten Komponente `_Collision` angehängt. Diese Präfixe können dann, wie im Kapitel Ändern von Präfix und Basis-URI beschrieben, angepasst werden, wodurch auch alle Elemente mit angepasst werden.

HINWEIS

Zuweisungen semantischer Elemente per Abfrage können hierdurch nicht entfernt werden.
Wenn ein Element Teil einer Systemkomponente ist und durch Anpassen des Präfixes zu einer benutzerdefinierten Komponente zugewiesen wurde, kann der ursprüngliche Präfix, welcher von der Systemkomponte vergeben wurde, nicht wiederhergestellt werden.

1.8.2.11. Graphbewertung

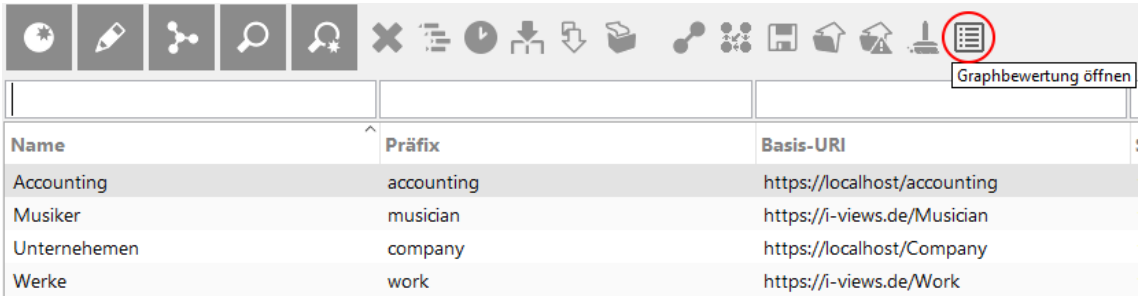


Figure 13. Der Knopf zum Öffnen des Werkzeugs zur Graphbewertung.

Dieses Werkzeug analysiert Probleme der Zuweisung von Elementen sowie die Modellierung des Graphen in Bezug auf benutzerdefinierte Komponenten.

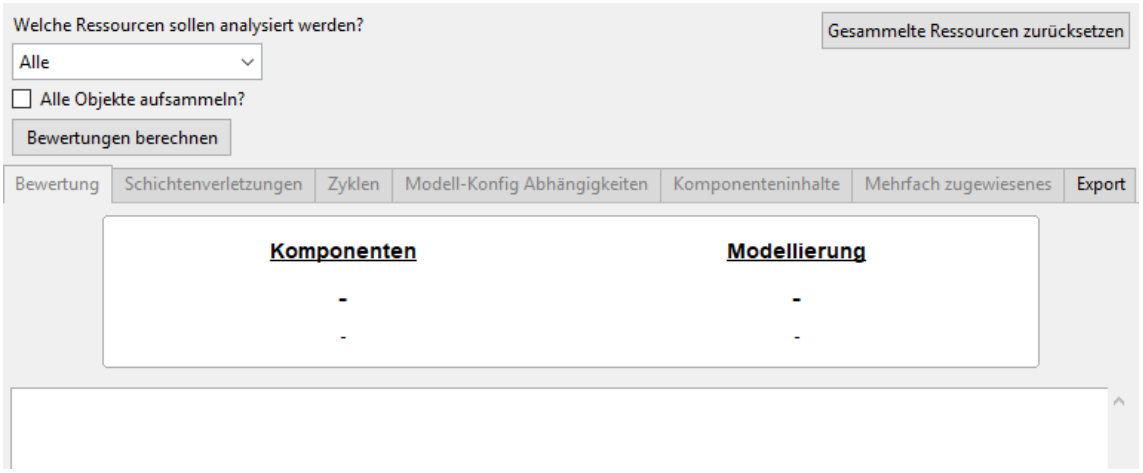


Figure 14. Die initiale Ansicht des Werkzeugs zur Graphbewertung.

Im oberen Bereich kann eingestellt werden, was alles analysiert werden soll.
Dabei gibt es folgende Optionen:

Alle

Alle Elemente, welche einer benutzerdefinierten Komponente zugewiesen werden können.

Unzugewiesene

Alle Elemente, welche einer benutzerdefinierten Komponente zugewiesen werden können, aber noch keine Zuweisung haben.

Zugewiesene

Alle Elemente, welche einer benutzerdefinierten Komponente zugewiesen sind.

Komponenten

Die Objekte, welche die im Graph definierten benutzerdefinierten Komponenten repräsentieren.

Zusätzlich kann eingestellt werden, ob alle Objekte analysiert werden sollen. Diese Einstellung hat keine Auswirkungen, wenn zuvor `Komponenten` ausgewählt wurde. Standartmäßig ist dies ausgeschaltet, da die Analyse durch diese vielen zusätzlichen Elemente deutlich länger dauert und für gewöhnlich keine weiteren Probleme findet. Abhängigkeiten von Elementen, welche keine

Objekte sind zu Objekten werden immer in der Analyse berücksichtigt.

Darunter kann dann die Analyse gestartet werden und oben rechts künnen die gesammelten Daten wieder zurückgesetzt werden, um eine neue Analyse zu starten, nachdem Änderungen vorgenommen wurden.
Im unteren Teil des Werkzeugs werden die Auswertungen der Analyse angezeigt, welche mithilfe der Tabs gewechselt werden können. Ist ein Tab ausgegraut, bedeutet das, dass er keine Inhalte hat. Entweder, weil die Analyse noch nicht gestartet wurde oder keine Probleme dieser Art gefunden hat.

HINWEIS	Alle Listen und Tabellen im unteren Beriech bieten die Möglichkeit per Doppelklick oder Rechtsklick zu den ausgewählten Elementen zu navigieren, sie in der Komponentenansicht des Graph-Editors anzuzeigen oder das Zuweisungswerkzeug mit ihnen zu öffnen.

1.8.2.11.1. Bewertung

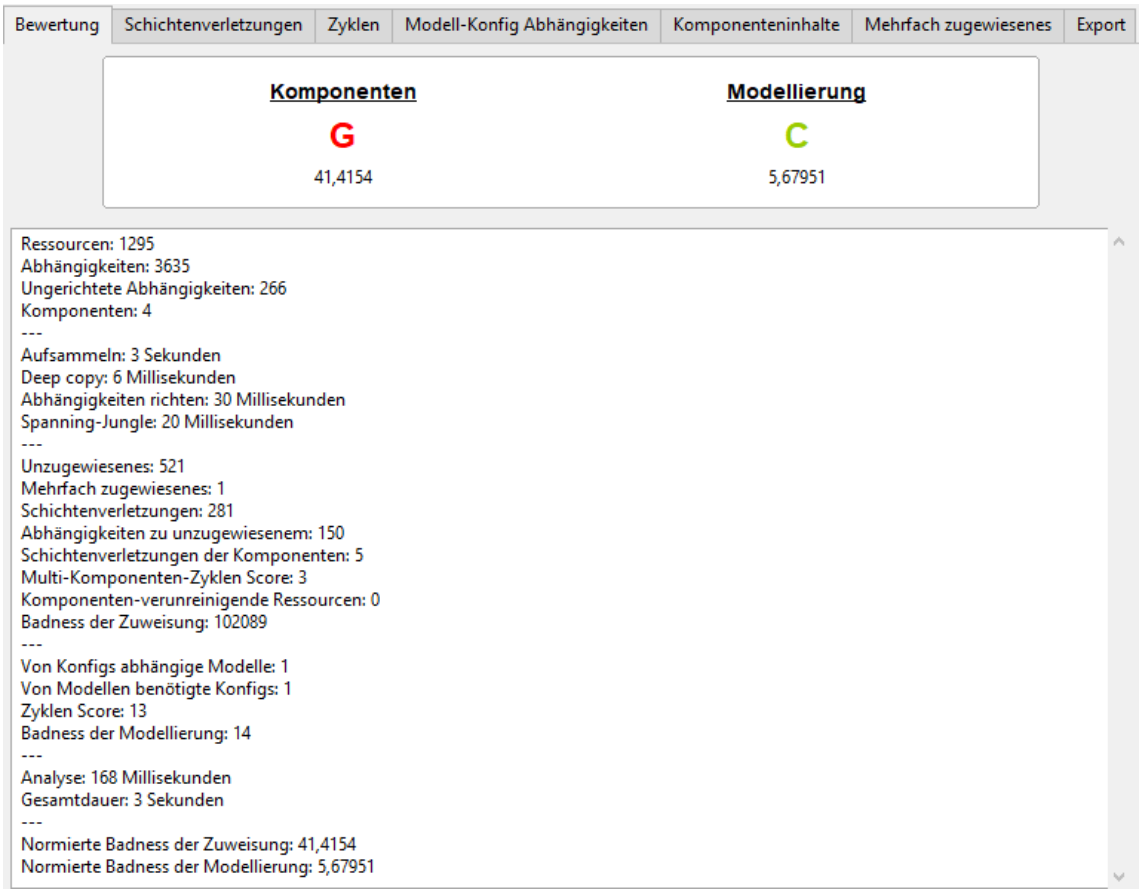


Figure 15. Der Bewertungs-Tab des Werkzeugs zur Graphbewertung.

Im oberen Bereich sind die Bewertungen der Komponentisierung und der Modellierung zu sehen. Da dieses Werkzeug nur Probleme analysiert, sind die Bewertungen als negative Werte zu verstehen und 0 ist die beste Bewertung, die erreicht werden kann. Da die Zahlenwerte sehr abstrakt sind, werden sie auf eine Skala von **A** bis **G** abgebildet, in der **A** die beste und **G** die schlechteste

Bewertung ist. Wie die Zahlenwerte übersetzt werden, wird angezeigt, wenn Sie den Cursor über die entsprechenden Zahlenwerte halten.

Buchstabe	Komponentenbewertung	Modellierungsbewertung
A	0	0
B	< 1	< 5
C	< 5	< 10[1]
D	< 10	< 25
E	< 20	< 50
F	< 30	< 100
G	>= 30	>= 100

[1]Sollter der Zyklen Score die einzige Metrik über 0 sein, kann die Modellierungsbewertung nicht schlechter als **C** ausfallen.

Darunter sind die einzelnen Statistiken aufgelistet, aus denen sich die Bewertungen zusammensetzen:

Ressourcen

Die Anzahl analysierter Elemente. (Siehe Komponenteninhalte)

Abhängigkeiten

Die Anzahl aller Abhängigkeiten zwischen den analysierten Elementen.

Ungerichtete Abhängigkeiten

Die Anzahl aller Abhängigkeiten zwischen den analysierten Elementen, welche keine vorgegebene Richtung haben.

Komponenten

Die Anzahl der benutzerdefinierten Komponenten, zu denen die analysierten Elemente zugewisen sind. (Siehe Komponenteninhalte)

Aufsammeln

Wie lange es gedauert hat, alle Elemente gemäß der Einstellungen zu sammeln.

Deep Copy

Wie lange es gedauert hat eine Deep copy des resultierenden Graphen zu erstellen.

Abhängigkeiten richten

Wie lange es gedauert hat, für alle ungerichteten Abhängigkeiten die möglichst problemfreie Richtung zu ermitteln.

Spanning-Jungle

Wie lange es gedauert hat, der Spanning Jungle zu erstellen.

Unzugewiesenes

Die Anzahl an unzugewiesenen Elementen. (Siehe Komponenteninhalte)

Mehrfach zugewiesenes

Die Anzahl an Elementen, welche zu mehreren benutzerdefinierten Komponenten gleichzeitig zugewiesen sind. (Siehe Mehrfach zugewiesenes)

Schichtenverletzungen

Die Anzahl an Schichtenverletzungen, welche aus den gesammelten Elementen resultieren. (Siehe Schichtenverletzungen)

Abhängigkeiten zu unzugewiesenem

Die Anzahl an Abhängigkeiten von Elementen, welche einer benutzerdefinierten Komponente zugewiesen sind von Elementen, welche keine Zuweisung haben. (Siehe Schichtenverletzungen)

Schichtenverletzungen der Komponenten

Die Anzahl an Schichtenverletzungen auf Komponentenebene.

Multi-Komponenten-Zyklen-Score

Ein Wert, welcher aus der Anzahl und Größe der Zyklen gebildet wird, welche über mehrere Komponenten verlaufen. (Siehe Zyklen)

Komponenten-verunreinigende Ressourcen

Die Anzahl an Elementen des Modells oder der Konfiguration, welche benutzerdefinierten Komponenten zugewiesen sind, die überwiegend aus Elementen der jeweils anderen Kategorie bestehen. (Siehe Komponenteninhalte)

Badness der Zuweisung

Die Bewertung der Zuweisung, welche aus den oben stehenden Statistiken errechnet wurde.

Von Konfigs abhängige Modelle

Die Anzahl an Elementen des Modells, welche von Elementen der Konfiguration abhängig sind. (Siehe Modell-Konfig Abhängigkeiten)

Von Modellen benötigte Konfigs

Die Anzahl an Elementen der Konfiguration, welche von Elementen des Modells banötigt werden. (Siehe Modell-Konfig Abhängigkeiten)

Zyklen Score

Ein Wert, welcher aus der Anzahl und Größe aller Zyklen gebildet wird. (Siehe Zyklen)

Badness der Modellierung

Die Bewertung der Modellierung, welche aus den oben stehenden Statistiken errechnet wurde.

Analyse

Wie lange die Analyse der obigen Statistiken gedauert hat.

Gesamtdauer

Wie lange der gesamte Prozess inklusive dem Aufsammeln der Elemente und der Analyse gedauert hat.

Normierte Badness der Zuweisung

Die Badness der Zuweisung, welche mithilfe der Größe des Graphen normiert wurde, um sie besser mit anderen Graphen vergleichen zu können.

Normierte Badness der Modellierung

Die Badness der Modellierung, welche mithilfe der Größe des Graphen normiert wurde, um sie besser mit anderen Graphen vergleichen zu können.

HINWEIS	Es ist durchaus möglich, dass ein fertig modellierter und zugewiesener Graph keine perfekten Werte in diesen Bewertungen erzielt, da zum Beispiel Zyklen innerhalb einer benutzerdefinierten Komponente nicht unbedingt problematisch sein müssen.

1.8.2.11.2. Schichtenverletzungen

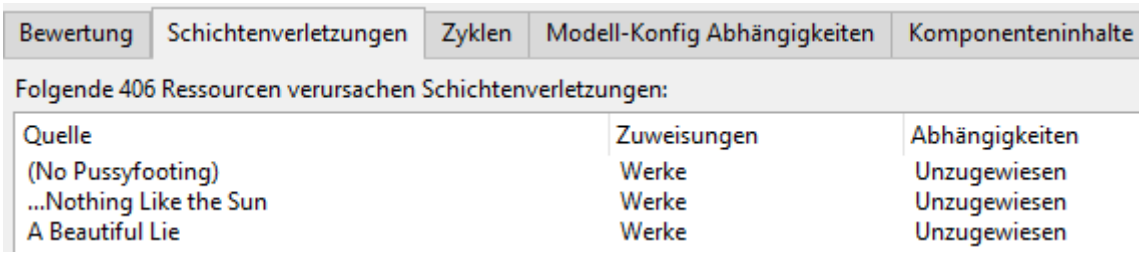


Figure 16. Der Schichtenverletzungs-Tab des Werkzeugs zur Graphbewertung.

Hier werden alle Schichtenverletzungen aufgelistet.
In der ersten Spalte sehen Sie das relevante Element, in der zweiten, welche Zuweisungen das Element hat und in der dritten, von welchen anderen Komponenten es abhängt, die nicht als Abhängigkeit eingetragen sind.

1.8.2.11.3. Zyklen

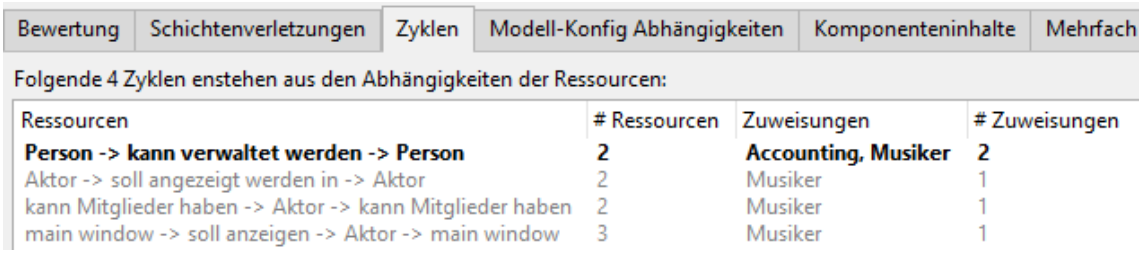


Figure 17. Der Zyklen-Tab des Werkzeugs zur Graphbewertung.

Auf diesem Tab werden alle zyklischen Abhängigkeiten zwischen den analysierten Elementen aufgelistet.
In der ersten Spalte werden die Elemente aufgelistet, welche einen Zyklus bilden. Die zweite Spalte enthält die Größe des Zyklus. In der dritten Spalte stehen alle Komponenten, über die der Zyklus verläuft, gefolgt von der Anzahl dieser Komponenten in der vierten Spalte.

Da Zyklen innerhalb einer Komponente keine Probleme bei der Zuweisung und dem Transfer verursachen, sind diese ausgegraut.

1.8.2.11.4. Modell-Konfig Abhängigkeiten

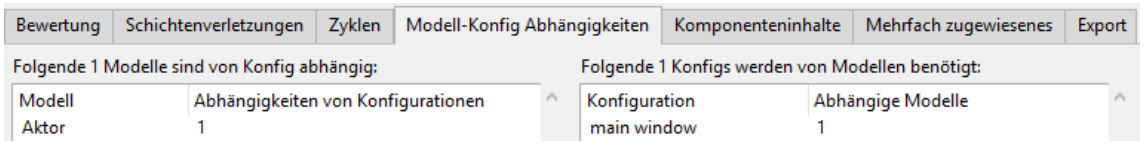


Figure 18. Der Modell-KonfigurationsAbhängigkeiten-Tab des Werkzeugs zur Graphbewertung.

In dieser Ansicht werden alle Abhängigkeiten aufgeslistet, welche vom Modell, also z.B. vom Schema, zu Elementen der View-Konfiguration oder REST-Konfigurations gehen.
Die Linke Tabelle zeigt alle Modellelemente mit solchen Abhängigkeiten und wie viele solcher Abhängigkeiten jedes Element hat. Rechts wird die andere Richtung dargestellt, also alle Konfigelemente von denen Modellelemente abhängen und ebenfalls wie viele solcher Abhängigkeiten auf die jeweiligen Konfigelementen zeigen.

1.8.2.11.5. Komponenteninhalte

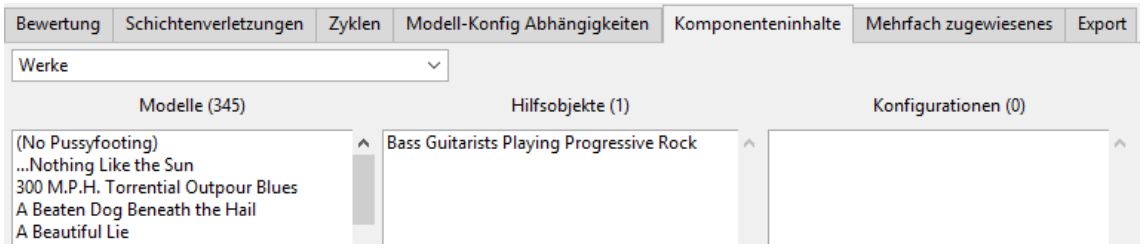


Figure 19. Der Komponenteninhalte-Tab des Werkzeugs zur Graphbewertung.

In diesem Tab werden alle analysierten Elemente, gruppiert anhand ihrer Zuweisung aufgelistet. Oben links kann eine Komponente ausgewählt werden, von der dann alle zugewiesenen Elemente angezeigt werden. Die Elemente werden weiter in Modellelemente (Schema), Hilfsobjekte (Skripte, Abfragem, etc.) und Konfigurationen (View-Konfiguration, REST) aufgeteilt. Hier können Sie also auch sehen, ob eine Komponente Modelldefinitionen mit Konfigurationen vermischt.

Anstatt sich die Inhalte einer spezifischen Komponente anzeigen zu lassen, können auch alle Elemente angeziegt werden, die keiner Komponente zugewiesen sind.

1.8.2.11.6. Mehrfach zugewiesenes

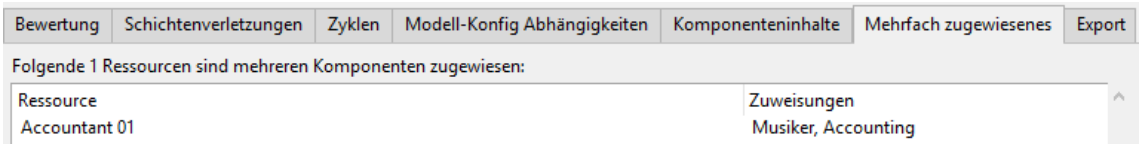


Figure 20. Der Tab mit mehrfach zugewiesenen Elementen des Werkzeugs zur Graphbewertung.

Dieser Tab zeigt alle Elemente an, welche zu mehr als einer Komponente gleichzeitig zugewiesen sind und was diese Zuweisungen sind.

1.8.2.11.7. Export

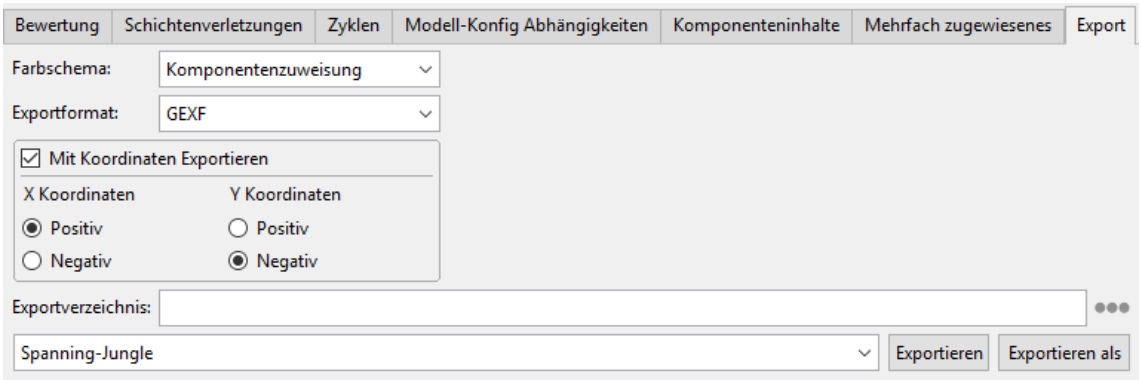


Figure 21. Der Export-Tab des Werkzeugs zur Graphbewertung.

Über diesen Tab können Abhängigkeitsgraphen der analysierten Elemente exportiert werden. Diese können verschiedene Formen annehemen und in verschiedenen Formaten exportiert werden.

Farbschema

Bestimmt wie die exportierten Knoten eingefärbt werden:

Komponentenzuweisung

Die Knoten werden anhand ihrer zugewiesenen Komponente gefärbt. Die Farben sind dieselben, welche auch von der Komponentenansicht des Graph-Editors verwendet werden.

Ressourcentyp (Modell/Konfig)

Die Knoten werden danach eingefärbt, ob sie zu Modell (*Rot*) oder Konfig (*Blau*) gehören. Diese Farben werden dann gegebenenfalls angepasst, wenn die Knoten von nichts abhängen (*Grün*) oder gar keine Verbindungen besitzen (*Weiß*).

Exportformat

Gibt an, in welchem Format der gewählte Graph exportiert werden soll.

Format	Farben	Koordinaten	Knoten	Kanten	gemischter Graph
GEXF - Graph Exchange XML Format	Ja	Ja	Ja	Ja	Ja
PajekNET	Ja	Ja	Ja	Ja	Nein

Format	Farben	Koordinaten	Knoten	Kanten	gemischter Graph
GDF - Graph Description Format	Ja	Ja	Ja	Ja	Ja
GML - Graph Modelling Language	Ja	Ja	Ja	Ja	Nein
CSV	Nein	Nein	Ja	Ja	Nein
CSV (Knoten)	Ja	Ja	Ja	Nein	Nein
CSV (Kanten)	Nein	Nein	Wenn sie Kanten haben	Ja	Nein

Koordinaten

Es kann ausgewählt werden, ob für die Knoten der exportierten Graphen Koordinaten berechnet werden sollen. Dies geht nur bei vollständig gerichteten Graphen, bei denen die Koordinaten versuchen eine hierarchische Struktur zu erzielen. Dabei haben Knoten, welche tifer in der Hierarchie sind eine größere y-Koordinate.
Die beiden Achsen der Koordinaten können beliebig invertiert werden, sodass der Graph eine frei wählbare Ausrichtung hat. Standarmäßig wird davon augegangen, dass die x-Achse positiv nach rechts und die y-Achse positiv nach oben verläuft.

Exportverzeichnis

Wenn kein Exportverzeichnis ausgewählt ist, wird die Datei im selben Ordner wie der Knowledge Builder abgelegt. Sobald ein Verzeichnis ausgewählt wird, wird dieses am Nutzer gespeichert, sodass es nicht bei jedem export ernet gewählt werden muss.

Graph

Gibt an, welcher Graph exportiert werden soll:

Spanning-Jungle

Ein Baumartiger Graph, in dem alle Knotenin eine Hierarchie eingefügt werden.

Abstrahierter Komponentengraph

Ein Graph, bei dem alle Knoten mit derselben Zuweisung zu einem Knoten zusammengefasst wurden.

Gerichteter abstrahierter Komponentengraph

Wie der Abstrahierte Komponentengraph, bei dem dann aber allen Kanten eine Richtung zugewiesen wurde.

Gerichteter Graph

Der Graph aller analysierter Elemente, in dem allen Kanten eine Richtung zugeordnet wurde.

Originalgraph

Der Graph aller analysierter Elemente.

Der ***Exportieren*** Knopf exportiert den Graph mit einem generierten Namen, der den Namen des Volumes sowie alle eingestellten Optionen enthält und speichert ihn im gewählten Exportverzeichnis.

Mit dem ***Exportieren als*** Knopf können der Name und das Exportverzeichnis selbst gewählt werden.

1.8.2.12. Einstellungen

Um zu den Einstellungen für benutzerdefinierte Komponenten zu kommen müssen Sie auf das Zahnrad oben rechts im Knowledge Builder klicken.

Dort gibt es zwei Arten von Einstellungen:

- Einstellungen zur Darstellunge von benutzerdefinierten Komponenten im ***Persönlich*** Tab
- Einstellungen zu Zuweisungen zu benutzerdefinierten Komponenten im ***System*** Tab

1.8.2.12.1. Persönliche Einstellungen

Diese Einstellungen ändern die Darstellung von benutzerdefinierten Komponenten für den Nutzer. Sie sind an den Nutzer gebunden und haben daher keinen Einflussen auf andere Nutzer.

Komponentenzugehörigkeit im Banner anzeigen

Wenn aktiviert, werden bei Elementen ihre zugewiesenen Komponenten auf der rechten Seite ihrer Bannerregion angezeigt.

Spalte mit Komponentenzugehörigkeit im Tabellen anzeigen

Wenn aktiviert, wird in den meisten standard Tabellen eine zusätzliche Spalte mit den zugewisenen Komponenten der Elemente in der Tabelle angezeigt.

HINWEIS	Unabhängig von dieser Einstellung kann die Spalte für benutzerdefinierte Komponenten konfiguriert werden. Dazu muss in den Knowledge-Builder-Einstellungen unter ***Persönlich > Editoren*** die Option ***Einstellungen für Tabellenspalten anzeigen*** aktiviert werden. Dann kann das Menü oben rechts an Tabellen geöffnet werden und es können beliebige Spalten ausgewählt werden. Diese Einstellungen überschreiben die Einstellung zum Anzeigen der Spalte für benutzerdefinierten Komponenten.

Objekte zur Berechnung von Komponenten verwenden

Wenn aktiviert, werden zur Ermittlung der Abhängigkeiten von Komponenten, die Objekte von Typen miteinberechnet. Außerdem werden diese Objekte auch im Zuweisungswerkzeug angezeigt.

WARNUNG	Diese Option ist standardmäßig ausgeschaltet, da die Performance (vorallem

> beim Öffnen des Zuweisungswerkzeugs) stark beeinträchtigt werden kann, wenn es sehr viele Objekte gibt und sich am Resultat meistens nicht viel ändert.

Dialog zum Überschreiben der Handhabung überschüssiger Elemente anzeigen

Wenn aktiviert, geht vor jedem Import ein Dialog auf, um auszuwählen, ob Sie die an der Komponente konfigurierte Handhabung überschüssiger Elemente überschreiben möchten. Das Überschreiben betrifft nur diesen konkreten Import und wird nicht gespeichert.

Einstellungen für die Komponentenansicht des Graph-Editors

Diese Einstellungen wirken sich auf Graph-Editoren mit der speziellen Konfiguration für benutzerdefinierte Komponenten aus:

Farbe der unzugewiesenen Elemente

Hier kann die Farbe der Legendenkategorie für Elemente ohne Zuweisung definiert werden.

Farbe der benutzerdefinierten Komponenten

Hier kann die Farbe der Legendenkategorie für Komponentenobjekte selbst definiert werden.

Das Icon einer Komponente für ihre Elemente verwenden

Wenn aktiviert, werden Icons, die an Komponenten definiert sind, auch für alle Knoten verwendet, welche ihnen zugewiesen sind und Icons anzeigen können.

1.8.2.12.2. System-Einstellungen

Diese Einstellungen ändern das Verhalten der benutzerdefinierten Komponenten beim Zuweisen von Elementen. Sie sind Systemweit und beeinflussen daher auch andere Nutzer.

Beim Zuweisen den Präfix zu Konfigurationsnamen hinzufügen

Wenn aktiviert, wird Konfigurationsnamen von View-Konfigurations-Elementen beim Zuweisen zu einer Komponente der Präfix der Komponente hinzugefügt.

Beim Zuweisen fehlende Konfigurationsnamen erstellen

Wenn aktiviert, wird beim Zuweisen von View-Konfigurations-Elementen ohne Konfigurationsnamen, ein neuer Konfigurationsname erstellt wenn möglich.

Beim Zuweisen fehlende interne Namen erstellen

Wenn aktiviert, wird beim Zuweisen von sematischen Elementen ohne internen Namen, ein neuer interner Name erstellt wenn möglich.

Beim Zuweisen eines Mappings auch die verwendete Datenquelle zuweisen

Wenn aktiviert, wird beim Zuweisen von Abbildungen auch die verwendete Datenquelle zur ausgewählten Komponente zugewiesen.

Nur wenn die Datenquelle noch keiner Komponente zugewiesen ist

Dies ist eine Einschränkung für die vorige Option. Wenn aktiviert, werden Datenquellen nur mit der Abbildung zugewiesen, wenn sie noch nicht zu einer anderen Komponente zugewiesen sind.

Beim Zuweisen einer Datenquelle den Registrierungsschlüssel des eindeutig zugeordneten Mappings als Fallback übernehmen

Wenn aktiviert, werden Datenquellen, die keinen Registrierungsschlüssel und auch keine andere Möglichkeit als ihre Private ID haben einen zu erstellen, den Registrierungsschlüssel ihrer Abbildung kopieren.

Beim Zuweisen eines Objektes auch alle verwendeten Erweiterungen zuweisen

Wenn aktiviert, werden beim Zuweisen eines Semantischen Elemnets auch alle Erweiterungsobjekte dieses elements zur ausgewählten Komponente zugewiesen.

Nicht, wenn die Erweiterungen einer anderen Komponente als ihr Kern-Objekt zugewiesen sind

Dies ist eine Einschränkung für die vorige Option. Wenn aktiviert, werden Erweiterungen nur mit ihrem Kern-Objekt zugewiesen, wenn sie nicht schon einer anderen Komponente zugewiesen sind.

1.8.2.13. Transfer

Beim Transfer von Komponenten aus einem Graph in einen anderen, werden alle Objekte von Typen, die in beiden Graphen existieren und identifiziert werden können, wie folgt synchronisiert:

- Attribute werden immer vom Import überschrieben.
- Bei Relationen kommt das Verhalten darauf an, in welcher Komponente sich das Relationsziel befindet.

Komponente des Relationsziels	Wird vom Import überschrieben
Die importierte Komponente	Ja
Eine Komponente, von der die Importierte abhängt	Ja
Eine von der Importierten abhängige Komponente	Nein
Eine von der Importierten unabhängige Komponente	Nein
Keine Komponente	Nein

Analog dazu werden Relationen gar nicht erst exportiert, wenn die Zuweisung des Zielobjekts wie folgt ist:

- Unzugewiesen
- Zugewiesen zu einer Komponente, welche keine definierte Abhängigkeit mit der Exportierten

hat

- Zugewiesen zu einer Komponente, welche abhängig von der Exportierten ist

HINWEIS	Wenn ein Typ der Komponente einen Obertyp hat, muss dieser Obertyp entweder in derselben Komponente oder Teil einer Komponente sein, die von der Komponente des Subtyps als Abhängigkeit konfiguriert ist. Ist dies nicht der Fall, wird der Typ nach einem Import nicht mehr als Untertyp, sondern als unabhängiger Typ existieren. Die einzige Ausnahme davon ist, wenn der Obertyp Teil einer Softwarekomponente ist, da es momentan noch nicht möglich ist, die Abhängigkeit einer benutzerdefinierten Komponente von einer Softwarekomponente zu konfigurieren.

Komponenten können auch mit Hife des Batch-Tools exportiert und importiert werden, wie im Kapitel Kommandozeilen Befehle beschrieben ist.

1.8.2.13.1. Export

Eine ausreichend konfigurierte Komponente kann jederzeit im Knowledge Builder in der Detailansicht der Komponente durch Klick auf den -Knopf exportiert werden.

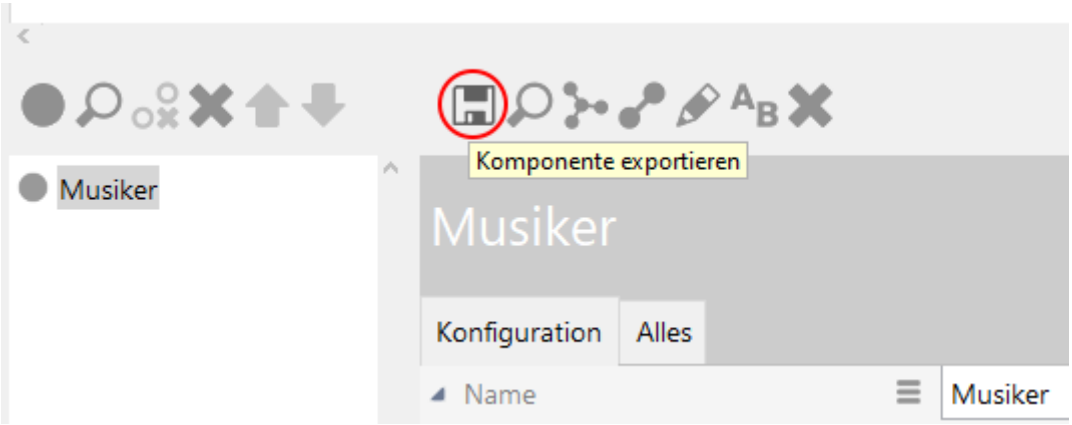


Figure 22. Der Knopf um einzelne Komponenten zu exportieren.

Ebenfalls kann unter ***TECHNIK > Benutzerdefinierte Komponenten*** der -Knopf über der Tabelle verwendet werden. Dieser exportiert alle markierten Komponenten in eine Datei.

HINWEIS	Beim Export einer Komponente erhalten alle semantischen Elemente eine RDF-URI, wenn sie noch keine haben. Wenn die Komponente ein Attribut zur Identifikation beim Transfer definiert hat, werden auch alle fehlenden IDs für semantische Elemente erzeugt.

Beim Exportieren haben Sie dann zwei Optionen:

1. **Alles:** Exportiert die Komponente mitsamt allen zugewiesenen Elementen
2. **Definition:** Exportiert nur die Komponente selbst und ihre Unterobjekte ohne jegliche zugewiesene Elemente

HINWEIS	Wenn nur die Definition einer Komponente erneuert wird, hat das Attribut ***Handhabung von überschüssigen Elementen*** keinen Effekt, da es alle Elemente der Komponente betreffen würde.

Im Exportdialog kann dann auch eine Version für den Dateinamen angegeben werden. Exportieren Sie nur eine Komponente und gibt eine Version an, wird diese Version auch an der Komponente gesetzt.

Bevor eine Komponente exportiert werden kann, müssen einige Bedingungen erfüllt sein:

- Die Komponente hat einen Namen.
- Die Komponente hat einen Präfix.
- Die Komponente hat eine valide Basis-URI.
- Der Graph hat eine valide Basis-URL (Diese kann in den Knowledge-Builder-Einstellungen unter ***System > RDF*** eingestellt verden).
- Die Komponente weiß wie sie mit üerschüssigen Elementen umgehen soll.
- Die Komponente hat keine zyklischen Abhängigkeiten.
- Wenn die Komponente ein Attribut zur Identifikation beim Transfer definiert hat, muss dieses eine RDF-URI, einen internen Namen und einen Eindeutigkeitsindex haben.
- Wenn die Komponente ein Attribut zur Identifikation beim Transfer definiert hat, muss dieses sowohl an sich selbst, als auch an der Komponente angebracht werden können.

Zusätzlich gibt es noch Dinge, die nicht notwendig, aber sehr hilfreich sind:

- Wenn die Komponente ein Attribut zur Identifikation beim Transfer definiert hat, sollte dieses ein Zeichekettenattribut sein, da sonst keine fehlenden IDs automatisch generiert werden können.
- Wenn die Komponente ein Attribut zur Identifikation beim Transfer definiert hat, sollte es der Komponente zugewiesen sein, von der es verwendet wird.

HINWEIS	Mit dem  -Knopf über der Tabelle aller Komponenten, können Sie die konfigurierten Abhängigkeiten aller aktuell markierten Komponenten markieren.

1.8.2.13.2. Import

Eine exportierte Komponente kann über das Admin Tool oder per Knowledge Builder importiert werden.

Import per Admin Tool:

1. Menü ***Systemkonfiguration > Komponenten*** aufrufen.
2. Knopf ***Modellkomponente hinzufügen*** klicken und ***Komponente importieren*** wählen.

3. Exportierte Datei auswählen oder Datei-URL angeben.
4. Auswählen, ob Sie die Handhabung überschüssiger Elemente für diesen Import der zu importierenden Komponente überlassen oder ihn selbst wählen.
5. **Ergebnis:** Die importierte Komponente erscheint in der Liste der Komponenten. Falls die Komponente `Benutzerdefinierte Komponenten` noch nicht vorhanden ist, wird sie automatisch hinzugefügt.

Nach dem Import ist das Komponentenobjekt im Knowledge Builder zu finden unter: ***Technik > Benutzerdefinierte Komponenten***.

Import per Knowledge Builder:

HINWEIS	Diese Variante setzt voraus, dass die Komponente `Benutzerdefinierte Komponenten` bereits dem System hinzugefügt wurde.

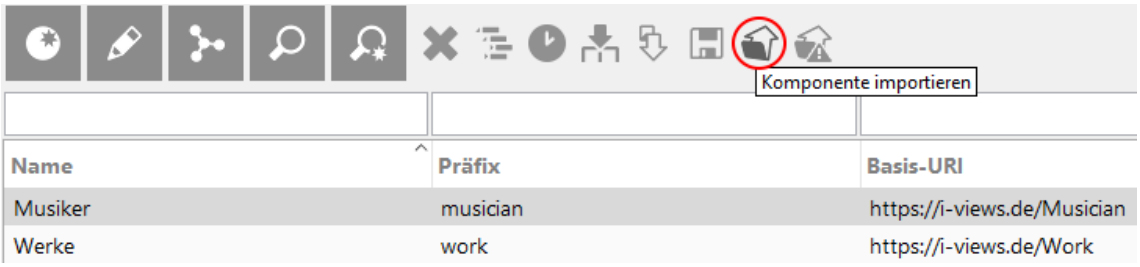


Figure 23. Der Knopf zum importieren einer Komponente.

1. Zum Menüpunkt ***Technik > Benutzerdefinierte Komponenten*** navigieren.
2. Auf den -Knopf klicken, welcher ganz rechts in der obersten Leiste zu finden ist.
3. Exportierte Datei auswählen oder Datei-URL angeben.
4. Auswählen, ob Sie die *Handhabung überschüssiger Elemente* für diesen Import der zu importierenden Komponente überlassen oder ihn selbst wählen.

Warnungen nach dem Import:

Sind beim Import keine Probleme aufgetreten und es gab keine überflüssigen Elemente, erscheint unter dem Ladebalken ein ***Schließen*** Knopf, mit dem der Import abgeschlossen werden kann.
Andernfalls öffnet sich ein Fenster, welches auf der linken Seite alle Warnungen anzeigt, die während dem Import aufgefallen sind. Rechts erscheinen alle überflüssigen Elemente, und wie mit ihnen umgegangen wurde.
Sollte eine dieser beiden Listen leer sein, wird sie nicht angezeigt.

Diese Warnungen können zum Beispiel sein, dass Indizes synchroniert werden sollen oder, dass Definitionsbereiche nicht gelöscht werden konnten, weil sie im Zielgraph noch in Verwendung sind. Hauptsächlich werden es aber Warnungen sein, dass Elemente nicht gefunden wurden. Das kann meistens behoben werden, indem das fehlende Element im Quellgraph gefunden und der passenden Komponente zugewiesen wird.

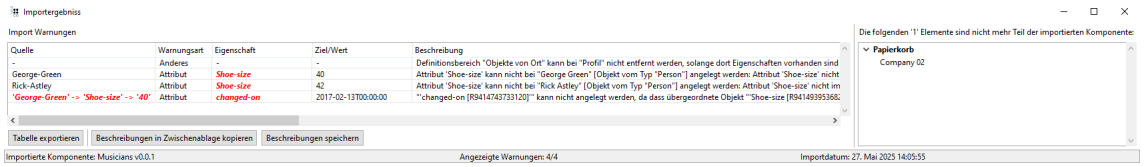


Figure 24. Beispielhafte Importwarnungen nach dem Import.

In dieser Tabelle werden alle Elemente mit ihrem RDF-Namen angezeigt, da sie eventuel nicht im Zielgraphen existieren und somit einfach ihre Import-Identifikatoren verwenden. Die hervorgehobenen Elemente wurden im Zielgraph nicht gefunden.

Um diese Probleme leichter zu beheben, kann die Tabelle mit dem Knopf unten links exportiert werden. Anschließend kann sie vom Quellgraphen wieder eingelesen werden.

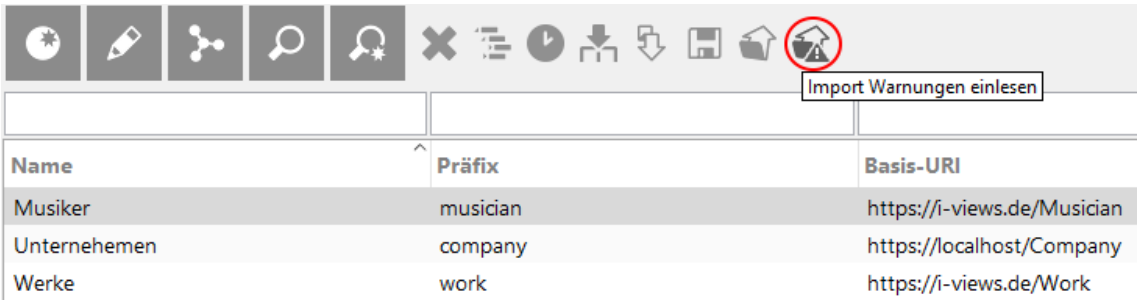


Figure 25. Der Knopf zum Öffnen von exportierten Importwarnungen.

Von dort aus können die Probleme direkt bohben werden, indem die Elemente per Rechtsklick bearbeitet werden. In dieser Tabelle werden Elemente auch mit ihrem Richtigen Namen angezeigt, solange sie nicht vor dem Öffnen der Tabelle gelöscht wurden. In diesem Fall wird `Element nicht gefunden` angezeigt. Über das Kontextmenü können Zeilen auch ausgeblendet werden, um die Tabelle übersichtlicher zu machen.

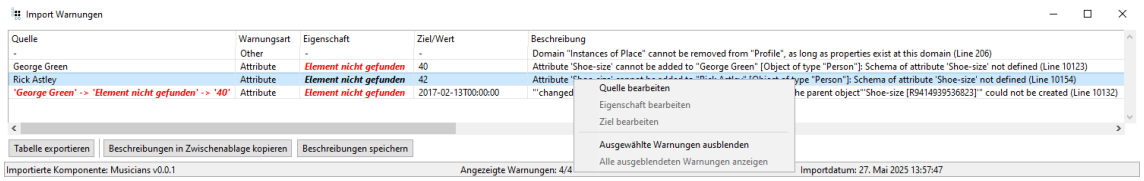


Figure 26. Beispielhafte, exportierte Importwarnungen, die im Quellgraph geöffnet wurden.

Falls es keine bearbeitbaren Element in der Tabelle gibt, können eventuell IDs von relevanten Elementen in der Beschreibung auf der rechten Seite gefunden werden. Diese können im Knowledge Builder durch Klicken auf das Menü oben rechts, unter ***Administrator > ID nachschlagen*** zu den gesuchten Elementen führen.

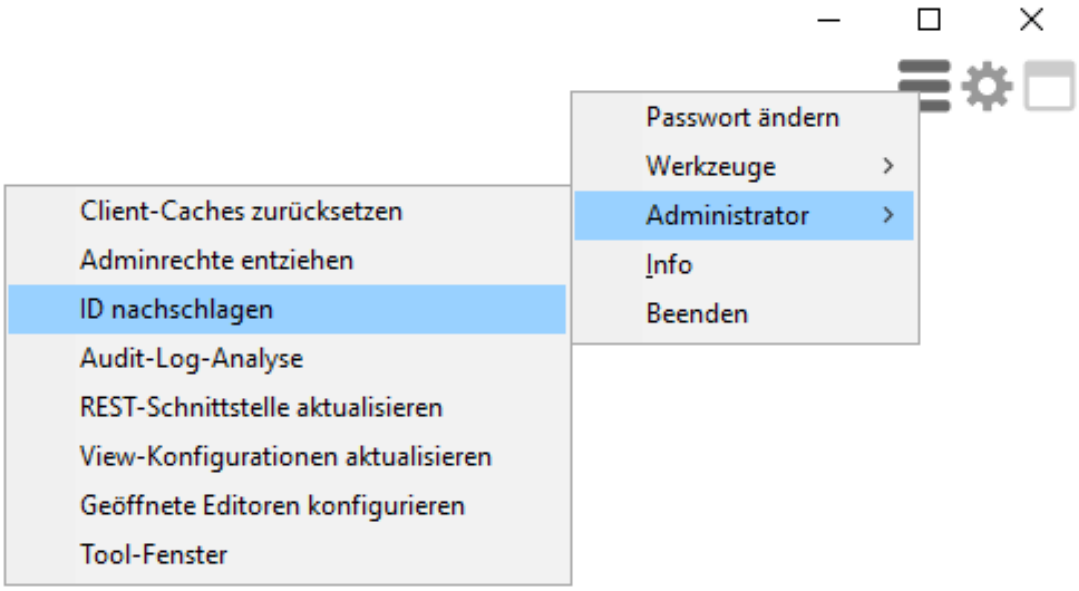


Figure 27. Das Menü zum finden von Elementen anhand ihrer ID.

1.8.2.13.3. Entfernen einer importierten Komponente

Die importierte Komponente kann auch vom Knowledge Builder ausgehend entfernt werden. Hierfür zu Komponenten unter ***Technik > Benutzerdefinierte Komponenten*** navigieren und den ✖ -Knopf auf der rechten Seite in der Detailansicht klicken.

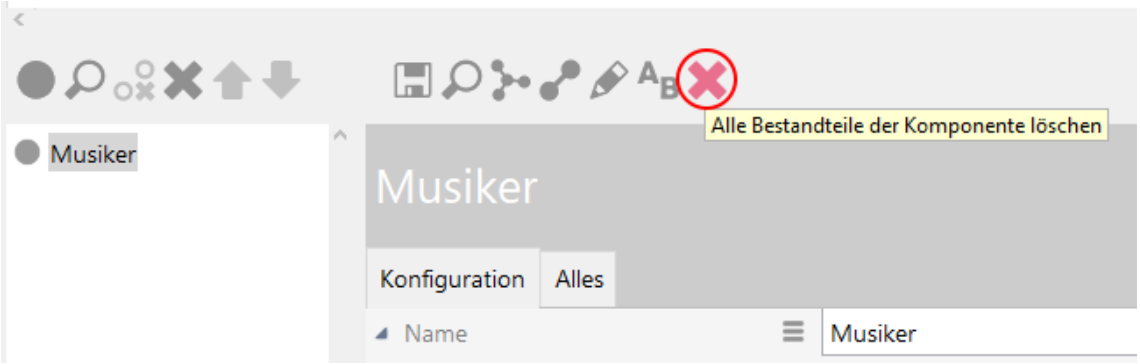


Figure 28. Der Knopf zum löschen einer Komponente.

Beim Entfernen einer Komponente können Sie sich aussuchen, ob Sie nur die Elemente und ihre Unterobjekte oder auch alle zugewiesenen Elemente entfernen (semantische Elemente, Abfragen, usw.).

HINWEIS	Alle Eigenschaftstypen, welche ausschließlich für die zu löschenden Elemente definiert wurden, werden ebenfalls gelöscht, da sie ohne definierte Domäne nicht existieren können.
WARNUNG	Wenn Sie versuchen eine Komponente so zu entfernen wie andere Elemente auch, z.B. über ihr Kontextmenü oder die Knöpfe über der Tabelle, wird nur das Komponentenobjekt selbst und seine Unterobjekte zu zusätlichen

> Auswahl entfernt, aber nicht ihre zugewiesenen Elemente.

1.8.2.14. Kommandozeilen Befehle

Das Batch-tool stellt folgende Befehle für benutzerdefinierte Komponenten zur Verfügung:

Befehl	Parameter	Wert	Optional
ImportCustomComponent	file	Dateiname der zu importierenden Komponente	Nein
	surplusHandling	`keep`, `bin` oder `delete` um die Handhabung überschüssiger Elemente der zu importierenden Komponente zu überschreiben	Ja
ExportCustomComponent	file	Dateiname für die exportierte Komponente	Nein
	uri	Die Basis-URI der zu exportierenden Komponente	Nein
	includeDependencies	Boolean. True, um alle anderen Komponenten, von der die Exportierte abhängt, ebenfalls zu exportieren.	Ja

1.8.3. Sprachen-Komponente

Die Sprachen-Komponente enthält Schema und Funktionalität, um Sprachen als Objekte des Knowledge-Graph konfigurieren und nutzen zu können.

1.8.3.1. Schema

Folgende Objekttypen stehen zur Modellierung zur Verfügung:

- **Sprache** Repräsentiert eine Sprache, die mittels ISO-639-1/2-Code (z.B. de, eng, …) oder Locale-Code (z.B. de-AT) identifiziert werden kann. Eine Sprache hat einen oder mehrere Sprachidentifikatoren (mindestens nach BCP 47) und eine Bezeichnung, die übersetzt und frei wählbar ist.
- **Kontextabhängige Sprache** Repräsentiert keine konkrete Sprache und ist vom jeweiligen Kontext abhängig. Es gibt genau zwei kontextabhängige Sprachen:
  - präferierte Sprache : steht für die zur Zeit für den Benutzer gewählte Sprache
  - beliebige Sprache : steht als Platzhalter für irgendeine aktuell zur Verfügung stehende Sprache
- **Sprachgruppe** Repräsentiert gleichzeitigen mehrsprachigen Zugriff auf alle in der Gruppe enthaltenen Sprachen. Die in der Gruppe enthaltenen Sprachen haben eine Reihenfolge, über die eine Sortierung in der Anzeige sowie eine Präferenz-Reihenfolge abgebildet werden.

1.8.3.2. Konfiguration von Sprachen und Sprachpräferenzen

Eine der Hauptaufgaben der Sprachen-Komponente ist die Regelung des Zugriffs (Anzeige und Eingabe) auf mehrsprachig definierte Attribute (siehe Kapitel "Mehrsprachigkeit").

Die weitreichendste Konfiguration zur Auswahl angezeigter Sprachen geschieht über das Objekt, welches die Anwendung (TECHNIK > View-Konfiguration > Anwendung) repräsentiert (z.B. "Knowledge-Builder" oder "View-Configuration-Mapper"). Über die Relation "Einschränken auf Sprache(n)" wird konfiguriert, welche Sprachen in der gesamten Anwendung überhaupt zur Anzeige kommen. Diese Einstellung gilt für alle mehrsprachig definierten Attribute, die innerhalb der Anwendung zur Anzeige kommen.

Zusätzlich kann an jeder Eigenschaft-Konfiguration für eine mehrsprachig konfigurierte Eigenschaft ebenfalls über die Relation "Einschränken auf Sprache(n)" eingestellt werden, mit welchen Sprachen die Eigenschaft im Kontext der Eigenschaft-Konfiguration dargestellt werden soll.

Ohne einschränkende Konfiguration werden alle für die Eigenschaft definierten Sprachen angezeigt.

1.9. Externe Indexierung

Im Gegensatz zur internen Indexierung werden bei der externen Indexierung Daten aus dem Knowledge-Graph an ein Fremdsystem übertragen, um dessen Features bei der Suche verwenden zu können. Zur Übertragung der Daten kommen die Werkzeuge zur Abbildung von Datenquellen zum Einsatz. Die Aktualisierung der externen Daten wird durch Trigger realisiert, und für die Suchfunktionalität stehen für das Fremdsystem spezialisierte Implementierungen bereit.

1.9.1. Anwendungsgebiete

- Realisierung von Funktionen (Aggregation, Linguistik, Pfad-Algorithmen, etc.), die von i-views nicht angeboten werden
- Beschleunigung der Suche, Ergebnis-Darstellung und Facettierung (speziell bei großen Datenmengen)
- Entkopplung in der Architektur der Anwendung (z.B. UI direkt auf externem Index)
- "Überhang"-Daten — d.h. im externen Index gibt es mehr Objekte als dem Knowledge-Graph bekannt sind
- Kopplung/Datenaustausch mit anderen Systemen

2. Admin-Tool

Das Admin Tool kann verwendet werden, neue Knowledge Graphen zu erzeugen, alle Knowledge Graphen eines Mediators zu verwalten und individuelle Knowledge Graphen zu konfigurieren.

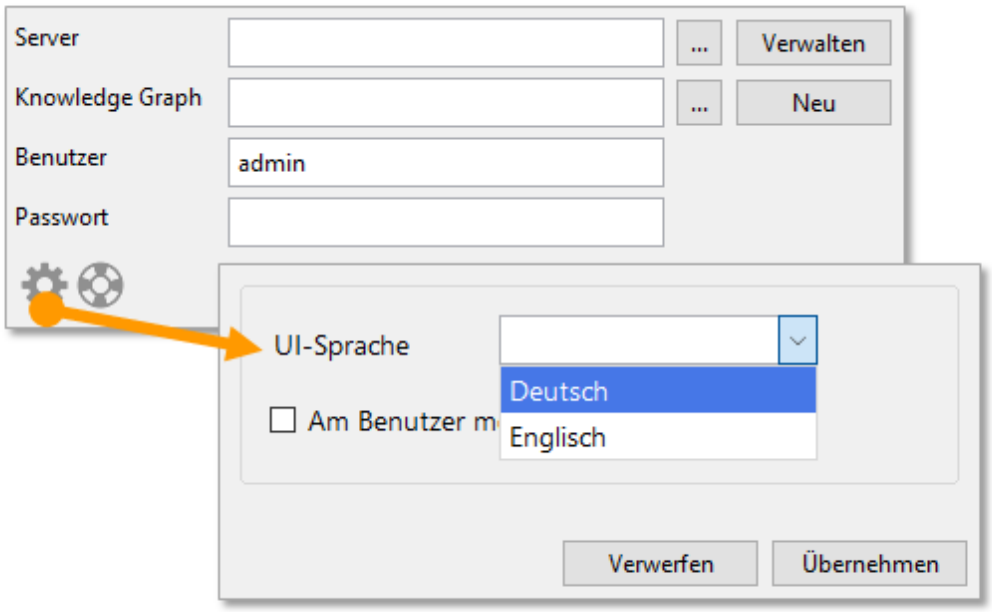
Das Admin Tools wird standardmäßig aufgerufen mit:

- Windows: `admin.exe`
- Mac OS: `admin`
- Linux: `visual admin.im`

Das Admin Tool funktioniert auch problemlos mit geänderten Dateinamen.

2.1. Admin-Tool Konfiguration

Das Admin-Tool kann wie der Knowledge-Builder in deutscher oder in englischer Sprache gestartet werden. Die voreingestellte Sprache ist Deutsch. Wenn das Admin-Tool in englischer Sprache gestartet werden soll, ist die Einstellung per Auswahldialog, ini-Datei oder Kommandozeilenparameter vorzunehmen. Die Sprachauswahl befindet sich links unten im Startfenster:



HINWEIS

Wenn mit dem Admin-Tool ein neuer Knowledge-Graph erzeugt wird, dann werden die Systemattribute und die Systemrelationen des Knowledge-Graphen

> in derselben Sprache angelegt, mit der das Admin-Tool gestartet wurde.

Abgesehen vom Einstellen der Sprache des Admin-Tools per Auswahldialog ist das Einstellen der (initialen) Default-Sprache nach wie vor möglich durch Verwendung einer ini-Datei oder eines Kommandozeilenparameters.

Der Inhalt der ini-Datei "admin.ini" für das englischsprachige Admin-Tool ist wie folgt:

```
[Default]
language=eng
```

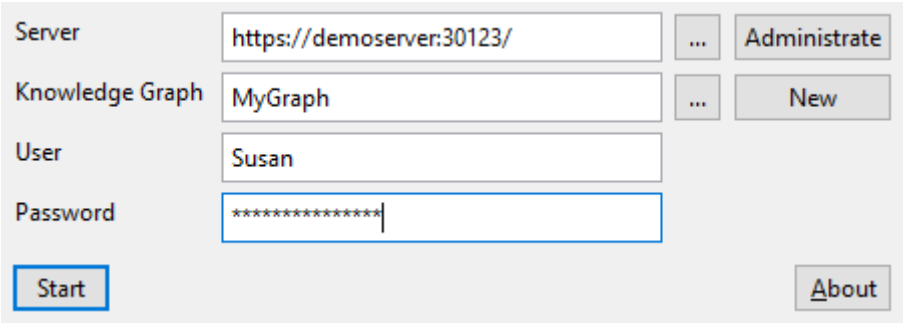
Der Kommandozeilenparameter zum Ändern der Sprache ist:

```
... -language eng
```

Zu beachten ist, dass ohne weiterführende Konfiguration die ini-Datei sich im selben Dateiverzeichnispfad befinden muss wie das Admin-Tool selbst.

2.2. Startfenster

Nach dem Start des Admin Tools erscheint das **Start Fenster**.



2.2.1. Server

Die URL des Servers wird im Feld **Server** eingegeben. Gültige URLs beginnen mit einem der Protokolle `cnp://`,`cnps://`,`http://` oder `https://` gefolgt von `[<Hostname oder IP-Adresse>[:<Portnummer>]]`. Wird kein Protokoll angegeben, wird `cnp://` verwendet. Wird kein Port angegeben, wird der Standard Port des Protokolls verwendet. Das Format entspricht der `interface` Konfiguration am Mediator.

Falls der verwendete Mediator auf dem selben Rechner läuft, wie das Admin Tool, kann auch der Hostname `localhost` verwendet werden.

Falls das Feld leer gelassen wird, werden die Knowledge Graphen zugegriffen, die im Unterordner `volumes` relativ zum Admin Tool liegen. Für diese Art von Zugriff wird kein Mediator benötigt.

Eingaben in dieses Feld werden gespeichert. Der **...** Knopf erlaubt es diese Werte in einem getrennten Fenster auszuwählen.

Mit dem **Verwalten** Knopf erreicht man die **Server Verwaltung**. Dieser benötigt das Server Passwort.

2.2.2. Knowledge-Graph

Im Feld **Knowledge Graph** wird der zu verwaltende Knowledge Graph eingetragen.

Eingaben in diesem Freitextfeld werden gespeichert. Mit dem **...** Knopf kann man die Werte in einem getrennten Fenster auswählen. Um alle Knowledge Graphen anzuzeigen, wird das Server Passwort abgefragt.

2.2.3. Verwalten, Neu und Start

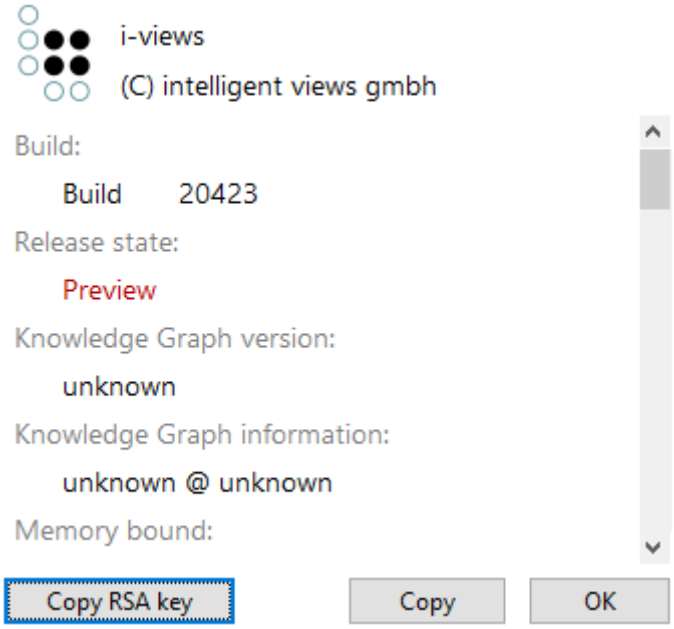
Administrate is used to access the **server administration**, for which authentication using the server password is required.

New forwards to Knowledge Graph generation.

Start forwards to the individual graph administration. The entries **user name** and **password** are used for this for logging in with an administrator account.

2.2.4. Info

You can use the **About** button to retrieve version-specific information in a separate window via the Admin tool.



Specifically, you can retrieve:

- The version number (*Build*),
- the publication status (*Release state*),
- the maximum system memory in bytes that can be used (*Memory bound*),
- the amount of memory at which garbage collection starts (*GC threshold*),
- the version number and the digital finger print of the execution environment (*VM version*),
- the configured HTTP proxy configuration (*HTTP Proxy*),
- External libraries found in the installation (*External Libraries*),
- the language setting active in the operating environment (*Locale*),
- the fonts used (*Fonts*),
- the software components including version numbers present in the software (*Software components*) and
- the core packages including version number used in the Admin tool (*Packages*).

Information on the *License*, *Knowledge Graph version* and *Knowledge Graph information* is not decisive here.

The information is shown in an invisible text field, which has a context menu that can be activated by right-clicking:

- **Copy:** copies the selected text area to the clipboard of the operating system.
- **Find:** allows a string to be input in a separate window, and its next occurrence in accordance with the read direction in relation to the position of the cursor set by clicking the mouse. The query is case-sensitive.
- **Find Again:** searches for the selected text area and finds its next occurrence in according to the read direction.
- **Select All** : selects all the text. Alternatively, the mouse pointer can be used to mark any text segment.

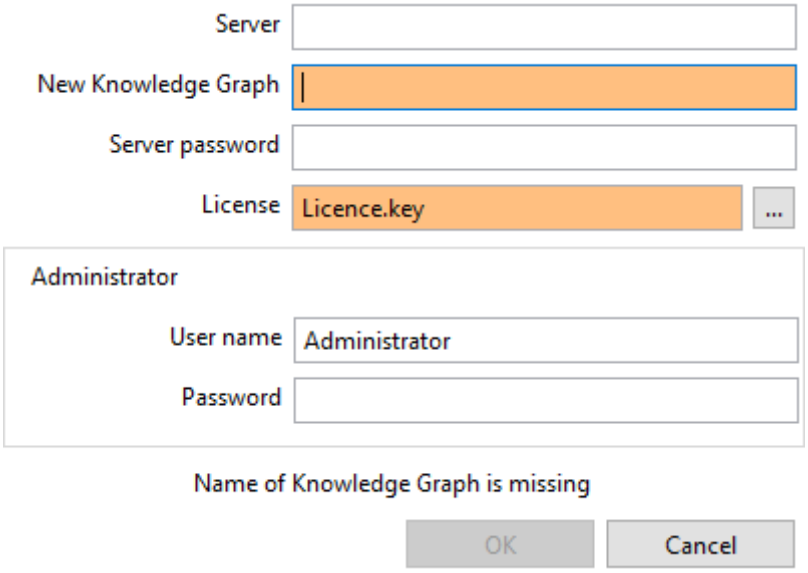
The **Copy** button copies all information to the clipboard of the operating system.

The **Copy RSA key** button copies the unique key for this build of the Admin tool to the clipboard of the operating system. This key can be used in the configuration of a mediator to restrict the tool build versions allowed to access the mediator.

The **OK** button enables you to return to the start window.

2.3. Create a new Knowledge Graph

A new Knowledge Graph is created via a separate **Knowledge Graph creation window**. It can be reached via the **New** button on the **start screen**. Any inputs in the **Knowledge Graph** free text field of the start screen is ignored.



2.3.1. Server

The name or the IP address of the computer on which the mediator is running is specified in the free text field **Server**. The value is copied from the start screen. If you use the server field in this dialog the same conditions as in the start screen apply.

If the field remains blank, the Knowledge Graph is generated in the `volumes` subfolder relative to the location of the Admin tool.

2.3.2. New Knowledge Graph

The name of the Knowledge Graph is specified in the free text field **New Knowledge Graph**. The characters allowed for this purpose are specified by the file system of the operating system on which the Knowledge Graph is to be stored. To ensure that the data can also be stored in different file systems, consider the following best practices:

- 64 characters maximum
- No whitespace at the start or end
- Characters used: Upper and lower case ASCII letter characters, numbers, spaces and `-_+.`. Permitted but not advised are also all Latin letters, `!@#$%&'()+-.[]^\_\`{}~Œœ` and ASCII characters 160-255
- The following character codes are not allowed: AUX, CON, NUL, PRN as well as COM0-COM9

and LPT0-LPT9

A name must be specified.

The name can subsequently be changed only during copy processes of the Knowledge Graph or by changing file and directory names. If you make a change, keep in mind that the name of the Knowledge Graph might be referenced in initialization files and that the license might have been adapted to the name.

2.3.3. Server password

The mediator server supports authentication via a password. If a password has been set for the mediator it will be used to create the new Knowledge Graph. The server password must be entered in the **Server password** free text field. If no password has been assigned, the field must remain empty.

2.3.4. License

A Knowledge Graph must have a valid license so that Knowledge Builder and other software components (with the exception of the Admin tool) can access it. Use the **…** button to access the file system of the operating system in order to load a license key (file name: `<License name>.key`).

2.3.5. User name

The name of the first user registered in the Knowledge Graph is specified in the **User name** free text field. The type and quantity of permitted characters is not restricted. The Administrator default setting is simply a suggestion. This field must not remain empty.

The name can be changed later on in the Admin tool or the Knowledge Builder. The user created in this way automatically is granted administrative rights.

2.3.6. Password (user)

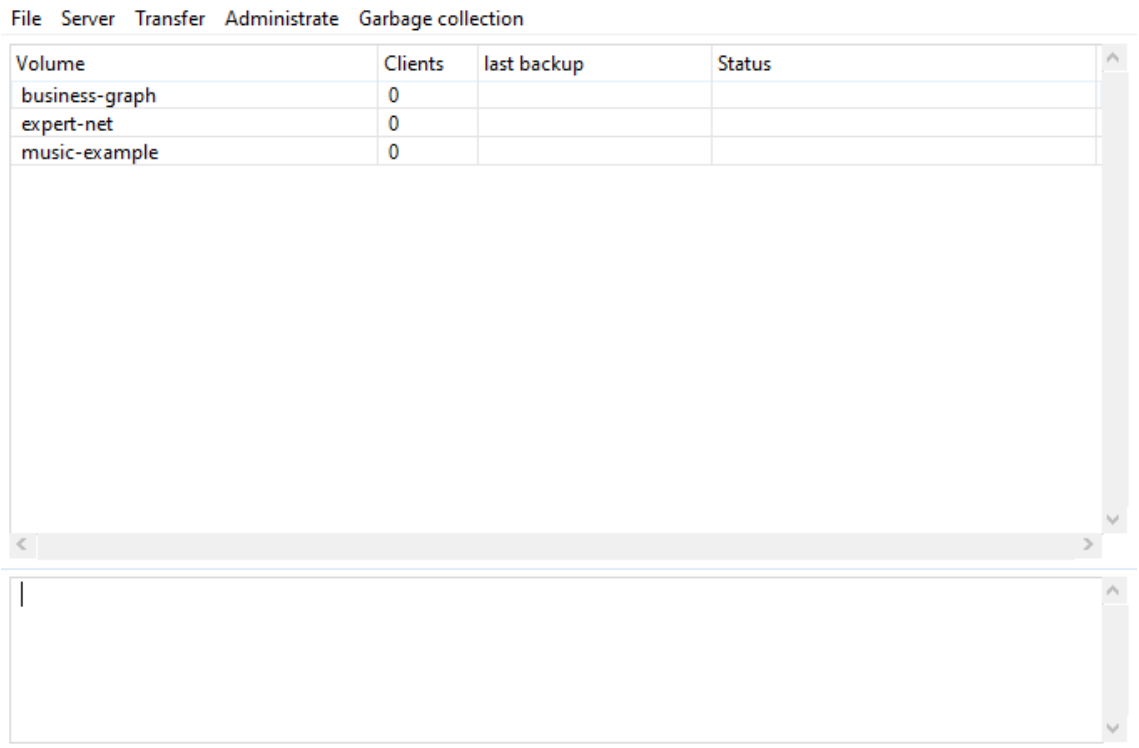
In the **Password** free text field, you should enter a password for the first user registered in the Knowledge Graph. This password will be required later on when this user attempts to log in to the Knowledge Builder or the Admin tool for the new Knowledge Graph.

2.3.7. Ok and Cancel

The **OK** button creates the Knowledge Graph, factoring in the data entered. The **Cancel** button cancels the process. In both cases, the system returns to the **start screen**.

2.4. Server administration

The overall Knowledge Graph administration allows the administration of all Knowledge Graphs of a mediator, or the local subfolder *volumes* respectively. It can be reached via the **Administrate** button on the **start screen**. A corresponding entry in the **Server** field is necessary for this. Any entries in the **Knowledge Graph** field of the **start screen** are ignored. If the Knowledge Graphs to be administrated are addressed using a mediator, the correct mediator password must also be specified in a separate window.



The **overall graph administration window** is comprised of a **graph overview** in the form of a table, a **message area** and a **menu line** at the top.

2.4.1. Graph overview

The **graph overview** in the form of a table provides details about

- the name (*Volume*)
- the number of clients currently connected (*Clients*),
- the date and time of the last backup (*last backup*) and
- the last status message (*Status*) of the respective Knowledge Graph.

The individual columns can be sorted by clicking on the head of the column.

The data is only updated when triggering operations, and therefore is not always up-to-date. A manual update can be forced at any time using the menu item **Server** > **Refresh Knowledge Graph**

list.

2.4.2. Message area

The **Message area** outputs all status reports for all Knowledge Graphs. Status reports are created when activities are triggered in the Admin tool. They are not saved when the Admin tool is closed. To prevent this, they can be exported via the menu option **File** > **Write administration log**. The **Message area** can be edited, but changes are ignored during export.

2.4.3. Menu line

The **menu line** consists of the following menu tabs:

File

Save administration log

saves all entries in the message window in a text file (default file name: *admin.log*). You can freely choose the name and storage location in a saving dialog. This operation requires the Admin tool to be connected to a mediator.

Reset session password

Log off

Closes server administration and opens the log-in window again.

Exit

Closes server administration

Server

Refresh Knowledge Graph list

Reloads the data collected in the **graph overview** in the **overall graph administration window**.

Re-import ini file

Triggers the server to reload its ini file. Note, that not all options can be updated during operation. The server outputs a message about updated options.

Download log

Downloads a copy of the mediator log file if present on the server (default file name: *mediator.log*) to the local machine. You can freely choose the name and storage location of the file in a saving dialog. The mediator log file keeps a log of all the mediators activities from its first commissioning.

Server connections

Shows the client id and individual IP address of all software clients currently connected to the mediator in the **message field**. The output contains a total count and is grouped by Knowledge Graph. The client id is generated sequentially by the mediator and assigned

(reusing free ids) whenever a new software component registers.

Transfer

Download Knowledge Graph

Creates a copy of the Knowledge Graph selected in the **graph overview** and saves it locally in the *volumes* subfolder that is located relative to the location of the Admin tool. A new name can be assigned to this copy in a separately appearing dialog.

Copy Knowledge Graph

Creates a copy of the selected Knowledge Graph and saves it on the same server as the original Knowledge Graph. A new name must be assigned to this copy in a separate dialog.

Upload Knowledge Graph

Uploads a local Knowledge Graph to the server. In a separate dialog, a local graph can be selected from a list, which is filled from the *volumes* directory relative to the Admin tools directory location. A new name can be assigned to this copy and it must differ from all graph names already present on the server.

Replace Knowledge Graph

Exchanges the contents of a selected Knowledge Graph with the contents of a locally present graph. In the process, the uploaded copy is given the name of the Knowledge Graph it has replaced. The local Knowledge Graph, which must be stored in the *volumes* subfolder that is relative to the position of the Admin tool, is selected in a separate selection window.

As a result of the copy processes initiated by transfer operations, the block allocation of the clusters and blobs within the Knowledge Graph copies is redefined, and their space consumption is optimized in the process. The resulting compression effect is identical to the one achieved by the operation **Manage** > **Compress volume**.

With the exception of the **Copy volume** operation, all these operations require the Admin tool to be connected to a mediator.

Administrate

Open Admin tool

Opens the Admin tool on the selected Knowledge Graph. Since this administration interface used authentication on the server level, no further authentication is required. In a separate window a list of accounts with administrative access in the selected Knowledge Graph is presented. Here the account to use when opening the Admin tool in "per graph" mode can be selected.
This makes it possible to access the user management of the volume if the administrator password has been lost.

Create backup

Creates a backup of the Knowledge Graph selected in the **Knowledge Graph overview** and saves it in the *backup* folder on the server. Every backup is a full copy of the original Knowledge Graph.

When the backup is initiated, a separate window asks whether the user wants to wait until the copy process is complete. If applicable, further use of the Admin tool is blocked until this time. Otherwise the copy process starts in the background, and there is no message regarding the process or completion of the copy process.

Restore backup

Offers a list of available Knowledge Graphs with backups present on the server. After picking a knowledge graph a second dialog offers the list of timestamps for backups of that graph. When selecting a timestamp, a new name must be assigned for the backup to be restored to.

Delete backup

Deletes a selected backup. To select this backup, two separate selection windows must be navigated: in the first, the Knowledge Graph must be selected; in the second, the backup timestamp must be selected from a list sorted by creation date.
The block assignment of clusters and blobs within the original Knowledge Graph is not modified when a Knowledge Graph copy is created. The copy process initiated by the backup operations therefore creates no compression effect.

Delete Knowledge Graph

Deletes the Knowledge Graph selected in the **Knowledge Graph overview**.

Compress Knowledge Graph

Remove free blocks in the files comprising the selected Knowledge Graph. Free blocks originate from changed data freed by the garbage collection.
The copying processes for clusters and blobs first move all unused blocks to the file end and then release them in the file system of the operating system.

Update volume version

Updates the version of the internal file system of the Knowledge Graph selected in the **Knowledge Graph overview**. If the Knowledge Graph is addressed via a mediator, the version it contains is used; otherwise, the version included in the Admin tool is used. The update should be performed, when upgrading from previous i-views versions.

Garbage collection

Garbage collection is a procedure that deletes objects that are no longer referenced (according to a programming terminology reading) from the Knowledge Graph. It thereby minimizes reduces the storage usage of the Knowledge Graph. Use of the garbage collection requires that the Knowledge Graph that is to be cleaned up is accessed via a mediator.

Start

Initiates a new garbage collection run for the Knowledge Graph selected in the **Knowledge Graph overview** or continues a garbage collection paused for it. No confirmation is sent when the process is completed. You can determine its progress via the **Status** menu option.

Pause

Halts the execution of the active garbage collection for the Knowledge Graph selected in the **Knowledge Graph overview**.

Stop

Terminates the execution of the active garbage collection for the Knowledge Graph selected in the **Knowledge Graph overview**.

Status

Fetches the status of the current garbage collection process for the Knowledge Graph selected in the **Knowledge Graph overview** and displays it in the status column of the **Knowledge Graph overview** and in the **message field**. If garbage collection is active, feedback on its progress is provided in percent.

2.5. Individuelle Knowledge-Graph Administration

Individual Knowledge Graph administration allows you to manage an individual Knowledge Graph. It can be reached via the **Start** button on the start screen. This requires the corresponding entries in the fields **Server**, **Knowledge Graph**, and authentication field combination, by default **Authentication** set to `Name and password`, **User** and **Password** of the start screen.

2.5.1. Nutzerauthentifizierung

To access the **Knowledge Graph administration window** a user account with administrative rights needs to be authenticated.

If you no longer have access to the Knowledge Graph, you can access the Knowlede Graph through authentication on the server by logging on to the **Server administration**.

Valid authentication methods for logging in to a specific Knowledge Graph depend on the configuration of the mediator and the graph. These details need to be provided to the users of the Admin tool.

Possible authentication methods are:

Name and password

The fields **User** and **Password** are used to authenticate against accounts defined in the Knowledge Graph

JSON Web Token

(*experimental*) A valid JWT needs to be pasted into the **Password** field. The token issuer needs to include the allowed operations as claims in the token. The **User** field contents are ignored.

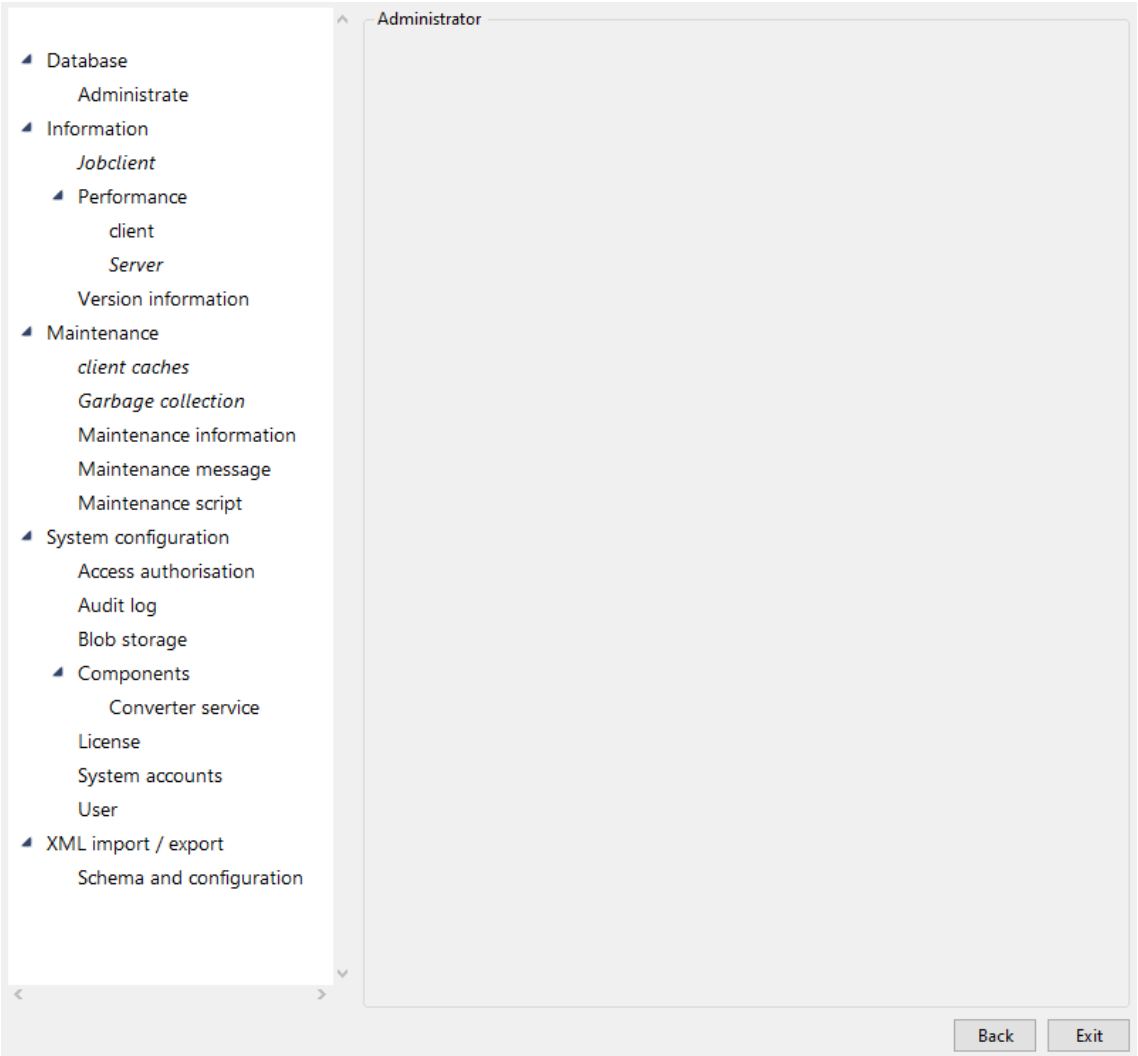
OAuth

To log in the user is redirected to the configured OAuth authentication provider service in the browser. After successful authentication and if the users account is authorized to log in to the installation, the Admin tool opens.

Windows negotiate

Use the Windows negotiate API to authenticate the user with the Knowledge Graph.

2.5.2. Fenster der individuellen Knowledge-Graph Administration



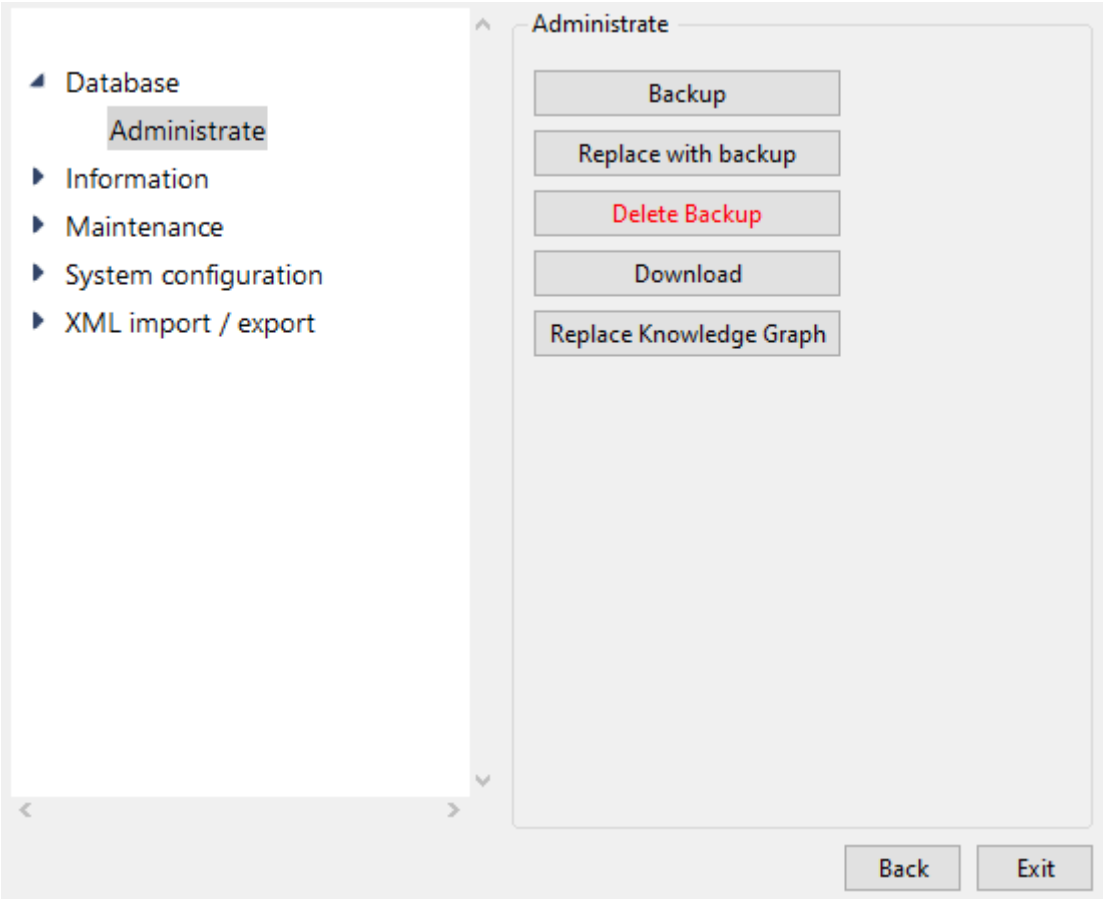
The **Knowledge Graph administration window** has a menu list with a multilevel structure on the left, and an operation window on the right. The content of the operation window depends on the menu option selected in the menu list.

The **Back** button returns you to the start window.

The **Exit** button closes the Admin tool.

If the Knowledge Graph to be administrated is addressed without a mediator, other users cannot access the Knowledge Graph via the Knowledge Builder or another instance of the Admin tool for as long as the **Knowledge Graph administration window** is open.

2.5.2.1. Datenbestand verwalten



Create backup creates a backup of the Knowledge Graph and saves it (on the server) in the *backup* folder, which lies in a parallel position relative to the position of this Knowledge Graph. There a separate subfolder is created for each backup; its name contains the time, precise to the second, at which this copy was created. Every backup is a full copy of the original Knowledge Graph.

Before the backup is created, a separate window asks whether the user wants to wait until the copy process is complete. If applicable, further use of the Admin tool is blocked until this time. Otherwise the copy process starts in the background, and there is no message regarding the process or completion of the copy process.

Restore backup replaces the current Knowledge Graph with a backup (afterwards you are logged off automatically). The backup is selected according to the time of the relevant backup.

Delete backup deletes an individual backup of this Knowledge Graph.

The block assignment of clusters and blobs within the original Knowledge Graph is not modified when a Knowledge Graph copy is created. The copy process initiated by the backup operations therefore creates no compression effect.

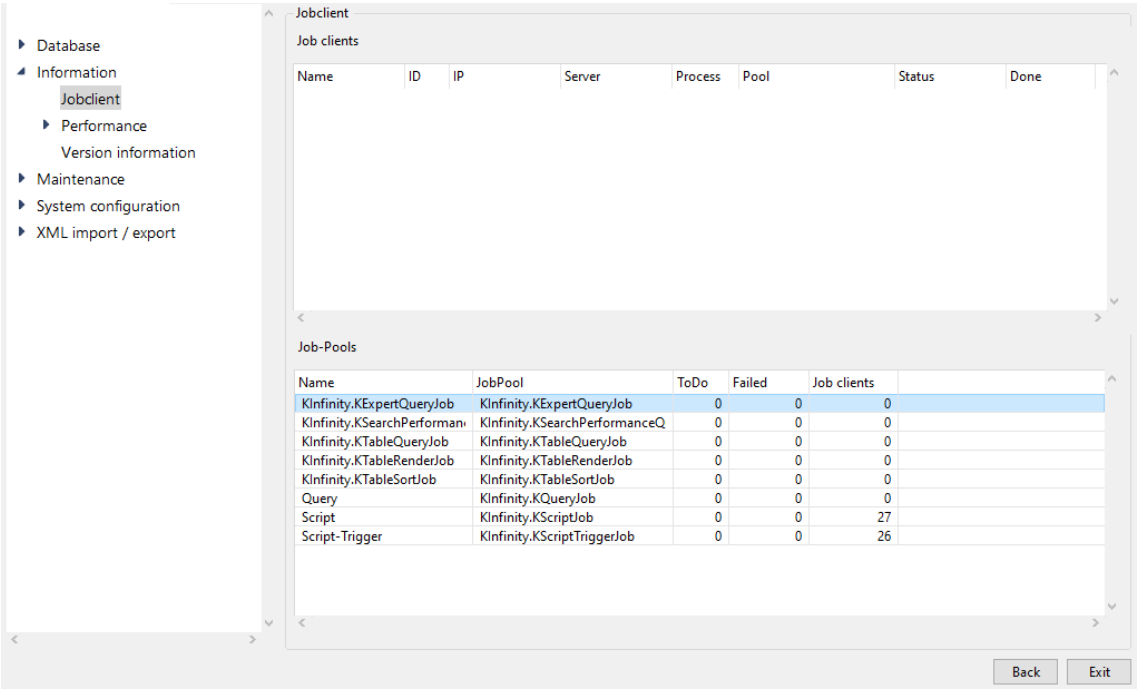
Download creates a copy of the Knowledge Graph and saves it locally in the *volumes* subfolder that is located relative to the position of the Admin tool. A new name can be assigned to this copy in a separately appearing free text field.

Upload volume transfers a locally stored Knowledge Graph and replaces the current Knowledge Graph with this Knowledge Graph (afterwards you are logged off automatically)

2.5.2.2. Information

2.5.2.2.1. Job-Client

In order to relieve the workload on the Knowledge Builder for specific, processor-intensive processes such as indexing, and querying Knowledge Graphs and executing scripts, some of these processes can be optionally performed by Job-Clients while others are exclusively performed as jobs by Job-Clients (a software service). To do so, the user interface of the Knowledge Builder or a script must be used to trigger a job, or the conditions for triggering it must be defined. Moreover, at least one Job-Client must be configured and started which can perform jobs of this job type (job pool). The Admin tool largely functions as an observer in this case. Jobs not completed appear in the Knowledge Builder under the entry *Tasks* in the *Technology* category. In order to use the Admin tool to manage Job-Clients, the Admin tool must be connected to a mediator.



The **Job-Clients overview** table shows the following for each job-client that is currently running:

- its name in the format [Job-Client-Name]@[Mediator-Name] (*name*),
- its job-client number (*ID*),
- its IP address (*IP*),
- the name of the mediator connected to it (*server*),
- the process number assigned by the operating system (*process*),
- the job types assigned to it (*pool*),

- its work status (*status*) and
- the number of jobs it has completed (*completed*).

The Job-Client number is generated sequentially by the mediator and a new number is assigned with each new log-in. The Job-Client name and the job types assigned to the Job-Client are defined in the initialization file for the respective Job-Client (default file name: *jobclient.ini*) under the key *name* or the key *jobPools* respectively. Each job type of a Job-Client is shown in a row of its own in the Job-Client overview, so that a Job-Client regularly takes up several rows.

The individual columns of the **Job-Clients overview** can be sorted by clicking on the head of the column. Right-clicking a row also opens a context menu:

- **Display information** displays all data listed in the selected row, with the exception of the job type and the completed number of jobs, in a new window. Added are
  - the date and time of the last time the Job-Client was started (*startUpTime*),
  - the maximum working memory capacity available for use by it in bytes (*max Memory*),
  - the name of its log file (*logFileName*) and
  - its specific name, under which it can be forced to shut down (a concatenation of the string "jobclient” and the Job-Client number) (*shutDownString*).
- The data there can be copied to the clipboard of the operating system (**Copy to clipboard** button) or be exported to any location as a text file that can be given any name using a saving dialog (**Save** button).
- The operation triggered using the menu item **Display information** can, alternatively, be performed by double-clicking a row in the Job-Clients overview.
- **Remove Job-Client** ends the Job-Client selected in the **Job-Clients overview**.
- **Remove all Job-Clients** ends all Job-Clients listed in the **Job-Clients overview**.

The **job pools overview** in the form of a table lists all job types that are assigned to at least one Job-Client in the **Job-Clients overview**. For each job type,

- its name (*name*),
- its technical name used in the Job-Client’s initialization file (*JobPool*),
- the number of uncompleted jobs of this job type (*ToDo*),
- the number of failed jobs of this job type (*failed*) and
- the number of Job-Clients available to it (*Job-Clients*)

are named.

The individual columns of the **job pools overview** can be sorted by clicking on the head of the column. Right-clicking a Job-Client also opens a context menu:

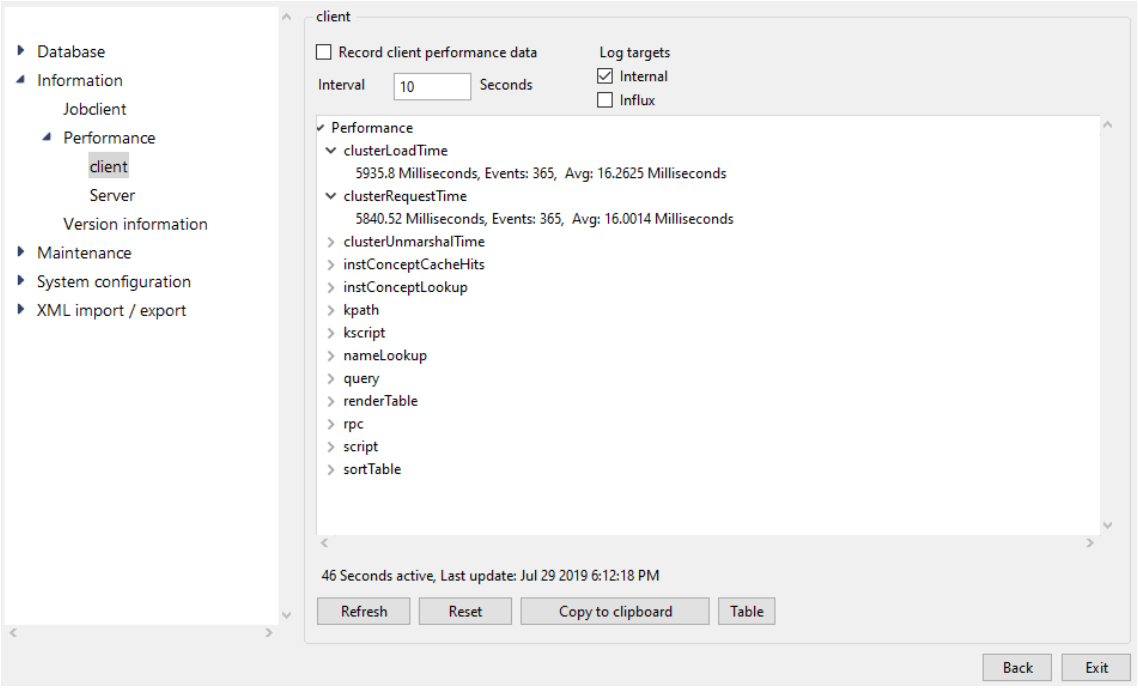
- **Empty job pool** deletes all uncompleted and failed jobs of the job type selected in the **job pools**

overview. This operation is only possible when no Job-Client is running.

- **Configure error messages to ignore** allows specific error messages to be blocked when executing jobs of the job type selected in the **job pools overview**. If an error message is blocked this way, the job related to the error is not factored in when determining the number of failed jobs in the **job pools overview**. This operation is only possible when there are already jobs of the job type selected in the **job pools overview** waiting to be processed, or that were already processed.
- The error messages to be blocked are administrated in a separate window:
  - All error messages to be blocked are listed in the alphabetically sorted **error message list**. An error message is blocked when its output text matches a text in the **error message list**.
  - **+** allows input of an error message to be blocked using a separate window. The error message appears in the **error message list**.
  - **...** allows the error message selected in the **error message list** to be changed.
  - **-** deletes the error message selected in the **error message list**.

2.5.2.2.2. Leistung

Client



Record client performance data starts and ends the collection of diverse key performance indicators that are coupled to activities by the software components connected to the Knowledge Graph. These key performance indicators can be used for the performance analysis.

Interval sets the required time period in seconds until a software component sends another data packet with key performance indicators to the Admin tool. It cannot be changed after recording starts. The preset is 10 seconds.

The key performance indicators are output in nested list items in the **key performance indicator overview**. Clicking on the triangle symbols to the right of the categories allows listed subitems to be expanded and collapsed. Alternatively, this can be implemented using a context menu, which can be accessed by right-clicking a list item:

- **Expand** opens all directly listed subitems in the list item selected.
- **Expand fully** opens all directly and indirectly listed subitems in the list item selected.
- **Contract fully** collapses all listed subitems in the list item selected.

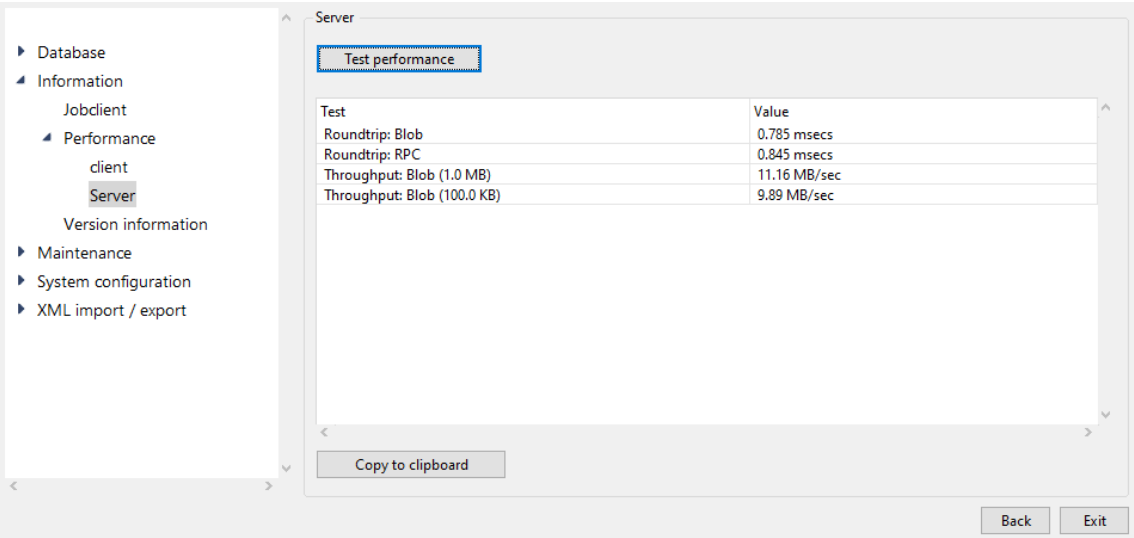
Double-clicking on a list item allows all key performance indicators stored below it to be shown at a glance in a separate window. There, they can be copied to the clipboard of the operating system (**Copy to clipboard** button) or be exported to any location as a text file that can be given any name using a saving dialog (**Save** button).

Update refreshes the key performance indicators shown in the **key performance indicator overview**.

Reset deletes the key performance indicators shown in the **key performance indicator overview**.

Copy to clipboard copies the key performance indicators shown in the **key performance indicator overview** to the clipboard of the operating system.

Server



Check performance starts a test process that evaluates the performance of the mediator connected. This sends four requests to the mediator, and the responses sent to the Admin tool are evaluated. Measurements are taken of

- the times until a small file is send back (*roundtrip: Blob*) and
- the result of an index search request (*roundtrip: RPC*) and
- the average transmission rate when sending several 1 MB files (*throughput: Blob (1.0 MB)*) and

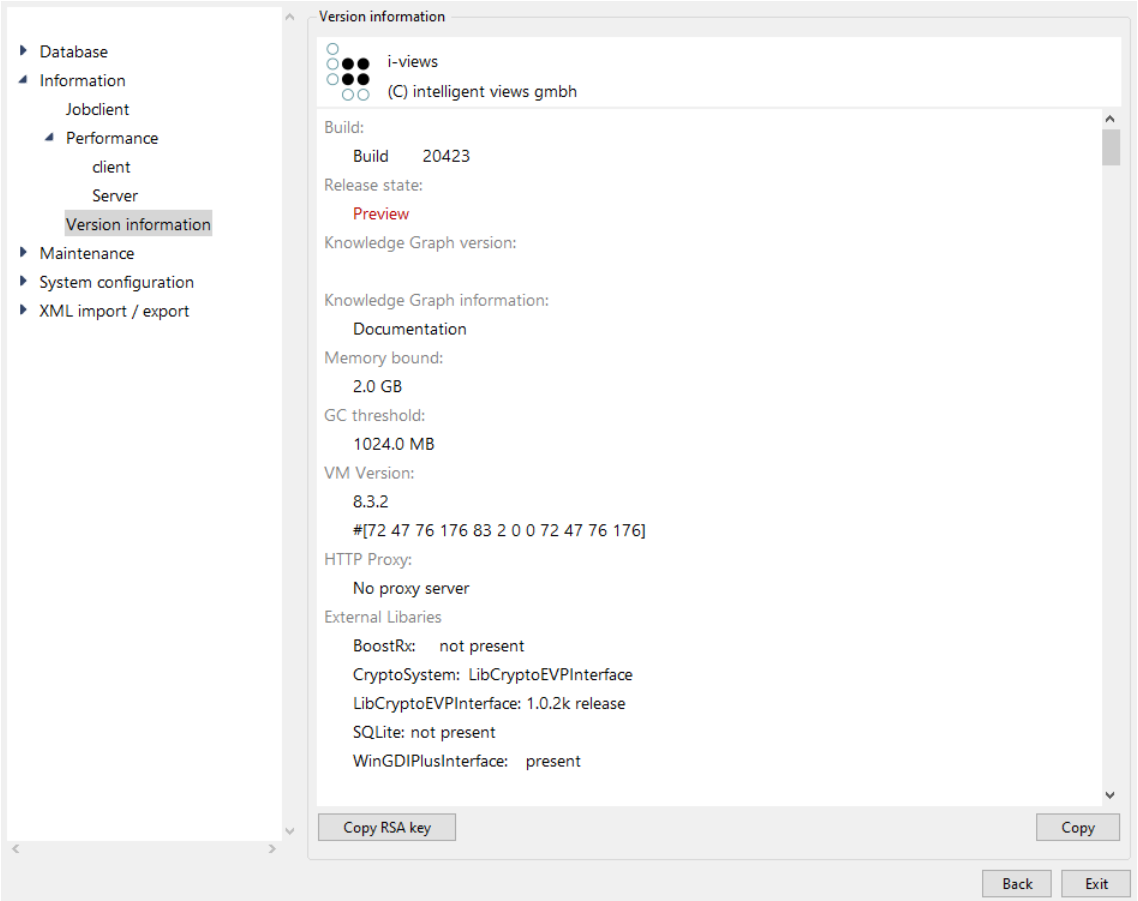
- the average transmission rate when sending several 100 KB files (*throughput: Blob (100.0 KB)*).

The test results are written to the **results list** provided. The individual columns of the table can be sorted by clicking on the head of the column.

Copy to clipboard copies the test results in the **results list** to the clipboard of the operating system as plain text.

2.5.2.2.3. Versionsinformation

This menu item can be used to retrieve version-specific information for the Knowledge Graph and Admin tool.



Specifically, you can retrieve:

- The version number of the Admin tool (*Build*),
- The publication status of the Admin tool (*Release*),
- The version number of the Knowledge Graph (the Knowledge Graph version), the name of the Knowledge Graph and the mediator used (*volume information*),
- The maximum system memory in bytes that can be used by the Admin tool (*Memory limit*),
- The version number and the digital finger print of the execution environment used by the

Admin tool (*VM version*),

- The language setting active in the operating system (*Locale*),
- The fonts provided in the Admin tool (*Fonts*),
- The Knowledge Graph components installed in the Knowledge Graph and their version numbers (*software components*) and
- The small talk packages including version number used in the Admin tool (*Packages*).

The information is output in an invisible text field, which has a context menu that can be activated by right-clicking:

- **Select All** selects all the text. Alternatively, the mouse pointer can be used to mark any text segment.
- **Copy** copies the selected text area to the clipboard of the operating system.
- **Find Again** searches for the selected text area and finds its next occurrence in according to the read direction.
- **Find** allows a string to be input in a separate window, and its next occurrence in accordance with the read direction in relation to the position of the cursor set by clicking the mouse. The query is case-sensitive.

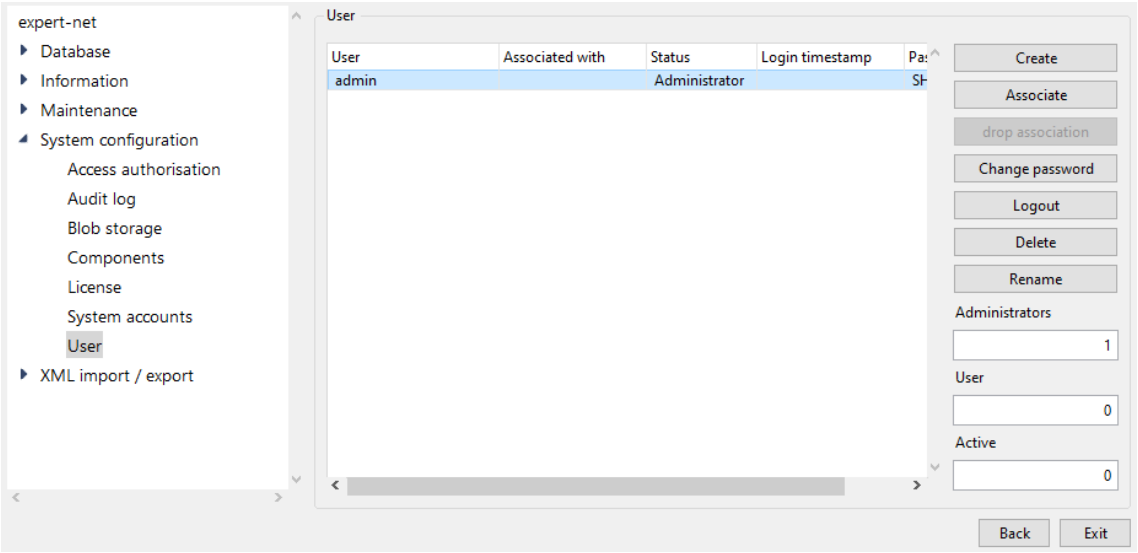
The **Copy** button copies all information to the clipboard of the operating system.

The **Copy RSA key** button copies the unique key for each compiled Admin tool to the clipboard of the operating system. This key can be entered into the initialization file of a mediator (default file name: *mediator.ini*) and thus restricts this mediator's access via an Admin tool to Admin tools with this specific key.

2.5.2.3. Systemkonfiguration

2.5.2.3.1. Benutzer

The user administration compares the ones in the Knowledge Builder, with the exception that no links between users and objects of the user-generated subgraph can be processed.



The **user overview** in the form of a table shows, for every user registered in the Knowledge Graph,

- the user name (*user*),
- the object of the user-generated subgraph the user is linked to (*linked to*),
- which status the user currently has (*status*),
- on which date and at which time the user logged into the Knowledge Graph using the Knowledge Builder (*log-in date*) if the user is still logged in, and
- which method was used to encrypt the password (*password type*).

The individual columns of the table can be sorted by clicking on the head of the column.

The *status* provides information about whether a user has administrator rights, whether a user with administrator rights does not have a password and whether a user is logged into the Knowledge Graph using the Knowledge Builder. Names of users with administrator rights without a password are marked in red.

Create creates a new user. User name (obligatory) and password (optional) are defined in a separate window. The type and quantity of permitted characters is not restricted.

Change password changes the password of the user selected in the **user overview**. The new password is entered two times in two windows that appear consecutively.

Log out logs out the user selected in the **user overview** from the Knowledge Graph following a security confirmation. To ensure this operation has its effect, this user must be currently logged into the Knowledge Graph using the Knowledge Builder.

Delete deletes the user selected in the **user overview** following a security confirmation. At least one user with administrator rights must remain.

Rename allows a new user name to be assigned for the user selected in the **user overview** by means of a free text field in a separate window. If the free text field remains blank, no renaming

occurs.

Notification uses a free text field in a separate window to send a message to the user selected in the **user overview**. The message is buffered in the Knowledge Graph and appears to the user addressed in a separate window in the Knowledge Builder as soon as the user uses it to log into the Knowledge Graph. The user cannot reply to this message.

Administrator assigns the administrator rights to the user selected in the **user overview**, or takes them away. A user must have a password to obtain administrator rights. Once the user has administrator rights, deleting the password is then possible. At least one user must have administrator rights.

Operations opens a new window in which the user selected in the **user overview** can, from a list of operations, these being

- Create backup,
- Delete backup,
- Restore backup,
- Garbage collection,
- Copy,
- Download log,
- Download volume,
- Upload volume,
- Delete volume,

select those operations that this user may execute within the scope of individual Knowledge Graph administration in future, without input of the mediator password. To confirm the selection, the correct mediator password must be entered in the free text field **Server password for operations**.

The **Operations** operation can only be selected by a user with administrator rights. Its use also requires that a mediator password has been set.

The field **Administrators** specifies the number of all users with administrator rights registered in the Knowledge Graph.

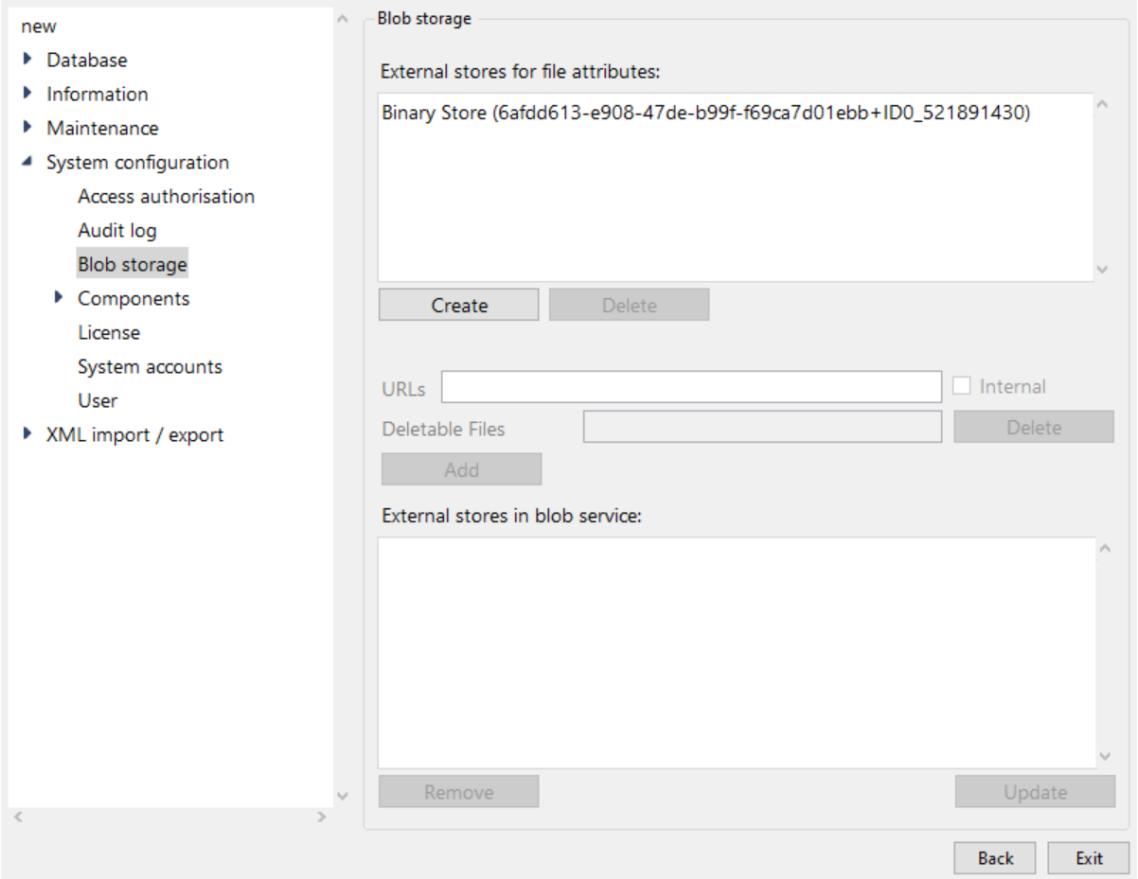
The field **Users** specifies the number of all users without administrator rights registered in the Knowledge Graph.

The field **Active** specifies the number of all users currently logged into the Knowledge Graph using the Knowledge Builder.

2.5.2.3.2. Blob-Speicherung

Attribute values of attributes with the attribute value type *file* (called blobs) can also be stored in a blob store outside the Knowledge Graph. The advantage of this is that they can be managed

independently of the Knowledge Graph and can thus be managed in a different system environment. To store blobs in a blob store, the blob store must be set up and connected to a configured blob service (a software service).



Create generates a new blob store. Using the name format *[Knowledge Graph ID]+[blob store ID]*, the **blob store overview** appears in the text field above it.

Delete deletes the blob store selected in the **blob store overview**.

The numeric field **Deletable files** shows the number of blobs no longer required in the **blob store overview** of the selected blob store. Blobs are no longer required when their respective attributes have been deleted from the Knowledge Graph or if the connection between blob service and blob store has been removed using the Admin tool.

Delete deletes all blobs that are no longer required in the blob store selected in the **blob store overview**.

You can identify a blob service in the free text field **URLs**. This is done by entering the network address of the initialization file of the corresponding blob service (default file name: *blobservice.ini*) stored under the *interfaces* key including the prefix *http*. If the blob service is supposed to be addressed via several network addresses, these can be entered in comma-separated form.

Alternatively, the blob service integrated in the mediator can also be addressed. In the initialization

file of the mediator (default file name: *mediator.ini*), the value *true* must be set under the key *startBlobService* and the free text field **URLs** must be left blank. The **internal** checkbox to the right of the free text field **URLs** indicates whether the integrated blob service or an external blob service is addressed. The blob service integrated into the mediator is not configured via the mediator initialization file but via a separate initialization file (default file name: *blobservice.ini*).

Add connects the blob store selected in the **blob store overview** to the blob service identified via the free text field **URLs**. To do so, the blob service must be active. If linking is successful, the blob store using the name format *[Knowledge Graph ID]+[blob store ID]* appears in the text field below, the **overview of registered blob stores**.

Update updates the **overview of registered blob stores**. To do this, a blob store must be selected in the **blob store overview**.

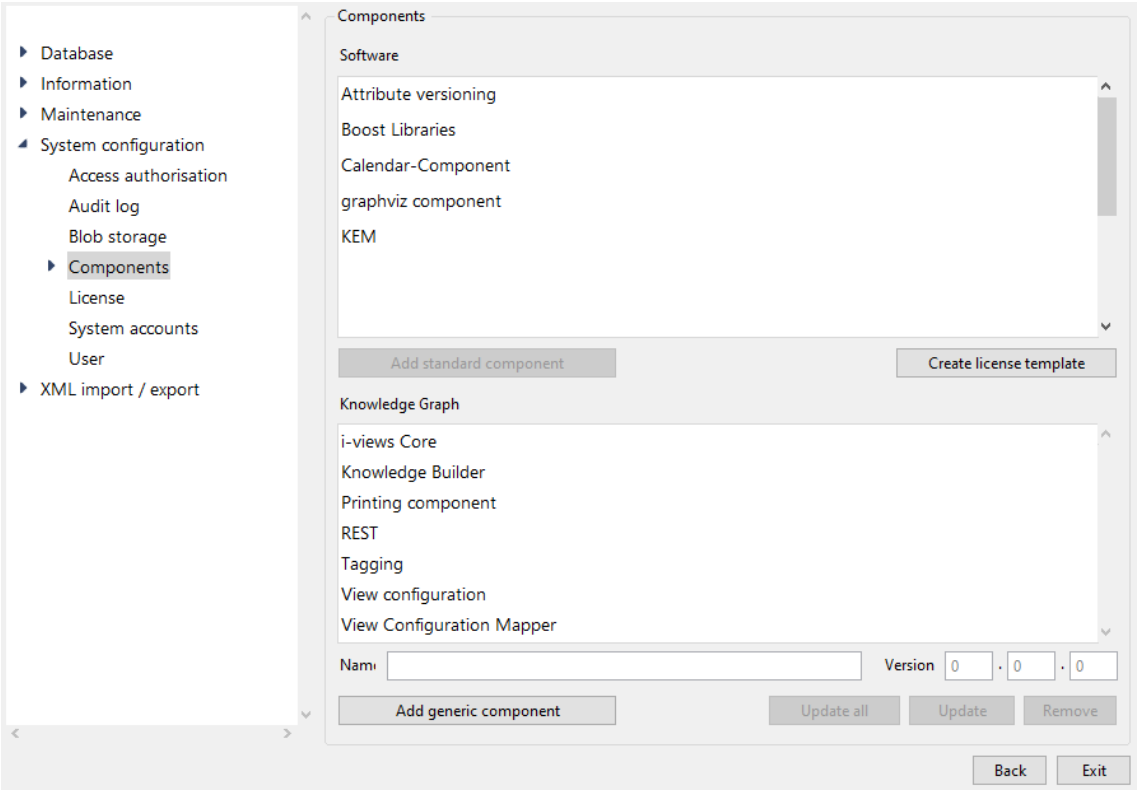
Remove interrupts the connection of the blob store selected in the **overview of registered blob stores** to the blob service and removes the blob store from the overview. In doing so, all blobs stored in the blob store irrevocably lose their internal references to the respective attributes in the Knowledge Graph and can no longer be retrieved in the Knowledge Graph. To ensure removal is successful, the blob store selected in the **overview of registered blob stores** must also be selected in the **complete blob store overview**.

All blobs stored via a blob service are stored in a subfolder called *blobs* that is located relative to the position of the blob service. The internal assignment of every blob to its blob store and its Knowledge Graph is established using an SQLite database.

2.5.2.3.3. Komponenten

Komponenten erweitern das Schema und ggf. auch die Funktionalität eines Knowledge-Graphen. Dabei gibt es zum einen die "Software-Komponenten", die mit jeder i-views-Software zusammen ausgeliefert werden und jederzeit installierbar sind sowie "Benutzerdefinierte Komponenenten", die über ein Datei-Archiv geladen werden.

Eine Sonderrolle nehmen die "Release State-Komponenten" ein, deren Funktion lediglich darin besteht, festzulegen, mit welchen Software-Release-Zuständen der Knowledge Graph betrieben werden darf.



Die **Software Liste** im oberen Bereich enthält eine alphabetische Auflistung verfügbarer Software-Komponenten sowie deren Versionen. Über die Schaltfläche **Softwarekomponente hinzufügen** kann eine in der Liste ausgewählte Komponente installiert werden, worauf sie in der unteren Liste der installierten Komponente erscheint.

Die **Knowledge-Graph Liste** im unteren Bereich listet die aktuell installierten Komponenten sowie deren Versionsnummern. Komponenten, die in rot dargestellt werden, benötigen ein Update und können aktuell nicht verwendet werden. Über entsprechende Schaltflächen lassen sich Komponenten **aktualisieren** (auf die aktuelle Version), **erneuern** (wenn die Version bereits aktuell aber möglicherweise unvollständig/beschädigt ist) oder auch **entfernen** (falls die jeweilige Komponente dies unterstützt).

Folgende Softwarekomponenten sind wählbar:

Komponente	Funktion
Attributversionierung	Bietet Strukturen und Funktionen, um Modellveränderungen aufzuzeichnen und zu visualisieren.
Aufgaben	Schema und Funktionalität um Aufgaben, die ein Jobclient ausführt, als Objekte im Knowledge-Graph zur Verfügung zu stellen. Auf diese Weise sind die Erstellung neuer Aufgaben und die Übersicht über die Abarbeitung von Aufgaben im konfigurierten Web-Frontend möglich.

Komponente	Funktion
Benutzerdefinierte Komponenten	Diese Komponente ermöglicht das Anlegen und Verwenden von Benutzerdefinierten Komponenten. Siehe auch: Benutzerdefinierte Komponenten
Boost libraries	Über diese Komponente kann man sicherstellen, dass die Boost-Bibliotheken für komplexe reguläre Ausdrücke zur Verfügung stehen.
Dokumentenerstellung	Ermöglicht die Erstellung von Dokumenten auf Basis von Vorlagen mit Inhalten des Knowledge-Graphen. Die Komponente bietet auch die generische Erstellung von Tabellen sowie eine Druckfunktion im Knowledge-Graph.
EXP Knowledge Graph	Bietet für die Integration des Knowledge-Graphen in die Empolis Plattform die notwendigen Konfigurationsmöglichkeiten für die Ausleitung von K-Pack Klassen, die Daten für die Simple-Concept-Hit Anzeige, die Related-Content-Plus Gruppen und das Verhalten der Ingest-Extension, um Metadaten von Dokumenten mit dem Graph zu verarbeiten.
Graphviz	Ermöglicht die Darstellung von Graphen mittels der Graph-Visualisierungssoftware "Graphviz".
i-views Core	Die Basiskomponente, die für den Betrieb eines Knowledge-Graphen zwingend notwendig ist. Enthält das Basis-Schema.
Kalender	Stellt einen REST-Schnittstelle zum Austausch von iCalendar-Dateien zur Verfügung.
LaTeX-Dokumentenerstellung	Ermöglicht die Erstellung von Dokumenten auf Basis von View-Konfiguration und LaTeX-Templates mit Inhalten des Knowledge-Graphen. Die Komponente beinhaltet eine Schnittstelle zu Overleaf, über die die erstellten LaTex-Dokumente in PDF gerendert werden können.
Maßeinheiten	Schema und Funktionalität zur Deklaration und Nutzung (insbesondere Umrechnung) von Maßeinheiten. Numerische Attributwerte können dann mit Maßeinheiten verwendet werden.
MQTT	Ermöglicht die Nutzung des leichtgewichtigen Publish/Subscribe-Messaging-Protokols MQTT (Message Queuing Telemetry Transport). Über einen neuen Bridge-Typen können so MQTT-Clients realisiert werden, die Zugriff auf den Knowledge-Graphen haben.
OAuth-Anmeldung	Ermöglicht die Anmeldung über OAuth.
Release State: Preview	Erlaubt den Betrieb des Knowledge-Graphen mit Preview-Software.

Komponente	Funktion
Release State: Release	Stellt sicher, dass der Knowledge-Graph nur mit Software-Status "Release" betrieben wird.
Release State: Release Candidate	Erlaubt den Betrieb des Knowledge-Graphen mit Release-Candidate-Software.
REST	Ermöglicht die Definition und Bereitstellung von REST-Schnittstellen.
Translator	Export und Import von übersetzten Attributen bzw. Schema in Tabellenform, um diese von einem Übersetzungsdienst bearbeiten lassen zu können.
View-Configuration-Mapper	Ermöglicht die Nutzung der View-Configuration-Mapper-Anwendung für Web-Browser basierte Benutzeroberflächen.
View-Configuration	Enthält Schema-Bausteine zur Definition von Modell-Ansichten. Diese können im Knowledge-Builder, View-Configuration-Mapper oder bei der Dokumenterstellung verwendet werden.
Validator	Framework zur Validierung von Attributwerten - insbesondere HTML.
ZMQ	Die Integration von ZeroMQ ermöglicht die Komposition von Laufzeit-Komponenten mittels PUBLISH/SUBSCRIBE, SEND/RECEIVE sowie PUSH/PULL-Pattern. Dazu stehen eine neue Bridge-Art, neue Script-Funktionen sowie eingebaute Broker-Unterstützung im Mediator zur Verfügung.

Boost libraries 1.18.0

Dieses Konfigurationsmenü wird nur angezeigt, wenn die Komponente „boost libraries“ installiert ist.

Mit Ausnahme des Blob-Dienstes und des Mediators können alle Softwarekomponenten JavaScript interpretieren. Um den Umfang und die Geschwindigkeit der Interpretation von in JavaScript eingebetteten regulären Ausdrücken zu verbessern, ist es möglich, deren Interpretation an die Boost.Regex-Bibliothek zu übertragen. Unter Windows und Linux muss sich die Bibliothek (Dateiname unter Windows: *boost_regex.dll*, Dateiname unter Linux: *libboost_regex.so*) im Ausführungsverzeichnis befinden. Unter Mac OS ist die Bibliothek bereits integriert.

Die Knowledge-Graph-Komponente *boost libraries* stellt sicher, dass der Zugriff auf die Bibliothek Boost.Regex möglich ist.

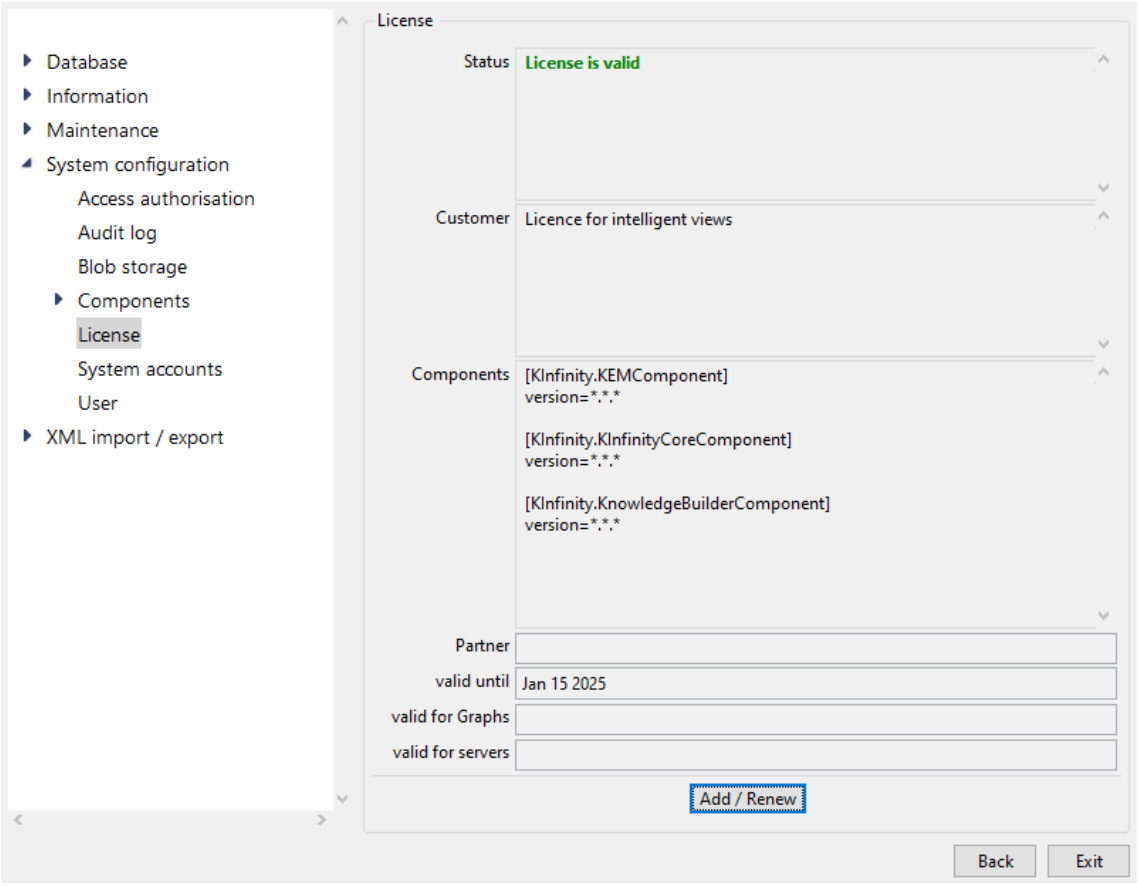
Wenn die Option **Boost-Bibliotheken für alle erforderlich, einschließlich Admins** ausgewählt ist, können alle Softwarekomponenten außer dem Admin-Tool nur dann auf den Knowledge Graph zugreifen, wenn sie Zugriff auf die Bibliothek Boost.Regex haben.

Wenn die Option **Boost-Bibliotheken für alle außer Admins erforderlich** ausgewählt ist, können alle Softwarekomponenten außer dem Admin-Tool nur dann auf den Knowledge Graph zugreifen, wenn sie Zugriff auf die Boost.Regex-Bibliothek haben. Die einzigen Ausnahmen von dieser Zugriffssperre sind Benutzer mit Administratorrechten, die über den Knowledge Builder auf den Knowledge Graph zugreifen.

Wenn die Option **Boost-Bibliotheken nicht erforderlich, nur Protokollierung** ausgewählt ist, gibt jede Softwarekomponente eine entsprechende Warnung in ihrer jeweilige Protokolldatei aus, sofern vorhanden, wenn sie beim Start nicht auf die Boost.Regex-Bibliothek zugreifen kann. Der Zugriff auf den Knowledge Graph bleibt dennoch möglich.

2.5.2.3.4. Lizenz

A Knowledge Graph must have a valid license so that Knowledge Builder and other software components (with the exception of the Admin tool) can work with it.



The **Status** field specifies whether the license is currently valid or invalid. If it is invalid, a reason is also stated. Reasons for an invalid license can be exceedance of the validity date or maximum number of allowed registered users.

The **Customer** field describes the client for whom the license was issued. In addition to the name, address and department may also be listed.

The **Components** field displays the content of the component license configuration file *[Knowledge Graph].componentLicenseTemplate.ini* used to generate the license key. This specifies

- The licensed versions of individual components (*version*),
- The maximum number of registered users with administrator rights (*maxAdminUsers*) and
- The maximum number of registered users without administrator rights (*maxUsers*)

The **Partner** field contains the name of the partner via which the license is forwarded.

The **Valid to** field contains the date on which the license expires.

The **Valid for Knowledge Graphs** field contains a list of names of all Knowledge Graphs to which the license is restricted. This can be entered using a regular expression.

The **Valid for servers** field contains a list of all IP addresses and port numbers that can be used to reach a mediator connected to the Knowledge Graph.

The fields **Partner**, **Valid to**, **Valid for Knowledge Graphs** and **Valid for servers** can be left blank.

All fields have a context menu that can be activated by right-clicking.

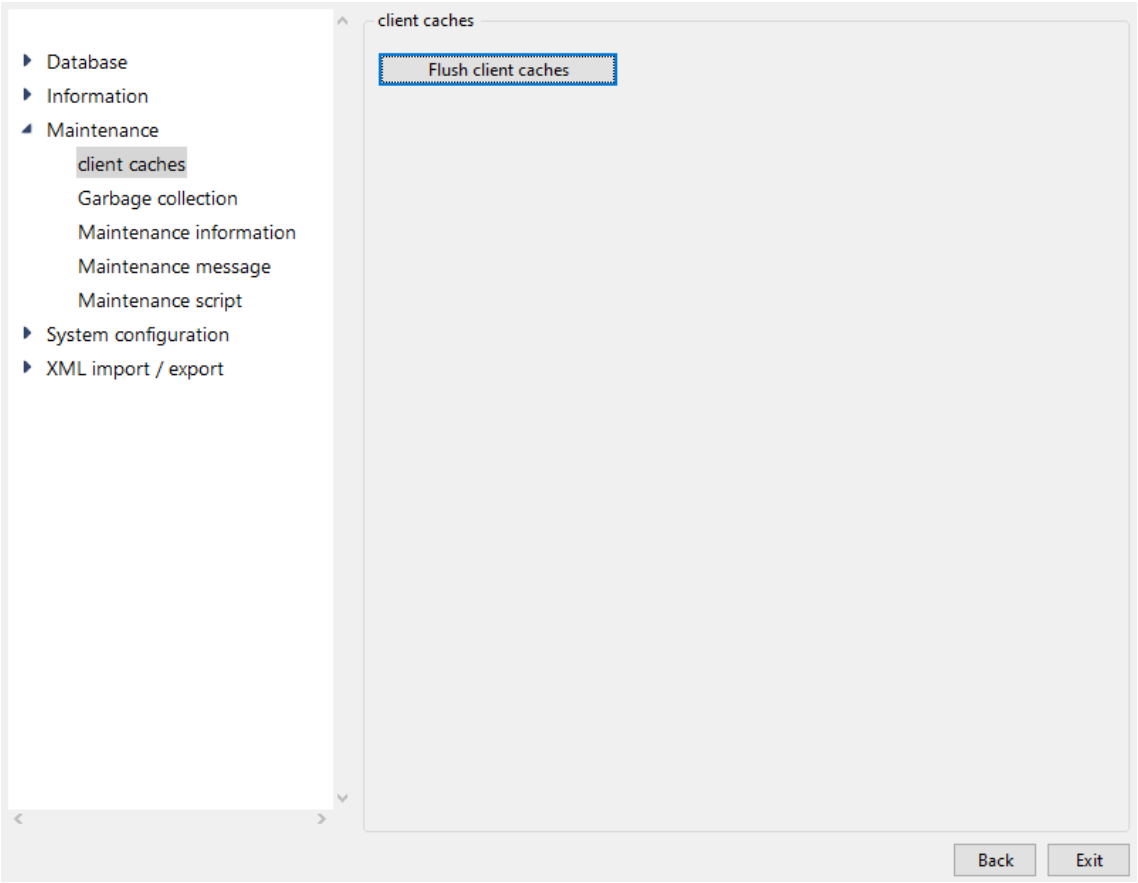
- **Select All** selects all the text. Alternatively, the mouse pointer can be used to mark any text segment.
- **Copy** copies the selected text area to the clipboard of the operating system.
- **Find Again** searches for the selected text area and finds its next occurrence in according to the read direction.
- **Find** allows a string to be input in a separate window, and its next occurrence in accordance with the read direction in relation to the position of the cursor set by clicking the mouse. The query is case-sensitive.

Add / Renew makes it possible to load a new license key (file name: *[License name].key*) via the file system of the operating system.

2.5.2.4. Wartung

2.5.2.4.1. Client-Caches

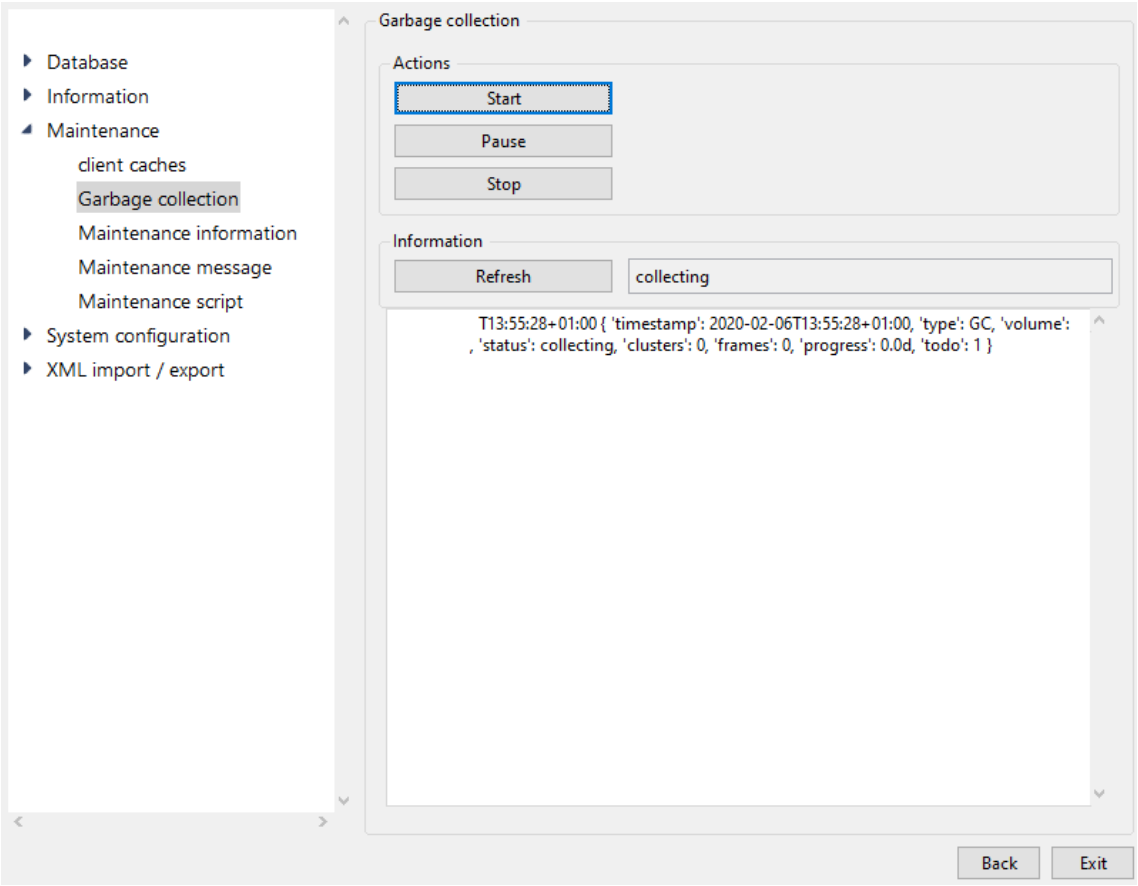
To improve performance, software components accessing the Knowledge Graph often fall back on their own buffers (cache). These buffer the schema and configuration data of the Knowledge Graph so they can access them more quickly if they need to use them later on.



Reset client caches deletes these buffered data. This makes sense if they are obsolete due to changes to the schema or the configuration. This operation requires that the Knowledge Graph is activated via a mediator.

2.5.2.4.2. Garbage Collection

Garbage collection is a procedure that deletes objects that are no longer referenced (according to a programming terminology reading) from the Knowledge Graph and thereby minimizes the memory usage of the Knowledge Graph. Use of the garbage collection requires that the Knowledge Graph that is to be cleaned up is activated via a mediator.



Start launches a new garbage collection for the Knowledge Graph or continues a paused garbage collection. No confirmation is sent when the process is completed. You can determine its progress via the **Refresh** menu option.

Pause interrupts the execution of the active garbage collection for the Knowledge Graph.

Stop cancels the execution of the active garbage collection for the Knowledge Graph.

Refresh writes the current state of the garbage collection for the Knowledge Graph to the neighboring text field. If garbage collection is active, feedback on its progress is provided in percent.

2.5.2.4.3. Wartung

Perform maintenance now checks

- the license (*license*)
- indexes (*indexes*),
- registered objects (*the registry*),
- rights (*access rights*),
- triggers (*trigger*) and
- installed Knowledge Graph components (*active components*)

for faults. Over the course of the check, the statistics for property frequencies per object (metrics) that can be viewed using the Knowledge Builder are updated.

Any faults found are collected in a **fault overview** in the form of a table. For each fault,

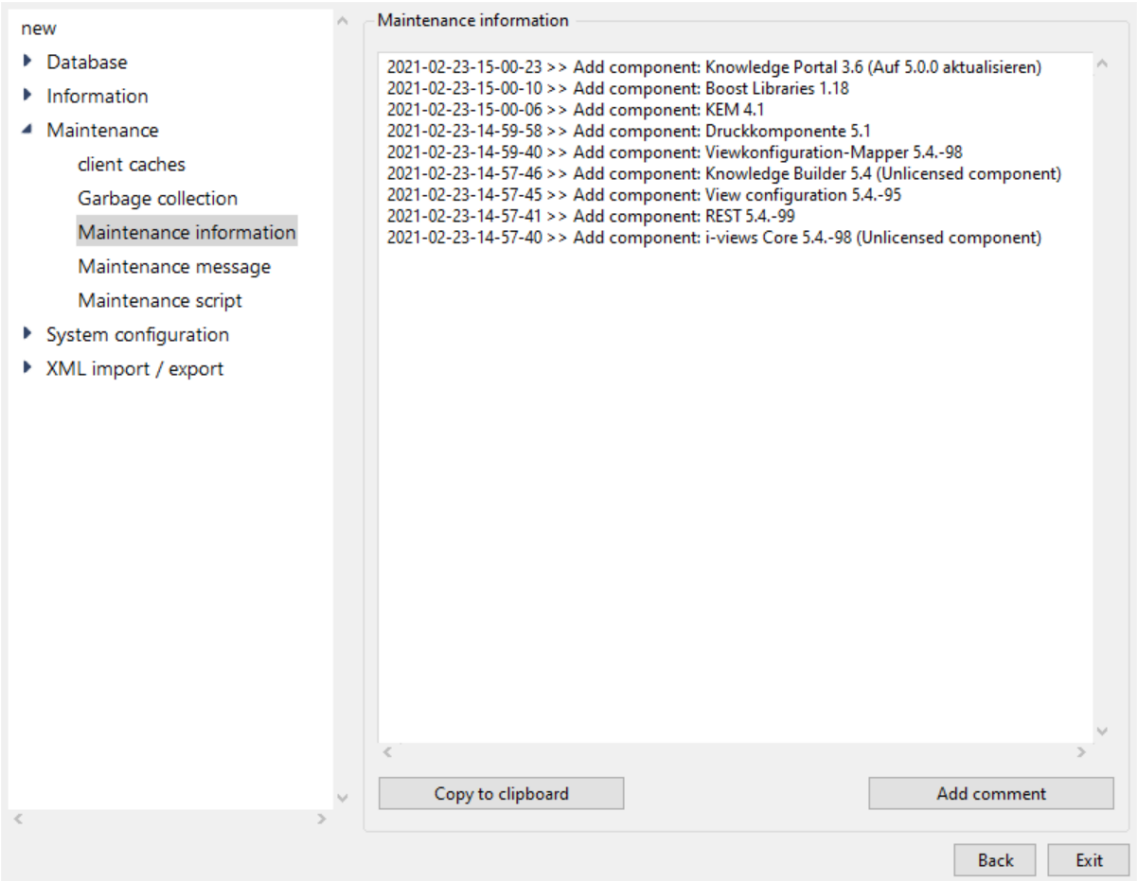
- a short description, if relevant including the cluster ID and the frame ID (format *cluster ID/frame ID*) of the faulty object (in the terminology interpreted by the program) (*notification*),
- the superordinate semantic element affected by the fault (*object*),
- its type (*type*),
- the severity of the fault (*priority*) and
- the first point in time at which it was identified in the form of a date (*date*)

are output. The individual columns of the table can be sorted by clicking on the head of the column.

Details displays all data listed in the **fault overview** of the selected fault in a new window. The time of the first point in time at which it was identified and date and time of the last time it was identified are added. The data there can be copied to the clipboard of the operating system (**Copy to clipboard** button) or be exported to any location as a text file that can be given any name using a saving dialog (**Save** button). The operation triggered using the **Details** button can, alternatively, be performed by double-clicking a fault in the fault overview.

Remove deletes the fault selected in the **fault overview**. This does not effect the first point in time at which the fault was identified.

2.5.2.4.4. Wartungsinformation



This menu option can be used to call up a chronologically ordered **maintenance history** of all essential administration processes in the Knowledge Graph since its creation. It contains backup and transfer processes, component installations and updates, and the execution of maintenance scripts and garbage collection, each with the time and date.

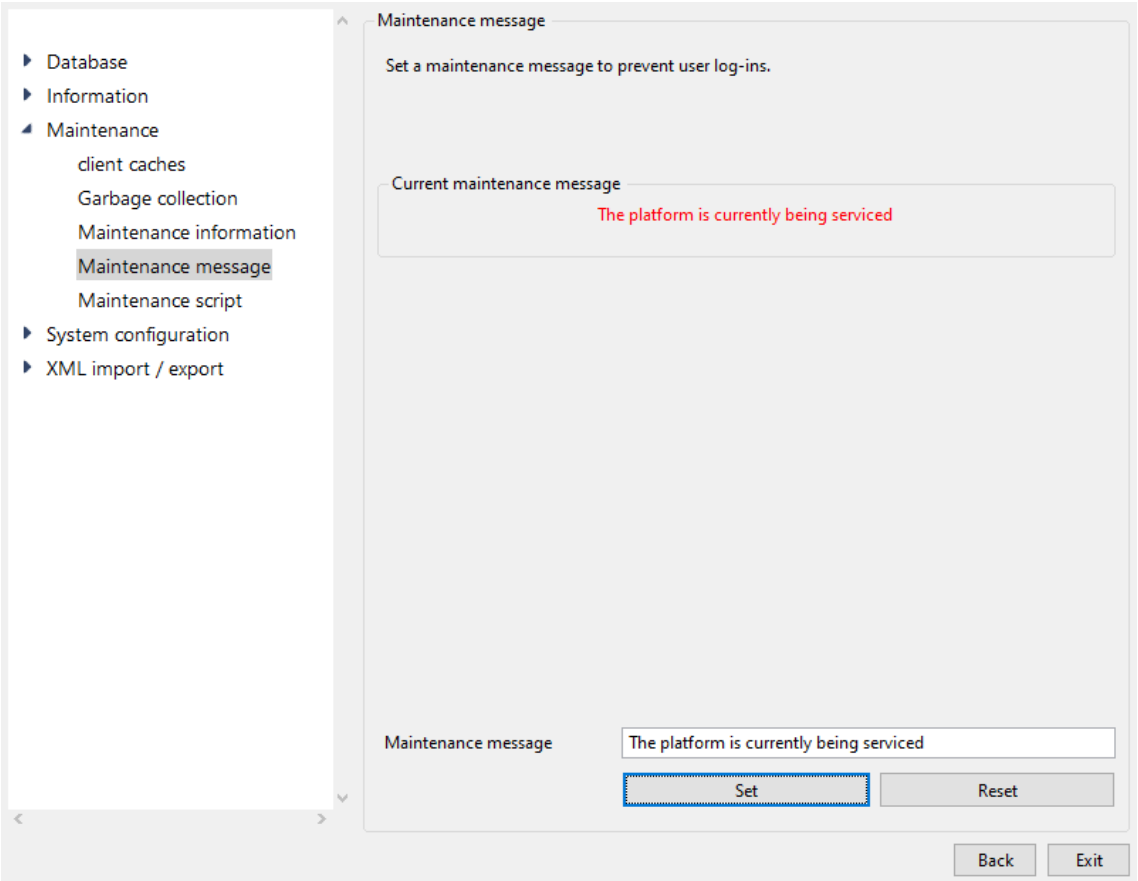
The **maintenance history** has a context menu that can be activated by right-clicking:

- **Select All** selects all the text. Alternatively, the mouse pointer can be used to mark any text segment.
- **Copy** copies the selected text area to the clipboard of the operating system.
- **Find Again** searches for the selected text area and finds its next occurrence in according to the read direction.
- **Find** allows a string to be input in a separate window, and its next occurrence in accordance with the read direction in relation to the position of the cursor set by clicking the mouse. The query is case-sensitive.

Copy to clipboard copies the entire **maintenance history** to the clipboard of the operating system.

Add comment allows a note to be entered via a free text field in a separate window. It is given a timestamp and added to the **maintenance history**. Notes added to the **maintenance history** cannot be deleted.

2.5.2.4.5. Wartungsnachricht

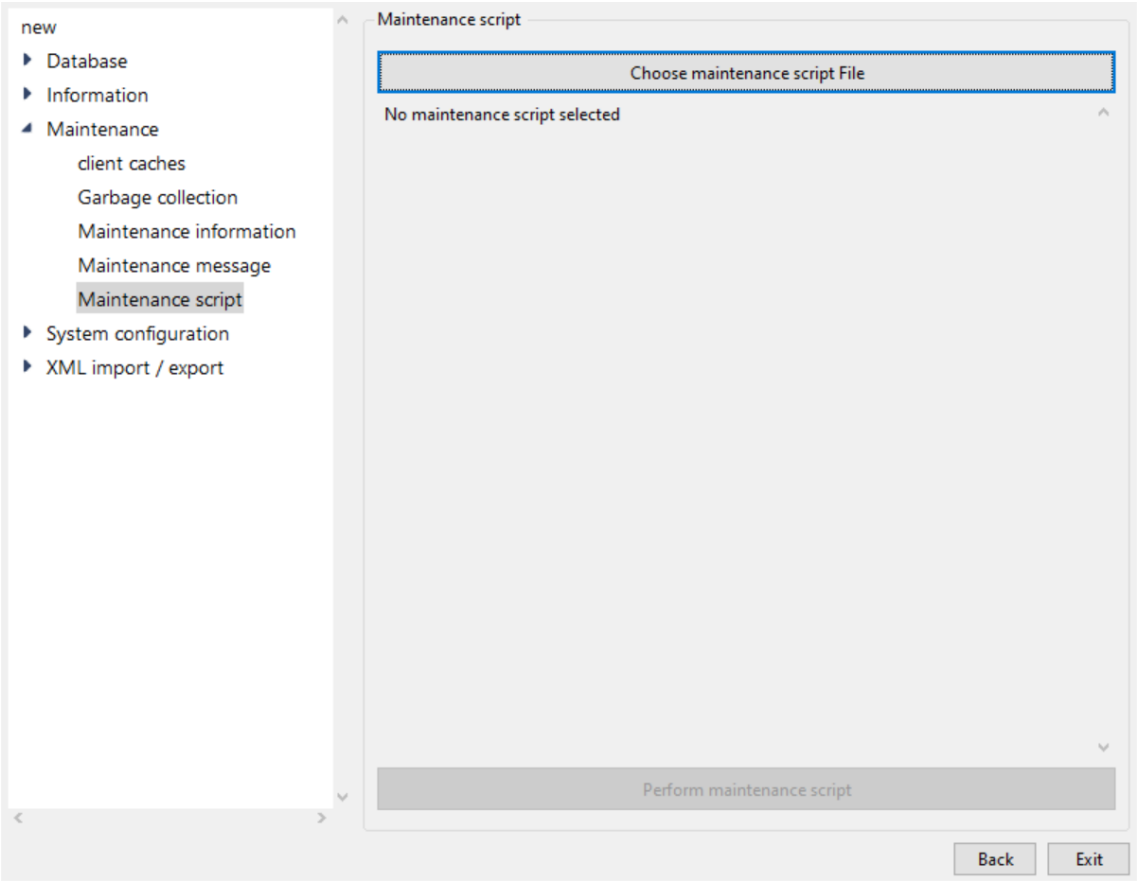


The **Set** button activates a maintenance block that prevents all users from accessing the Knowledge Graph via the Knowledge Builder. To do this, a maintenance notification must be written.

The maintenance notification is written in the free text field **Maintenance notification**. It is displayed as an error message shown to all users who try to access the Knowledge Graph via the Knowledge Builder when the maintenance block is active.

The **Reset** button removes the previously set maintenance block and deletes the maintenance notification.

2.5.2.4.6. 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

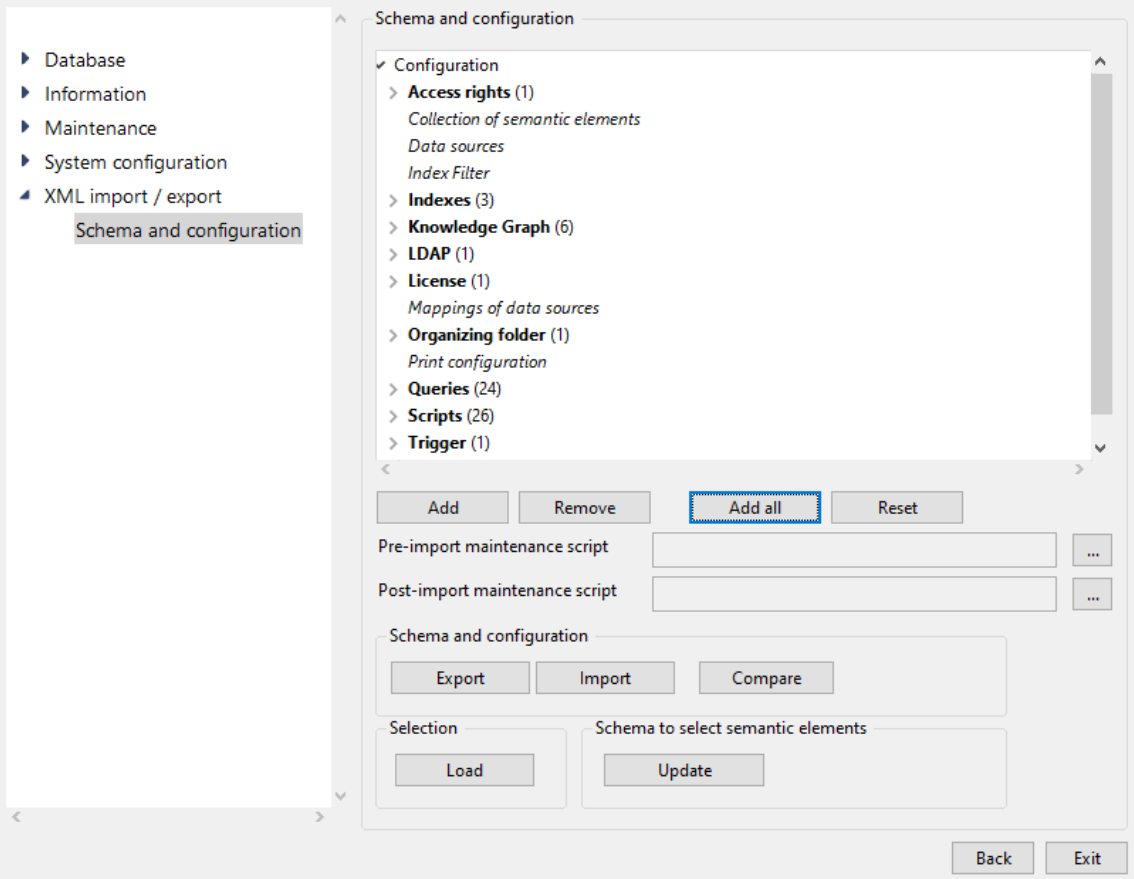
Ausführungsinformationen oder erlaubt skriptspezifische Ausführungsoptionen.

2.5.2.5. XML-Import/Export

2.5.2.5.1. Schema und Konfiguration

Ein Knowledge-Graph im weiteren Sinne besteht neben den nutzergenerierten und über Komponenten eingebrachten Teilgraphen (Schemata mit Nutzdaten) noch aus diversen weiteren Bausteinen (Konfigurationen), die diesen Teilgraph funktional erweitern, konfigurieren oder damit arbeiten. Im Rahmen dieses Menüpunkts werden Schemata und Konfigurationen zusammenfassend als Konfigurationen bezeichnet.

Zahlreiche Konfigurationen eines Knowledge-Graphen lassen sich gezielt exportieren und importieren.



Vorbereiten des Schemas für den Objekte-Transfer

Für den Transfer vereinzelter semantischer Elemente — insbesondere Instanzen (individuelle Objekte, Attribute und Relationen eines Typs) — und für die Steuerung des Verhaltens von Export und Import werden vorkonfigurierte XML-Attribute benötigt.

Vorbereitung der XML-Attribute

Um die XML-Attributtypen zu generieren, fügt man im **Admin-Tool** per **Aktualisieren** dem Knowledge-Graph die folgenden Boolesche hinzu, falls sie noch nicht existieren:

- *XML-Schematransfer: Alle Objekte exportieren*
- *XML-Schematransfer: Direkte Objekte exportieren*
- *XML-Schematransfer: Nicht überschreiben*
- *XML: Typ und alle Untertypen nicht exportieren*
- *XML: Untertypen nicht exportieren*

Diese Attributtypen werden benötigt, um bei einem Export einer Konfiguration des Konfigurationstyps Knowledge-Graph 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. Wenn ein Objekt oder Objekttyp exportiert wird, dann werden alle direkt mit ihm verbundenen Attribute und Relationen sowie deren Attributtypen oder Relationstypen ebenfalls exportiert.

Die **Konfigurationsübersicht** bietet einen listenartigen Überblick über alle mittels der im Folgenden beschriebenen Operationen prinzipiell transferierbaren Konfigurationstypen eines Knowledge-Graphen. 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*)
- Druckkonfiguration (*Druckkonfiguration*)
- Menge aller innerhalb der Rubrik *Ermittlung* der View-Konfiguration definierten Bausteine (*Ermittlung der View-Konfiguration*)
- Einzelne Indexfilter (*Indexfilter*)
- Einzelne Indexerkonfigurationen (*Indizes*)
- LDAP-Authentifizierung (*LDAP*)
- Lizenz des Knowledge-Graphen (*Lizenz*)
- Einzelne registrierte Sammlungen semantischer Objekte (*Sammlung von Knowledge-Graph Elementen*),
- Einzelne registrierte Skripte (*Skripte*)
- Arbeitsordner (*Strukturordner*)
- Menge aller innerhalb der Rubrik Trigger definierten Bausteine (*Trigger*)

- Einzelne Teilgraphen (*Knowledge Graph*)
- Menge aller innerhalb der Rubrik *Rechte* definierten Bausteine (*Zugriffsrechte*)

Darstellung

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.

Navigation

Das Ein- und Ausklappen von Listenunterpunkten in der **Konfigurationsübersicht** geschieht per Klick auf die Dreiecksymbole links neben den Listenpunkten. Alternativ steht ein Kontextmenü zur Verfügung, das über einen Klick mit der rechten Maustaste auf einen Listenpunkt aufgerufen wird:

- **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/entfernen von Konfigurationen

Hinzufügen fügt der **Konfigurationsübersicht** eine Konfiguration des dort ausgewählten Konfigurationstyps hinzu. Wenn mehr als eine Konfiguration für den ausgewählten Konfigurationstyp im Knowledge-Graph existiert, dann folgt eine Auswahlmöglichkeit in einem separaten Fenster. Die Auswahl erfolgt dort entweder einzeln per Klick auf die jeweiligen Konfigurationen 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 Knowledge-Graph existierenden Konfigurationen hinzu und verteilt diese auf die jeweils passenden Konfigurationstypen.

Wartungsskripte

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 Textfeld links von der jeweiligen Schaltfläche. Wenn im Anschluss Konfigurationen importiert werden, dann wird das Wartungsskript ausgeführt. Wenn im Anschluss Konfigurationen exportiert werden, dann 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. Die Ausführung erfolgt entweder vor dem Start des Importprozesses oder nach dem Ende des Importprozesses.

Export und Import

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: *[Knowledge Graph].tar*) oder des Ordners (kein Standardname) angegeben werden. Die Archivdatei oder 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 oder des Ordners frei festzulegen.

Import importiert nach einer Bestätigungsfrage Konfigurationen in den Knowledge-Graphen. 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: *[Knowledge Graph].tar*) oder des Ordners (kein Standardname) angegeben werden. Archivdatei oder Ordner werden im gleichen Ordner wie das Admin-Tool gesucht. Alternativ kann über **Wählen** auf das Dateisystem des Betriebssystems zugegriffen werden, um eine Archivdatei oder einen Ordner an einer beliebigen Stelle auszuwählen.
- Wenn die zu importierende Archivdatei oder der zu importierende Ordner gewählt wurde, dann erscheint in einem weiteren Fenster eine Übersicht über alle darin enthaltenen Konfigurationen. Diese Übersicht kann in die Zwischenablage des Betriebssystems kopiert werden (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 wird:
  - **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 gesetzten 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 diesen Knowledge-Graph 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 diesen Knowledge-Graph aus einer XML-Datei (Standarddateiname: *instruction.xml*) zu laden.

3. Web-Frontend

Das Web-Frontend ermöglicht die Darstellung von Daten aus dem Knowledge-Graph im Browser. Dazu muss eine *Anwendung* im Bereich ***TECHNIK > View-Konfiguration*** erstellt werden. Alle Möglichkeiten der View-Konfiguration können im Web-Frontend genutzt werden. So können auch komplexe Web-Anwendungen (z.B. Editieren von Daten, Tabellenfunktionen, interaktiver Graph, Navigationsbäume) mit einfachen Mitteln erstellt werden.

Über eine ***i-views-Bridge*** können Nutzer mit ihrem Web-Browser auf das konfigurierte Frontend zugreifen. Die Software-Komponente ***View-Configuration-Mapper*** legt die dafür benötigten Ressourcen im Bereich ***TECHNIK > REST*** an.

3.1. Konfiguration

Für die Konfiguration des Web-Frontends sind im Knowledge-Builder Einstellungen in den Bereichen ***TECHNIK > REST*** und ***TECHNIK > View-Konfiguration*** nötig. Damit sie durchgeführt werden können, muss die Software-Komponente *View-Configuration-Mapper* im Knowledge Graph installiert sein.

Die Komponente legt im Knowledge-Graph folgende Objekte an:

- Den REST-Service *viewconfig* unter ***TECHNIK > REST***
- Die *Anwendung Viewkonfiguration-Mapper* unter ***TECHNIK > View-Konfiguration***

Die angegebenen Namen von Rest-Service und *Anwendung* sind die Standardnamen, die von der Software-Komponente initial vergeben werden. Im weiteren Verlauf verwendet die Dokumentation diese Standardnamen.

Folgendes kann für das Web-Frontend konfiguriert werden:

- Optionen bezüglich Sprache und Aussehen
- Authentifizierungsmethode
- Sichten und Interaktionen der angelegten *Anwendung* (die eigentliche Funktionalität des Web-Frontends)

3.1.1. Optionen

Die Optionen für das Web-Frontend finden Sie an der REST-Ressource *vcm/options* am REST-Service *viewconfig*. Dort können CSS-Regeln, bevorzugte Sprachen, Übersetzungstexte und Themes konfiguriert werden. Es kann auch eine zweite REST-Ressource für Optionen hinterlegt werden. Diese ist für Web-Frontends mit Authentifizierung gedacht und wird erst nach erfolgreicher Authentifizierung eines Benutzers aufgerufen. Sie ist dann sinnvoll, wenn z.B. benutzerspezifische Einstellungen in den Optionen berücksichtigt werden sollen.

Es können folgende Optionseigenschaften konfiguriert werden:

CSS

In dieses Textfeld können Sie CSS-Regeln in Standard-CSS-Syntax eintragen. Sie werden in das CSS des Web-Frontends integriert und wirken sich entsprechend aus.

Skript für Optionen

Mit dem Skript können Sie u.a. angepasste Übersetzungstexte oder bevorzugte Sprachen an das Web-Frontend übergeben. Dafür steht in dem Skript ein Objekt vom Typ `$k.VCMOptions` zur Verfügung. Detaillierte Informationen dazu finden Sie in der JS-API-Dokumentation. Eine Liste der Standardübersetzungstexte finden Sie hier: Übersetzungstexte im Web-Frontend

```
function configure(options, request) {
  // Beispielhafte Übergabe von Schlüssel-Wert-Paaren für die
entsprechenden Sprachen.
  // Eine Übersicht über alle im Web-Frontend verwendeten Schlüssel und
Texte finden Sie weiter unten.
  options.setTranslations({
    de: {'login.username.failure': 'Bitte geben Sie einen Benutzername
an'},
    en: {'login.username.failure': 'Please supply a user name'}
  })
}
```

Optionen nach Authentifizierung

Hier können Sie bei Bedarf eine zusätzliche REST-Ressource für Optionen verknüpfen. Diese wird erst nach einer erfolgreichen Authentifizierung aufgerufen. So ist es z.B. möglich, benutzerspezifische Einstellungen in den Optionen zu berücksichtigen. Ein Beispiel hierfür wäre die bevorzugte Sprache für das Web-Frontend.
Der empfohlene Weg zum Anlegen der neuen Ressource ist über den Button  im Menü. Wählen Sie *Options resource* aus der angebotenen Liste. Es wird eine zweite Ressource mit Namen *vcm/options* angelegt. Nachdem Sie diese neue Ressoure hier verknüpft haben, ändert sich ihr Name zu *vcm/options-post-authentication*.

Verwende Theme

Verknüpfen Sie hier eine oder mehrere Theme-Instanzen. Weitere Informationen dazu finden Sie unter [technical-handbook-web-frontend-configuration-options-Themes].

3.1.1.1. Themes

Themes ermöglichen es, das visuelle Erscheinungsbild des Web-Frontends anzupassen. Sie werden als *Theme*-Instanzen im Knowledge-Graph des REST-Service *viewconfig* verwaltet und über die Relation ***Verwende Theme*** an der *vcm/options*-Ressource verknüpft.

Das Anlegen eines Themes mit dem Namen `vcmLight` überschreibt das Standard-Theme des Web-Frontends. Diese Änderung wirkt sich sofort auf das gesamte Web-Frontend aus, ohne dass eine Panel-Konfiguration notwendig ist.

Themes können auch gezielt an einzelne Panels gebunden werden. Weitere Informationen dazu finden Sie unter Panels.

3.1.1.1.1. Konfiguration eines Themes

Konfigurationseigenschaften eines Themes

Name

Der Name der *Theme*-Instanz. Er wird verwendet, um das Theme zu referenzieren, z.B. bei der Verknüpfung mit einem Panel. Der Name darf nur Buchstaben, Ziffern, Bindestriche und Unterstriche enthalten.

Theme-Definition

Ein JSON-Objekt, das ein benutzerdefiniertes Vuetify-Theme beschreibt. Weitere Informationen zur Struktur finden Sie in der Vuetify-Dokumentation.

```
{
  "colors": {
    "primary": "#1976D2",
    "secondary": "#424242"
  }
}
```

Mit Hintergrund

Ist diese Option aktiviert, wird die Hintergrundfarbe des Themes auf das verknüpfte Panel angewendet. Wenn das Theme nicht an ein Panel gebunden ist, hat diese Option keine Auswirkung.

3.1.1.2. Übersetzungstexte im Web-Frontend

Schlüssel	Deutsch	Englisch
button.cancel	Abbrechen	Cancel
button.ok	OK	OK
confirmation.accept.label	Ja	Yes
confirmation.decline.label	Nein	No
errors.apiRequestError	Leider ist ein Fehler aufgetreten. Weitere Informationen finden Sie in der Konsole Ihres Browsers.	Unfortunately an error has occurred. Please check your browser's console for more information.

Schlüssel	Deutsch	Englisch
errors.authenticationErrorBasic	Bitte die Seite neu laden und den Browser-Login verwenden. Weitere Informationen finden Sie in der Konsole Ihres Browsers.	Please reload the page and use the browser login. Please check your browser’s console for more information.
errors.initializingError	Fehler bei der Initialisierung. Weitere Informationen finden Sie in der Konsole Ihres Browsers.	Error during initialization. Please check your browser’s console for more information.
errors.misconfiguredSwitchLanguageAction	Followup "switch-language" ohne konfigurierte Sprache. Weitere Informationen finden Sie in der Konsole Ihres Browsers.	Followup "switch-language" without configured language. Please check your browser’s console for more information.
errors.missingActionReceiver	Die Aktion ({0}) kann nicht ausgeführt werden. Der Empfänger konnte nicht ermittelt werden. Weitere Informationen finden Sie in der Konsole Ihres Browsers.	The action ({0}) cannot be executed. The receiver could not be determined. Please check your browser’s console for more information.
errors.unknownError	Leider ist ein Fehler aufgetreten. Weitere Informationen finden Sie in der Konsole Ihres Browsers.	Unfortunately an error has occurred. Please check your browser’s console for more information.
errors.unsupportedAuthentication	Nicht unterstützte Authentifizierung. Weitere Informationen finden Sie in der Konsole Ihres Browsers.	Unsupported authentication. Please check your browser’s console for more information.
facets.showAll	Alle anzeigen	Show all
login.button.label	Anmelden	Login
login.failure.message	Die Anmeldung ist fehlgeschlagen. Bitte versuchen Sie es erneut.	Login invalid. Please try again.
login.message		
login.password.label	Passwort	Password
login.password.placeholder		
login.title	Anmeldung	Login
login.username.failure	Benutzername darf nicht leer sein.	A username is required.

Schlüssel	Deutsch	Englisch
login.username.label	Benutzer	Username
login.username.placeholder		
pager.default.total	{0}-{1} von {2}	{0}-{1} of {2}
pager.more	Mehr anzeigen ({0})	Show more ({0})
passwordChange.new	Neues Passwort	New password
passwordChange.old	Altes Passwort	Former password
passwordChange.repeat	Neues Passwort wiederholen	Repeat new password
passwordChange.repeat.warning	Die eingegebenen Passwörter stimmen nicht überein.	The passwords do not match.
plugin.maps.button.home	Ansicht zurücksetzen	Reset view
plugin.maps.coordinateSystem.background	Hintergrund	Background
plugin.maps.coordinateSystem.grid	Koordinatenraster	Coordinate grid
property.container.manualSpawn.label	Hinzufügen	Add
property.flextime.label.day	Tag	day
property.flextime.label.hours	Stunde	hour
property.flextime.label.minutes	Minute	minute
property.flextime.label.month	Monat	month
property.flextime.label.year	Jahr	year
property.flextime.validation.invalidFormat	Der Flextime-Wert hat ein falsches Format. Erlaubte Formate sind: {0}.	The flextime value has a wrong format. Allowed formats are: {0}.
property.flextime.validation.invalidValue	In die Felder dürfen nur Ganzzahlen eingegeben werden.	Each input value must be a number.
property.geo.getCurrentPositionError	Die aktuelle Position kann nicht bestimmt werden.	The current position cannot be determined.
property.interval.divider.label	bis	to
property.interval.max.label	Bis	Max
property.interval.min.label	Von	Min
property.interval.validation.max	'Bis'-Wert: {0}	Max value: {0}

Schlüssel	Deutsch	Englisch
property.interval.validation.min	'Von'-Wert: {0}	Min value: {0}
property.interval.validation.minGreaterThanMax	Der 'Von'-Wert darf nicht größer als der 'Bis'-Wert sein.	The min value must not be greater than the max value.
ui.boolean.false	Nein	No
ui.boolean.true	Ja	Yes
validation.date.notADate	Der Wert ist kein korrektes Datum.	The value is not a correct date.
validation.date.notATimeValue	Der Wert ist keine korrekte Uhrzeit.	The value is not a correct time.
validation.edit.invalid.message	Es gibt fehlerhafte Einträge in den Werten. Speichern ist nicht möglich.	There are validation errors. Saving the values is not possible.
validation.property.blob.uploading	Die Datei wird noch übertragen.	The file upload is still in progress.
validation.property.float.nan	Die Eingabe muss eine Zahl sein.	The input must be a number.
validation.property.integer.nan	Die Eingabe darf nur aus Zahlen bestehen.	The input may only consist of numbers.
validation.property.pattern.message	Die Eingabe entspricht nicht dem Muster '{0}'.	The value does not match the pattern '{0}'.
validation.propertyGroup.minOccurrences.message	Die minimale Anzahl an Werten für diese Eigenschaft beträgt {0}.	The minimal number of values for this property is {0}.
validation.propertyGroup.minOccurrences.required.message	Das ist ein Pflichtfeld.	This field is required.
validation.required	Pflichtfeld	Mandatory field

3.1.2. Authentifizierung

An der Eigenschaft *Authentication* des REST-Service kann eine Authentifizierungsmethode für das Web-Frontend eingestellt werden. Initial ist die Methode *Bearer* konfiguriert. Das Web-Frontend unterstützt darüber hinaus die Authentifizierungsmethode *Basic*. Sollte keine Authentifizierung gewünscht sein, muss explizit eine Authentifizierung vom Typ *No authentication* hinterlegt werden. Genaueres finden Sie im Kapitel REST-Services.

3.1.2.1. Login-Formular

Wenn die Authentifizierungsmethode *Bearer* aktiviert ist, wird beim ersten Aufruf des Web-Frontends ein Login-Formular angezeigt. Die Texte des Formulars können Sie bei Bedarf über die

Optionen anpassen. Die entsprechenden Text-Schlüssel und ihre Verwendung finden Sie weiter unten. Zur Verdeutlichung folgen zwei Screenshots: Einer mit den Standardtexten und einer, in dem die Text-Schlüssel an entsprechender Stelle angezeigt werden.

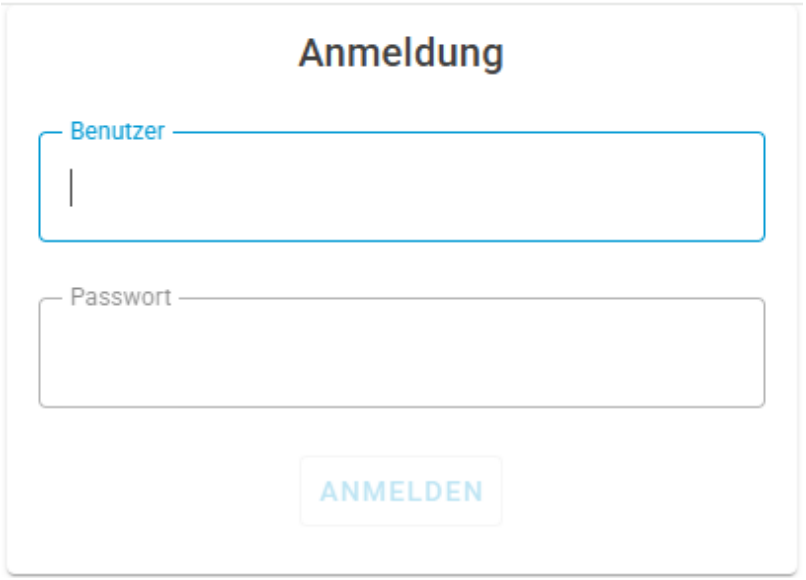


Figure 29. Login mit Standardtexten

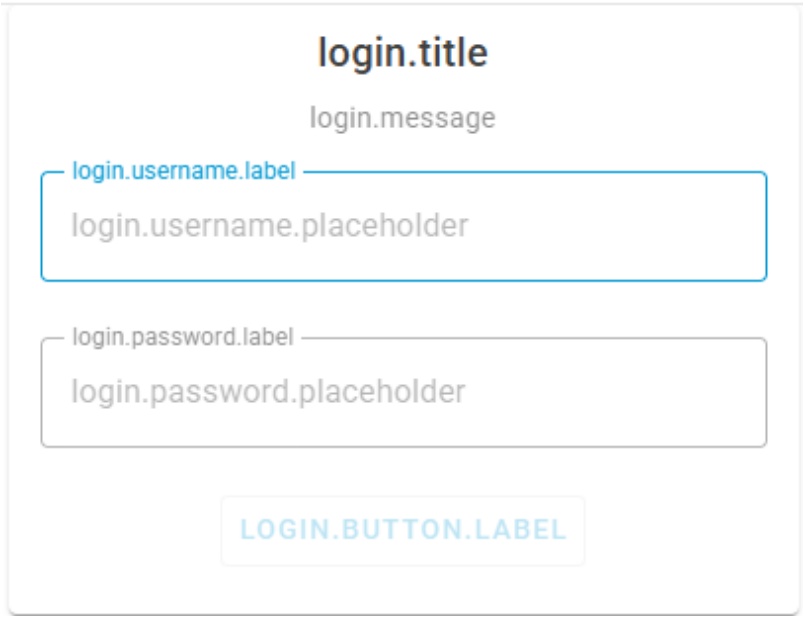


Figure 30. Login mit Text-Schlüsseln

Textschlüssel für das Login-Formular

Schlüssel	Text	Verwendung
login.button.label	Anmelden	Beschriftung des Login-Buttons
login.failure.message	Die Anmeldung ist fehlgeschlagen. Bitte versuchen Sie es erneut.	Fehlermeldung bei fehlgeschlagener Anmeldung

Schlüssel	Text	Verwendung
login.message		Nachricht unterhalb des Formulartitels
login.password.label	Passwort	Beschriftung des Passwortfeldes
login.password.placeholder		Platzhalter des Passwortfeldes
login.title	Anmeldung	Titel des Login-Formulars
login.username.failure	Benutzername darf nicht leer sein.	Fehlermeldung unterhalb des Feldes für den Benutzernamen
login.username.label	Benutzer	Beschriftung des Feldes für Benutzernamen
login.username.placeholder		Platzhalter des Feldes für Benutzernamen

3.1.2.2. Logout

Für das Logout steht im Web-Frontend eine spezielle Aktion zur Verfügung. Genaueres finden Sie hier.

3.1.3. Anwendungsidentifikator

Ein Web-Frontend besteht aus zwei verbundenen Komponenten: einem REST-Service und einer *Anwendung*. Diese werden durch einen eindeutigen Identifikator verknüpft, der sowohl in der Eigenschaft *Identifikator* der *Anwendung* als auch in der Eigenschaft *Application ID* des REST-Service eingegeben werden muss.

Beim initialen Setup wird eine *Anwendung* mit dem Namen *Viewkonfiguration-Mapper* im Bereich ***TECHNIK > View-Konfiguration*** erstellt. Der Standardidentifikator lautet `viewConfigMapper`. Sie können diesen bei Bedarf anpassen. Für zusätzliche Web-Frontends müssen Sie einen neuen Identifikator selbst definieren.

3.2. Anwendung

Eine *Anwendung* ist die zentrale Konfigurationseinheit für ein Webfrontend. Sie definiert das Gesamtlayout, die verfügbaren Datenansichten (*Views*) und die Benutzerinteraktionen. *Anwendungen* werden im Knowledge-Builder unter **TECHNIK > View-Konfiguration** angelegt und stellen den Startpunkt für die Konfiguration eines Webfrontends dar.

Das Layout einer *Anwendung* wird durch *Panels* strukturiert. In diese Panels werden *Views* mit den entsprechenden Datenansichten (Suche, Eigenschaften etc.) integriert. Für Interaktionen und Geschäftslogik stehen Aktionen und Skripte zur Verfügung.

Konfigurationseigenschaften

Identifikator

Die hier eingetragene Zeichenkette dient als eindeutige Kennung und muss beim entsprechenden REST-Service als *Application ID* konfiguriert werden.

Sprachen

Definiert die im Webfrontend angezeigten Sprachen. Wenn nur eine Auswahl der konfigurierbaren Sprachen angezeigt werden soll, wird hier die Liste der anzuzeigenden Sprachen festgelegt.

Zuletzt ausgeklappte Knoten wiederherstellen

Globale Einstellung zur Speicherung des Ausklappzustands von Knoten in Hierarchien und Bäumen.

Zuletzt gewählte Alternativen wiederherstellen

Globale Einstellung zur Speicherung des zuletzt aktiven Reiters in Alternativen-Ansichten.

Zuletzt gewählte Sortierung/Spaltenfilterung wiederherstellen

Globale Einstellung zur Speicherung von Tabellenkonfigurationen wie Sortierung und Spaltenfiltern.

3.2.1. Panels

Mit Panels wird das Layout eines Web-Frontends aufgebaut. Komplexe Layout-Strukturen können durch die Schachtelung von Panels realisiert werden. Das Anzeigen konkreter Daten erfolgt schlussendlich über Views, die in die Panels eingehängt werden.

Für das Anzeigen von Daten greifen Views auf ihr Modell (auch Domain-Model genannt) zu. Das Domain-Model kann ein Element des Knowledge-Graphs, eine Liste von Elementen, eine (parametrisierte) Suchdefinition oder Suchparameter sein. Dieses Modell kann über die Beeinflussungskette der Panel-Aktivierung an Views weitergegeben oder von ihnen selber bestimmt werden.

Beim ersten Anzeigen des Web-Frontends ist standardmäßig das Wurzel-Panel der Anwendung, das *Hauptfensterpanel*, aktiv und damit sichtbar. Welche Sub-Panels aktiv sind, ergibt sich aus dem Typ

der Panels und der Beeinflussungskette. Beides wird weiter unten beschrieben. Mit Aktionen können Sie die Aktivierung von Panels weiter beeinflussen. Eine Navigation durch das Web-Frontend wird damit möglich.

Panels ermöglichen darüber hinaus auch das Anzeigen von modalen Dialogen.

Es gibt folgende Panel-Typen:

Festgelegte Ansicht

Sie können hier eine beliebige View eingehängen. Durch die Verwendung einer Layout-View sind auch komplexere Strukturen möglich.

Flexible Ansicht

Sie können hier beliebige Views eingehängen. Das Panel wählt die am besten geeignete View basierend auf dem Domain-Modell und der *Anwenden auf*-Konfiguration der Views aus.

Lineares Layout

Dient der vertikalen oder horizontalen Anordnung weiterer Panels.

Wechselndes Layout

Dieses Panel kann mehrere Unterpanels enthalten, wobei immer nur eines aktiv sein kann. Ein Unterpanel wird auf drei Arten aktiviert:

1. Über eine *Skript für Aktivierung*- bzw. *Suche für Aktivierung*-Konfiguration am Unterpanel.
2. Basierend auf dem *Modus zur Aktivierung von Sub-Panels* (erstes oder letztes Panel).
3. Manuell über die Konfiguration *Ergebnis anzeigen in Panel* an einer ausgeführten Aktion.

3.2.1.1. Spezial-Panels

Abgesehen von den Panel-Typen gibt es noch folgende Spezial-Panels:

Dialog-Panel

Dialog-Panels sind spezielle Anzeigebereiche, deren Inhalte in einem Dialogfenster angezeigt werden.

Fenstertitelpanel

Nur in Dialog-Panels und im Hauptfensterpanel: Der Kopfbereich des Panels.

Fußzeilenpanel

Nur in Dialog-Panels: Der Fußbereich des Panels.

Hauptfensterpanel

Das Wurzel-Panel der *Anwendung*.

3.2.1.2. Konfiguration

Konfigurationseigenschaften von Panels

Paneltyp

`Festgelegte Ansicht`, `Flexible Ansicht`, `Lineares Layout` oder `Wechselndes Layout`

Ausrichtung

Nur für *Lineares Layout* und *Wechselndes Layout*: `horizontal` oder `vertikal`.

Bookmark-Pfad

Siehe Bookmarks und Historie.

Rolle

Markiert das Panel mit der entsprechenden Rolle. Wird z.B. von Aktionen benötigt, die durch eine bestimmte Rolle ausgeführt werden.

Beeinflusst

Legen Sie hier Panels hinterlegen, die Teil der Beeinflussungskette sein sollen. Bei Aktivierung eines Panels werden auch alle von ihm beeinflussten Panels aktiviert.

Session-Grenze

Das Aktivieren von Panels mit der Markierung *Session-Grenze* erzeugt eine neue Session auf dem Session-Stapel, deren Lebensdauer bis zum Deaktivieren des Panels andauert.

Kontextelement

Veraltet! Eine Möglichkeit, das Domain-Modell zu setzen.

Nicht durch äußeres Kontextelement überschreibbar

Ein übergebenes Domain-Model ersetzt nicht das durch *Kontextelement* oder *Skript für Kontextelement* bestimmte Modell.

class

CSS-Klassen für das Panel.

Breite / Höhe

Die exakten Maße des Panels können Sie hier jeweils setzen.

Maximale Breite / Höhe

Legen Sie hier die Höchstmaße des Panels fest. Das Panel nimmt sich so viel Platz wie es benötigt, ohne diese Werte zu überschreiten.

Flex-grow / -shrink

Geben Sie hier die Werte für die jeweilige CSS-Eigenschaft für den Wachstums- bzw. Schrumpffaktor des Panels an. Ein Element mit einem Wert von 2 für flex-grow erhält beispielsweise doppelt so viel Platz wie ein Element mit Wert 1.

overflow-x / -y

Legen Sie hier die Darstellung von Scrollbars in der Anwendung fest, 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.

Theme anwenden

Verknüpfen Sie hier eine *Theme*-Instanz, um dem Panel ein eigenes Design zu geben. Die Theme-Konfiguration muss zuvor an der VCM-Options-Ressource über ***Verwende Theme*** hinterlegt werden. Wenn die Option **Mit Hintergrund** am Theme aktiviert ist, wird die Hintergrundfarbe des Themes auf dieses Panel angewendet. Weitere Informationen zur Konfiguration von Themes finden Sie unter Optionen.

3.2.1.3. Dialog-Panel

Dialog-Panels sind spezielle Anzeigebereiche, deren Inhalte in einem Dialogfenster angezeigt werden. Dialogfenster werden inhaltlich in die folgenden drei Bereiche unterteilt:

- Fenstertitel
- Inhaltsbereich
- Fußzeile

Die Inhalte und das Layout innerhalb der drei Bereiche können Sie jeweils über eine eigene Panel-Konfiguration festlegen. Das Dialog-Panel selbst steht dabei stellvertretend für den Inhaltsbereich. Zur Konfiguration von Fenstertitel und Fußzeile legen Sie am Dialog-Panel eine Unterkonfiguration vom Typ Fenstertitel- oder Fußzeilen-Panel an.

Die Dialogfenster werden automatisch sichtbar, wenn das entsprechende Dialog-Panel aktiviert wird. Die Aktivierung kann dabei wie bei anderen Panels auch gezielt über bestimmte Aktionen erfolgen.

Zum Ausblenden ("Schließen") von Dialogen verwenden Sie Aktionen. Wenn Sie in einer Aktionskonfiguration das Attribut *Panel schließen* anhaken, führt die Ausführung dieser Aktion in einem Dialogfenster dazu, dass das Fenster ausgeblendet wird. Die Aktion muss dazu im Dialog-Panel selbst oder einem seiner untergeordneten Panels konfiguriert werden.

3.2.1.4. Panel-Aktivierung und Beeinflussungskette

Panels kennen zwei grundsätzliche Zustände: aktiv und inaktiv. Ein Panel ist sichtbar, wenn es aktiv ist. Die Aktivierung von Panels funktioniert über folgende grundlegende Mechanismen:

- Initiales Laden eines Web-Frontends: Das Haupfensterpanel der Anwendung wird aktiviert.
- Ausführen einer Aktion: Der Ausführungsort (*Ergebnis anzeigen in Panel* an der Aktion) bestimmt, welches Panel aktiviert wird.

Wenn ein Panel aktiviert wird, wird auch sein Inhalt neu berechnet. Ausnahmen hiervon sind Schritt 6 der Folge-Aktivierung weiter unten und bestimmte Einstellungen des Aktivierungsmodus bei Panel-Beeinflussung und Aktionsausführung. Ausgehend von den zwei Aktivierungsmechanismen wird eine Kette von Folge-Aktivierungen in dieser Reihenfolge ausgelöst:

1. Beeinflusste Panels (*Beeinflusst* am Panel) werden aktiviert.
2. Spezial-Panels werden aktiviert.
3. Unter-Panels werden aktiviert.
4. Im Falle eines Panels mit wechselndem Layout: Das aktive Panel wird bestimmt und seine Geschwister-Panels werden deaktiviert.
5. Wieder zu 1. bis keine weiteren Panels aktiviert werden können.
6. Zuletzt wird sichergestellt, dass alle Panels oberhalb der aktivierten Panels auch aktiv sind. Dabei wird keine Neuberechnung der Inhalte ausgelöst.

Folge-Aktivierungen transportieren jeweils das aktuelle Domain-Model. Wenn also beispielsweise Panel A das Objekt "Herr Meier" anzeigt, dann zeigt das aktivierte Unterpanel B ebenso "Herr Meier" an.

3.2.1.4.1. Aktivierungsmodus

Bei Panel-Aktivierung über Beeinflussung oder Aktionsausführung können Sie über den *Aktivierungsmodus* die Berechnung der Panel-Inhalte optimieren.

Es lässt sich damit vermeiden, dass Panel-Inhalte neu berechnet werden, die aktuell nicht angezeigt werden, weil sie trotz Aktivierung nicht im Sichtbarkeitsbereich liegen (z.B. ein Warenkorb).

Aktivierungsmodi

Model und Ansicht aktualisieren

Aktualisiert die Panel-Inhalte nur, wenn das Panel schon aktiv ist.

Nur Ansicht aktualisieren

Aktualisiert die View-Inhalte, behält den View-Status und das Domain-Model.

Standard

Aktiviert das Panel und aktualisiert die Panel-Inhalte. Dies ist die Standard-Einstellung, wenn keine der anderen oben beschriebenen Optionen ausgewählt wurde.

3.2.2. Aktionen

Aktionen dienen der Interaktion mit dem Web-Frontend. Sie ermöglichen das Navigieren und das Bearbeiten von Daten des Knowledge-Graphen.

Einige der Konfigurationseigenschaften von Aktionen sind in Kategorien unterteilt. Es werden zunächst diejenigen ohne Kategorie beschrieben.

3.2.2.1. Konfigurationseigenschaften

Konfigurationseigenschaften ohne Kategorie

Bookmark-Pfad

Siehe Bookmarks und Historie.

Aktionsart

Mögliche Aktionsarten sind: `Abbrechen`, `Anzeigen`, `Auswahl`, `Download`, `Drucken`, `NN-Expand`, `NN-Hide`, `NN-Pin`, `Passwortänderung`, `Relationsziel wählen`, `Skript`, `Speichern`, `Suchen`, `Web-Link`

Skript (benutzerdefiniert)

Wird beim Ausführen der Aktion ausgewertet. Es darf Elemente des Knowledge-Graphen verändern und bestimmt das Aktionsergebnis (`ActionResult`). Dieser Skript-Typ ist verfügbar, wenn einer der folgenden Aktionsarten ausgewählt wurde: `Auswahl`, `Relationsziel wählen`, `Skript`

Ausführen durch

Hier wird eine Rolle hinterlegt, durch die die Aktion ausgeführt werden soll.

Transaktion

`beginnen` oder `beenden`. Mit der Transaktionsart `beginnen` können Sie einen temporären Zustand (bzw. ein temporäres Element) erzeugen, bis eine andere Aktion die Transaktion mittels der Transaktionsart `beenden` bestätigt.
Beispiel: Ein Objekt soll in einem Dialogfenster neu angelegt werden. Erst bei Betätigen eines Speichern-Buttons soll das Objekt tatsächlich im Knowledge-Graph erzeugt werden (Aktionsart `Speichern` mit Transaktionsart `beenden`). Ansonsten sollen bei Abbruch der Transaktion kein Objekt erzeugt oder Änderungen verworfen werden (Schließen des Dialogfensters mit Aktionsart `Abbrechen`, ohne Angabe einer Transaktionsart).

Intent

Markiert die Aktion mit einer beliebigen Zeichenkette. Sie kann dann von Custom-Views gezielt adressiert und ausgewertet werden.

Darstellung

Bildcontainer

Bild für die Schaltfläche der Aktion. In dem gewählten Bildcontainer können Sie bei *Vordefiniertes Bild* ein Icon der Material Design Icons verwenden. Tragen Sie dafür die auf der Web-Seite dokumentierte Zeichenkette des gewünschten Icons ein.

Vor der Ausführung

Skript (vor der Aktion)

Nur für Aktionsart `Speichern`. Code, der vor dem Speichern der Daten ausgeführt wird.

Frage vor der Ausführung

Der hier angegebene Text wird dem Nutzer vor dem Ausführen der Aktion in einem Dialogfenster angezeigt. Sie können die Aktion abbrechen oder annehmen.

Skript für Frage vor der Ausführung

Der Text für die *Frage vor der Ausführung* kann mit diesem Skript erzeugt werden. Der Dialog wird nur angezeigt, wenn das Skript einen Text zurückgibt.

Nach der Ausführung

Benachrichtigung

Text, der nach der Aktion im Web-Frontend angezeigt wird.

Skript für Benachrichtigung

Skript zur Erzeugung des Benachrichtigungstextes.

Skript (nach der Aktion)

Nur für Aktionsart `Speichern`. Code, der nach erfolgter Speicherung der Daten ausgeführt wird.

Skript (recall)

Skript für eine Recall-Aktion.

Skript (ActionResponse)

Ein hier angegebenes Skript löst eine ActionResponse nach der Aktion aus.

Nach der Ausführung (Panels)

Panel neu aufbauen

Hier können Sie manuell festlegen, ob das Panel neu berechnet werden soll.

Panel wird neu aufgebaut

Zeigt an, ob das Panel neu berechnet wird und was die Standardeinstellung für die Aktionsart ist.

Ergebnis anzeigen in Panel

Ein Panel, das durch diese Aktion aktiviert wird. Über Meta-Attribute können Sie das Verhalten genauer steuern:

- *Aktivierungsmodus*: `Standard`, `Modell und Ansicht aktualisieren`, `Nur Ansicht aktualisieren`
- *Skript für Aktivierung*: Mit diesem Skript können Sie steuern, welche der konfigurierten Panel-Aktivierungen tatsächlich ausgeführt werden sollen.
- *Skript für Zielobjekt*: Das zurückgegebene Objekt wird zum Modell des Ziel-Panels.

Recall-Ergebnis anzeigen in Panel

Wie *Ergebnis anzeigen in Panel*, aber für das Ergebnis des *Skript (recall)*.

Panel schließen

Gibt an, ob ein Dialog-Panel nach der Aktion geschlossen wird.

3.2.2.2. Styles

Für Aktionen und Menüs

extra

Typ: `object`
Konfiguriert das Erscheinungsbild der Aktionsschaltfläche. Wird dieses Feld an einem Menü gesetzt, gelten die Werte als Standard für alle Aktionen des Menüs. Aktionsseitige Werte haben Vorrang vor menüseitigen Werten.

appearance

Typ: `object`
Objekt zur Konfiguration des Erscheinungsbilds.

variant

Typ: `string`
Bestimmt die Darstellungsart der Schaltfläche. Mögliche Werte: `"flat"`, `"outlined"`, `"plain"`, `"text"`, `"tonal"`, `"elevated"`.

color

Typ: `string`
Legt die Farbe der Schaltfläche fest. Es können CSS-Farbnamen, Hex-Werte oder benannte Theme-Farben wie `"primary"` oder `"secondary"` angegeben werden.

size

Typ: `string`
Legt die Größe der Schaltfläche fest. Mögliche Werte: `"x-small"`, `"small"`, `"default"`, `"large"`, `"x-large"`.

density

Typ: `string`
Legt die Kompaktheit der Schaltfläche fest. Mögliche Werte: `"default"`, `"comfortable"`, `"compact"`.

Beispiel

```
{
  appearance: {
    variant: 'flat',
    size: 'small',
    density: 'compact',
    color: 'primary'
  }
```

```
}
```

3.2.2.3. Spezielle Aktionsarten

Einige Aktionsarten erfordern eine spezifische Konfiguration. Die folgenden Abschnitte beschreiben diese Aktionsarten im Detail.

3.2.2.3.1. Logout-Aktion

Konfigurieren Sie eine Aktion mit der Aktionsart `Skript` und dort ein *Skript (ActionResponse)*. Erzeugen Sie in dem Skript ein Objekt vom Typ $k.ActionResponse. Setzen Sie an dem Objekt mit `setFollowup('logout')` den Followup-Wert und geben Sie das Objekt zurück.

```
function actionResponse(element, context, resultModel) {
  const actionResponse = new $k.ActionResponse()
  actionResponse.setFollowup('logout')

  return actionResponse
}
```

3.2.2.3.2. Sprachumschaltungsaktion

Konfigurieren Sie eine Aktion mit der Aktionsart `Skript` und dort ein *Skript (ActionResponse)*. Erzeugen Sie in dem Skript ein Objekt vom Typ $k.ActionResponse. Rufen Sie an dem Objekt `setFollowup('switch-language')` auf. Rufen Sie am gleichen Objekt `setData()` auf und übergeben Sie ein Objekt mit der Eigenschaft `language`. Der Wert der Eigenschaft kann ein Array oder eine Zeichenkette mit einer kommaseparierten Liste von Sprachcodes sein. Es können Sprachcodes im Format ISO 639-1 ('de', 'en') oder ISO 3166-1 (`de-DE`, `en-US`) angegeben werden.

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

  actionResponse.setFollowup('switch-language')
  actionResponse.setData({
    language: "de-DE"
  })

  return actionResponse
}
```

3.2.2.4. Spezifische Anwendungsfälle

Die folgenden Abschnitte beschreiben häufige Anwendungsfälle im Umgang mit Aktionen.

3.2.2.4.1. Recall-Aktionen

Es ist manchmal nötig, am Ende einer Sequenz von Aktionen wieder zu einer früheren Aktion der Sequenz zurückzukehren, um dort abschließenden Code auszuführen. Für solche Fälle steht das Konzept der Recall-Aktion zur Verfügung.

Zwei Schritte sind dafür nötig:

1. An der Aktion, zu der eine Rückkehr erfolgen soll, wird ein *Skript (recall)* angelegt. Durch dieses Skript wird die Aktion als Recall-Aktion markiert.
2. An der Aktion, von der aus die Rückkehr erfolgen soll, wird im Skript-Code `action.recallMarkedAction()` aufgerufen.

Das Recall-Skript wird dann mit der gleichen Umgebung ausgeführt (View, Aktion, Parameter), die vorhanden war, als die Aktion erstmals ausgeführt wurde. Über `action.recallingAction()` können Sie innerhalb des Recall-Skripts auf die Aktion zugreifen, von der aus der Recall ausgelöst wurde. Die notwendige Umgebung zur Ausführung eines Recall-Skripts wird in der aktuellen Session gespeichert und wird daher beim Beenden der Session verworfen. Mit der Funktion `action.dropMarkedAction()` können Sie die Umgebung aus der Session entfernen, falls die gesamte Sequenz von Aktionen abgebrochen werden soll. Es ist häufig sinnvoll, Recall-Aktionen mit langlaufenden Transaktionen zu kombinieren.

3.2.2.4.2. Formular-Eingabewerte über Aktionen oder Skripte auslesen

Damit die Werte aus Formulareingaben weiterverarbeitet werden können, benötigt eine Aktion oder ein Skript eine Möglichkeit, diese zu adressieren. Eine Aktion kann entweder direkt über ein Menü an der Formulareingabe platziert werden oder an einer separaten Stelle, wobei die Aktion dann über eine Rollenzuweisung den Inhalt der jeweiligen Formulareingabe referenziert (*Ausführen durch*). Eine Rollenzuweisung zu einer Formulareingabe kann auch von einem Skript genutzt werden, um die zugehörige View zu ermitteln und dort den Eingabewert abzufragen.

HINWEIS	Bei der Verwendung einer Rolle muss darauf geachtet werden, dass der Bezeichner keine Leerzeichen enthält. Da eine View mehrere Rollen haben kann, werden die Rollen in leerzeichengetrennter Form verarbeitet. Ein Rollenbezeichner mit Leerzeichen würde als mehrere Rollen fehlinterpretiert werden und zu Fehlern führen.

Folgende Anwendungsfälle sind möglich:

Mit einer Aktion mehrere Formulareingaben unter einem gemeinsamen Layout auslesen, die Einträge aber einzeln verarbeiten

Verbinden Sie die Aktion über eine Layout-View mit der Rolle, lesen Sie den einzelnen Wert über eine individuelle Rolle an der jeweiligen Formulareingabe-View mit Adressierung über `this.viewsWithRole( roleName )[0].value()` aus.

Nur eine Formulareingabe auslesen

Die Aktion wird direkt über eine Rolle mit der Formulareingabe verbunden, mit Adressierung

über `this.value()`.

Alle Formulareingaben auf einmal auslesen (mehrere Werte gleichen Typs oder Reihenfolge der Werte spielt keine Rolle)

Verbinden Sie die Aktion über eine Layout-View mit der Rolle, lesen Sie die Werte über `this.viewsWithRole(roleName)` mit der Rolle `roleName` an allen Formulareingaben aus, mit Adressierung über `this.viewsWithRole(roleName)`, und verarbeiten Sie dann jedes Array-Element einzeln.

HINWEIS	Das Auslesen aller Formulareingaben durch gleichzeitige Zuweisung einer Rolle an alle Formulareingaben und an die Aktion mit Adressierung über `this.value()` funktioniert nicht, da Rollen im Web-Frontend eindeutig zuordenbar sein müssen.

Beispiel

Eine Layout-View enthält drei Formulareingaben: "Auswahl", "Boolean" und "Zeichenkette".

Wenn eine Aktion nur Zugriff auf eine bestimmte Formulareingabe benötigt (z. B. "Zeichenkette")

Die Zeichenketten-View erhält eine Rolle mit dem Konfigurationsnamen `input` und wird über die *Ausführen-durch*-Relation mit der Aktion verbunden. Die Aktion ist vom Typ Skript und enthält ein *Skript (benutzerdefiniert)*. In jedem Fall ist `this` das Element, das über die Rolle mit der Aktion verbunden ist. Der Wert des Eingabefelds wird dann über `this.value()` ausgelesen.

Wenn alle drei Formulareingaben gleichzeitig über eine einzige Aktion mit individueller Verarbeitung ausgelesen werden sollen

Die Aktion benötigt eine Verbindung zur Layout-View über eine dort platzierte Rolle (z. B. `form`), und die einzelnen Formulareingaben erhalten jeweils eine eigene Rolle. Zur Adressierung der Eingabe wird die Aktion in diesem Fall über die *Ausführen-durch*-Relation mit der Rolle `form` verbunden. Die Aktion hat den Aktionstyp *Skript (benutzerdefiniert)*. Diesmal ist `this` die Formular-View. Um im Skript auf das Eingabefeld zuzugreifen, müssen Sie die View mit der Rolle `input` adressieren. Der Wert des Eingabefelds wird dann über `this.viewsWithRole('input')[0].value()` ausgelesen. Da `viewsWithRole` ein Array zurückgibt, ist die einzige vorhandene Eingabe-View das erste (und einzige) Element des Arrays mit dem Index 0.

3.2.2.4.3. Transaktionen und Aktionssequenzen

Aktionen, die den Knowledge-Graph modifizieren, führen automatisch eine Transaktion aus, die eine konsistente Alles-oder-Nichts-Modifikation sicherstellt. Es gibt jedoch Situationen, in denen Änderungen, die Sie am Knowledge-Graph vornehmen, in eine Sequenz aufeinanderfolgender Aktionen aufgeteilt werden. Dies ist insbesondere der Fall, wenn Benutzerinteraktion notwendig ist, um weitere Aktionsparametrisierung zu bestimmen oder um den bisherigen Prozess abzubrechen.

Beispiel: Ein neues Objekt muss innerhalb eines Dialogs erstellt werden. Um dem Benutzer zu ermöglichen, die Erstellung des neuen Objekts durch Drücken eines Abbrechen-Buttons des Dialogs abzubrechen, startet die Dialog-aufrufende Aktion eine Transaktion. Der Abbrechen-Button bricht die Transaktion ab (Aktionstyp: `Abbrechen`) und ein Speichern-Button committet die Transaktion. Um eine Sequenz von Aktionen in eine Transaktion zu kapseln, markieren Sie die erste Aktion mit "Transaktion — **beginnen**" und die finale Aktion mit "Transaktion — **beenden**".

WARNUNG

Risiko von Datenverlust durch nie endende Transaktion

Wenn Aktionen nur mit "Transaktion — beginnen" konfiguriert sind, wird eine einzige nie endende Transaktion erstellt. Eine nie endende Transaktion hat das Potenzial, kontinuierlich zu wachsen, sobald sie gestartet wurde, und erfasst alle Aktionen seit dem Beginn der Transaktion. Danach wird das System zuerst langsamer und bricht dann zusammen. Zusätzlich werden Änderungen nie gespeichert, da das Speichern nur am Ende einer Transaktion ausgelöst wird, um die Konsistenz aufrechtzuerhalten. Um diese Effekte zu vermeiden, stellen Sie sicher, dass Sie ein "**Transaktion — beenden**" setzen, sobald die Aktionssequenz vollständig genug ist, um den erforderlichen Zustand zu erreichen.

Setzen Sie eine Aktion nur auf "Transaktion — beginnen", wenn Sie auch eine nachfolgende Aktion auf "**Transaktion — beenden**" setzen.

WARNUNG

Risiko des Verlusts der Datenintegrität bei wiederholter Transaktionsausführung

Transaktionen dürfen nicht auf Elementen mit variabler Reihenfolge ausgeführt werden.

Die Transaktionshistorie einer Transaktion zeichnet auf, welche Aktion auf welchem semantischen Element einer bestimmten Reihenfolge ausgeführt wird. Bei der Ausführung einer weiteren Aktion innerhalb einer Transaktion wird die Transaktionshistorie mit ihrer festen Reihenfolge von Aktionen wiederholt und die neue Aktion wird dieser Historie hinzugefügt.

Wenn sich die Reihenfolge semantischer Elemente beim Wiederholen der Transaktion ändert (z.B. durch Erstellen von Elementen mittels eines Skripts oder durch Bestimmen von Elementen durch Ausführung einer Abfrage), führt dies zu einer Fehlausrichtung der Reihenfolge von Aktionen zu ihrem jeweiligen Element einer bestimmten Reihenfolge. Dies hat zur Folge, dass Aktionen auf falschen Elementen verarbeitet werden.

Stellen Sie daher bei der Verarbeitung semantischer Elemente in einer Transaktionssequenz sicher, dass die Reihenfolge der Elemente deterministisch bleibt. Um eine deterministische Reihenfolge beizubehalten, müssen Sie die Elemente nach einer festen Eigenschaft sortieren.

> **Beispiel:** Eine Aktion **A** führt eine Abfrage für Mahlzeiten aus. Das Suchergebnis ist "Pudding" und "Fisch". Die Aktion erstellt zusätzlich zwei Bewertungsobjekte und verknüpft die Bewertungen mit "Pudding" und "Fisch". Sie werden befähigt, einen Kommentar (Attribut) zu jeder Bewertung zu schreiben. Die nächste Aktion **B** speichert die Kommentare bei den Bewertungsobjekten und beendet die Transaktion. Die Ausführung der Aktion erfolgt wie folgt: **A**, **A** + **B**. Aktion **A** wird daher zweimal ausgeführt.
>
> Ein wichtiger Aspekt ist die deterministische Reihenfolge der gefundenen Mahlzeiten: Da das Ergebnis einer Abfrage keine spezifische Reihenfolge hat, geschieht die Zuordnung des ersten Kommentars zu "Pudding" oder zu "Fisch" zufällig. Daher müssen Sie das Abfrageergebnis zuerst sortieren, um eine korrekte Zuordnung des ersten und zweiten Bewertungsobjekts zur relevanten Mahlzeit sicherzustellen. Nur dies macht es möglich zu verhindern, dass der Pudding die Bewertung "Sehr zart, aber zu viele Fischgräten" erhält.

Der Transaktions-Commit kann auch dynamisch über die Skriptfunktion `setTransactionCommit()` herbeigeführt werden.

Wenn Sie die Transaktion abbrechen möchten, können Sie dies durch eine Aktion vom Typ `Abbrechen` erreichen. Abbrechen bedeutet, dass alle vorherigen Änderungen am Knowledge-Graph, die innerhalb der Transaktion durchgeführt wurden, rückgängig gemacht werden. Mit der Skriptfunktion `setFailed()` können Sie dynamisch einen Abbruch einleiten. Da eine Transaktion immer mit der Dauer einer Session gekoppelt ist (siehe unten), wird eine Transaktion automatisch abgebrochen, wenn die Session endet, in der die Transaktion gestartet wurde.

Wenn Sie beispielsweise einen Dialog zu Beginn der Transaktion öffnen und dieser Dialog geschlossen wird, bevor die Transaktion abgeschlossen wurde, wird die Transaktion automatisch abgebrochen. Dies gilt nicht für Dialoge, die geöffnet werden, während eine Transaktion bereits läuft: Das Öffnen eines Dialogs erstellt eine neue Session auf dem Session-Stack, die unabhängig von der aktuell laufenden Transaktion ist. Dialogsequenzen (ein Dialog wird geschlossen und ein anderer Dialog wird sofort geöffnet) unterbrechen die Transaktion ebenfalls nicht.

HINWEIS	Nur eine Transaktion kann gleichzeitig verarbeitet werden. Eine Transaktion innerhalb einer anderen Transaktion wird nicht unterstützt.

3.2.2.4.4. Sessions

Im Sinne der View-Konfiguration dient eine *Session* als temporärer Speicher für Variablenwerte, die innerhalb von Skripten der View-Konfiguration gelesen und geschrieben werden können. Dies wiederum ermöglicht die Darstellung des aktuellen Zustands einer Anwendung.

Sessions sind Laufzeitobjekte, die während der Ausführung einer i-views Web-Anwendung instanziiert werden. Sessions bilden einen Stack. Die erste Session dauert die gesamte Dauer der Web-Session. Das heißt von der Zeit, zu der Sie die Anwendung aufrufen, bis das jeweilige Browserfenster geschlossen wird. Sie können die erste Session jederzeit durch Verwendung der

Funktion `$k.Session.main()` aufrufen.

Das Öffnen eines Dialogs erzeugt eine neue Session auf dem Stack. Das Schließen des Dialogs entfernt die entsprechende Session wieder vom Stack.

Die Aktivierung von Panels, die mit einer *Session-Grenze* markiert sind, erzeugt ebenfalls eine neue Session auf dem Stack, die anhält, bis das Panel deaktiviert wird. Das Element der neuen Session wird auf das aktuelle Element des Panels gesetzt und kann zukünftig durch Verwendung der Funktion `element()` auf dieser Session verwendet werden. Verwenden Sie die Funktion `$k.Session.actual()`, um auf die oberste Session des Stacks zuzugreifen.

Werte schreiben Sie in eine Sessions-Variable mittels `$k.Session.actual().setVariable()` und lesen Sie aus einer Sessions-Variable mittels `$k.Session.actual().getVariable()`.

3.2.3. Bookmarking und Historie

Bookmarking ermöglicht:

- Direktsprünge zu Inhalten des Web-Frontends per URL (inklusive einer Vorbelegung des Anwendungszustands wie z.B. Suchparameter).
- Verwendung der Browser-Historie zum Vor- und Zurücknavigieren durch das Web-Frontend.
- Suchmaschinenoptimierung

Da das Web-Frontend eine Single-Page-Webanwendung ist, ändert sich die URL beim Navigieren durch die Anwendung normalerweise nicht. Mit Bookmarks aber kann definiert werden, welche konkrete URL für den aktuell angezeigten Inhalt ausgebildet wird. Das wiederum ermöglicht dem Nutzer den direkten Sprung in einen spezifischen Anwendungszustand. Weiterhin erhöht es die automatische Indexierbarkeit der Anwendung durch eine Web-Suchmaschine.

Da eine Web-Anwendung beliebig strukturiert sein kann, muss das Bookmarking auch auf die konkrete Navigationsstruktur der Anwendung konfiguriert werden. Dabei muss die Konfiguration in zwei Richtungen funktionieren:

- Aus einer gegebenen Anwendungssituation bestehend aus aktuell sichtbaren (= aktiven) Panels und deren Inhalten muss das System jederzeit ein Bookmark bestehend aus Pfad und Parametern bilden können. Die resultierende URL wird dann in der Adresszeile des Browser angezeigt.
- Ausgehend von einer gegebenen URL muss das System ggf. eine Anwendungssituation herstellen können. Dabei müssen ausreichend Informationen zu den angezeigten Inhalten in der URL enthalten sein, sonst kann eine Anwendundungssituation nicht vollständig (wieder) hergestellt werden.

3.2.3.1. Bookmark Ressource

Die Definition von Bookmarks beginnt mit der *Bookmark-Ressource*. Die Aufgabe dieser Ressource ist die Auflösung einer gegebenen URL, die entweder auf einen konfigurierten Bookmark passt oder

- falls das nicht der Fall ist - auf einen anderen konfigurierten REST-Pfad der VCM-Schnittstelle hinweist. Eine weitere Aufgabe der Ressource ist die Steuerung der initialen Auslieferung der Web-Anwendung.

Die *Bookmark-Ressource* kann in ***TECHNIK > REST*** an einem REST-Service angelegt werden. REST-Services, die mit der Vorlage *viewconfig (View-Configuration-Mapper)* angelegt werden, haben automatisch diese Ressource.

HINWEIS	Es ist darauf zu achten, dass die *Bookmark-Ressource* als *Authentication* ein Objekt mit dem *Authentication type* `No authentication` erhält. Sie löst HTTP-Redirects aus, die auch beim initialen Laden des Web-Frontends, und damit vor einem Login, funktionieren müssen.

Konfigurationseigenschaften

Authentication

Muss vom Typ `No Authentication` sein.

Static Path

Verweist auf den statischen Pfad zur Web-Anwendung (default = viewconfigmapper)

Path pattern

Veraltet, wird nicht mehr verwendet.

Initial File

Verweist auf das initial auszuliefernde File der Web-Anwendung (default = index.html)

Has fallback path

Siehe Bookmark-Fallback

3.2.3.2. Bookmark-Pfad

Die *Bookmark-Ressource* erlaubt die Definition einer beliebigen Anzahl von Unterelementen des Typs *Bookmark-Pfad*. Die Aufgabe von Bookmark-Pfaden ist die eindeutige Zuordnung anzuzeigender Panels zu einer URL und umgekehrt. Jeder *Bookmark-Pfad* hat die Eigenschaft *Path pattern*, die zur Bildung bzw. Identifikation der URL verwendet wird. Ein *Path pattern* ist ein Adressmuster, das in der Anwendung verwendet werden kann. *Path pattern* müssen eindeutig sein. Eine konkrete URL muss also immer zweifelsfrei genau einem *Path pattern* zuzuordnen sein. Weiterhin darf es nicht zu Überschneidungen mit anderen Ressourcen im gleichen REST-Service kommen. Beispiele hierfür sind die Ressourcen des initial angelegten REST-Service *viewconfig*, die Pfade wie `perform/cancel` oder `accessToken/login` haben. Auch die Namen von statischen Ressourcen des Web-Frontends dürfen nicht verwendet werden. Ein Beispiel hierfür wäre der Pfad `viewconfigmapper/index.html`.

Ein *Path pattern* besteht aus statischen und variablen Anteilen. Dynamische Anteile sind in geschweifte Klammern gefasst, z.B. `help/{topic}` oder `performance/{company}/{year}`. Für die variablen Anteile müssen Bookmark-Parameter konfiguriert werden.

Konfigurationseigenschaften

Path pattern

Das Adressmuster, das dieser Bookmark-Pfad repräsentiert.

Panel

Das Panel, das beim Aufrufen des Bookmarks aktiviert werden soll. Sollen mehrere Panels aktiviert werden, muss das über die Beeinflussungskette realisiert werden.

Has fallback path

Siehe Bookmark-Fallback

3.2.3.3. Bookmark-Parameter

Über Pfad- und *Bookmark-Parameter* wird der Inhalt (= domain model) des zugehörigen Panels adressiert. Ein *Bookmark-Parameter* repräsentiert oft ein Element des Wissensnetzes und wird - wenn nicht abweichend konfiguriert - bei der Bildung einer Adresse als ID des Elements dargestellt (z.B. ID1527_373749). Zeigt ein Panel zum Beispiel Informationen zu einem technischen Bauteil an, dann repräsentiert ein Parameter dieses Bauteil. Ist der Inhalt des Panels nicht durch ein Element sondern beispielsweise durch eine Query bestimmt, dann sollten die Parameter des Bookmarks, auf die der Query abgebildet werden, um beim Aufruf des Bookmarks die gleiche Trefferliste zu produzieren.

Durch Definition eines *Parameter conversion*-Skripts kann eine Umrechnung der Parameter definiert werden. Dadurch lässt sich Folgendes erreichen:

- Elemente sprechender darzustellen
- Verwendung externer IDs (z.B. Teilenummer) für die Adressierung durch Fremdsysteme
- Verwendung stabiler Identifikatoren, die über die Lebensdauer der internen ID hinweg ihre Gültigkeit behalten

Ein häufiger Anwendungsfall ist die Anzeige eines Objektnamens anstatt der Objekt-ID. Das produziert zum einen lesbare URLs, erzeugt aber ggf. Probleme bei der Auflösung des Namens bei mehreren gleichnamigen Objekten.

Konfigurationseigenschaften

Parameter name

Der Name des Parameters, wie er im Bookmark-Pfad steht

Parameter conversion

Siehe weiter unten

Panel

Das Panel, dessen Inhalt über die Parametrierung ermittelt wird bzw. dessen Inhalt die Parametrierung definiert.

3.2.3.4. Zusammengesetzter Parameter

Ein *Zusammengesetzter Parameter* ist eine Variante des *Bookmark-Parameters*, die die Adressierung des Panel-Inhalts über einen strukturierten Pfad ermöglicht, z.B. `/{mainChapter}{subChapter}/{version}`. Für jeden Teil-Parameter eines *Zusammengesetzten Parameters* muss ein entsprechendes *Bookmark-Parameter*-Objekt unterhalb des *Zusammengesetzten Parameters* konfiguriert werden. Außerdem wird ein Skript für *Parameter conversion* benötigt, das die jeweiligen Teil-Parameter behandelt und daraus den Inhalt berechnet bzw. bei gegebenem Inhalt die Teilparameter berechnet.

HINWEIS	Im Konvertierungs-Skript kann auf Session-Variablen zugegriffen werden. Das ermöglich die Ansteuerung eines Anwendungszustands, der sich nicht alleine aus den angezeigten Inhalten ergibt.

Konfigurationseigenschaften

Parameter name

Der Name des Parameters spielt bei zusammengesetzten Parametern im Prinzip keine Rolle. Für die bessere Lesbarkeit empfiehlt sich allerdings die Konkatenation der Sub-Parameter.

Parameter converserion

Siehe weiter unten

Panel

Panels, die das initiale Element für den Parameter liefern bzw. ein über *Parameter conversion* bestimmtes Element als Modell erhalten

3.2.3.5. Skript *Parameter conversion*

Das Skript für Parameter-Konvertierung beinhaltet zwei Funktionen:

- `identifier(optionalElement)`: Das Panel wird als Bestandteil der Bookmark-URL wiedergegeben. Diese Funktion bestimmt, wie das im Panel dargestellte semantische Element in einen Textidentifikator für die URL umgewandelt wird (Verarbeitung des Bookmark-Output).
- `element(parameterValue)`: Diese Funktion bestimmt, wie eine eingegebene URL interpretiert wird, um damit das im Panel anzuzeigende semantische Element zu ermitteln (Verarbeitung des Bookmark-Input).

In diesem Beispiel wird auf die Variable (z. B. `optionalElement.name()`) in der Funktion `identifier()` zugegriffen, in Kombination mit einer Zuweisung der Variable (z. B. `$k.Registry.elementAtValue('name', parameterValue)`) in der Funktion `element()`:

```
/**
 * Returns an (element-) identifier for the parameter
 * @function
 * @param {$k.SemanticElement} optionalElement The element for which the
```

```
identifier shall be returned (optional)
 * @returns {string}
**/
function identifier(optionalElement) {
  if (optionalElement)
    return optionalElement.name()
  else
    return undefined
}
/**
 * Returns an element for the given parameter value
 * @function
 * @param {string} parameterValue The parameter value
 * @returns {$k.SemanticElement}
**/
function element(parameterValue) {
  return $k.Registry.elementAtValue('name', parameterValue)
}
```

3.2.3.6. Verknüpfung mit Panels

Path patterns, wie im vorangegangenen Kapitel erklärt, können jeweils mit einem Panel verknüpft werden. Damit wird ausgedrückt, dass das Pattern zur Adressbildung verwendet wird, sobald das Panel aktiv und damit sichtbar ist.

HINWEIS	Beim Design der Anwendung ist darauf zu achten, dass es zu keiner Zeit mehr als ein aktives Panel mit *Path pattern* gibt, weil das Web-Frontend sonst nicht entscheiden kann, welches Adressmuster verwendet werden soll. Zum Aktivieren mehrerer Panels kann die Beeinflussungskette verwendet werden.

Das Modell, das im aktiven Panel sichtbar ist, wird verwendet, um die Parameter des *Path pattern* zu bilden. Es ist darauf zu achten, dass das aktive Panel dieses Modell beim Vorgang der Aktivierung gesetzt bekommt, sodass es die entsprechenden Parameter für das Panel ermittelt werden können.

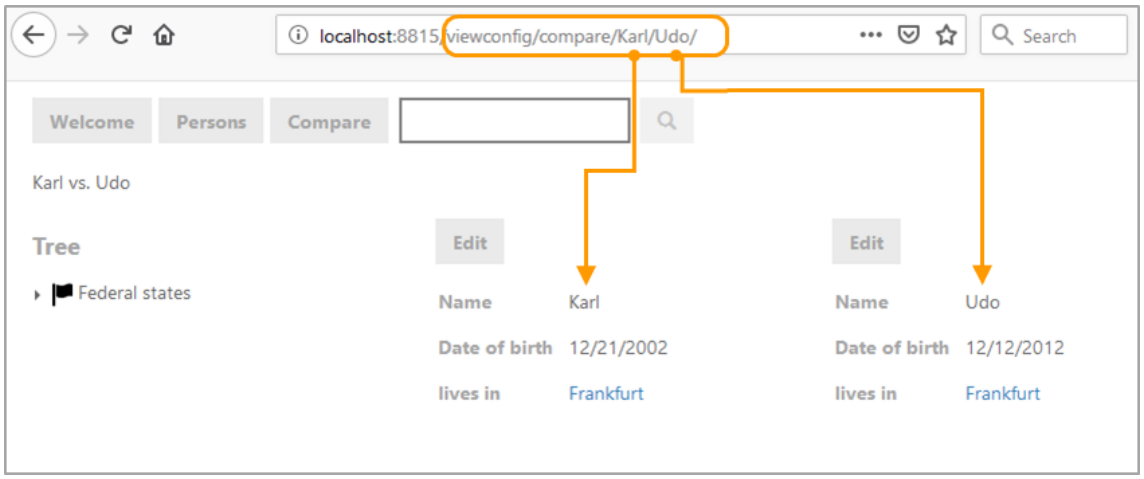
Es gibt Fälle, in denen eine Anwendungssituation nicht durch ein einziges Panel und dessen Inhalt bestimmt wird. Beispielsweise enthält eine Vergleichansicht zwei Panels mit unterschiedlichen Modellen nebeneinander. Um dies zu realisieren können zusätzlich zum Panel des Bookmark-Pfades weitere Panel über weitere Parameter dieses Pfades verknüpft werden. Auf diese Weise kann man bspw. eine Vergleichsansicht zweier Produkte adressieren (compare/product_A/product_B).

Beim Aufruf eines Bookmark-Links (Eingabe in die Adresszeile des Browsers) wird die Konfiguration im Prinzip in umgekehrter Richtung verwendet:

1. Das passende *Path pattern* wird ermittelt.
2. Das zugehörige Panel wird aktiviert und ggf. mit der im Parameter definierten Vorschrift mit

dem Element zur Anzeige versehen. Die Anzeige des Elements selbst wird durch die Parameter-Regeln definiert.

3. Panels, die über Parameter verknüpft sind, werden ebenfalls aktiviert und ggf. mit der im Parameter definierten Vorschrift mit einem Element zur Anzeige versehen.
4. Die Aktivierungs-Ketten werden ausgeführt und die Anwendung ist im gewünschten Zustand sichtbar.



Auch Dialoge sind über den beschriebenen Mechanismus adressierbar. Bei der Definition von *Path pattern* für Dialoge ist allerdings darauf zu achten, dass im Aufruf des Bookmark-Links auch definiert ist, welche Inhalte (Panels) innerhalb des Dialogs zur Anzeige kommen sollen. Das lässt sich durch Verknüpfung eines Parameters mit einem Panel des Hauptfensters bewirken.

3.2.3.7. Bookmark-Fallback

Unter Umständen ist ein Bookmark, das sich ein Anwender gespeichert hat, veraltet und kann nicht mehr aufgerufen werden. Zum Beispiel können sich die Daten oder der Aufbau der Anwendung geändert haben. Es kann auch sein, dass das Bookmark eines anderen Anwenders mit abweichenden Zugriffsrechten aufgerufen wird. Damit es in diesen Fällen nicht zu unerwünschten Fehlermeldungen kommt, kann für Bookmarks ein Fallback-Verhalten konfiguriert werden.

Sowohl die allgemeine Bookmark-Ressource, als auch die einzelnen Bookmark-Pfad-Objekte, können über der Relation *Has fallback path* mit einem anderen Bookmark-Pfad-Objekt verknüpft werden. Kommt es beim initialen Aufruf eines Bookmarks zu einem Fehler, wird die Anfrage stattdessen an den Fallback-Pfad umgeleitet. Dieser könnte zum Beispiel auf die Startseite oder eine allgemeine Infoseite zum Umgang mit Fehlern leiten.

Für komplexere Szenarien können mehrere Fallback-Pfade konfiguriert werden. Diese benötigen jeweils die Meta-Eigenschaft *Script for bookmark fallback*. Darüber wird der Pfade ermittelt, zu dem die Anfrage umgeleitet werden soll. Das erste Skript, das `true` zurückliefert, bestimmt den Zielpfad. Über den Parameter `originalBookmark` der Skriptfunktion `bookmarkFallback()` können Daten zum Kontext abgefragt werden, so zum Beispiel der original angefragte Pfad, sowie die Belegung sämtlicher Bookmark-Parameter.

3.3. Frontend-spezifische View-Konfiguration

Einige Views haben Konfigurationseigenschaften, die nur im Web-Frontend Auswirkungen haben. Im folgenden werden nur diese frontend-spezifischen Eigenschaften beschrieben. Die detaillierte Beschreibung aller Views findet sich hier

3.3.1. Alternative

Alternativen werden hier beschrieben.

Konfigurationseigenschaften

Zuletzt gewählte Alternative wiederherstellen

Beim Setzen dieses Wertes merkt sich das Web-Frontend den zuletzt aktivierten Reiter der Alternative.

3.3.2. Facetten-Ansicht

Facetten bieten eine strukturierte Filterung für Suchergebnisse, wie man sie z.B. von Verkaufsplattformen kennt. Dort werden neben dem Suchergebnis häufig Kategorien wie "Produktart", "Marke" etc. angezeigt. Innerhalb der Kategorien gibt es entsprechende Unterkategorien wie z.B. "Laufschuhe", "Marke-XY" mit einer Angabe der Anzahl der Produkte in dieser Unterkategorie. Durch Klick auf eine Unterkategorie werden nur noch Suchergebnisse der gewählten Art angezeigt. Es ist auch möglich, mehrere Unterkategorien anzuwählen, um z.B. alle Laufschuhe einer bestimmten Marke zu sehen.

Die Kategorien entsprechen in diesem Fall einer Facette, die Unterkategorien entsprechen den zur Facette gehörenden Termen.

Facetten können hierarchisch sein. Dann sind ihre Terme selber Facetten und bilden auch Terme aus.

Konfigurationseigenschaften

Abfrage zur Termermittlung

Suche zur Ermittlung der Terme der Facette. Die Eigenschaft, die als Term zurückgeliefert werden soll, muss mit dem Bezeichner `term` versehen werden. Es ist auch möglich, dass der Bezeichner `term` mehrfach in einer Strukturabfrage verwendet wird. Dann bestimmt der bei *Term-Operator* definierten Wert, ob alle oder nur manche der Termbedingungen zutreffen müssen.

Abfrage zur Elternermittlung

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` muss am Eltern-Element gesetzt werden.

HINWEIS	• Damit eine Facettenhierarchie aufgebaut werden kann, muss die *Abfrage zur Termermittlung* außer den anzuzeigenden Termen auch die

Elternterme enthalten. Diese Elternterme werden dann nachgelagert durch die *Abfrage zur Elterntermermittlung* zur Hierarchiebildung herangezogen.

- Derzeit können nur Terme des gleichen Typs eine Hierarchie bilden.

Ausblenden ab Anzahl von Termen

Wenn eine Facette mehr Terme hat, als hier definiert ist, werden die überzähligen Terme im Web-Frontend ausgeblendet. Ein Hinweistext und die Möglichkeit zum Einblenden der fehlenden Terme werden angezeigt. Der Standardwert ist 10.

Kindterme initial anzeigen

Falls die Facette hierarchisch aufgebaut ist, kann über diese Option definiert werden, ob die Unter-Facetten initial angezeigt werden sollen. Standardmäßig werden bei einer Hierarchie die Kindelemente erst angezeigt, wenn das dazugehörige Elternelement ausgewählt wurde.

Leere Facetten ausblenden

Falls zu einer Facette keine Terme gefunden werden, wird sie standardmäßig trotzdem angezeigt. Mit dieser Option wird sie ausgeblendet.

Leere Terme anzeigen

Terme, zu denen keine Suchergebnisse gefunden werden, werden standardmäßig ausgeblendet. Mit dieser Option werden sie trotzdem angezeigt.

Maximale Anzahl an Termen

Wie *Ausblenden ab Anzahl von Termen* mit dem Unterschied, dass es keinen Hinweis und keine Möglichkeit zum Einblenden der überzähligen Terme gibt.

Term-Anzahl nicht anzeigen

Die Anzahl der Suchtreffer für einen Term wird im Web-Frontend ausgeblendet.

Term-Operator

Bestimmt die Art, wie die mit `term` markierten Bedingungen in der *Abfrage zur Termermittlung* verknüpft werden:

- *und*: Alle Termbedingungen müssen zutreffen.
- *oder*: Mindestens eine Termbedingung muss zuzutreffen.

Termart

Wenn keine Termart ausgewählt wird (Standardverhalten), werden die Terme anhand der Abfrage an der Facetten-Konfiguration ermittelt. Zusätzlich zum Standardverhalten stehen folgende Einstellungsmöglichkeiten zur Verfügung:

- Dynamisch: Die Wertebereiche der Terme werden automatisch ermittelt. Die Werte, die zur Termbildung verwendet werden, müssen in der *Abfrage zur Termermittlung* mit dem vordefinierten Parameter `termValue` versehen sein.

- Statisch: Terme müssen einzeln angelegt und konfiguriert werden. Es muss jeweils eine Suche konfiguriert werden, die die Treffermenge für den Term bestimmt.

Terme absteigend 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 Facetten-Terme alphabetisch nach dem Namen sortiert. Durch das setzen dieses Flags werden die Terme nach der Anzahl ihres Vorkommens sortiert.

Skript für initiale Terme

Legt die initial für eine Filterung ausgewählten Terme fest.

3.3.3. Formulareingabe

Formulareingabe-Views dienen zur Erfassung von Inhalten, die unabhängig von der Existenz eines semantischen Elements sind. Die in den Formular-Feldern erfassten Inhalte können mittels Aktion per Skript ausgelesen und anschließend weiterverarbeitet werden. Sie können auch als Eingabe eines Suchverbunds genutzt werden, um eine Suche mit Parametern zu versorgen. Es gibt Formulareingabefelder für folgende Typen: *Boolean*, *Datum und Uhrzeit*, *Zahl*, *Auswahl*, *Zeichenkette*, *Eingabe mit Vorschlägen*

Das Auslesen der Feldinhalte wird im Bereich über Aktionen beschrieben.

HINWEIS	Eine Aktion mit der Aktionsart *Speichern* hat keinen direkten Einfluss auf Formulare und führt nicht zum Persistieren der Werte.

Konfigurationseigenschaften

Annehmen-Aktion

Eine Aktion, die nach Änderung des Eingabewertes ausgeführt wird. Kann als Ersatz oder Ergänzung zu einer Aktion, die über Rollen auf die Formularwerte zugreift, verwendet werden.

Benötigt

Gibt an, ob eine Eingabe zwingend erfolderlich ist.

Initialisierungsskript

Hier kann mit `this.setValue()` ein Initialwert gesetzt werden.

Skript für Auswahl

Durch Rückgabe eines Arrays von Zeichenketten oder semantischen Elementen aus der Funktion `choices()` werden die möglichen Auswahlwerte festgelegt. Die Rückgabe eines entsprechenden Wertes aus der Funktion `initialChoice()` legt den Initialwert fest.

Skript für Vorschlagswerte

Details weiter unten

Skript zur Validierung

Details weiter unten

Suche nach Vorschlagswerten

Details Formulareingabe##input-with-proposals[weiter unten]

3.3.3.1. Formulare als Eingabe für Suchverbünde

Formulare können als Parameter-Eingabe für eine Suche genutzt werden. Dafür muss eine Formulareingabe mit der *Eingabe von*-Relation mit einem Suchverbund verknüpft werden. Zudem muss der *Parametername* konfiguriert werden. Dieser legt fest, welcher Parameter der Suche durch die Formulareingabe bestückt wird. Wenn die Formulareingabe als "Benötigt" markiert ist und keine Nutzereingabe vorliegt, wird die zugehörige Suche nicht ausgeführt. Andernfalls wird bei leerer Eingabe der zugehörige Suchparameter deaktiviert. Für weitere Details siehe Kapitel Suchverbünde.

3.3.3.2. Eingabevalidierung

Formulareingaben können als *Benötigt* markiert werden. In diesem Fall wird das Eingabefeld mit einer Markierung versehen, die dem Nutzer anzeigt, dass eine Eingabe zwingend erforderlich ist. Dies wirkt sich auch darauf aus, wie eine fehlende Parametereingabe behandelt wird, wenn die Formulareingabe Teil eines Suchverbunds ist (siehe oben).

Der zweite Validierungsmechanismus ist das *Skript zur Validierung*, welches für alle Arten von Formulareingaben konfiguriert werden kann. Das folgende Beispiel zeigt ein Skript zur Validierung der Eingabe eines Zahl-Felds:

```
function validateFormValue (value) {
  if (value < 0 || value > 10) {
    this.setValidationErrorMessage('Only values between 0 and 10 are
allowed.')
    return false
  }
  return true
}
```

Die Validierungsfehlermeldung wird dem Nutzer in der Bedienoberfläche angezeigt, damit die Eingabe korrigiert werden kann. Wenn das Validierungsskript `false` zurückgibt, ist der invalide Wert nicht für Skripte und Suchen zugreifbar.

Es ist außerdem möglich, zur Validierung die Werte anderer Formularfelder abzufragen, indem eine der oben beschriebenen Referenzierungsmöglichkeiten genutzt wird. Dabei muss jedoch beachtet werden, dass keine zirkulären Abhängigkeiten zwischen mehreren Validierungsskripten entsteht. In so einem Fall ist die Wertabfrage nicht möglich und es wird stattdessen `undefined` als Wert geliefert.

3.3.3.3. Eingabe mit Vorschlägen

Für eine Eingabe mit Vorschlägen kann auf zwei Arten spezifiziert werden, wie sich Vorschlagswerte aus der Nutzereingabe ableiten. Wenn eine *Suche nach Vorschlagswerten* definiert ist, werden dem Anwender beim Ausfüllen des Feldes die Ergebnisse der Suche vorgeschlagen. Um Suchergebnisse in Abhängigkeit von der Nutzereingabe zu erhalten, kann der vordefinierte Suchparameter "searchString" genutzt werden.

Alternativ kann ein *Skript für Vorschlagswerte* konfiguriert werden:

```
function valueProposals (searchString) {
  return [
    // a function applied to the user input
    new $k.TypeAheadProposal(searchString.toUpperCase()),
    // a static number with label
    new $k.TypeAheadProposal(42, 'forty-two'),
    // a semantic element of the knowledge graph
    new $k.TypeAheadProposal($k.Registry.element('myElement'))
  ]
}
```

In beiden Fällen gibt es zwei zusätzliche Konfigurationsoptionen:

Anlaufschwelle

Definiert die Anzahl an Zeichen, die der Anwender tippen muss, bevor die erste Anfrage nach Vorschlagswerten abgesendet wird. Für teure Abfragen sollte dieser Wert entsprechend hoch gewählt werden, um negative Auswirkungen auf die Performanz zu minimieren. Der Standardwert für die Anlaufschwelle ist 3.

Eingabe auf Vorschläge beschränken

Wenn diese Checkbox aktiviert ist, ist der Anwender nicht in der Lage, Werte abzuschicken, die ihm nicht vorgeschlagen wurden. Andernfalls ist er frei darin, die Vorschlagswerte vor dem Abschicken zu editieren.

3.3.3.4. Eingabe für Dateiupload

Mit der *Eingabe für Dateiupload* kann eine Datei hochgeladen werden. Wie bei allen Eingabefeldern kann der hinterlegte Wert dann per Skript abgefragt und weiterverarbeitet werden. Hochgeladene Dateien werden nicht automatisch persistiert. Sie werden nach dem Upload für eine gewisse Zeit (mindestens eine Stunde) aufbewahrt, bevor sie zur automatischen Speicherbereinigung freigegeben werden. Häufig ist daher das Speichern in einem Dateiattribut des Knowledge Graphs notwendig.

Referenz-Dateiattribut

Wenn eine Datei in einem Dateiattributen abgelegt werden soll, konfigurieren Sie an der *Eingabe für Dateiupload* das *Referenz-Dateiattribut*. Die hochgelade Datei wird dann direkt in

der am referenzierten Attributtyp konfigurierten Dateispeicherung hinterlegt.

3.3.4. Hierarchie

Hierarchien werden hier beschrieben.

Konfigurationseigenschaften

Ausgabe bis zur Tiefe

Gibt an, bis zu welcher Ebene die Hierarchie initial aufgeklappt ist.

Klick-Aktion

Aktion, die Beim Klick auf einen Hierarchieknoten ausgeführt wird

Initialisierungsskript

Skript für den Initialzustand der Hierarchie

Zuletzt ausgeklappten Knoten wiederherstellen

Beim Setzen dieses Wertes merkt sich das Web-Frontend den zuletzt aufgeklappten Hierarchieknoten.

3.3.4.1. Rendermodes

breadcrumbs

Zeigt die Hierarchie als Breadcrumb-Navigation an.

3.3.5. Layout

Layouts werden hier beschrieben.

3.3.5.1. Styles

style

Um die Größenverhältnisse der Subviews des *Layouts* zu beeinflussen, kann hier die CSS-Eigenschaft `flex` konfiguriert werden.
Mit beispielsweise `flex: 0 0 250px` wird ein Subview eines horizontalen *Layouts* auf eine Breite von `250px` gesetzt. Die restlichen Subviews passen dabei ihre Breite automatisch innerhalb des zur Verfügung stehenden Bereichs an.
Bei vertikalen Layouts kann auf diese Weise die Mindesthöhe von Subviews festgelegt werden.

collapsed

Gibt an, ob ein *Layout* mit Rendermode `card` inital zusammen- oder aufgeklappt ist. Wird diese Eigenschaft nicht konfiguriert, ist das *Layout* nicht klappbar.

3.3.5.2. Rendermodes

card

Das Layout erhält im Web-Frontend einen dezenten Rand mit abgerundeten Kanten. Karten können auf- und zugeklappt werden, wenn der *Style* `collapsed` gesetzt wird.

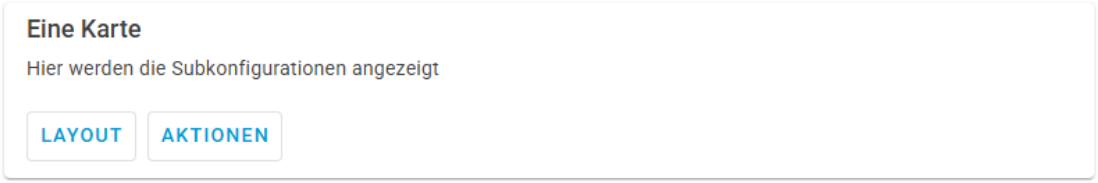


Figure 31. Layout mit Rendermode `card`

3.3.6. Eigenschaft

Eigenschaften werden hier beschrieben.

Konfigurationseigenschaften

Klick Aktion

Die Aktion, die beim Klick auf die Eigenschaft ausgeführt werden soll.

Tooltip

Hilfstext der Eigenschaft.

Platzhaltertext

Nur innerhalb einer *Edit*-View: Der Text wird angezeigt, wenn das Eingabefeld leer ist.

Skript für Platzhaltertext

Nur innerhalb einer *Edit*-View: Skript für die Bestimmung des Platzhaltertextes.

Skript für Tooltip

Skript für die Bestimmung des Tooltips.

Maskiert

Nur innerhalb einer *Edit*-View: Verhindert die gleichzeitige Eingabe mehrerer Eigenschaften. Sobald ein Wert in eine maskierte Eigenschaft eingegeben wird, werden alle anderen ausgewählten Eigenschaften deaktiviert.

3.3.6.1. Styles

Für Eigenschaften in einem Edit-Formular

readOnly

Deaktiviert das Eingabefeld

propertyValidation

Ein Skript, das ein Objekt für die Validierungskonfiguration zurückgibt. Folgende Eigenschaften können angegeben werden:

- `required`: Durch Angabe von `true` als Wert wird die Eigenschaft zum Pflichtwert.

> **HINWEIS** Die Interpretation als Pflichtfeld kann auch durch Setzen des Wertes 1 bei *minOccurrences* im *Schema* des zur Eigenschaft gehörenden Attributs bewirkt werden.

- pattern: Die Eingabe wird auf das Muster geprüft, das hier als Wert gesetzt ist. Das Muster kann eine Zeichenkette oder ein regulärer Ausdruck sein.

Für Eigenschaftstypen `Fließkommazahl`, `Ganzzahl` *und ihre Intervalle*

numberFormat

Hier kann ein Objekt vom Typ Intl.NumberFormatOptions hinterlegt werden

Für Eigenschaftstypen `Datum`, `Zeit`, `Datum und Uhrzeit` *und ihre Intervalle*

dateFormat

Hier kann ein Objekt vom Typ Intl.DateTimeFormatOptions hinterlegt werden

Für den Eigenschaftstyp `Datei`

extra

Typ: `object`
Zusätzliche Optionen für die Bilddarstellung.

imageAsDownload

Typ: `boolean`
Wenn `true`, wird ein Bild nur als Download angeboten, anstatt direkt angezeigt zu werden (wie alle anderen Dateitypen).

imageWithoutLink

Typ: `boolean`
Wenn `true`, wird ein Bild ohne anklickbaren Link dargestellt.

imageSize

Typ: `object`
Objekt zur Definition der Bildgröße. Unterstützt folgende Eigenschaften:

maxHeight

Typ: `string`
Maximale Höhe des Bildes (z. B. `"200px"` oder `"50%"`).

minHeight

Typ: `string`
Minimale Höhe des Bildes.

maxWidth

Typ: `string`
Maximale Breite des Bildes.

minWidth

Typ: `string`
Minimale Breite des Bildes.

width

Typ: `string`
Feste Breite des Bildes.

height

Typ: `string`
Feste Höhe des Bildes.

Beispiel für nicht klickbare Bilder in der Größe 64x64 Pixel

```
{
  imageWithoutLink: true,
  imageSize: {
    width: "64px",
    height: "64px"
  }
}
```

3.3.6.2. Rendermodes

Für den Eigenschaftstyp `Boolesch`

icon

Stellt den booleschen Wert als Icon dar

Für den Eigenschaftstyp `Zeichenkette`

email

Zeigt die Zeichenkette als Mailto-Link an.

html

Zeigt eine mit HTML ausgezeichnete Zeichenkette entsprechend formattiert im Web-Frontend an.

multiline

Zeigt die Zeichenkette mehrzeilig an.

pre

Zeigt die Zeichenkette mehrzeilig an und verwendet eine Monospace-Schriftart.

3.3.7. 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

Konfigurationsname

Der Konfigurationsname dient zur Identifizierung und Wiederverwendung einer Konfiguration.

Beschriftung

Eine Beschriftung findet nur Anwendung, wenn diese Konfiguration in einer anderen Konfiguration wie z.B. *Alternative* eingebettet ist.

Skript für Beschriftung

Mittels Skript kann die Beschriftung dynamisch ermittelt werden.

Skript für Sichtbarkeit

Skript, das einen Booleschen Wert zurückgibt, ob die View sichtbar sein soll oder nicht.

Automatisch speichern

Wenn aktiviert, werden Änderungen beim Verlassen eines Eingabefelds einer *Eigenschaft* -Konfiguration direkt gespeichert (Kein Speichern-Button notwendig).

HINWEIS

- Sollte nur für Attribute und wenige Eigenschaften verwendet werden!
- Zu erwartende Seiteneffekte sind:
  - Da durch das automatische Speichern intern eine Aktion ausgelöst wird, werden gleichzeitige Klicks auf Aktionsschaltflächen unter Umständen nicht ausgeführt. Das passiert, wenn sich während des Klicks auf eine Aktionsschaltfläche der Fokus in einem Eingabefeld befindet, das ein automatisches Speichern auslöst. Hierdurch würden zwei Aktionen in direkter Folge ausgelöst, was nicht möglich ist.
  - Eingabefelder für Intervalle mit Datums- und/oder Zeitwerten verhalten sich anders, als andere Intervalle. Bei ihnen wird nach jeder Teiländerung des Intervallwertes ein automatisches Speichern ausgelöst. Bei anderen Intervallen passiert das erst nach dem Verlassen des gesamten intervallfeldes.

3.3.8. Suche

Suchen im Web-Frontend haben eine Besonderheit in Bezug auf die Positionierung der zur View gehörenden Aktionen: Sie werden unterhalb der View angezeigt.

Zum Ausführen der Suche wird nicht unbedingt eine konfigurierte Such-Aktion benötigt. Im Web-Frontend werden Suchen auch durch drücken der `Eingabe`-Taste ausgelöst.

Die Qualität eines Suchtreffers kann als Wert in einer Spalte der Ergebnistabelle angezeigt werden. Die Standarddarstellung ist ein Prozentwert über einem Fortschrittsbalken. Mit einem *Style* am entsprechenden *Spaltenelement* lassen sich folgende CSS-Klassen setzen, um diese Darstellung zu beeinflussen:

`vc-property-value-quality–percent`

Zeigt nur den Prozentwert an.

`vc-property-value-quality–plain`

Zeigt einen Wert zwischen 0 und 1 an.

Konfigurationseigenschaften

Hits verwenden

Wenn diese Eigenschaft angehakt ist, liefert die Suche Hit-Objekte anstatt einfacher Topics zurück. Hit-Objekte sind vom Typ $k.Hit und enthalten Metainformationen über den Suchtreffer, wie z.B. Qualität. Detaillierte Informationen dazu finden Sie in der JS-API-Dokumentation.

Für Web-Frontends können an der Eigenschaft *Abfrage* als Meta-Eigenschaft Parameter definiert werden. Mit ihnen können Eingabefelder konfiguriert werden, über die Parameterwerte in die Suche gegeben werden. Alternativ kann eine Parameterübergabe auch mithilfe eines eines Suchverbundes in Kombination mit Formulareingaben realisiert werden.

Eigenschaften von Parametern

Parametername

Name des Parameters, wie er auch in der Suche verwendet wird.

Skript für Wertermittlung

Wenn bei *Wertermittlung* eine Ermittlungsart mit Skript ausgewählt wurde, kann hier das entsprechende Skript hinterlegt werden.

Skript für geparsten Wert

Ein Skript, das nach erfolgter Wertermittlung ausgeführt wird. In ihm kann der Wert vor Übergabe an die Suche verändert werden.

Wertermittlung

Es gibt folgende Auswahlmöglichkeiten: `Skript`, `Skript, überschreibbar durch Benutzereingabe` und `Benutzereingabe`. Bei `Skript` ist keine Benutzereingabe nötig, der Wert wird immer über das Skript berechnet. Bei `Skript, überschreibbar durch`

`Benutzereingabe` stellt das Skript einen Default-Wert bereit, der vom Benutzer überschrieben werden kann. Bei `Benutzereingabe` erfolgt keine Berechnung, der Wert wird vom Benutzer eingegeben oder bleibt leer.

Wertart

Die Auswahlmöglichkeiten sind `Obligatorisch` und `Optional`.

Typ

Hier kann über eine Auswahl der Wertetyp des Parameters gewählt werden (`xsd:integer`, `xsd:string` etc.).

Beschriftung

Die Beschriftung des Eingabefeldes im Web-Frontend. Wird hier nichts eingetragen, erhält das Feld keine Beschriftung.

Suche nach Vorschlagswerten

Wenn für den Parameter eine Autovervollständigung gewünscht ist, kann hier eine Suche hinterlegt werden, die die Vorschläge ermittelt. Die Meta-Eigenschaften *Anlaufschwelle* und *Eingabe auf Vorschläge beschränken* können bei dieser Eigenschaft hinzugefügt werden (siehe weiter unten).

Skript für Vorschlagwerte

Analog zu *Suche nach Vorschlagswerten* aber mit einem Skript für die Ermittlung der Vorschlagswerte. Die Meta-Eigenschaft *Anlaufschwelle* kann bei dieser Eigenschaft hinzugefügt werden (siehe weiter unten).

Zusätzliche Meta-Eigenschaften für Vorschlagswerte

Anlaufschwelle: Definiert, ab dem wievielten Buchstaben der Eingabe nach Vorschlagswerten gesucht werden soll. Standardmäßig ist der Wert 3.
Eingabe auf Vorschläge beschränken: Erzwingt die Auswahl eines der Vorgabewerte aus einer Liste. Eine Freitexteingabe ist nicht mehr möglich.

3.3.9. View-Verbund

Ein Verbund aus Views teilt sich den State, so dass alle beteiligten Views durch einen einfachen Mechanismus auf die gleichen Daten zugreifen können. Es gibt Spiegel- und Suchverbünde.

3.3.9.1. Spiegelverbund

Bei einem Spiegelverbund teilen sich Views einen State. Dadurch haben Aktionen in einer View auch Einfluss auf alle mit ihr gespiegelten Views. Wenn Sie zum Beispiel einen Knoten in einer Hierarchie auswählen, würde dieser Knoten auch in einer gespiegelten View aktiv.

Um Views miteinander zu verbinden, gehen Sie zum Reiter `Kontext` einer View und verknüpfen sie über `Spiegelung von` mit einem Spiegelverbund. Alle zu spiegelnden Views fügen Sie dann diesem Verbund hinzu.

3.3.9.2. Suchverbund

Ein Suchverbund verbindet mehrere Views miteinander über eine Suche. Bei den Views unterscheidet man zwischen Eingaben und Ausgaben. Ein einfacher Suchverbund besteht aus einer Suchdefinition und einer Tabellen-View, die als Ausgabe der Suche dient.

Für Eingaben, also Suchparameter, können Formulareingabefelder genutzt werden. Sie ersetzen die bislang genutzte Suchfeldansicht. Für jedes Eingabefeld muss im Suchverbund ein Parametername konfiguriert werden, damit dieser in der Suche verwendet wird.

Für Ausgaben können zusätzlich zu Tabellen auch noch andere Viewtypen wie z.B. die Graph-Ansicht genutzt werden.

Zur Filterung der Suchergebnisse kann eine Facetten-Ansicht konfiguriert werden.

Es ist möglich eine beliebige Anzahl von Eingaben, Ausgaben und Filtern zu konfigurieren, auch solche, die nicht immer in der Anwendung sichtbar sind.

Neue Suchverbünde können über den Kontextreiter jeder beteiligten View erzeugt werden.

3.3.10. Suchfeld-Ansicht

HINWEIS	Suchfeld-Ansichten sind zugunsten von Suchverbünden in Kombination mit Formulareingaben veraltet.

Mit Suchfeld-Ansichten können Eingabefelder für die Parameter einer Suche definiert werden.

Suchfeld-Ansichten können nur an *Panels* konfiguriert.

Die Konfigurationseigenschaften sind die gleichen, wie bei Paramtern von Suchen

3.3.11. Suchergebnis-Ansicht

HINWEIS	Suchergebnisansichten sind zugunsten von Suchverbünden in Kombination mit Formulareingaben veraltet.

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. Wenn es Parameter gibt, dann sollte die Suchergebnis-Ansicht mit einer Suchfeld-Ansicht verknüpft werden.

HINWEIS	Wenn Facetten das Suchergebnis beeinflussen sollen, muss sich die Suchergebnis-Ansicht direkt in einem eigenen Panel befindet (also nicht in einem Layout innerhalb eines Panels).

3.3.12. Skriptgenerierte View

Ein scriptgenerierte View ermöglicht die Definition eigener View-Komponenten. Mit einem Skript werden die Daten für diese View-Komponente erzeugt und an das Frontend weitergegeben. Dort muss eine speziell entwickelte Komponente existieren, die die Daten entgegennimmt und anzeigt.

Konfigurationseigenschaften

Skript

Das Skript, in dem die Daten der View-Komponente berechnet werden.

viewType

Frei wählbarer Bezeichner, über den im Frontend die konkrete Spezialkomponente identifiziert wird. Im Standard Web-Frontend sind Komponenten für die Bezeichner `chart` und `timeline` enthalten.

Nachfolgendes Skript ist ein Beispiel für die Datengenerierung für eine View-Komponente mit dem Bezeichner `timeline`:

```
/**
 * Customizes the view data
 * @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
}
```

3.3.13. Tabelle

Tabellen werden hier beschrieben.

Konfigurationseigenschaften

Anzahl Zeilen (Page size) `Standardwert: 20`

Die Maximale Anzahl an Zeilen, die von der Tabelle angezeigt werden. Die Gesamtmenge der anzuzeigenden Elemente wird auf mehrere Seiten mit dieser Anzahl an Zeilen aufgeteilt. Es wird eine Paginierung angezeigt, mit deren Hilfe durch die Seiten geblättert werden kann.

Klick-Aktion

Die Aktion, die beim Klick auf eine Tabellenzeile ausgeführt werden soll.

Label bei leerer Tabelle

Dieser Text wird als Ersatz für Beschriftung und Tabelle verwendet, wenn die Tabelle leer ist.

Ohne Spaltenfilter

Blendet die Zeile mit Spaltenfiltern aus.

Zuletzt gewählte Sortierung/Spaltenfilterung wiederherstellen

Sortierung und Spaltenfilter bleiben nach einem Wegnavigieren von der Tabelle bestehen.

3.3.13.1. Styles

cellValueSeparator

Über ein Skript an der Eigenschaft `extra` kann das Trennzeichen konfiguriert werden, das mehrere Einzelwerte in einer Tabellenzelle voneinander trennt. Standardmäßig ist das Trennzeichen ein Leerzeichen. Bei Verwendung des vordefinierten Bezeichners `NEWLINE` werden die Einzelwerte mit Zeilenumbruch dargestellt.

Beispiel für das Konfigurationsobjekt für `cellValueSeparator`

```
{
  table: {
    // Anzeige mehrerer Einzelwerte mit '/' als Trennzeichen:
    // E-Orgel/Keyboard/Klavier/Synthesizer
    cellValueSeparator: '/'
  }
}
```

pager

Über ein Skript an der Eigenschaft `extra` kann die Paginierung der Tabelle konfiguriert werden. Es muss ein Objekt mit der Eigenschaft `pager` zurückgegeben werden. Dieses Objekt kann die Eigenschaften `type` und/oder `pageSizes` haben.
Der Wert von `pageSizes` muss ein Array mit Zahlen sein. Diese werden im Web-Frontend als

mögliche Seitengrößen für die Paginierung zur Auswahl angezeigt.
Mit `type` kann der Typ der Paginierung konfiguriert werden. Es stehen `default`, `auto` und `more` zur Verfügung.

- `default` ist die Standardpaginierung mit Buttons zum Blättern durch die einzelnen Seiten.
- `auto` ist eine Paginierung, die automatisch beim Scrollen zusätzliche Elemente nachlädt.
- `more` zeigt einen Button an, bei dessen Aktivierung zusätzliche Elemente nachgeladen werden.

Beispiel für das Konfigurationsobjekt für `pager`

```
{
  pager: {
    type: 'auto'
    pageSizes: [5, 10, 20]
  }
}
```

3.3.13.2. Rendermodes

searchresults

Der Rendermode `searchresults` erlaubt das Anzeigen von Sucherergebnissen angelehnt an die Darstellung, wie man sie von Suchmaschinen kennt. Folgende Bereiche können per Spaltenkonfiguration befüllt werden: `left`, `heading`, `right`, `body`, `actions`. Die Spaltenkonfigurationen müssen die entsprechendnen Werte als *Beschriftung* gesetzt bekommen. Spalten mit anderen Werten für *Beschriftung* werden unterhalb von `body` angezeigt. Im Beispiel unten sind das "Veröffentlicht", "Stimmung" und "Stichworte".

Figure 32. Tabelle mit Rendermode `searchresults`

slideshow

Zeigt eine Tabelle als Slideshow an. Konfigurieren Sie dafür eine Tabelle mit drei Spalten. Die erste Spalte enthält die Bilder für die Slideshow, die zweite den Titel und die dritte eine Beschreibung. Titel und Beschreibung sind optional. Sie werden unterhalb des Bildes angezeigt.

chart

Siehe Chart

timeline

Siehe Timeline

3.3.13.3. Spaltenelemente

Für die Darstellung der Werte einer Spalte können die gleichen Styles und Rendermodes wie bei einer Eigenschaft verwendet werden

3.3.14. Diagramm

Das Web-Frontend bietet die Möglichkeit, unterschiedliche Arten von Diagrammen anzuzeigen. Es gibt zwei Arten der Konfiguration:

- Tabelle mit dem Rendermode `chart`
- Skriptgenerierte View mit dem *viewType* `chart`

Die Darstellung der Diagramme erfolgt mit der Software-Bibliothek Chart.js. Die Dokumentation findet sich hier

3.3.14.1. Tabelle als Diagramm

Standardmäßig wird jede Zeile der Tabelle als ein Datenpunkt des Diagramms interpretiert (z.B. Umsatz u. Gewinn pro Jahr). Die erste Spalte enthält dabei die Beschriftung des Datenpunktes (im Beispiel das Jahr), alle anderen Spalten die einzelnen Datenwerte. Bei der Konfiguration der Tabelle ist unter Umständen daran zu denken, den Wert für *Anzahl Zeilen (Page size)* zu setzen, falls mehr als 20 Datenpunkte im Diagramm angezeigt werden sollen. Über die Konfigurationseigenschaften der Styles kann die standardmäßige Interpretation der Zeilen und Spalten angepasst werden.

3.3.14.2. Skriptgenerierte View als Diagramm

Mit einer skriptgenerierten View lassen sich beliebig komplexe Diagramme erzeugen. Folgende Eigenschaften können an der View per Skript angebracht werden: `type`, `options` und `chartData`. Die genaue Struktur dieser Daten wird in der Chart.js-Dokumentation beschrieben. **Wichtiger Unterschied zur Dokumentation**: Die dort beschriebene Eigenschaft `data` entspricht der oben genannten Eigenschaft `chartData`. Die Eigenschaft `plugins` wird vom Web-Frontend nicht ausgewertet.

3.3.14.3. Styles

vcmPluginChartWidth `Standardwert: auto`

Breite des Diagramms im Browser in Pixeln.

vcmPluginChartHeight `Standardwert: auto`

Höhe des Diagramms im Browser in Pixeln.

vcmPluginChartType `Standardwert: line`

Typ des Diagramms. Folgende Typen werden unterstützt: `bar`, `doughnut`, `line`, `pie`, `radar`. Sollte nur bei Tabellen verwendet werden. Bei skriptgenerierten Views kommt der Typ aus dem Skript.

vcmPluginChartLabelColumn `Standardwert: 0`

Nur für Tabelle. Index der Spalte, die der Beschriftung der Datenpunkte dient.

vcmPluginChartDataColumns

Nur für Tabelle. Zeichenkette mit leerzeichengetrennten Indizes der Spalten, die als Werte des Datenpunktes dienen. Standardmäßig sind das alle Spalten außer der ersten.

vcmPluginChartDataMode `Standardwert: rows`

Nur für Tabelle. `rows` oder `columns`. Gibt an, ob Zeilen oder Spalten als Datenpunkte interpretiert werden. Wird das auf `columns` gesetzt,

vcmPluginChartOptions

Optionen für Chart.js (siehe oben verlinkte Dokumentation)

3.3.15. Timeline

Eine Timeline dient der Anzeige von Ereignissen auf einem Zeitstrahl. Es gibt zwei Arten der Konfiguration:

- Tabelle mit dem Rendermode `timeline`
- Skriptgenerierte View mit dem *viewType* `timeline`

3.3.15.1. Tabelle als Timeline

Die Tabelle benötigt zwei Spalten. Die erste wird als Titel eines Ereignisses interpretiert, die zweite als Beschreibung. Mit einem Skript für die Style-Eigenschaft `extra` kann die genaue Darstellung konfiguriert werden. Es muss ein Objekt mit der Eigenschaft `options` zurückgegeben werden, das die weiter unten beschriebenen Eigenschaften haben kann.

3.3.15.2. Skriptgenerierte View als Timeline

Im Skript können zwei Eigenschaften an das `view`-Objekt gehängt werden:

`events` **(obligatorisch)**

Ein Array von Objekten folgender Art:

```
{
  title: "Titel des Ereignisses", // Eine beliebige Zeichenkette
  content: "Beschreibung des Ereignisses", // Eine beliebige
```

```
Zeichenkette
  elementId: "IDxxx_yyyyyyy" // Die Element-ID eines semantischen
Elements, das beim Ausführen einer Klickaktion addresiert werden soll.
}
```

`options` **(optional)**

Das Optionsobjekt für die Timeline.

3.3.15.3. Optionsobjekt

Das Optionsobjekt für die Timeline kann folgende Eigenschaften haben:

direction

`"horizontal"` oder `"vertical"`

Die Richtung der Timeline. Der Standardwert ist `"horizontal"`

side

`"start"`, `"end"` oder `undefined`

Gibt, abhängig von der Richtung der Timeline, die Seite an, auf der die Ereignisse angezeigt werden. `"start"` bedeutet über einer horizontalen und links von einer vertikalen Timeline. `"end"` bedeutet unter einer horizontalen und rechts von einer vertikalen Timeline. Wird der Wert weggelassen, werden die Ereignisse abwechselnd auf beiden Seiten angezeigt.

event

Ein Objekt mit der Eigenschaft `size`. Gibt die Größe des Kreises auf der Timeline an, der ein Ereignis repräsentiert. Folgende Werte können für `size` angegeben werden: `"xx-small"`, `"x-small"`, `"small"`, `"default"`, `"large"`, `"x-large"`. Der Standardwert ist `"small"`.

3.3.15.4. Label

Die Label-View wird verwendet, um den Titel und das Icon von Dialog-Panels sowie des Browser-Tabs zu konfigurieren.

Die Label-View wird dazu beim entsprechenden Fenstertitelpanel (Titel-Konfiguration bei Hauptfensterpanel bzw. Dialogen) konfiguriert.

Konfigurationseigenschaften

Beschriftung

Die Beschriftung wird als Dialog- bzw. Browser-Tab-Titel (Titel der Website) angezeigt.

Bildcontainer

Das konfigurierte Bild wird als Icon in der Dialog-Kopfzeile beziehungsweise als Favicon im Browser-Tab angezeigt.

3.3.16. Passwortänderung

Diese View ermöglicht das Ändern des Passworts des aktuell angemeldeten Anwenders. Damit die Passwortänderung ausgelöst werden kann, muss eine Aktion mit der *Aktionsart* `Passwortänderung` an der View angelegt werden.

Wenn das Passwort bestimmten Richtlinien entsprechen soll, können diese das über das Rechtesystem geprüft werden. Dazu muss ein *Eigenschaftsfilter* für die Eigenschaft *password* angelegt werden. Die Meldung, die im Web-Frontend bei Verletzung der Richtlinien angezeigt wird, kann über den Namen des *Zugriff verweigern*-Entscheiders konfiguriert werden.

3.4. Betrieb des Web-Frontends

Das Web-Frontend wird als ZIP-Datei ausgeliefert. Es kann entweder direkt über die **i-views Bridge** bereitgestellt oder auf einem beliebigen Webserver betrieben werden.

3.4.1. Standardinstallation

Bei Verwendung der Standardpfade ist keine zusätzliche Konfiguration erforderlich.

3.4.2. Betrieb hinter einem Proxy

Beim Betrieb des Web-Frontends hinter einem Proxy mit modifizierten Pfaden ist sicherzustellen, dass die erforderlichen Header für die i-views Bridge konfiguriert werden.

Parametername	Beschreibung	Standardwert
iv-root-url	Der Einstiegspunkt der Bridge. Entweder eine URL mit Schema, Host und Port oder ein absoluter Pfad. Beispiele: • `http://localhost:8815` • `https://localhost:8815` • `https://i-views.com/myproject/x/y` • `/myproject/x/y`	`/`
iv-path-viewconfig	Absoluter oder relativer (zu `iv-root-url`) Pfad zum Viewconfig-Service für das Web-Frontend. Beispiele: • `viewconfig/` • `my-service/` • `/myproject/x/y/service`	Pfad des zum REST-Request gehörenden Services
iv-path-bookmarks	Absoluter oder relativer (zu `iv-root-url`) Pfad, aus dem die Lesezeichen-URLs gebildet werden.	Identisch zu `iv-path-viewconfig`

Parametername	Beschreibung	Standardwert
iv-path-static	Absoluter oder relativer (zu `iv-root-url`) Pfad zu den statischen Ressourcen. Beispiele: • `viewconfig/viewconfigmapper` • `viewconfig/my-statics` • `my-service/viewconfigmapper` • `/static/`	`<iv-path-viewconfig>/viewconfigmapper`
iv-cookie-path	Definiert den Pfadbereich (Scope) des Authentifizierungs-Cookie	Identisch zu `iv-path-viewconfig`
iv-secure-cookies	Legt fest, ob das `secure`-Flag für Cookies gesetzt werden soll. Mögliche Werte: `true` oder `false`	`false`

3.4.2.1. Content Security Policy

Das Web-Frontend erfordert die folgenden minimalen **Content Security Policy (CSP)**-Direktiven:

Directive	Wert	Beschreibung
default-src	`'self'`	Ressourcen nur aus derselben Quelle (Origin) laden
script-src	`'self'`	Skripte nur aus derselben Quelle (Origin) laden
style-src	`'self' 'unsafe-inline'`	Styles nur aus derselben Quelle (Origin) laden; `unsafe-inline` wird benötigt, wenn benutzerdefiniertes CSS über den REST-Service konfiguriert wird
img-src	`'self' data: https://*.tile.openstreetmap.org`	Bilder aus derselben Quelle lade, sowie Data-URLs. Optional bei Verwendung des Maps-Plugin die OpenStreetMap-Kacheln

Directive	Wert	Beschreibung
font-src	`'self' data:`	Schriftarten nur aus derselben Quelle laden, sowie Data-URLs
connect-src	`'self'`	Erlaubt nur Netzwerkverbindungen zur selben Quelle
frame-ancestors	`'none'`	Optional. Verhindert das Einbetten der Seite in Frames
base-uri	`'self'`	Beschränkt die Basis-URI auf dieselbe Quelle
form-action	`'self'`	Erlaubt nur Formular-Übermittlungen zur selben Quelle

3.5. Interaktionsmuster und Rezepte

Interaktionsmuster helfen dabei, die Dynamik einer Anwendung sinnvoll und konsistent zu gestalten. Gleichzeitig unterstützen sie Benutzer dabei, das Verhalten einer Anwendung intuitiv zu verstehen, da sie auf etablierten Mechanismen basieren, die aus vielen anderen Anwendungen bereits vertraut sind.

Typische Beispiele für solche Interaktionsmuster sind:

- Navigationsleisten
- Einkaufswagen
- Assistenten (Wizards)
- einfache Suchfunktionen

Dieses Kapitel erhebt keinen Anspruch auf eine vollständige Übersicht aller Interaktionsmuster — entsprechende Sammlungen finden sich in der einschlägigen Fachliteratur. Stattdessen werden ausgewählte Muster exemplarisch dargestellt, wie diese mit dem i-views Web-Frontend konkret umgesetzt werden können.

3.5.1. Navigationsleiste

Ähnlich wie bei der Visualisierung einer *Alternative*-Ansicht können auch Panel-Registerkarten mit einer Navigationsleiste verwendet werden. Der Vorteil von Panel-Registerkarten mit Navigationsleiste im Vergleich zur Alternative: Panel-Registerkarten können weitere Unter-Panels enthalten, was die Konfiguration spezifischerer Layouts sowie die Nutzung aller panelbezogenen Funktionen ermöglicht, über die eine Ansicht nicht verfügt (Ansichtsmodelle, Session-Grenzen, Interaktion usw.).

Um Panel-Registerkarten mit einer Navigationsleiste zu konfigurieren, gehen Sie wie folgt vor:

1. Konfiguration der Panel-Struktur:
    - Erstellen Sie ein Panel mit einem Menü, das sich am oberen oder seitlichen Rand des Bildschirms befindet.
    - Je nach gewünschtem Layout des Navigationsleistenmenüs wählen Sie den Menütyp *Werkzeugleiste* für horizontales Layout oder *Liste* für vertikales Layout.
    - Erstellen Sie für jeden anzuzeigenden Bereich eine Schaltfläche.
    - Erstellen Sie ein weiteres Panel vom Typ *Wechselndes Layout*, das den restlichen Teil des Anzeigebereichs abdeckt.
    - Erstellen Sie für jeden Abschnitt ein Unter-Panel dieses Panels und markieren Sie jedes Panel als *Session-Grenze* (Checkbox auf true gesetzt).
2. Verknüpfung von Aktion und Panel:
    - Verknüpfen Sie jede Schaltfläche (= Aktion) mit dem entsprechenden Bereichspanel über die Relation *Ergebnis anzeigen in Panel*. Dies bewirkt, dass ein Panel aktiviert wird, wenn

seine Schaltfläche gedrückt wird.

- Verknüpfen Sie jede Schaltfläche (= Aktion) mit dem Panel des Menüs, in dem sich die Aktion selbst befindet. Dies bewirkt, dass das Styling der Schaltfläche aktualisiert wird, wenn die Schaltfläche gedrückt wird.

3. Erstellen des Styles für die Aktion:

   - Für eine bessere Benutzerfreundlichkeit, erstellen Sie einen *Style* `buttonActive`, der eine visuelle Anzeige der Buttonauswahl gibt. Erstellen Sie den Style zunächst für eine Aktion und verwenden Sie ihn dann auch für die anderen Aktionen (weisen Sie ihn zu).
   - Fügen Sie das folgende Skript unter *class (script)* zu jeder Schaltfläche hinzu:

```
function additionalPropertyValue (element) {
  const isActive = isActiveForSession($k.Session.actual(), this
.getPanelsToActivate())
  return isActive ? 'yourButtonClass buttonActive' :
'yourButtonClass'
}

function isActiveForSession (session, panelsToActivate) {
  const sessionBoundaryActivatedConfig = session.
panelConfiguration()
  let isActive = panelsToActivate.indexOf
(sessionBoundaryActivatedConfig) > -1
  if (isActive === false && session.parent()) {
    isActive = isActiveForSession(session.parent(),
panelsToActivate)
  }
  return isActive
}
```

   Ersetzen Sie `yourButtonClass` durch den Namen der Klasse, die für die Schaltfläche verwendet werden soll.

4. Definieren der CSS-Klasse für den Style:

   - Fügen Sie für den Style eine Klasse für die aktive Schaltfläche zur Options-Ressource des REST-Services der Anwendung "viewconfig" hinzu.

   Navigieren Sie dazu mit dem Organizer im Knowledge Builder zu *TECHNIK* > *REST*. Wählen Sie in der Objektliste *REST Service* auf der rechten Seite den Service mit der ID "viewconfig" aus. Im Detail-Editor des Services wählen Sie "vcm/options" und bearbeiten den Eintrag für *CSS*.

   Nehmen wir an, dass bereits eine Klasse `.navigation-button` für die Buttons

verwendet wird. Dann wird eine weitere Klasse `.navigation-button.buttonActive` für das Styling des aktiven Zustands des Buttons benötigt:

```
.navigation-button {width: 200px;}
.navigation-button.butttonActive {background-color: red
!important;}
```

5. Schnittstellen von REST und VCM aktualisieren:
   - REST-Service und ViewConfig aktualisieren und das Web-Frontend neu laden (da Panels erstellt wurden).

     Ergebnis: Wenn Sie auf eine Schaltfläche klicken, wird das repräsentative Panel angezeigt und die Schaltflächen werden durch einen Style als aktiv markiert.

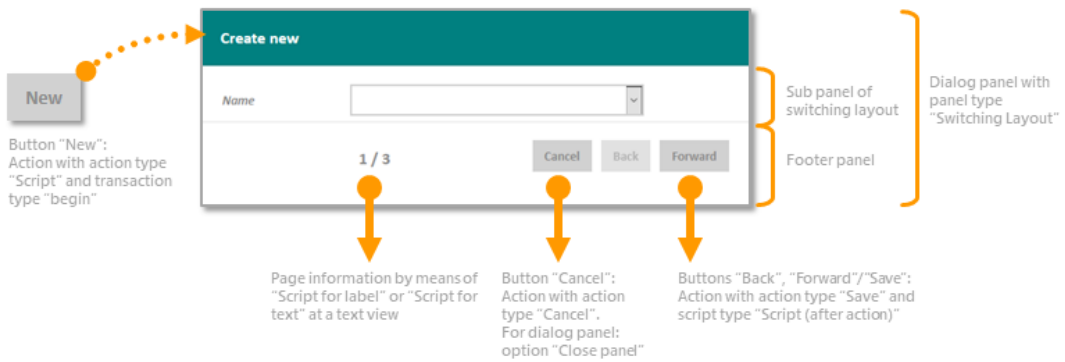
3.5.2. Dialog (modal)

Konfigurieren Sie ein Dialog-Panel. Vergewissern Sie sich, dass das Panel einen richtigen Titel und ein Menü mit einer Schaltfläche "X" zum Schließen des Dialogs hat. Aktivieren Sie die Option *Panel schließen* für die Schließaktion. Konfigurieren Sie den Hauptteil des Dialogs wie gewünscht. Konfigurieren Sie optional einen Fußbereich mit den Menüschaltflächen "Ok" und/oder "Abbrechen" — beide mit der Option *Panel schließen* wie oben.

Schließlich konfigurieren Sie eine Aktion zum Öffnen des Dialogs. Verbinden Sie die Aktion mit dem Dialog-Panel über die Relation *Ergebnis anzeigen in Panel*. Stellen Sie sicher, dass das Ergebnis der Aktion das Domain-Model ist, das Sie im Dialog-Panel anzeigen möchten.

3.5.3. Assistent

Ein Assistent ("Wizard") wird verwendet, um den Benutzer durch einen Eingabeprozess zu führen, der mehrere Schritte umfasst. Die folgende Beispielkonfiguration eines Assistenten sieht wie folgt aus:



- Eine Aktionsschaltfläche gibt das zu bearbeitende oder zu erstellende semantische Element zurück und zeigt es in einem anderen Panel an oder öffnet einen Dialog. Dabei startet die Aktion eine Transaktion, sodass die Bearbeitung jederzeit abgebrochen werden kann.
- Innerhalb des Panels oder Dialogs wird jeder Schritt auf einer separaten, untergeordneten Seite dargestellt.
- Eine Unterkonfiguration zeigt Fortschrittsinformationen an und ist mit den Schaltflächen "Zurück"/"Weiter"/"Speichern"/"Abbrechen" ausgestattet. Im Falle eines Dialogs eignet sich das Dialog-Fußzeilen-Panel für diesen Zweck.

Konfigurieren Sie ein Panel vom Typ *Wechselndes Layout*. Jeder Schritt des Assistenten wird dann in einem Unter-Panel dieses Panels dargestellt. Betten Sie das Layout-Panel in ein Dialog-Panel ein oder sorgen Sie dafür, dass es außer "Weiter/Speichern", "Zurück" und "Abbrechen" keine weiteren Navigationsmöglichkeiten gibt. Konfigurieren Sie ein (Fußzeilen-)Panel unterhalb des Layout-Panels mit den Navigationsschaltflächen "Weiter" und "Zurück".

Der Assistent arbeitet mit einer *Edit*-Ansicht in jedem Unter-Panel des Layout-Panels. Das übergeordnete Panel (z.B. Dialog-Panel) wird durch eine Aktion aktiviert, die eine *Transaktion* auslöst: Wenn der Assistent ohne zu speichern verlassen wird ("Abbrechen"), werden die Änderungen abgebrochen. Jeder Schritt löst eine Zwischenspeicherung ("Weiter"/"Zurück") innerhalb der Transaktion aus.

HINWEIS

Jedes Unter-Panel des Layout-Panels *muss* eine Edit-Ansicht enthalten. Andernfalls funktionieren die Aktionsschaltflächen nicht. Wenn ein einleitender Text für einen Schritt benötigt wird, kann er daher nicht allein in einem separaten Unter-Panel ohne Edit-Ansicht platziert werden.

Dieser Assistent kann verwendet werden, wenn die Schaltflächen entweder im gleichen Bereich wie die Edit-Ansicht oder in einem anderen Bereich als der Edit-Ansicht platziert werden (z. B. in der Fußzeile eines Dialogs).

Registrieren Sie ein Skript mit dem Schlüssel `wizard`:

```
function currentPageIndex () {
  const index = $k.Session.actual().getVariable('currentPageIndex')
  return index || 1
}

function numberOfPages () {
  const switchingConfig = $k.Session.actual().panelConfiguration()
  const switchingPanel = $k.PanelConfiguration.from(switchingConfig)
  return switchingPanel.subPanels().length
}

function nextPage (view) {
  const currentPage = $k.PanelConfiguration.from(view.panelView
```

```
().configurationElement())
    const switchingPanel = currentPage.parent()
    const pages = switchingPanel.subPanels()
    const currentIdx = pages.indexOf(currentPage)
    let nextPage = null
    if (currentIdx < (pages.length - 1)) {
      nextPage = pages[currentIdx + 1] // index based on start value 1
      $k.Session.actual().setVariable('currentPageIndex', currentIdx + 2)
    }
    return nextPage
  }

  function previousPage (view) {
    const currentPage = $k.PanelConfiguration.from(view.panelView
().configurationElement())
    const switchingPanel = currentPage.parent()
    const pages = switchingPanel.subPanels()
    const currentIdx = pages.indexOf(currentPage)
    let previousPage = null
    if (currentIdx > 0) { // 0 based
      previousPage = pages[currentIdx - 1] // index based on start value 1
      $k.Session.actual().setVariable('currentPageIndex', currentIdx)
    }
    return previousPage
  }

export {
    currentPageIndex,
    numberOfPages,
    nextPage,
    previousPage
  }
```

Dieses Skript enthält die folgenden Funktionen:

`currentPageIndex()`, `numberOfPages()`, `nextPage(view)` und `previousPage(view)`, die die aktuelle Seite, die Anzahl der Seiten, die nächste Seite und die vorherige Seite zurückgeben. Verwenden Sie diese Informationen für die Aktions-, Beschriftungs- und Aktivierungsskripte der Schaltflächen "Weiter" und "Zurück".

Fügen Sie für die Fußzeile eine Textansicht für die Anzeige der Seiten und ein Menü hinzu. Fügen Sie im Menü eine Aktion für die Schaltfläche "Weiter" hinzu: Um zwischenzeitliche Änderungen zu speichern, wählen Sie den Aktionstyp *Speichern*.

Beschriftungsskript für die Schaltfläche **Weiter**:

```
import { currentPageIndex, numberOfPages } from 'wizard'

function label (element) {
  if (currentPageIndex() === numberOfPages()) {
    return 'Save'
  }
  return 'Forward'
}
```

Aktionsskript für **Weiter**-Schaltfläche — z.B. Aktionstyp *Speichern* mit *Skript (nach Aktion)*:

```
import { nextPage } from 'wizard'

function postAction (element, action) {
  const next = nextPage(this)
  if (next === null) {
    action.setClosePanel(true)
    action.setTransactionCommit(true)
  } else {
    action.result().activatePanelConfiguration(next)
  }
}
```

Wenn der letzte Schritt erreicht ist (= letztes sichtbares Unterfenster), fungiert die Schaltfläche "Weiter" als Speichern-Schaltfläche und schreibt die Transaktion fest — mit der Folge, dass alle im Dialog vorgenommenen Änderungen gespeichert werden. Das Beenden der Transaktion speichert auch alle zwischengespeicherten Änderungen, die während der Transaktion vorgenommen wurden.

Aktionsskript für die Schaltfläche **Zurück** — z.B. Aktionstyp *Speichern* mit *Skript (nach Aktion)*:

```
import { previousPage } from 'wizard'

function postAction (element, action) {
  const previous = previousPage(this)
  if (previous !== null) {
    action.result().activatePanelConfiguration(previous)
  }
}
```

Aktivierungsskript für die Schaltfläche **Zurück**:

```
import { currentPageIndex } from 'wizard'
```

```
function actionEnabled (element) {
  return currentPageIndex() > 1 // index based on start value 1
}
```

Fügen Sie der Textansicht in der **Fußzeile** ein *Label* hinzu, das die aktuelle Seitenzahl und die Gesamtsumme zur Fortschrittsanzeige anzeigt. Stellen Sie ein Label-Skript wie folgt bereit:

```
import { currentPageIndex, numberOfPages } from 'wizard'

function label (element) {
  return `${currentPageIndex()} / ${numberOfPages()}`
}
```

Konfigurieren Sie je nach Bedarf weitere Funktionen für jeden Schritt des Assistenten.

3.5.4. Transaktion

Konfigurieren Sie die erste Aktion als *Transaktion: **beginnen*** und die letzte Aktion als *Transaction: **beenden***. Für jede dazwischen liegende Aktion sollte es eine alternative Aktion geben, die es den Benutzern ermöglicht, die Transaktion abzubrechen. Konfigurieren Sie Abbruchaktionen mit der Aktionsart *Abbrechen*.

Verwenden Sie die Muster [technical-handbook-web-frontend-interaction-patterns-interaction-patterns-and-recipes-transaction-pattern-dialog] oder [technical-handbook-web-frontend-interaction-patterns-interaction-patterns-and-recipes-transaction-pattern-wizard], um den Umfang der Transaktion eindeutig anzugeben.

3.5.5. Geführte Eingabe

Hängen Sie ein Menü an die Eigenschaftskonfiguration an, für die Sie eine geführte Eingabe bereitstellen möchten. Fügen Sie dem Menü eine Aktion hinzu und konfigurieren Sie die Aktion so, dass sie alles Notwendige tut, um den Prozess zu initiieren. Konfigurieren Sie ein *Skript (recall)*, das nach Beendigung des geführten Prozesses ausgeführt werden soll:

```
function customActionRecall(action, actionResult) {
  this.setNewValue(actionResult.element())
  actionResult.activatePanel(this.panelView())
}
```

In diesem Skript wird der Eingabewert in das Eigenschafts-Eingabefeld geschrieben (Funktion `setNewValue()`) und das Ergebnis der Aktion wird mit dem Inhalt des aktuellen Panels versehen, was notwendig ist, da wir sonst die Werte anderer Eingabefelder im selben Panel verlieren könnten.

Verbinden Sie die Aktion mit einem Dialog-Panel oder einem Schwesterpanel, indem Sie die Relation *Ergebnis anzeigen in Panel* verwenden. Konfigurieren Sie das Ziel-Panel so, dass es durch den Prozess der Bestimmung der Eingabewerte führt (siehe z.B. Muster Assistent oder Dialog).

Die letzte Aktion des Prozesses muss das Recall-Skript aufrufen und sicherstellen, dass mögliche Dialogfelder geschlossen werden. Zusätzlich muss der Wert des Eingabefeldes an das Recall-Skript übergeben werden, indem z.B. das Aktionsergebnis entsprechend gesetzt wird.

```
function customAction(action, actionResult) {
  actionResult.setModel(action.selectedElement())
  action.recallMarkedAction()
}
```

Bieten Sie dem Benutzer die Möglichkeit, den Vorgang abzubrechen. Die Abbruchaktion muss die Rückrufaktion aus der Sitzung entfernen:

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

3.5.6. Suchen und Filtern

Die Suche nach einem bestimmten Element des Knowledge Graphs ist oft eine komplexe Aufgabe, die durch verschiedene Elemente und Funktionalitäten der Benutzeroberfläche unterstützt werden muss. Zunächst muss die Suche auf die Bedürfnisse des Benutzers parametrisiert werden. Nach dem Start der Suche werden die Ergebnisse in einer Weise visualisiert, die es dem Benutzer ermöglicht, zwischen verschiedenen Elementen zu unterscheiden und schließlich eines oder mehrere auszuwählen. Optional ist eine weitere Filterung der Suchergebnisse erforderlich.

In jedem Fall ist die Konfiguration eines Suchverbunds erforderlich, der zum einen die zu verwendende Suche festlegt und zum anderen die Zustände aller beteiligten Ansichten zusammenführt, welche für gewöhnlich über mehrere Panels verstreut sind.

3.5.6.1. Parameter

Die Eingabe von Parametern erfolgt mittels Formulareingaben. Damit die Nutzereingaben in den passenden Suchparametern verarbeitet werden, müssen die Formulareingaben als Teil eines Suchverbunds konfiguriert werden wie im Kapitel zu Formulareingaben beschrieben.

Web-Anwendungen haben häufig eine global verfügbare Suche mit einem einzigen Parameter in einer stets sichtbaren Kopfleiste. Komplexer parametrisierte Suchfunktionalität benötigt mehr Raum für die Parameter und wird daher nur bei Bedarf sichtbar gemacht. In jedem Fall hat es sich als ratsam erwiesen, die Parameter auf einem separaten Panel zu platzieren.

3.5.6.2. Auslösen der Suche

Der Suchprozess kann ausgelöst werden, sobald der Nutzer alle erforderlichen Suchparameter spezifiziert hat. Wenn es nur einen Parameter gibt, dann kann die Auslösung direkt durch eine "Annehmen-Aktion" an der Formulareingabe erfolgen. Ansonsten ist eine separate Schaltfläche erforderlich.

Die Aktion zur Auslösung der Suche kann entweder die Aktionsart "Skript" oder keine Aktionsart haben. Wichtig ist, dass bei der Auslösung der Suche die erforderlichen Werte aus den Formulareingaben für die Parameter vorliegen. Dazu muss die Aktion an einer Ansicht ausgeführt werden, die sich auf demselben Panel wie die Formulareingaben für die Parameter befindet.

Die Aktion zur Auslösung der Suche muss außerdem das Panel mit der Ansicht des Suchergebnisses bzw. das Panel mit der Ansicht zur Filterung des Suchergebnisses aktivieren.

3.5.6.3. Filtern

Mittels einer Facettenansicht kann der Nutzer das Suchergebnis weiter einschränken. Dazu muss die Facettenansicht Teil eines Suchverbundes sein. Eine Interaktionskonfiguration der Facettensicht ist nicht erforderlich. Bei einer Änderung der Facettenauswahl wir automatisch eine Neu-Berechnung des Panels mit der aktualisierten Facettenauswahl ausgelöst.

Damit die Ansicht des Suchergebnisses an die Facettenauswahl angepasst wird, muss das Panel der Facettenauswahl das Panel mit der Ansicht des Suchergebnisses beeinflussen. Das ist natürlich nicht erforderlich, wenn sich beide auf demselben Panel befinden. Aus Performance-Gründen empfiehlt sich jedoch die Konfiguration auf separate Panels.

3.5.6.4. Suchergebnis anzeigen

Zum Anzeigen von Suchergebnissen können verschiedene Ansichten als Ausgabe im Suchverbund konfiguriert werden:

- Tabelle
- Graph
- Layout
- Alternative

Im Falle von Layout und Alternative benötigt es ein *Skript für Domain-Model*, welches die Elemente des Suchergebnisses auf die Unteransichten verteilt:

```
function domainModel () {
  return this.domainModel().elements()
}
```

3.5.7. Gespiegelter Zustand

Wechselt der Nutzer durch Navigation auf ein neues Panel, dann werden die darauf enthaltenen Ansichten auf Basis eines "leeren" Basiszustands neu berechnet. Möchte man an dieser Stelle bereits mit einem Zustand starten, den der Anwender durch Manipulation von Ansichten anderer Panels hergestellt hat, dann kann man die entsprechenden Ansichten über einen *Spiegelverbund* als Quelle und Ziel einer Zustandsspiegelung konfigurieren.

> Panel A befindet sich auf der Hauptseite der Anwendung und enthält eine Formulareingabe F1 für eine globale Suche. Panel B enthält das Suchergebnis und zeigt außerdem noch einmal den eingegebenen Suchparameter in einer Formulareingabe F2. Verbindet man F1 über die Relation *Gespiegelt von* und F2 über die Relation *Spiegelung von* mit einem Spiegelverbund, dann wird bei der Initialisierung von F2 der Zustand von F1 gespiegelt.

Die Spiegelung ist unidirektional, kann aber bei Bedarf auch bidirektional konfiguriert werden, wenn die beteiligten Ansichten jeweils sowohl "Spiegelung" als auch "gespiegelt" sind.

3.5.8. Benutzerdefinierter Relationszieldialog

Der Standard-Relationszieldialog stellt folgende Funktionen zur Verfügung:

- Eine Liste für die Auswahl des Relationsziels anhand dessen Primärnamens. Ein Klick auf einen Listeneintrag schließt den Dialog und erzeugt eine Relation zum ausgewählten Relationsziel.
- Ein Dropdown-Formulareintrag erlaubt die Auswahl des Relationszieltyps. Eine "Neu"-Schaltfläche legt ein neues Objekt des gewählten Typs als Relationsziel für die neue Relation an.
- Eine "Abbrechen"-Schaltfläche schließt den Dialog ohne weitere Aktion.

Jedoch kann es vorkommen, dass der Standard-Relationszieldialog für spezielle Web-Frontend angepasst werden muss. Beispielsweise wird für die Anzeige in der Liste eine andere Eigenschaft benötigt oder der Relationszieldialog darf das Anlegen von Objekten durch den Anwender nicht zulassen etc.

Ein benutzerdefinierter Relationszieldialog kann auf einfache Art und Weise wie folgt angelegt werden:

1. Eigenschaftskonfiguration der Relation auswählen und neues Menü anlegen.
2. Am Menü die Menüart "View-spezifische Aktionen" wählen.
3. Dem Menü eine neue Aktion hinzufügen und dabei die Aktionsart *Relationsziel auswählen* wählen.
4. Im Strukturbaum des View-Config-Mappers den Knoten *Dialog-Panels* wählen und neues Dialog-Panel anlegen, dabei im folgenden Dialog die Vorlage `RelationTargetDialog` wählen.
5. Ein nachfolgender Dialog fordert zur Eingabe eines Namens auf. Dieser Namen wird anschließend durch automatisches Hinzufügen der Endung `.relationTargetDialog` dazu

verwendet, die Konfigurationsnamen sämtlicher Bestandteile des Dialog-Panels zu generieren.

6. Auf dem *Kontext*-Reiter die Relation *Ergebnis anzeigen aus Aktion* zur zuvor angelegten Aktion erstellen.
7. ViewConfig-Elemente nach Bedarf anpassen (Tabelle, Menü-Aktionen etc.). Beispiel: Wenn die "Neu"-Schaltfläche nicht benötigt wird, diese löschen. Wenn sowohl die Typauswahl als auch die "Neu"-Schaltfläche nicht benötigt werden, das gesamte Panel des Menüs löschen.

3.5.8.1. Neuen Standard-Relationszieldialog bestimmen

Der Standards-Relationszieldialog ist als Dialog-Panel des View Configuration Mappers vorkonfiguriert. Er besitzt die Rolle `RelationTargetDialog` und wird standardgemäß bei der Auswahl eines Relationsziels angezeigt durch Klick auf die Such-Schaltfläche "+" an der Eigenschafts-View.

Durch Neu-Zuweisen der Rolle `RelationTargetDialog` kann ein anderer Dialog als Standard-Relationszieldialog festgelegt werden.

HINWEIS	Durch die Benutzung eines Standard-Dialog-Panels mit der Rolle `RelationTargetDialog` muss an der zu editierenden Eigenschaftskonfiguration keine benutzerdefinierte Aktion hinzugefügt werden.

3.6. Spezialisierte Komponenten

Dieses Kapitel beschreibt die Realisierung von Frontend-Funktionalität für Spezialfälle, die mittels der allgemeinen und generischen Konfigurationselemente nicht realisiert werden kann.

3.6.1. Knowledge-Builder als Web-Anwendung

Ein Knowledge-Builder lässt sich auch als VCM-Webanwendung realisieren. Dabei können zum großen Teil die gewohnten View-Konfigurationselemente und Techniken verwendet werden, die in jeder VCM-Webanwendung zum Einsatz kommen.

Ein fertig konfigurierter Knowledge-Builder als Web-Anwendung (kurz: "Web-KB") ist als vorkonfigurierte benutzerdefinierte Komponente erhältlich. Dieser Web-KB kann in einen beliebigen Knowledge-Graph importiert werden und ist dann sofort nutzbar. Wenn gewünscht lässt er sich dann noch erweitern, um beispielsweise zusätzliche Funktionalitäten zu realisieren:

- Projekt-spezifische Objektansichten
- Einbindung in bereits existierende Frontends
- Herausgreifen von Teilaspekten für dedizierte Nutzergruppen
- u.s.w.

Natürlich ließe sich ein Web-KB auch von Grund auf selbst konfigurieren. Neben den bekannten Techniken benötigt man dazu noch einige spezialisierte Schema-Elemente, die im folgenden Abschnitt beschrieben werden.

3.6.1.1. View-Konfiguration für Schema-Elemente

Für die Pflege von Schema-Elementen sind besondere Eingabeformen erforderlich, die mittels Spezialisierung von generischen View-Konfigurationen über einen sogenannten "Systemidentifikator" realisiert werden. So lässt sich zum Beispiel über eine Tabelle mit Systemidentifikator "subConcepts" die Menge der Subtypen eines Typs anzeigen und pflegen. View-Konfigurationen mit Systemidentifikator bieten die besondere Aktionsart "Schema", die dann noch einmal durch eine "Aktionsunterart" näher spezifiert wird. Die folgende Tabelle listet die Systemidentifikatoren, an welcher View-Konfiguration sie sich anbringen lassen und die unterstützten Aktionsunterarten.

Systemidentifikator	Schema-Element	View-Konfiguration	Aktionen
mainDirection	Hauptrichtung	Eingabe für booleschen Wert	Annehmen
abstractType	Abstrakt	Eingabe für booleschen Wert	Annehmen
oneWayRelation	Einseitige Relation	Eingabe für booleschen Wert	Annehmen

Systemidentifikator	Schema-Element	View-Konfiguration	Aktionen
multipleOccurrences	Kann mehrfach vorkommen	Eingabe für booleschen Wert	Annehmen
indexed	Indexiert	Eingabe für booleschen Wert	Annehmen
maxOccurrencesGuideline	Richtwert für maximales Auftreten	Eingabe für Zahl	Annehmen
minOccurrencesGuideline	Richtwert für minimales Auftreten	Eingabe für Zahl	Annehmen
internalName	Interner Name	Eingabe für Zeichenkette	Hinzufügen, Neu, Entfernen
attributeDefinitions	Attribute der Objekte	Tabelle	Hinzufügen, Neu, Entfernen
relationDefinitions	Relationen der Objekte	Tabelle	Hinzufügen, Neu, Entfernen
domainDefinitions	Definiert für	Tabelle	Hinzufügen, Entfernen
supplementalRelationDefinitions	Ergänzungstypen	Tabelle	Hinzufügen, Neu, Entfernen
superConcepts	Obertypen	Tabelle	Hinzufügen, Entfernen
subConcepts	Untertypen	Tabelle	Hinzufügen, Neu, Entfernen

Um Operationen auf Schema-Elementen ausführen zu können, wird auf das Recht "Schema Modifizieren" geprüft.

3.6.2. Aufgaben-Komponente

Die Aufgaben-Komponente enthält Schema und Funktionalität, um Aufgaben, die ein Jobclient ausführt, als Objekte im Knowledge-Graph zur Verfügung zu stellen. Auf diese Weise sind die Erstellung neuer Aufgaben und die Übersicht über die Abarbeitung von Aufgaben im konfigurierten Web-Frontend möglich.

3.6.2.1. Einrichtung

Zunächst muss die Komponente "Aufgaben" mittels des Admin-Tools im Knowledge-Graph aktiviert werden. Als Resultat erhält man Schema für Aufgaben-Objekte, welche zur Ausführung der Aufgaben angelegt werden und nach Ausführung das Ergebnis bzw. ggf. Fehlermeldungen beinhalten. Aktuell (Stand 6.1) gibt es an möglichen Aufgaben-Typen nur Import-Aufgaben, die einen Datenimport über eine gegebene Abbildung ermöglichen.

Für die Ausführung der Aufgaben benötigt man einen Job-Client, der zur Ausführung von Script-Jobs konfiguriert ist.

Als nächster Schritt benötigt man View-Konfiguration, mittels derer die Anwender dann im Web-Frontend zu importierende Daten hochladen können und den Import starten sowie das Ergebnis überprüfen können.

Für einen Import könnte eine "Eingabe für Dateiupload" verwendet werden, um eine neu erstellte Import-Aufgabe mit Daten zu versehen. Die Aktion, die dann den Import auslöst, würde beispielsweise wie folgt aussehen:

```
function customAction(action, actionResult) {
    const mapping = $k.Registry.element("myMapping")
    let task = new $k.ImportTask()
    task.setImportData(this.value())
    task.setMapping(mapping)
    task.insertJob()
    actionResult.setModel(task.semanticElement())
}
```

Die wichtigen Schritte des Skripts sind:

- Task-Objekt erstellen
- Import-Daten eintragen
- Mapping verknüpfen
- Job einwerfen
- optional: Task-Objekt als Ergebnis der Aktion setzen, um es ggf. in einem anderen Panel anzuzeigen

Alle anderen Daten (Verknüpfung zum Nutzer, Zeitstempel, etc.) werden automatisch gestetzt.

Zur Anzeige der laufenden und abgearbeiteten Aufträge empfiehlt sich beispielsweise eine Tabelle - wahlweise getrennt je Auftraggeber/Anwender oder gesammelt für alle Aufgaben.

3.6.2.2. Betrieb

Eine Aufgabe durchläuft folgende Zustände:

1. "erstellt": Die Aufgabe ist erstellt, die Bearbeitung hat noch nicht begonnen
2. "in Arbeit": Die Aufgabe wird aktuell vom Job-Client bearbeitet
3. "abgeschlossen"/"Fehler": Die Aufgabe ist bearbeitet ohne/mit Fehler

Ist die Bearbeitung abgeschlossen, dann findet sich das Ergebnis in dem Attribut "Ergebnis". Wenn es bei der Bearbeitung Fehler gab, dann gibt es zusätzlich noch das Attribut "Detailliertes Ergebnis",

welches eine Datei mit allen Fehlereinträgen enthält.

4. i-views-Dienste

4.1. Allgemeines

Die i-views-Dienste können über Kommandozeilen-Parameter oder eine Konfigurationsdatei konfiguriert werden.

Falls es sowohl in der ini-Datei als auch auf der Kommandozeile Einstellungen für den gleichen Parameter gibt, hat die Kommandozeile Vorrang.

4.1.1. Konfigurationsdatei

Einige Einstellungen können über eine Konfigurationsdatei (Dateiendung .ini) festgelegt werden. Der Aufbau der Datei sieht folgendermaßen aus:

```
[Abschnitt]
parameterName1=parameterWert1
parameterName2=parameterWert2
...
```

Der Name der Standard-Datei hängt von der Art des i-views Werkzeugs ab, also z.B. heißt die ausführbare Datei eines KnowledgeBuilders standardmäßig "kb.exe" (bzw. "kb.im"), daher wird diese ohne obige Konfiguration nach der Konfigurationsdatei "kb.ini" suchen. Zu beachten ist, dass der Dateiname des Werkzeugs keinen Einfluss hat auf den Standard Namen der ini-Datei, die das Werkzeug sucht.

Der Name der Ini-Datei kann über den Kommandozeilen-Parameter "inifile" angegeben werden:

```
-inifile <Dateiname>, -ini <Dateiname>
```

Im folgenden sind Konfigurationen aufgeführt, die für jeden Dienst verwendet werden können. Für dienstspezifische Einstellungen siehe den Abschitt "Konfigurationsdatei" des entsprechenden Dienstes.

4.1.1.1. Makros

In Konfigurationsdateien können weitere Konfigurationsdateien eingebunden werden:

```
$(include:Dateiname)
```

Dadurch können von mehreren Dateien benötigte Information wie z.B. Hostname in eine gemeinsame Datei auslagert werden.

- In der Zeile darf vor/nach der Include-Anweisung nichts stehen, sonst wird sie nicht als include erkannt
- Include entspricht einer textuellen Ersetzung
- Include darf geschachtelt werden, die eingebundene Datei darf also andere Dateien einbinden
- Der Dateiname kann auch Pfadangaben enthalten; unter Windows wird der Schrägstrich `/` automatisch durch den umgekehrten Schrägstrich `\` ersetzt

Des Weiteren ist das Einbinden von Umgebungsvariablen möglich:

```
$(env:Variablenname)
```

Aus “`$(env:USERDNSDOMAIN)`” wird bspw. "i-views.com"

HINWEIS	Dieses Makro kann nur in Schlüsselwerten verwendet werden, bei Kategorien / Schlüsselnamen wird es nicht ersetzt.

Beispiel:

`jobclient.ini`

```
$(include:../shared/host.ini)
$(include:../shared/volume.ini)
jobPools=lucene, KLuceneAdminJob
[JNI]
classPath=...
```

`bridge.ini`

```
$(include:../shared/host.ini)
[KHTTPRestBridge]
$(include:../shared/volume.ini)
port=8815
```

`../shared/host.ini`

```
host=$(env:COMPUTERNAME).$(env:USERDNSDOMAIN)
```

`../shared/volume.ini`

```
volume=master
```

4.1.1.2. Logging-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>,... ]
```

Durch Komma getrennte zusätzliche Prefixe, die bei jeder Log-Ausgabe hinzugefügt werden.

Präfix	Beschreibung
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.

4.1.1.2.1. 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.

HINWEIS	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.

4.1.1.3. Text-Extraktion

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
```

4.1.1.4. HTTP Proxy Konfiguration

Abhängig von der Netzwerk Infrastruktur sind ggfs. HTTP Verbindungen nicht direkt möglich, sondern setzen die Verwendung einer vorhandenen HTTP Proxy Infrastruktur voraus. Ohne weitere Konfiguration versucht i-views nicht, Proxies für HTTP Verbindungen zu verwenden, kann aber dazu konfiguriert werden.

Die einfachste Konfigurationsvariante versucht eine im Betriebssystem hinterlegte HTTP Proxy Konfiguration zu ermitteln. Um dieses Verhalten zu aktivieren, muss der ini-Datei des Werkzeugs folgender Abschnitt hinzugefügt werden:

```
[NetClient]
HttpClient.useProxy=true
```

Beim nächsten Start des Werkzeugs wird i-views dann versuchen, die Proxy Konfiguration des Betriebssystems zu ermitteln und zu verwenden.

Falls dies nicht funkioniert, besteht die Möglichkeit die Proxy-Angaben manuell in die ini-Datei einzutragen:

```
[NetClient]
HttpClient.proxyHost=HOSTNAME
HttpClient.proxyPort=PORT
```

Die Werte von HOSTNAME und PORT können entweder durch das Netzwerk Administration Team zur Verfügung gestellt werden oder man kann versuchen, die Werte aus der Konfiguration des Betriebssystems oder eines Browsers zu ermitteln. Hier sind einige bekannte Quellen dokumentiert:

- Windows: wenn man in einem PowerShell Fenster folgenden Befehl ausführt

```
[System.Net.WebProxy]::GetDefaultProxy()
```

kann man im Feld "Address" die Werte für Hostname und Port-Nummer getrennt von einem Doppelpunkt finden.

- auf Windows nutzen Google Chrome und Microsoft Edge/Internet Explorer genau diese Einstellung von Windows
- Firefox: Einstellunge, Menü "Werkzeuge, Optionen", auf der Seite "Allgemein", Punkt "Verbindungs-Einstellungen", Eintrag "HTTP Proxy"

4.1.1.5. TLS-Konfiguration

If a service provides a HTTPS interface, then a certicate for TLS must be specified in the category **[tls]**. The configuration depends on the operating system.

Windows

```
[tls]
certificateName=MyCert
```

certificateName specifies the "friendly name" property of the certificate in the Windows certificate store. Note that this property is not a part of the certificate itself.

The certificate must be put in the personal certificate store of the user. This means that the service must be run under a user account, not with the local system account.

The private key must be stored, too.

A self-signed certificate for testing can be generated with Powershell:

```
New-SelfSignedCertificate -CertStoreLocation Cert:\CurrentUser\My -DnsName
"mycomputer.mydomain.org" -FriendlyName "MyCert" -NotAfter (Get-Date)
.AddYears(10)
```

Linux

```
[tls]
certificatePath=myCert.cert
privateKeyPath=myCert.key
```

certificatePath is the path to the certificate file. It must be stored in PEM format. *privateKeyPath* is the path to the private key file. It must be stored in PEM format, without password protection.

4.2. Mediator

4.2.1. Allgemeines

The i-views server provides consistent and persistent data storage, and ensures that the data on the i-views clients that are connected are up-to-date.

Data is managed in an object-oriented database that uses an optimistic transaction system to allow cooperative work on the Knowledge Graph.

Functioning as a communication center, the i-views server ensures clients and services are synchronized. As a basic mechanism, it makes a shared object space and active updates available for this.

The i-views server can be operated in three modes:

1. Classic/Compact: The server starts as an individual process in this mode — the so-called "mediator".
2. Multiprocess: The server starts at least two processes in this mode. This results in higher memory usage than in compact mode, however many jobs can be executed in parallel.
3. Distributed: The server components "stock" and "dispatcher" can be configured and operated separately in this mode. This makes it possible to distribute the server components across different computer nodes.

4.2.2. Systemvoraussetzungen

Der i-views-Server ist Plattform-unabhängig und läuft auf allen gängigen Betriebssystemen, z. B. Windows und Linux. GUI Komponenten werden auch auf MacOS unterstützt. Andere Plattformen auf Anfrage.

OS	Version	Prozessor	Unterstützt
Windows	Windows 11, 10 22H2; Windows Server 2022, 2025	x86	ja
Linux	Kernel >= 3.10, glibc >= 2.17	x86	ja
MacOS	macOS >= 12.x	x86, Apple Silicon	ja

4.2.3. Betriebsmodi

Folgende Kommandozeilenparameter unterscheiden zunächst grundsätzlich über den Modus, in dem der Server gestartet wird. Ohne Parameter startet 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.

Alternativ kann der Modus auch über die Umgebungsvariable `IV_SERVER_MODE` gewählt werden. Als Wert kann `stock`, `dispatcher`, `server` oder `mediator` angegeben werden. Die Kommandozeilenparameter haben Vorrang vor der Umgebungsvariable.

4.2.3.1. Multi-Prozess Modus (-server)

Mit dem Startparameter `-server` wird automatisch ein Stock und ein Dispatcher gestartet. Der Dispatcher öffnet einen Server auf dem Standard-Port (30069). Der Port des Stock wird automatisch ausgewählt. Authentifizierungs-Tokens zwischen den beiden Prozessen werden automatisch erzeugt und müssen nicht konfiguriert werden.

HINWEIS	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. Konfiguration des Stock

The stock is responsible for storing the data on the hard drive. A simple is example of this is the configuration file `stock.ini`

```
[Default]
```

```
interfaces=cnp://0.0.0.0:4998
```

This configuration ensures that the stock listens on port 4998 and communicates via the native Coast protocol.

The following parameters can be used:

```
port=<port number>
```

Starts the stock with port number <num>. Without this entry, port 30069 is used.

This parameter is obsolete. It is replaced by the "interfaces" parameter. The entry "port=1234" corresponds to the entry "interfaces=cnp://0.0.0.0:1234". In contrast to the start parameter, multiple values are possible here, which can be listed consecutively in comma-separated form.

```
interfaces=<interface-1>,<interface-2>,...<interface-n>
```

This parameter determines the addresses and protocols used to access the server. Several values are permissible and are separated by a comma.

Possible protocols are:

- http
- https
- cnp
- cnps

The abbreviation "cnp" stands for "Coast Native Protocol" or "Coast Native Protocol Secure". The syntactic structure of an interface definition is equivalent to a URL with schema, host and port.

The host component is used to manage which network address(es) is/are used to access the server. For example:

- `0.0.0.0` binds to all IPv4 interfaces
- `[::1]` binds to the IPv6 loopback only

The "http" and "https" protocols can be rerouted via proxies, allowing the server to be accessed using an IIS running on port 443, for example.

```
baseDirectory=<Directory>
```

Sets the directory in which the "volumes" directory is located. If this value is supposed to end on

volumes, this directory is used directly without creating an additional "volumes" directory below it.

```
volumesDirectory=<Directory>
```

The Knowledge Graphs are stored in this directory. Here, "volumes" is entered as the default value.

```
backupDirectory=<Directory>
```

Specifies the directory to which the Knowledge Graph backups are written and also read for restoring. Only complete directory names are allowed, no relative paths.

```
networkBufferSize=<Size in bytes>
```

This specifies the size of the buffer that is used for sending/receiving data. The default value is 20480. In some infrastructures you can specify

```
networkBufferSize=4096
```

to achieve a higher throughput.

```
flushJournalThreshold=<Number of clusters>
```

Specifies the maximum value that "changed cluster" + "index cluster" may reach in a saving process. If the value for "changed clusters" has already been exceeded, no "index clusters" are saved; these are kept with the journal instead.

A low value (e.g. 50) guarantees fast saving time but can potentially generate a large journal.

A value of "0" deactivates journaling. The default value is "2000".

A "flush" of the journal is executed after complete saving at the latest. This in turn is triggered if:

- The mediator is closed
- The last client of the corresponding volume is logged off
- Saving is triggered by a full-save job (see jobs.ini)

```
autoSaveTimeInterval=<Wait interval in seconds>
```

Specifies the maximum wait time in seconds until automatic saving takes place again after the last

cluster was saved. The default value is 15.

```
clientTimeout=<Timeout in seconds>
```

Specifies the time in seconds that a connected client may not have sent an Alive message before the mediator regards it as inactive and excludes it.

```
password.flavour=190133293071522928001864719805591376361
password.hash=111995451824586607054955998020526241717349657914270806386949
54247035513239844
```

The mediator password is calculated together with a random flavor to produce a (SHA256) hash value. These two pieces of information then suffice for the mediator to check an authentication request. During authentication on the server, the user name must be specified as "Server.admin". To determine these values, you can use

```
password.update=new_password
```

Trigger the server to compute a new flavor and suitable hash value and write these to the ini file. The "password.update" entry is removed in this process.

```
password=<String>
```

The obsolete but still supported way of setting the mediator password. This variant must not be used at the same time as the SHA256 hash variant.

Changed

```
skipVolumesCheck=<true|false>
```

Specifies whether the check of the existing volume that is normally performed after starting the mediator is skipped

Changed

4.2.3.2.1. Memory settings

The following three parameters are used to configure the memory allocation and usage. You may specify values either in megabytes or actual bytes, whereby it is assumed that values under 1048576 refer to megabytes.

```
maxMemory=<Integer, in MB>
```

Maximum base memory usage permitted. A minimum of 50 MB, the total physical base memory available (under Windows) or 512 MB by default.

```
baseMemory=<Integer, in MB>
```

Base memory usage after which efforts to free up memory increase. By default 0.6 * maxMemory. (alias: "growthRegimeUpperBound")

```
freeMemoryBound=<Integer, in MB> [10]
```

If memory that is being used, but is no longer needed, exceeds this limit, it is freed up for use again.

4.2.3.2.2. Blob service configuration

If the mediator is supposed to be started with an integrated BLOB service so that the BLOBs are stored separately from the database on the hard drive, the following setting must be entered in the `mediator.ini` file:

```
startBlobService=true
```

For more information on this, refer to the documentation of the BLOB service.

4.2.3.3. Konfiguration des Dispatchers

Der Dispatcher ist verantwortlich für Transaktionssteuerung und Koordination mehrere Clients. Eine einfacher Konfigurations-Datei ist

```
[Default]

interfaces=cnp://0.0.0.0:5000

[stock]
address=cnp://localhost:4998
authentication=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.

4.2.3.4. Failover

Im Failover-Modus werden identische Kopien der Knowledge-Graphen auf mehreren Servern vorgehalten, die einen Verbund bilden. Ein Server ist dabei aktiv, die anderen Server passiv. Clients verbinden sich ausschließlich mit dem aktiven Server. Änderungen werden vom aktiven Server an die passiven Server weitergeleitet. Falls der aktive Server ausfällt, wird einer der bisher passiven Server zum aktiven Server, und der Betrieb geht weiter. Sobald der ausgefallene Server wieder verfügbar ist, nimmt er als passiver Server am Verbund teil.

Die Entscheidung, welcher Server aktiv ist, wird pro Knowledge Graph individuell getroffen, d.h. ein Server kann für einen Graphen A als aktiver Server und für einen Graphen B als passiver Server dienen.

4.2.3.4.1. Wahl des aktiven Servers

Die Entscheidung, welcher Server aktiv ist, wird mit dem Raft-Algorithmus getroffen. Hierbei bestimmen die beteiligten Server durch ein Wahlverfahren untereinander, welcher Server als aktiver Server dient. Dieses Verfahren kommt ohne zentrale Kontrollinstanz aus.

4.2.3.4.2. Anzahl der Server

Grundsätzlich sollte bei Raft eine ungerade Anzahl an Servern verwendet werden. Minimal werden drei Server benötigt. Es kann dann der Ausfall eines Servers kompensiert werden. Mit fünf Servern kann der Ausfall von zwei Servern kompensiert werden.

4.2.3.4.3. Initiale Replikation

Falls ein Knowledge Graph auf einem beteiligten Servern nicht vorliegt, lädt sich der Server den Graph vom aktiven Server herunter.

4.2.3.4.4. Blobs

Intern gespeicherte Blobs werden ebenfalls repliziert. Für extern von einem Blob-Service verwaltete Blobs gibt es momentan keinen Failover-Mechanismus.

4.2.3.4.5. Konfiguration

Der Failover-Modus erfordert eine statische Konfiguration des Server-Verbunds. Jeder Server muss im Stock+Dispatcher-Modus betrieben werden. Stock und Dispatcher bilden dabei einen Knoten des Verbunds. Jedem Knoten des Verbunds muss eine eindeutige ID gegeben werden.

Die Konfiguration des Server-Verbunds erfolgt in den Konfigurationsdateien im Abschnitt `[mesh]`.

- Beim Stock müssen die Knoten-ID und die Adressen der anderen Stocks angegeben werden.
- Beim Dispatcher müssen die Knoten-ID und optional die öffentliche URL des Dispatchers angegeben werden.

Da Stock und Dispatcher zum selben Knoten gehören, muss beiden dieselbe ID gegeben werden.

HINWEIS	Es wird derzeit keine Validierung vorgenommen, ob die Konfiguration "sinnvoll" ist, z.B. dass mindestens drei Knoten vorliegen.

Knoten 1

Konfiguration der Stock-Komponente (`stock.ini`)

```
[Default]
interfaces=cnp://0.0.0.0:40001
; Öffentliche URL des Stocks
publicURL=cnp://node1.kg-server.local:40001

[mesh]
; ID des Knotens
id=node-1
; Adressen der Stocks der anderen Knoten
addresses=cnp://node2.kg-server.local:40001,cnp://node3.kg-
server.local:40001
```

Konfiguration der Dispatcher-Komponente (`dispatcher.ini`)

```
[Default]
interfaces=cnp://0.0.0.0:40002
; Öffentliche URL des Dispatchers
publicURL=cnp://node1.kg-server.local:40002

[stock]
address=cnp://127.0.0.1:40001

[mesh]
; ID des Knotens
id=node-1
```

Knoten 2

Konfiguration der Stock-Komponente (`stock.ini`)

```
[Default]
interfaces=cnp://0.0.0.0:40001
; Öffentliche URL des Stocks
publicURL=cnp://node2.kg-server.local:40001

[mesh]
; ID des Knotens
id=node-2
; Adressen der Stocks der anderen Knoten
addresses=cnp://node1.kg-server.local:40001,cnp://node3.kg-
server.local:40001
```

Konfiguration der Dispatcher-Komponente (`dispatcher.ini`)

```
[Default]
interfaces=cnp://0.0.0.0:40002
; Öffentliche URL des Dispatchers
publicURL=cnp://node2.kg-server.local:40002

[stock]
address=cnp://127.0.0.1:40001

[mesh]
; ID des Knotens
id=node2
```

Knoten 3

Konfiguration der Stock-Komponente (`stock.ini`)

```
[Default]
interfaces=cnp://0.0.0.0:40001
; Öffentliche URL des Stocks
publicURL=cnp://node3.kg-server.local:40001

[mesh]
; ID des Knotens
id=node-3
; Adressen der Stocks der anderen Knoten
addresses=cnp://node1.kg-server.local:40001,cnp://node2.kg-
server.local:40001
```

Konfiguration der Dispatcher-Komponente (`dispatcher.ini`)

```
[Default]
interfaces=cnp://0.0.0.0:40002
; Öffentliche URL des Dispatchers
publicURL=cnp://node3.kg-server.local:40002

[stock]
address=cnp://127.0.0.1:40001

[mesh]
; ID des Knotens
id=node3
```

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. Startparameter

A range of parameters can also be transferred to the mediator process when starting. Most parameters can, however, also be specified in the mediator.ini, allowing the mediator to be started using a simple command line. When doing so, the rule is that the parameters specified on the command line take precedence over any parameters specified twice in the .ini file.

The complete list of possible start parameters is output by the mediator when called up using the parameter "-?".

```
-interface <interface-1>
```

This parameter determines the addresses and protocols used to access the server. Possible protocols are: http, https, cnp, cnps. The abbreviation "cnp" stands for "Coast Native Protocol" or "Coast Native Protocol Secure". The syntactic structure of an interface definition is equivalent to a URL with schema, host and port. The host component is used to manage which network address(es) is/are used to access the server. For example: "0.0.0.0"=IPv4 all interfaces, "[::1]"=IPv6 loopback only.

The "http" and "https" protocols can be rerouted via proxies, allowing the server to be accessed using an IIS running on port 443, for example.

```
-clientTimeout <sec>
```

Sets the time within which a client must automatically answer to <sec> seconds. The value should be set to a minimum of 600 (which is also the default value).

```
-baseDirectory <directory>
```

Sets the directory in which the "Volumes" directory is located. Along with the "Volumes" subdirectory, the directories for backups and downloads are created. This parameter used to be called "-volumes".

The following parameters give commands to the mediator executable to run specific jobs, without functioning as a server for Knowledge Graph afterwards.

```
-quickRecover <volume> -recover <volume>
```

In the event that the mediator was not shut down properly (e.g. computer crash), lock files in volumes that were in use stop running. The volume will then not be able to be entered. In order to disable the lock, remove the lock by calling -quickRecover <volume>. It cannot be called when (possible) inconsistencies were found. In this case, the start parameter -recover must be used.

HINWEIS	The working directory called must be the directory that contains the "volumes" directory. The "volumes" parameter therefore does not function in this case.

```
-bfscommand <volume> <command>
```

Executes commands that are identified by the BlockFileSystem.

Command line parameter for logging:

```
-nolog
```

Disables logging

```
-loglevel <integer>
```

Configures the messages that should appear in the log:

- 0: All messages including debug outputs

- 10 (default value): All messages excluding debug outputs
- 20: Warnings and error messages only
- 30: Error messages only

```
-logfile <file name>, -log <file name>
```

Name of the log file that is used instead of the standard log file. It is important to change this parameter when several clients are being started in the same working directory.

```
-debug
```

Switches logging to debug mode

```
-log <logname>
```

Sets the log file to <logname>.

4.2.4.2. Konfigurationsdatei "mediator.ini"

A number of mediator settings can also be defined in the configuration file mediator.ini. The structure of the file is as follows:

```
[Default]
parameterName1=parameterValue1
parameterName2=parameterValue2
...
```

The following parameters can be used at this point:

Network communication

```
port=<port number>
```

Starts the server with port number <num>. Without this entry, port 30069 is used.

This parameter is obsolete. It is replaced by the "interfaces" parameter. The entry "port=1234" corresponds to the entry "interfaces=cnp://0.0.0.0:1234". In contrast to the start parameter, multiple values are possible here, which can be listed consecutively in comma-separated form.

```
interfaces=<interface-1>,<interface-2>,...<interface-n>
```

This parameter determines the addresses and protocols used to access the server. Several values are permissible and are separated by a comma. Possible protocols are: http, https, cnp, cnps. The abbreviation "cnp" stands for "Coast Native Protocol" or "Coast Native Protocol Secure". The syntactic structure of an interface definition is equivalent to a URL with schema, host and port. The host component is used to manage which network address(es) is/are used to access the server. For example: "0.0.0.0"=IPv4 all interfaces, "[::1]"=IPv6 loopback only.

The "http" and "https" protocols can be rerouted via proxies, allowing the server to be accessed using an IIS running on port 443, for example.

For SSL communication (cnps:// or https://), the file paths for certification and private key must also be specified in the configuration file:

```
certificate=name of the .crt file
privateKey=name of the .key file
```

Directories

```
baseDirectory=<Directory>
```

Sets the directory in which the "volumes" directory is located. If this value is supposed to end on volumes, this directory is used directly without creating an additional "volumes" directory below it.

```
volumesDirectory=<Directory>
```

The Knowledge Graphs are in this directory. "volumes" is entered as the default value at this position.

```
backupDirectory=<Directory>
```

Specifies the directory to which the Knowledge Graph backups are written and also read for restoring. Only complete directory names are allowed, no relative paths.

```
networkBufferSize=<Size in bytes>
```

This specifies the size of the buffer that is used for sending/receiving data. The default value is 20480. In some infrastructures, you can specify

networkBufferSize=4096

to achieve a higher throughput.

```
journalMaxSize=<Maximum size of the journal>
```

journalMaxSize=0 can be used to deactivate journaling, which is normally active. The default value is 5242880 (5 MB).

autoSaveTimeInterval=< Wait interval in seconds>

Specifies the maximum wait time in seconds until automatic saving takes place again after the last cluster was saved. The default value is 15.

```
clientTimeout=<Timeout in seconds>
```

Specifies the time in seconds that a connected client may not have sent an Alive message before the mediator regards it as inactive and excludes it.

```
password.flavour=190133293071522928001864719805591376361
password.hash=111995451824586607054955998020526241717349657914270806386949
54247035513239844
```

The mediator password is calculated together with a random flavor to produce a (SHA256) hash value. These two pieces of information then suffice for the mediator to check an authentication request. During authentication on the server, the user name must be specified as "Server.admin". To determine these values, you can use

```
password.update=new_password
```

Trigger the server to compute a new flavor and suitable hash value and write these to the ini file. The "password.update" entry is removed in this process.

```
password=<String>
```

The obsolete but still supported way of setting the mediator password. This variant must not be used at the same time as the SHA256 hash variant.

Changed

```
skipVolumesCheck=<true|false>
```

Specifies whether the check of the existing volume that is normally performed after starting the mediator is skipped

Logging

For the configuration options for logging, see the logging settings in Chapter 11.1.2 Configuration file.

Working memory

The following three parameters are used to configure the memory allocation and usage. You may specify values either in megabytes or actual bytes, whereby it is assumed that values under 1048576 refer to megabytes.

```
maxMemory=<integer, in MB>
```

Maximum base memory usage permitted. A minimum of 50 MB, the total physical base memory available (under Windows) or 512 MB by default.

```
baseMemory=<integer, in MB>
```

Base memory usage after which efforts to free up memory increase. By default 0.6 * maxMemory. (alias: "growthRegimeUpperBound")

```
freeMemoryBound=<integer, in MB> [10]
```

If memory that is being used, but is no longer needed, exceeds this limit, it is freed up for use again.

BLOB service configuration

If the mediator is supposed to be started with an integrated BLOB service so that the BLOBs are stored separately from the database on the hard drive, the following setting must be entered in the "mediator.ini" file:

```
startBlobService=true
```

For more information on this, refer to the documentation of the BLOB service (see link below).

4.2.4.3. Sicherheitskonzept des Mediators

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]
build=Build 00010101
rsa.n_1=93D516DF61395258AA21A91B33E8EE67
rsa.n_2=B07C6FC5023DBB18F2201CF723C8F5DD
rsa.n_3=78941FB7C10D20988FEDFC6BD02CF3B7
rsa.n_4=E4567751843C38F055ED791AA7505278
rsa.n_5=23D94BB9EAB2E23F21DBEAA3DD2D2776
rsa.n_6=CE8B81564645DA85C85E9A78BB6E6B41
rsa.n_7=28A646D4868C38E00AE4810601B1EE9F
rsa.n_8=4FF5C35F873E6ED4F65F0FE8B4B45307
rsa.e_1=010001
```

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 konfigurieren

In a number of application scenarios, it may be necessary to log all accesses to a Knowledge Graph in an access or audit log. This audit log contains entries for all log-in and log-out processes, write and read access to Knowledge Graph contents, search requests made, printouts, exports, etc.

The log must be activated in the "System configuration / Audit log" category in the Admin tool. The activation or deactivation of the log, in turn, results in a entry in the audit log.

An analysis tool can be opened in the administrator menu of the Knowledge Builder to view and search within the access log.

The log can be configured by creating a file named `log.ini` in the data directory of the volume. This configuration file is only read when the volume is opened. If the configuration was changed while the volume was opened, then the Mediator has to be restarted.

```
[Default]
; A comma-separated list of log names. The log is configured in the
section with the same name.
applicationLog=audit

[audit]
; Create a compressed backup every 28 days and start with a new empty log
backupInterval=28
; Max size of a JSON file, in MB
maxLogSize=5
; Do not flush the log immediately, for better performance
writeBackImmediately=false
```

4.2.5. Betrieb

4.2.5.1. Herunterfahren des Servers

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. Speicherung und Backup von Knowledge-Graphen

Directory structure

The basic directory of the i-views server has the following structure:

```
volumes/
   knowledgegraphName/
     knowledgegraphName.cbf
     knowledgegraphName.cdr
     knowledgegraphName.cfl
     knowledgegraphName.lock (if the Knowledge Graph is open)

backup/
  knowledgegraphName/
    <ten-digit number>/
      knowledgegraphName.cbf
      knowledgegraphName.cdr
      knowledgegraphName.cfl
```

Storage of Knowledge Graphs

Knowledge Graphs are stored in the file system in the "volumes" subdirectory of the basic directory of the i-views server. In this directory, a subdirectory with a corresponding name is created for each Knowledge Graph. A file with the ".lock" file extension indicates that a Knowledge Graph is currently in use.

Backup of Knowledge Graphs

The Knowledge Graph directories must never by copied while the server is running. For this purpose the server has a backup service, which copies a consistent state of the Knowledge Graph to a backup area. This backup area must be backed up at regular intervals (e.g. as part of an overall backup strategy).

The location where backups are created can be specified using the entry

```
backupDirectory=<directory>
```

in the `mediator.ini` file. Without this information, the "backup" subdirectory of the basic directory is used.

The backup service of the K-Infinity server can be initiated in two ways:

1. With a direct request to the server process (e.g. from the administrator tool)
2. With entries in the `jobs.ini` file in the working directory of the server. For each Knowledge Graph, this file can contain a category [name_of_graph] with the following entries:

Example jobs.ini

```
[volume1]
;Backup of Knowledge Graph "volume1"

;Time the backup starts
backupTime=00:45

;Interval in days -- daily in this case
backupInterval=1

;Keep the last 5 backups of this Knowledge Graph
backupsToKeep=5
```

"backupsToKeep" specifies the number of backups to be kept. This also includes backups that were created manually. The default value is 3.

When specifying the graph names in square brackets, you can use the wildcards "*" and "?"; the names are not case-sensitive.

4.2.5.3. Garbage Collection

Without Garbage Collection, the Knowledge Graph continues to grow through use. Hence, it makes sense to perform a cleanup (Garbage Collection) from time to time. Like a data backup, you can start the Garbage Collection manually at any time (e.g. with a special administrator tool) or it can be started automatically.

Depending on the size of the Knowledge Graph, the Garbage Collection might require a lot of time and memory. When running the Garbage Collection in large Knowledge Graphs, we recommend starting it without connected clients (e.g. Knowledge Builder and Job-Clients) and without other active processes (e.g. backup).

Automatic Garbage Collection: Structure of the jobs.ini file

Automatic Garbage Collection is configured through an entry in the `jobs.ini` file, e.g.

```
[volume1]
garbageCollectTime=00:55
garbageCollectInterval=7
```

This entry in jobs.ini ensures that a garbage collection in the Knowledge Graph called "volume1" is performed at "00:55" a.m. every "7" days. The default value for the interval is "1" (i.e. daily); the time of day must be specified.

When specifying the Knowledge Graph names in square brackets, you can use the wildcards "*" and "?"; the names are not case-sensitive.

Manual start of Garbage Collection

Alternatively, garbage collection can also be controlled via the Admin tool or by using the mediator REST api.

4.2.5.4. Betrieb unter Unix

In UNIX the server reacts to the following signals:

```
SIGTERM/SIGHUP
```

Shuts down the server

```
SIGUSR2
```

The server immediately begins to back up all Knowledge Graphs that are specified for backup in the jobs.ini file (see also the section on backups).

4.2.5.5. Betrieb im Cluster

The mediator can be operated in a cluster. A cluster environment usually mirrors the directories and therefore the Knowledge Graph constantly. If the part of the cluster on which the mediator is running fails, a new mediator that then manages access to the Knowledge Graph is started automatically

If the first mediator fails, it is possible that the mediator no longer has time to make the Knowledge Graph consistent and that the graph thus has an inconsistency and the "lock" file of the old mediator remains in the corresponding directory. To ensure that the new mediator is able to delete the "lock" file, the following parameter must be added to the mediator.ini file.

```
host=NameOfCluster
```

In this case, all mediators with this ini entry can also unlock locked volumes of other mediators that read the same value in the mediator.ini when started. "NameOfCluster" can be selected freely but must comply with the rules that apply to host names (no spaces, colon, or the like)

A consistency check of the volume is executed automatically when the mediator is started. To the extent possible, the Knowledge Graph is made consistent and operation continues as normal.

4.2.5.6. Problembehebung

If the i-views server was not shut down properly during operation (e.g. computer crash), then the locks remain in opened Knowledge Graphs. When a locked Knowledge Graph is opened, this lock is detected and removed, if possible.

If the mediator detects an inconsistency, then the Knowledge Graph can be checked and inconsistencies can be repaired to the extent possible by calling the mediator in the command line using the parameters -quickRecover / -recover.

If resolving the inconsistencies is, contrary to expectation, not possible, then a backup copy will need to be used.

4.2.5.7. Kommandos des BlockFileSystems

Die Befehle hinter -bfscommand ermögliche Oerationen auf dem BlockFileSystem und sind für Supportfälle vorgesehen. Ein solcher Befehl könnte zBsp so aussehen:

```
-bfscommand {target volume} quickCheck
```

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. Allgemeines

Die i-views bridge dient als Interface für Clients, die nicht über das i-views interne Protokoll kommunizieren. Folgende Protokolle werden unterstützt:

- REST: Stellt ein REST interface über HTTP oder HTTPS zur Verfügung
- Message Queue
  - MQTT: Stellt ein MQTT-Interface zur Verfügung
  - ZeroMQ: Stellt ein ZeroMQ-Interface zur Verfügung

Zur Aktivierung des gewünschten Modus, sind folgende Identifikatoren zu verwenden:

Identifikator	Identifikator (veraltet)	Description
rest	KHTTPRestBridge	REST-Interface
mqtt	-	MQTT-Interface
zmq	-	ZMQ-Interface

4.3.2. Gemeinsame Kommandozeilen-Parameter

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 <identifier> <portnumber>
```

Definiert den Port für den gewünschten Dienst.

Beispiel: REST-Bridge auf Port 8815 starten:

```
bridge -host server01:30000 -port rest 8815
```

```
-stop <hostname>
```

4.3.3. Konfigurationsdatei "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.

4.3.3.1. 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.

4.3.4. REST-Bridge

4.3.4.1. Einführung

Die REST-Bridge-Anwendung ermöglicht den Lese- und Schreibzugriff auf i-views über ein RESTful services architecture. Die Schnittstelle ist über HTTP oder HTTPS verfügbar.

Die REST-Bridge läuft innerhalb der Standard-Bridge von i-views.

Die Schnittstelle wird vollständig durch individuelle Konfigurationen im Knowledge Graph konfiguriert. Der Rückgabewert eines REST-Aufrufs ist eine beliebige Zeichenkette, normalerweise in einem Format, das der aufrufende Client leicht verarbeiten kann (z. B. XML oder JSON).

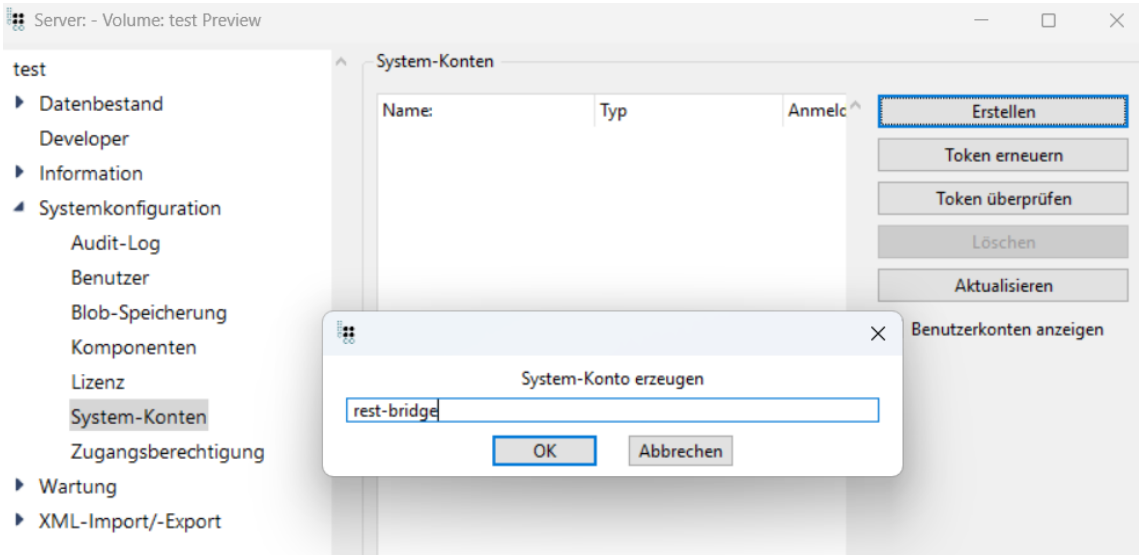
4.3.4.2. Installation

4.3.4.2.1. Volume vorbereiten

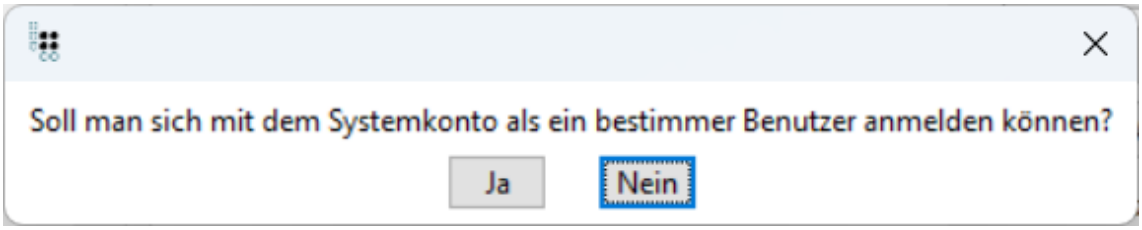
1. Anlegen eines System-Accounts für den Bridge-Service

Damit ein Bridge-Service auf ein Wissensnetz zugreifen darf, welches von einem Mediator-Service verwaltet wird, muss in dem Wissensnetz ein System-Konto für den Bridge-Service angelegt werden. Dies kann mit dem Admin-Tool erfolgen (unter Systemkonfiguration > System-Konten) oder mit dem Knowledge-Builder (Einstellungen/Zahnrad > Reiter "System" > System-Konten). Im Beispiel wird gezeigt, wie ein System-Konto mit dem Admin-Tool angelegt werden kann:

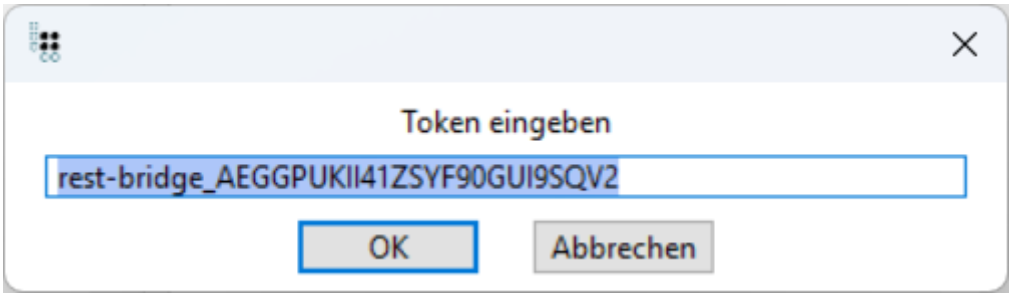
Schritt 1:



Schritt 2:



Schritt 3:

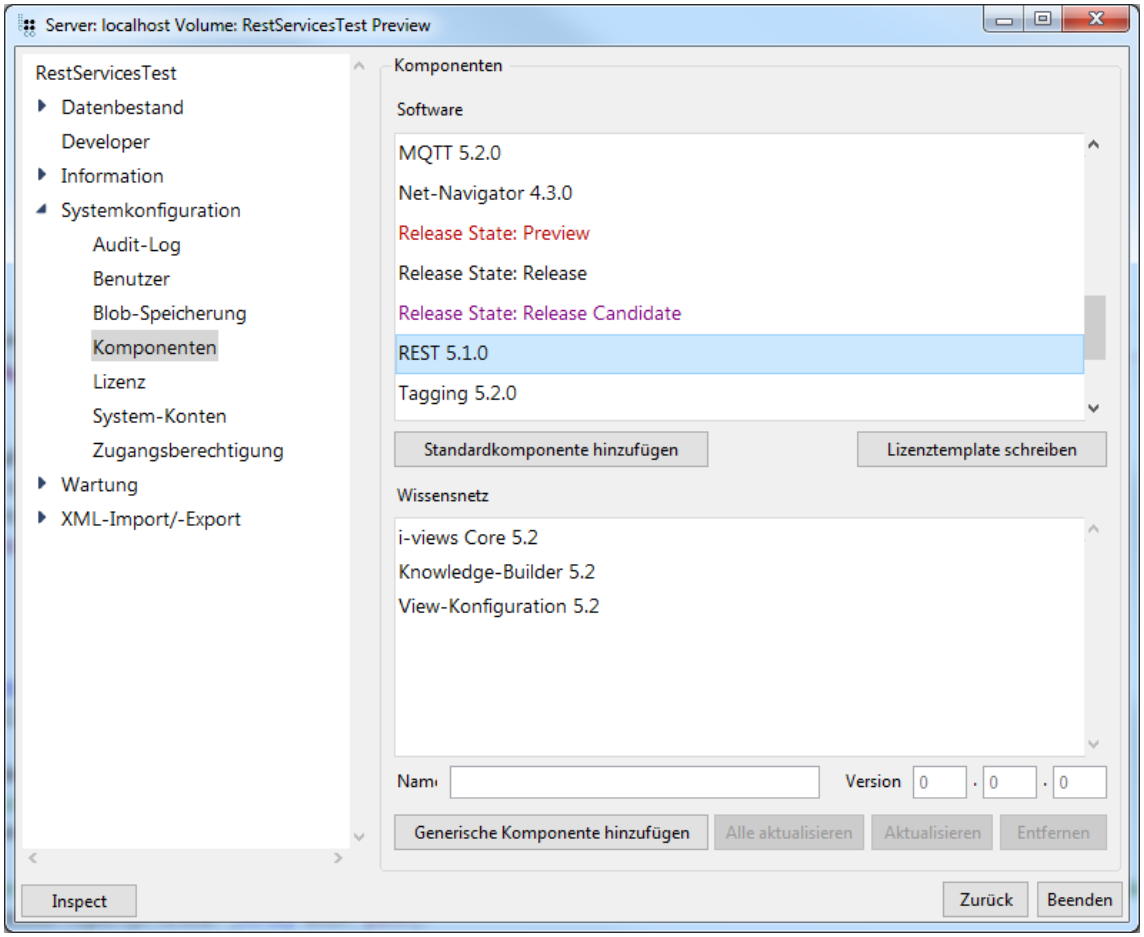


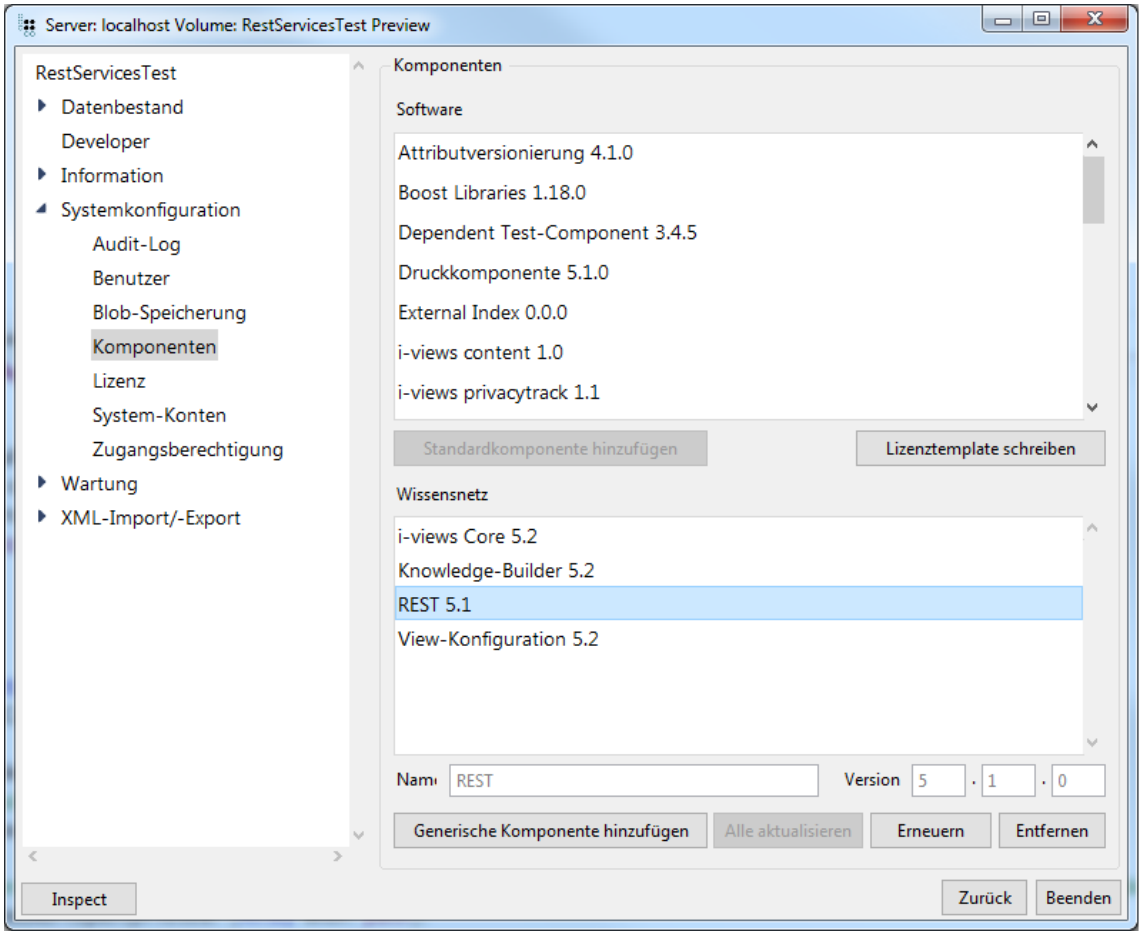
HINWEIS

Das im letzten Schritt angezeigte Anmelde-Token ("rest-bridge_...") wird bei der Konfiguration der Bridge (nächstes Kapitel) wieder benötigt. Deshalb sollte das Fenster "Token eingeben" geöffnet bleiben oder das Token an einem sicheren Ort gespeichert werden.

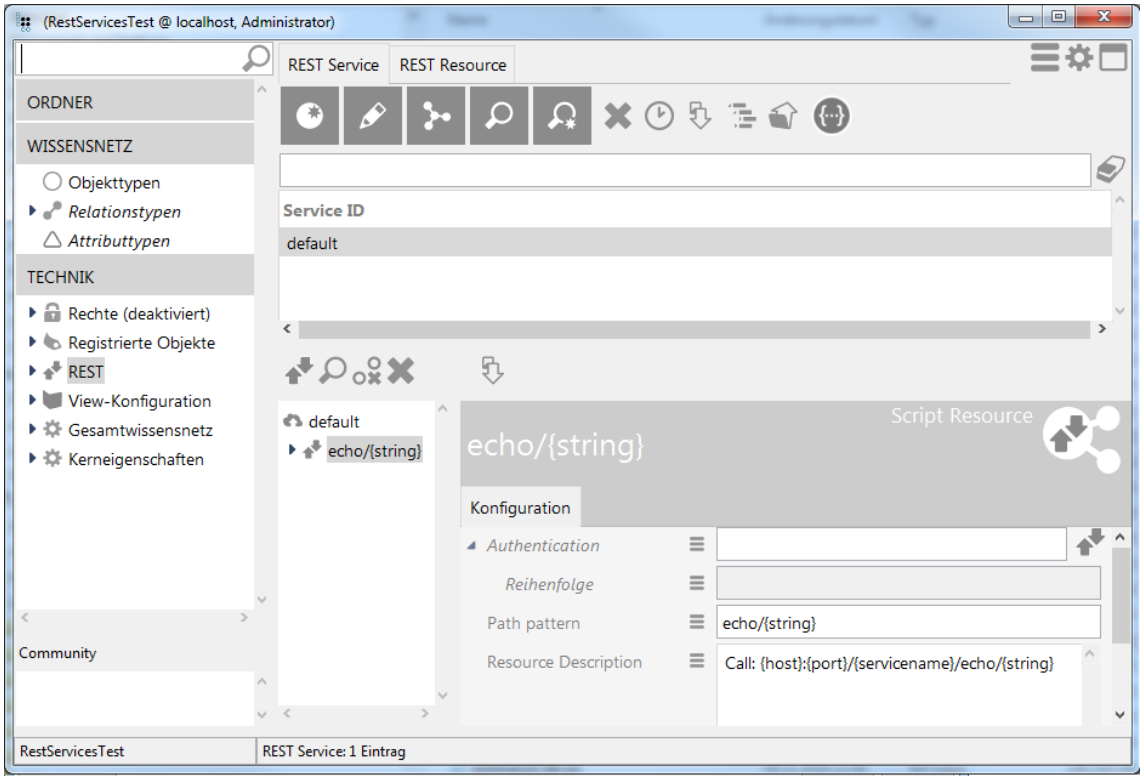
2. Aktivieren der REST-Komponente im Wissensnetz

Durch das Hinzufügen der Softwarekomponente "REST" im Admin-Tool wird im Wissensnetz das benötigte Schema für die Konfiguration der REST-Services 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 konfigurieren

Die REST-Schnittstelle wird durch die Standard-Bridge-Komponente von i-views bereitgestellt, sofern in der Konfigurationsdatei eine Kategorie `rest` eingetragen ist:

```
[rest]
;Name des Knowledge Graphs
volume=example-graph
;Optionale Liste von Interfaces, auf denen der Service erreichbar sein
soll.
;Standardwert ist http://0.0.0.0:8815
interfaces=http://0.0.0.0:8080
;Optionale Liste von Service-IDs
services=public
```

Falls ein Interface mit dem HTTPS-Protokoll angegeben wurde, müssen in der Konfigurationsdatei zusätzlich die Dateipfade für Zertifikat und privaten Schlüssel angegeben werden. Diese müssen im PEM-Format vorliegen und dürfen nicht passwortgeschützt sein.

```
[rest]
volume=example-graph
interfaces=https://0.0.0.0:8443,http://0.0.0.0:8080
;Name der .crt-Datei
```

```
certificate=bridge.crt
;Name der .key-Datei
privateKey=bridge.key
```

HINWEIS

Statt `rest` können auch die alte Bezeichnungen `KHTTPRestBridge` oder `KHTTPSRestBridge` verwendet werden. KHTTPRestBridge verwendet standardmäßig das HTTPS-Protokoll, falls keine Interface konfiguriert wurden.

Im Konfigurationsabschnitt der REST-Bridge 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. MQTT-Bridge

4.3.5.1. Einführung

Eine MQTT-Bridge verbindet sich zu einem MQTT-Broker und nimmt von diesem Nachrichten entgegen. Die im Knowledge-Graph hinterlegte MQTT-Konfiguration bestimmt dabei, welche MQTT-Topics abboniert werden (SUBSCRIBE) und wie der Empfang dieser Topics gehandhabt wird. An dieser Stelle werden Javascripts eingesetzt, die es dem Konfigurator erlauben, die Nachricht zu interpretieren, den Knowledge-Graph lesend und schreiben zuzugreifen sowie ggf. eine Reaktion in Richtung des Brokers zu formulieren (PUBLISH).

4.3.5.2. Konfiguration

Zur Nutzung der MQTT-Bridge muss zunächst die Komponente "MQTT" eingespielt werden.

4.3.6. ZMQ-Bridge

4.3.6.1. Einführung

Während MQTT stets dem PUBLISH/SUBSCRIBE-Muster sowie der Client/Server-Topologie folgt, ermöglicht ZeroMQ weitaus flexiblere Topologien und Kommunikationsmuster. Die Integration von ZeroMQ ermöglicht die Komposition von Laufzeit-Komponenten mittels PUBLISH/SUBSCRIBE, SEND/RECEIVE sowie PUSH/PULL-Pattern. Dazu stehen eine dedizierte Bridge-Art, Script-Funktionen sowie eingebaute Broker-Unterstützung im Mediator zur Verfügung.

4.3.6.2. Konfiguration

Zur Nutzung der ZMQ-Infrastruktur muss zunächst die Komponente "ZMQ" eingespielt werden. Weiterhin muss die shared library "libzmq" installiert sein.

4.3.6.3. Konfiguration des Mediators

Um die im Mediator eingebaute Broker-Funktionalität zu nutzen, muss ZMQ in der ini-Datei des Mediators eingeschaltet werden. Außerdem wird eine zusätzliche ini-Datei zur Konfiguration des Services bereitgestellt werden.

ZMQ-Service in der mediator.ini einschalten:

```
[Default]

...

startZMQService=true
startZMQWebSocket=true
```

Beispiel für eine zmqservice.ini mit üblichen Interaktionsmustern

```
[Default]
services=routerDealer,pushPull,pubSub

[routerDealer]
front=ROUTER@tcp://0.0.0.0:40000
back=DEALER@tcp://0.0.0.0:40001

[pushPull]
front=PULL@tcp://0.0.0.0:50000
back=PUSH@tcp://0.0.0.0:50001

[pubSub]
front=XSUB@tcp://0.0.0.0:60000
back=XPUB@tcp://0.0.0.0:60001
```

4.4. Job-Client

4.4.1. Allgemeines

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.

Clients können im Knowledge Graph Jobs hinzufügen, die von JobClients abgearbeitet werden. Die Jobs sind entweder synchron (Client wartet darauf, dass der Job bearbeitet wurde) oder asynchron (Client wartet nicht).

4.4.1.1. Funktionsweise

In dem vom i-views-Mediator bereitgestellten geteilten Objektraum werden die Aufträge der Clients an die Services in sogenannten Pools abgelegt. Sobald ein Job-Client frei ist, führt er den nächsten verfügbaren Job ist. Es wird dabei der älteste Job zuerst ausgeführt. Falls mehrere Job-Clients frei sind, wird der Job von einem beliebigen Job-Client ausgeführt.

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.

Für den Client ist es transparent, welcher Job-Client seinen Auftrag ausführt. Für den Job-Client transparent, 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.

4.4.2. Konfiguration des Job-Clients

4.4.2.1. Konfigurationsdatei "jobclient.ini"

Die Konfiguration des Job-Clients wird in der Ini-Datei vorgenommen. Falls diese nicht durch den Aufrufparameter "-inifile" beim Start des Job-Clients spezifiziert ist, wird "jobclient.ini" als Konfigurationsdatei verwendet. Die selbe Konfiguration kann auch über Umgebungsvariablen vorgenommen werden.

4.4.2.1.1. Allgemeine Parameter

Die folgenden Parameter können konfiguriert werden:

Parameter	Beschreibung	Syntax
host	Name oder IP-Adresse und Port des Servers.	`host=<host name:port number>`
volume	Der Name des Knowledge-Graphs, auf dem gearbeitet wird.	`volume=<volume name>`

jobPools	Legt fest, welche Jobs der Job-Client bearbeiten soll. Die Namen der zu startenden Job-Pools werden in Komma-separierter Form angegeben.
	
	Die Job-Pools sollten nach Kategorie angegeben werden (z.B. "index"). Alternativ können auch einzelne Pool-Namen angegeben werden (z.B. "KScriptJob").
	
	Siehe Job-Pool-Typen für die vollständige Liste möglicher Typen.
		`jobPools=<job name1>[,<job name2>, ...]`
		
		Beispiel:
		
		`jobPools=script, query`

cacheDir	Die Angabe des Verzeichnisses, in dem der Cache des Job-Clients abgelegt wird.	`cacheDir=<directory>`
maxCacheSize	Maximale Größe des Caches.	`maxCacheSize=<size in MB>`
shutDownTimeout	Wartezeit für das Beenden des aktiven Jobs beim Herunterfahren des Job-Clients. Nach Ablauf dieser Zeit wird ein aktiver Job abgebrochen. Der Standardwert beträgt 10 Sekunden.	`shutDownTimeout=<seconds>`
enableLowSpaceHandler	Diese Option aktiviert den Handler für Situationen mit wenig freiem Hauptspeicher. Für große Knowledge-Graphen sollte diese Option immer aktiviert sein.	`enableLowSpaceHandler=true/false`

Parameter	Beschreibung	Syntax
useProxyValueHolder	Mit dieser Option lässt sich steuern, ob der Job-Client den Index-Zugriff per RPC ausführt (`true`) oder die Indizes in den Speicher lädt (`false`). Der Standardwert ist `true`. Ist die Option deaktiviert, wird die RPC-Last auf dem Mediator reduziert. Ist sie aktiviert, werden weniger Updates vom Mediator an die Clients gesendet. Dabei muss jedoch sichergestellt sein, dass dem Job-Client genügend Speicher zur Verfügung steht. Wenn der Job-Client für Schreib-Jobs konfiguriert ist, hat diese Option keine Wirkung, da der Index-Zugriff dann immer per RPC erfolgt. Wird der Wert auf `false` gesetzt, erscheint beim Start eine Meldung im Log.	`useProxyValueHolder=true/false`
loadIndexes	Indizes in den Speicher laden (Standardwert: `false`). Im Gegensatz zur Option useProxyValueHolder bleibt dabei der Schreibzugriff möglich, wenn der Wert auf `true` gesetzt wird. Die Option kann für alle Clients aktiviert werden, auch für den Knowledge Builder.	`loadIndexes=true/false`
name	Unter diesem Namen wird der Job-Client im Admin Tool in der Übersichtsliste aller Job-Clients identifiziert.	`name=<Job-Client name>`
scheduledJobs	Eine Komma-separierte Liste von Jobs, die zeitgesteuert ausgeführt werden sollen.	`scheduledJobs=<Job name 1>[, <Job name 2>, ...]`

4.4.2.1.2. Speichereinstellungen

Die folgenden drei Parameter dienen der Konfiguration der Speicherzuweisung und -nutzung. Werte können entweder in Megabyte oder in Bytes angegeben werden, wobei angenommen wird, dass Werte unter 1048576 sich auf Megabyte beziehen.

Parameter	Beschreibung	Syntax
maxMemory	Maximal zulässige Nutzung des Hauptspeichers. Mindestens 50 MB, standardmäßig der gesamte verfügbare Hauptspeicher.	`maxMemory=<integer, in MB>`
baseMemory	Basisspeichernutzung, ab der die Bemühungen zur Freigabe von Speicher verstärkt werden. Standardmäßig der kleinere Wert von `0,3 * maxMemory` oder 1GB. (Alias: "growthRegimeUpperBound")	`baseMemory=<integer, in MB>`
freeMemoryBound	Überschreitet der Speicher, der belegt, aber nicht mehr benötigt wird, diese Grenze, wird er wieder zur Verwendung freigegeben.	`freeMemoryBound=<integer, in MB> [10]`
minAge	Mindestdauer (in Sekunden), die ein Cluster im Speicher verbleibt. Ein Cluster ist eine Menge von Objekten, die immer gemeinsam als Einheit geladen werden (z.B. ein Individuum mit all seinen (Meta-)Eigenschaften). Cluster, die längere Zeit nicht genutzt wurden, werden bei Bedarf entladen.	`minAge=<Integer> [30]`
unloadInterval	Mindestdauer (in Sekunden) zwischen dem Entladen zweier Cluster.	`unloadInterval=<Integer> [10]`
unloadSize	Mindestanzahl geladener Cluster, ab der entladen wird.	`unloadSize=<Integer> [4000]`
keepSize	Anzahl der Cluster, die beim Entladen behalten werden.	`keepSize=<Integer> [3500]`

4.4.2.1.3. Konfiguration des Lucene-Servers

Lucene wird über einen Job-Client eingebunden, dessen `jobclient.ini` entsprechend konfiguriert werden muss. Im Folgenden ist eine exemplarische Konfiguration dargestellt:

```
[lucene]
directory=lucene-index
port=5100
pageSize=100
; Wildcards at the start of a word are prohibited by default as they are
very slow
```

```
; Allow in this configuration
allowLeadingWildcards=true

[JNI]
classPath=lucene-6.4.1\\core\\lucene-core-6.4.1.jar;lucene-6.4.1\\
analysis\\common\\lucene-analyzers-common-6.4.1.jar;lucene-6.4.1\\
analysis\\queries\\lucene-queries-6.4.1.jar;lucene-6.4.1\\analysis
\\queries\\lucene-queries-6.4.1.jar
```

Das Verzeichnis *lucene-6.4.1* enthält die Lucene-Binärdateien. Der Index wird im Verzeichnis *lucene-index* abgelegt.

4.4.2.1.4. Zeitgesteuerte Jobs

Jobs können durch den Job-Client zeitgesteuert ausgeführt werden. Der Job-Client legt die Jobs dann zum festgelegten Zeitpunkt an.

HINWEIS	In der Regel sollte nur ein einzelner Job-Client Jobs zeitgesteuert einplanen. Zeitgesteuerte Jobs werden wie reguläre Jobs behandelt und können auch von anderen Job-Clients ausgeführt werden.

Um einzelne Jobs in der Konfigurationsdatei zu konfigurieren, muss für jeden Job ein eigener Abschnitt angelegt werden. Diese Abschnitte beginnen jeweils mit dem Namen des Jobs in eckigen Klammern. Anschließend folgen die jeweiligen Parameter des Jobs.

Die Job-Namen müssen am Parameter `scheduledJobs` aufgeführt sein.

Beispiel:

```
scheduledJobs=Job-Name1,Job-Name2

[Job-Name1]
<Parameter>=<value>
...

[Job-Name2]
...
```

Parameter zeitgesteuerter Jobs

Parameter	Beschreibung	Syntax
jobPool	Der Pool, in den der Job eingefügt werden soll. Legt den Typ des zeitgesteuerten Jobs fest.	`jobPool=<Job-Pool-Name>`

time	Zeitpunkt, zu dem der Job erstmalig ausgeführt werden soll.	`time=<Time>`
		Beispiel:
		`time=22:15`

interval	Legt fest, wie häufig der Job ausgeführt werden soll. Hierbei wird die Wartezeit zwischen der Beendigung eines Jobs und seiner nächsten geplanten Ausführung angegeben. (d=Tage, h=Stunden, m=Minuten, s=Sekunden)	`interval=<Exact time>`
command	Nur für KExternalCommandJob: Name einer externen Batch-Datei, die durch den Job ausgeführt werden soll.	`command=<File name.ext>`
scriptName	Nur für KScriptJob: Registrierungsschlüssel eines internen Skripts, das durch den Job ausgeführt werden soll.	`command=<Script resource>`
unique	Befindet sich bereits ein Job dieses Typs in der Warteschlange, wird kein weiterer Job eingeplant.	`unique=true/false`
user	Name eines Benutzerkontos, unter dem der Job ausgeführt werden soll.	`user=<User name>`
arguments	Nur für KExternalCommandJob: Argumente, die beim Aufruf des Skripts übergeben werden.	`arguments=<Argument1 [Argument2 ...]>`

4.4.2.1.5. Job-Pool-Typen

Die folgenden Typen von Job-Pools sind verfügbar:

Kategorie	Pool	Beschreibung
blob	KBlobGCJob	Führt eine BLOB-Garbage-Collection durch.
blob	KSwitchBlobStoreJob	Überträgt BLOBs zwischen Stores.

Kategorie	Pool	Beschreibung
index	KAddAllToIndexJob	Fügt alle Eigenschaften eines Eigenschaftstyps dem Index hinzu.
index	KLightweightIndexJob	Aktualisiert den Index einer Eigenschaft.
index	KRemoveIndexJob	Entfernt alle Eigenschaften eines Eigenschaftstyps aus dem Index.
index	KSyncIndexJob	Aktualisiert alle Eigenschaften eines Eigenschaftstyps.
index	KExternalIndexUpdateJob	Aktualisiert einen externen Index (z.B. IAS, Elasticsearch).
lucene, luceneAdmin	KLuceneAdminJob	Verwaltet Lucene-Indizes.
lucene, luceneQuery	KLuceneQueryJob	Führt Lucene-Abfragen aus.
script	KScriptJob	Führt ein Skript des Knowledge-Graphs aus.
script	KScriptTriggerJob	Führt ein Trigger-Skript aus.
print	KPrintJob	Erzeugt durch Anwendung einer Druckkonfiguration auf semantische Elemente einen Dokument-BLOB.
query	KQueryJob	Führt eine Abfrage aus und speichert das Ergebnis.
-	KExternalCommandJob	Führt einen externen Befehl aus (Executable, Shell-Skript).
-	KExtractBlobTextJob	Extrahiert den Text aus einem BLOB.

Index-Jobs

Die Indizierungs-Jobs sollten nur von einem einzigen Job-Client ausgeführt werden.

KExternalCommandJob

Mit den **KExternalCommandJobs** können ausführbare Programme gestartet werden, die Dateien verarbeiten oder verändern oder einfach nur aufgerufen werden sollen. Ein Job kann durch einen Skriptaufruf eingefügt werden.

Dies findet Über die JavaScript Klasse `ExternalCommandJob` statt, mit der Jobs direkt erzeugt und auch für die zeitgesteuerte Ausführung einplant werden können.

```
const externalCommandJob = new $k.ExternalCommandJob()

externalCommandJob.setCommand("filename.ext")
```

```
externalCommandJob.insert()
```

Vor der Ausführung eines ExternalCommandJobs prüft der Job-Client, ob der gesetzte Befehl in der Liste der zulässigen Befehle vorhanden ist und der entsprechende Befehl im Arbeitsverzeichnis des Job-Clients vorhanden ist. Die Liste der zugelassenen Befehle kann in der Konfiguration im Abschnitt `[ExternalCommandJob]` mit der Option `allowedCommands` als Komma-separierte Liste angegeben werden. Verzeichnisangaben werden sowohl in der Konfiguration als auch im Dateinamen des setCommand-Aufrufs ignoriert.

Wird ein aktuell anstehender Job von keinem Job-Client zur Bearbeitung angenommen, ist die Job-Warteschlange für den Benutzer, der den Job eingefügt hat, blockiert. Dieser Job muss dann manuell gelöscht werden.

4.4.2.2. Performance-Optimierungen

4.4.2.2.1. Vorladen

Beim Start können Job-Clients bei entsprechender Konfiguration auswählbare Strukturen vorladen. Dieser Vorgang erhöht den Speicherbedarf des Job-Clients, ermöglicht ihm aber auch eine schnellere Ausführung.

In der Ini-Datei des Job-Clients muss der Eintrag **keepClusterIDs** angegeben werden. Mögliche Werte für diesen Eintrag sind:

- **index**: In den Einstellungen der Indexkonfiguration gibt es die Option *Job-Client soll Index in den Hauptspeicher laden*. Ist sie für einen Index aktiviert, wird ein Teil der Indexstruktur beim Start geladen.

HINWEIS	Wird nur verwendet, wenn *useProxyValueHolder* auf `false` gesetzt ist. Andernfalls sendet der Job-Client RPCs, anstatt den Index in den Speicher zu laden.

- **protoOfSizes**: Die Anzahl der Individuen pro Begriff wird bereits beim Start ermittelt.
- **accessRights**: Das Wurzelobjekt des Rechtesystems wird in den Speicher geladen.

Zur Performance-Verbesserung trägt es ebenfalls bei, den Knowledge-Graph-Cache für den Job-Client zu aktivieren.

Beispiel für Einträge in der Ini-Datei:

```
[Default]
...
useProxyValueHolder=false
keepClusterIDs=index,protoOfSizes,accessRights
cacheDir=jobcache
maxCacheSize=1000
```

```
...
```

4.5. Batch-Tool

Das Batch-Tool ermöglicht das Ausführen von administrativen Befehlen und das Importieren/Exportieren von Daten per Kommandozeile. Der gewünschte Befehl muss als Kommandozeilen-Parameter ausgeführt werden, gefolgt von Kommando-spezifischen Parametern. Darüber hinaus ist es möglich, Serien von Kommandos auszuführen.

4.5.1. Allgemeine Kommandozeilen-Parameter

Alle Befehle teilen sich dieselben gemeinsamen Parameter:

```
-host
```

URL oder Host-Name und Portnummer des Servers

```
-volume
```

Name des Knowledge-Graph Volumes

```
-user
```

Überholt. Name des Benutzer-Accounts, welcher aktiviert werden soll, wenn die Befehlszeile ausgeführt wird. Stattdessen ist der Authentifizierungsschlüssel zu verwenden (siehe Konfigurationsdatei-Optionen), um ein Datenleck des Benutzers/Passwortes an andere Prozesse zu vermeiden.

```
-password
```

Passwort des Benutzers

4.5.2. Konfigurationsdatei-Optionen

4.5.2.1. Angaben zum Knowledge-Graph

```
host
```

URL oder Hostname des Mediators

```
volume
```

Name des Knowledge-Graph Volumes.

```
authentication
```

Authentifizierungsschlüssel des Systemkontos. Der Authentifizierungsschlüssel muss mit dem Admin-Tool erstellt werden.

4.5.3. Befehle

4.5.3.1. Importieren oder Exportieren von gemappten Daten

Die folgenden Befehle ermöglichen den Import oder den Export von Daten anhand eines im Volume definierten Mappings.

Das folgende Beispiel exportiert die Daten in eine Datei `data.csv` mithilfe des registrierten Mappings `example.export`:

```
batchtool -volume example -exportMapping example.export -file data.csv
-errorLogFile export-errors.log
```

4.5.3.1.1. Kommandozeilen-Parameter

Entweder

```
-exportMapping {ID}
```

Exportieren gemappter Daten unter Angabe des Registrierungsschlüssels

oder

```
-importMapping {ID}
```

Importieren gemappter Daten unter Angabe des Registrierungsschlüssels

4.5.3.1.2. Zusätzliche Kommandozeilen-Parameter

```
-encoding {name}
```

Name der Zeichen-Encodierung, z. B. utf-8. Bestimmt die Verbindungs-Codierung für Datenbank-Abbildungen und die Text-Codierung für Text-Dateien (z. B. CSV-Dateien).

```
-errorLogFile {logfile}
```

Dateiname des Fehlerberichts

```
-mapping {ID}
```

Registrierte ID des Daten-Mappings

```
-triggers {true/false}
```

Ein- oder Abschalten der Trigger während des Imports

```
-queryParameter {parameter name} {parameter value}
```

Setzt den Query-Parameter namens {parameter name} auf den Wert {parameter value} (nur beim Export einsetzbar)

4.5.3.1.3. Kommandozeilen-Parameter (nur für Datei-Abbildungen)

```
-caption {true/false}
```

Wahr, wenn die Tabelle Spaltenbezeichner enthält oder enthalten sollte

```
-file {filename}
```

Mögliche Werte: Name der zu im-/exportierenden Datei

```
-separator {separator string}
```

Zellen-Trennzeichen

4.5.3.1.4. Kommandozeilen-Parameter (nur für Datenbank-Mapping)

```
-binding {name value}
```

Name-Wert-Paar für Datenbank-Bindings. Werden nur von Datenbank-Abbildungen benutzt, welche ein benanntes Binding in der Abfrage enthalten

```
-dbEnvironment {string}
```

Zeichenkette für Datenbank-Verbindung

```
-dbHostname {string}
```

Datenbank-Host; nur für MySQL.

```
-dbPassword {string}
```

Datenbank-Passwort

```
-dbUsername {string}
```

Datenbank-Nutzername

4.5.3.2. Import und Export von RDF-Dateien

Die folgende Befehle ermöglichen einen Import oder einen Export von Dateien im Format RDF(S) oder OWL.

4.5.3.2.1. Kommandozeilen-Parameter

Entweder

```
-exportRDF {filename}
```

Exportiert eine Datei im Format RDFS oder OWL

oder

```
-importRDF {filename}
```

Importiert eine Datei im Format RDF, RDFS oder OWL

HINWEIS	Der Export verwendete immer RDFS oder OWL. Es ist nicht möglich, rein RDF-

basierte Daten zu exportieren.

4.5.3.2.2. Zusätzliche Kommandozeilen-Parameter

```
-parameter {name} {value}
```

Setzen eines Parameters für die Steuerung des RDF-Imports/Exports

```
-query {ID}
```

Setzen der Abfrage, welche die zu exportierenden Elemente enthält.

```
-queryParameter {name} {value}
```

Setzen des Abfrage-Parameters

4.5.3.2.3. Export-Parameter

Die folgenden Export-Parameter können gesetzt werden mithilfe der Angabe "-parameter {name} {value}"

```
abbreviateURIs
```

Abgekürzte URIs mittels rdf:ID und xml:base

```
baseURI
```

Basis-URI

```
blobHash
```

"True", falls zusätzliche Kommentare exportiert werden sollen

```
exportFrameIDs
```

Exportiert Objekt Frame-IDs

```
exportLabels
```

Exportiert den Namen als rdfs:label

```
exportMeta
```

"True", wenn Metaeigenschaften exportiert werden sollen. Diese werden dann als Vergegenständlichung exportiert.

```
exportPropertyIDs
```

Exportiert die IDs der Eigenschaften

```
exportReferencedTopics
```

"True", wenn externe Relationsziele als Stümpfe exportiert werden sollen

```
ignoreStoredIdentifier
```

Wenn auf "true" gesetzt, wird der RDF-Locator immer generiert. Wenn auf "false" gesetzt, wird das rdf:about/rdf ID Attribut verwendet, falls vorhanden.

```
qualifier
```

XML-Qualifier, welcher an die Basis-URL gebunden wird

```
schemaNameSpace
```

Standard Schema XML-Namensraum

```
schemaOnly
```

"True", wenn nur Typen exportiert werden sollen

```
updatePersistentIdentifier
```

"True", wenn die generierten Werte für rdf:ID / rdf:about als Attributwerte vergegenständlicht/persistiert werden sollen

```
useFrameURIs
```

"True", wenn URIs, welche aus der Objekt-ID erzeugt wurden, zur Identifizierung der Objekte verwendet werden sollen

```
useKRDF
```

"True", wenn die KRDF-Eigenschaften exportiert werden sollen (bspw. krdf:internalName)

```
useOWL
```

"True", wenn das OWL-Vokabular verwendet werden soll

4.5.3.2.4. Import-Parameter

```
activateTriggers
```

"True", wenn Trigger aktiviert sein sollen

```
allowExtensiveRestructurings
```

Durchführen von Schemaänderungen, auch wenn sie viele Objekte betreffen

```
avoidDuplicateProperties
```

"True", wenn Eigenschaften-Dupletten übersprungen werden sollen

```
baseURI
```

Basis-URI

```
changeCompatibleInstanceTypes
```

Ändern von Objekttypen, auch wenn der aktuelle Typ kompatibel mit dem definierten Typ ist

```
enableCreateSchema
```

"True", wenn Schema erzeugt/modifiziert werden können soll

```
enforceInverseRelationConcepts
```

"True": Erzeugt inverse Relationstypen, falls nicht definiert.

"False": Relationen ohne Inverse werden symmetrisch sein.

```
importCommentsAsAttributes
```

Importiert rdfs:comment als Attribut (muss im Schema definiert sein)

```
importReferencedResources
```

Importiert referenzierte, externe RDF-Ressourcen

```
importValuesAsAttributes
```

Importiert rdf:value als Attribut (muss im Schema definiert sein)

4.5.4. Skripte ausführen

Dieser Befehl ermöglicht das Ausführen beliebiger Skripte, welche in JavaScript geschrieben sind.

4.5.4.1. Kommandozeilen-Parameter

```
-script {registered script ID}
```

Führt ein registriertes Skript aus

```
-scriptfile {filename}
```

Führt ein Skript aus, welches aus einer Datei ausgelesen wird

4.5.4.2. Zusätzliche Kommandozeilen-Parameter

```
-argument {argument name} {argument value}
```

Setzt beim globalen Objekt "arguments" die Eigenschaft {argument name} mit den Wert {argument value}.

Beispiel:

1. Aufruf des Batchtools mit `-argument id test-1234`
2. Zugriff im Script auf die ID mit `const id = arguments.id`

```
-encoding {encoding name}
```

Setzt die Output-Codierung auf {encoding name}

```
-errorLogFile {filename}
```

Dateiname des Fehlerberichts

```
-output {file name}
```

Ausgabe allen Outputs in Datei namens {file name}

```
-stdout
```

Ausgabe allen Outpus nach stdout und aller Fehler nach stderr. Einträge werden nur in die Log-Datei geschrieben.

4.5.5. Importieren oder Exportieren von Schema

Die folgenden Befehle ermöglichen es, Schema in den Knowledge-Graphen zu importieren oder aus dem Knowledge-Graphen zu exportieren. Dies beinhaltet

- Typen
- View-Konfigurationen
- REST-Service Definitionen
- Registrierte Abfragen, Skripte, Mappings, Ordner

- Trigger
- Zugriffsrechte
- Index-Konfigurationen und Filter

4.5.5.1. Kommandozeilen-Parameter

Entweder

```
-exportSchema {schema file}
```

Exportiert das Schema als Datei

oder

```
-importSchema {schema file}
```

Importiert Schema von einer Datei

4.5.5.2. Zusätzliche Kommandozeilen-Parameter

```
-filter {filter file}
```

Spezifiziert eine Filter-Datei (siehe nächstes Kapitel)

4.5.5.3. Export-Filter

Der Filter definiert, welche registrierten Objekte exportiert werden. Jede Zeile definiert einen positiven Flag (beginnt mit "+") und einem negativen Flag (beginnt mit "-"), einer Kategorie und einem Muster, welche gegen die registrierte ID geprüft wird. Die Kategorie und das ID-Muster können dabei Wildcards "*" für eine Teilzeichenkette oder Raute "#" für ein einzelnes Zeichen enthalten. Eine Zeile mit Semikolon ";" am Zeilenanfang wird ignoriert.

Objekte ohne ID treffen nur zu, wenn das ID-Muster "*" ist.

HINWEIS	Objekte, welche mit keinem der Filter übereinstimmen, werden exportiert.

Schema-Teilnetze werden gegen den "RDF:about"-Wert des Top-Typs geprüft (z. B. "http://www.intelligent-views.de/kinfinity/component/rest/4.0/type/rest-ConfigTopConcept")

Mögliche Kategorien:

accessRights, dataConnections, editorDetection, indexers, indexFilter, ldap, license, mappings, organizingFolder, printConfigurations, queries, schema, scripts, topicCollection, triggers, unknown.

Beispiel:

```
+ queries custom.*
- * *
```

Dies exportiert ausschließlich Abfragen, deren ID mit dem Muster "custom.*" übereinstimmen.

4.5.6. Importieren von Lizenzen

Dieser Befehl ermöglicht den Import von Lizenz-Dateien. Dies kann notwendig sein im Rahmen von Installationsroutinen, weil andere Befehle aufgrund der abgelaufenen Lizenz nicht funktionieren werden.

Kommandozeilen-Parameter

```
-importLicense {license file}
```

Importiert die Lizenz aus einer Datei.

4.5.7. Upgrade von Komponenten

Dieser Befehl ermöglicht es, Software- oder Modell-Komponenten eines Volumes upzugraden. Es ist darüber hinaus möglich, das von Komponenten mitgebrachte Schema neu anzulegen.

4.5.7.1. Kommandozeilen-Parameter

```
-upgradeComponents {optional component names}
```

Upgradet spezifische Komponenten oder alle Komponenten, wenn keine festgelegt sind.

Mögliche Komponenten-Namen sind:

iviewsProducts, kintelligence, knowledgeBuilder, mqtt, netNavigator, printing, rest, translator, viewConfigMapper

4.5.7.2. Zusätzliche Kommandozeilen-Parameter

```
-updateSchema
```

Re-initialisiert das Schema der Komponenten

4.5.8. Ausführen einer Serie von Befehlen

Dieser Befehl ermöglicht es, eine Serie von Befehlen auszuführen. Dies ist effizienter, als das Batch-Tool für jeden Befehl separat auszuführen, weil Daten, welche bereits durch einen vorangegangenen Befehl geladen wurden, nicht nochmals geladen werden müssen.

4.5.8.1. Kommandozeilen-Parameter

```
-batchFile {file}
```

Durchführen aller Befehle, welch in der Stapelverarbeitungsdatei aufgelistet sind. Diese müssen UTF-8 codiert sein. Die Stapelverarbeitungsdatei darf keine Kommandozeilen-Parameter enthalten (host, volume etc.)

Beispiel:

```
batchtool -volume example -batchFile commands.txt
```

Mit commands.txt, welche die folgenden Zeilen enthält:

```
-exportMapping example.export1 -file data1.csv -errorLogFile export1-
errors.log
-exportMapping example.export2 -file data2.csv -errorLogFile export2-
errors.log
```

Dies wird zwei Exporte durchführen.

4.5.9. Beispiel: Import per Batch-Tool

Für den Import via Batchtool benötigt man den Zugang zu den zu importierenden Daten und eine Netzwerkverbindung zum Mediator. Wenn das Batch-Tool nicht auf dem Server ausgeführt wird, auf dem sich der Mediator für den Import befindet, müssen die folgenden Angaben in der batchtool.ini angepasst werden:

- Adresse/URL des Servers ("host=")
- Portnummer des Servers ("port=")
- Name des Volumes ("volume=")
- Token für Authentifizierung, zu erstellen mittels Admin-Tool ("authentication=")

Für den einmaligen Aufruf des Batchtools mit Steuerdatei (bspw. "import.data") in der Kommandozeile ist folgendes einzugeben:

```
batchtool -batchFile import.data
```

Falls das Batchtool nicht im selben Verzeichnis ist wie das Arbeitsverzeichnis der Kommandozeile (oder des Scheduled Tasks), so muss zumindest die ini-Datei sich im Arbeitsverzeichnis befinden oder alternativ als Parameter übergeben werden:

```
D:\\PFAD\\batchtool\\batchtool -ini D:\\PFAD\\batchtool\\batchtool.ini
-batchFile import.data
```

"PFAD" bezieht sich hierbei auf den Rechner, von dem der Aufruf erfolgt.

Die Steuerdatei kann leicht generiert werden und sieht folgendermaßen aus:

```
-importMapping MAPPING1 -file EXCELDATEI1 -errorLogFile MAPPING1-
errors.log
-importMapping MAPPING2 -file EXCELDATEI2 -errorLogFile MAPPING2-
errors.log
-importMapping MAPPING3 -file EXCELDATEI3 -errorLogFile MAPPING3-
errors.log
```

- `MAPPINGx` = registrierte Name des Import-Mappings in i-views
- `EXCELDATEIx` = Dateiname, ggfs. mit Pfad

Zum regelmäßigen Ausführen kann per Betriebssystem ein "Scheduled Task" (unter Windows die Funktion "Aufgabenplanung", unter Linux "cron") eingerichtet werden, der den Aufruf enthält.

Wenn der Import nach einem zuvor stattfindenden Export aus einem anderen System ausgeführt werden soll, dann kann man per Kapselung der Aufrufe in einer Batch-Datei (*.bat, *.cmd oder *.ps1) sicherstellen, dass der Import nur dann stattfindet, wenn der Export erfolgreich war.

4.6. Blob-Service

4.6.1. Einführung

The blob service is used to store the data of large files outside the Knowledge Graph but links to the file attributes in which these file contents are supposed to be stored. This has several advantages:

- It has the effect that the Knowledge Graph only receives the semantic information that is based on files and remains easy to backup and transfer.
- Storage locations of the Knowledge Graph and file contents can be configured differently.
- Several blob services can be connected to one Knowledge Graph, so that one storage location can be provided for each attribute definition.

The following chapter explains how to set up and operate blob services.

4.6.2. Konfiguration

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:

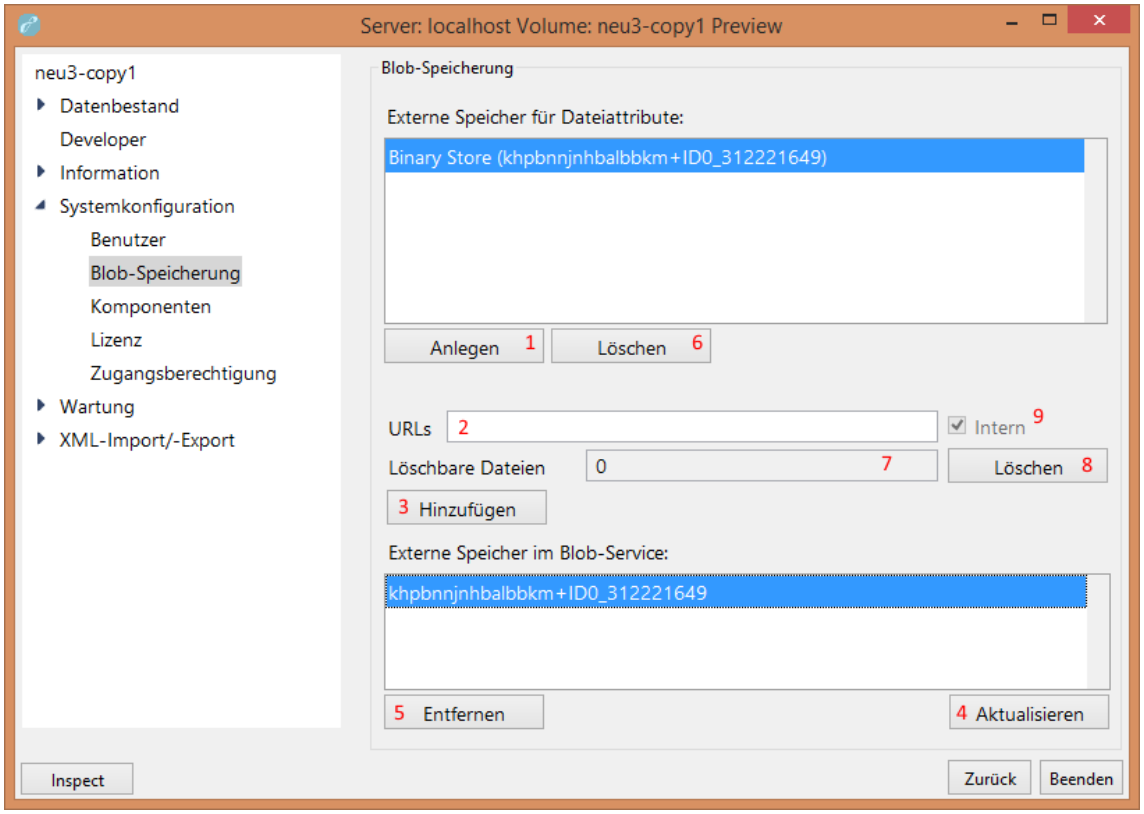
```
interfaces=https://0.0.0.0:30000
```

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.

To link the blob service with a blob store in the Knowledge Graph, the Admin tool offers the required tools under "System configuration > Blob storage":



Clicking on "Create" (1) creates a new logical store. After that, enter the URL (2) of the blob service specified in the ini file and then click on "Add" (3). The newly created blob store for external storage of file attributes is then linked to the blob service, which you can check by clicking on "Update" (4) in the lower display area.

You can also specify a comma-separated list of alternative URLs in the "URLs" area (2). For alternative URLs, i-views prefers a connection via a loop-back device where possible.

The "Deletable files" area (7) displays the number of files that are no longer required from the Knowledge Graph perspective. Use "Delete" (8) to de-reference them in the blob service and remove them if appropriate.

The indicator "Internal" (9) shows that this is a store that is integrated into a mediator. Internal stores are automatically transferred with the volume during a volume transfer (upload, download,

copy, backup, recover).

If you want to remove the link between a blob store and a blob service, select the desired blob store in the list "External stores in the blob service" and click "Remove" (5). Following that, you can select the blob store in the top section "External storage for file attributes" and then click "Delete" (6) to remove it completely. Alternatively, you can specify a new URL to link the blob store to another blob service.

HINWEIS	By removing a blob store's link to a blob service, all files stored therein are lost.

4.6.3. SSL Zertifikate

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, z.B. mit OpenSSL:

```
openssl rsa -in input.kez -out private/server.key -outform PEM
```

4.7. Login mit OAuth 2.0

Users of the Knowledge-Builder and the Admin-Tool can be authorized with the OAuth 2.0 framework. This requires an external authorization server that provides the tokens to access the Knowledge Graph. It is also necessary to install a bridge service that provides a REST interface for handling user data

4.7.1. Limitierungen

- The only supported grant type is the authorization code flow
- Server administration tasks (e.g. upload Knowledge Graphs) cannot be authorized with OAuth
- When creating Knowledge Graphs, the initial graph administrator account is created with username and password. The administrator can be authorized either with OAuth or username and password.

4.7.2. Autorisierungsablauf

When a user opens a Knowledge Graph that has been configured to use OAuth 2, a web browser will be opened and directed to the login URI. There, the user performs the necessary steps to login, e.g. confirm the login or enter credentials.

Afterwards, a request containg a grant is sent to a redirect URI which must point to the endpoint

```
/oauth/redirect
```

of the Knowledge graph server. This endpoint then requests a token from the authorization server. The token is validated using the public keys (JWKS). The token then allows access to the Knowledge Graph.

Once a user has been authorized, then the server sends a POST request containing the data to an endpoint of the REST interface provided by the bridge service. This allows to perform additional, customizable steps to create user objects in the Knowledge Graph.

4.7.3. Konfiguration

The OAuth framework can either be configured for the entire server, which affects all Knowledge Graphs, or for a single Knowledge Graph.

4.7.3.1. Konfiguration des Autorisierungsservers

The authorization server must be prepared for an authorization code flow. This usually requires registering a new application and generating a client ID and secret. It is also necessary to register a redirect URI that points to the OAuth redirect endpoint of the server.

PKCE is currently not supported.

4.7.3.2. Konfiguration von OAuth für den gesamten Server

The OAuth configuration for all Knowledge Graphs of a server is part of the server configuration file (mediator.ini). It can (but must not) be put in a separate file by using an include directive.

The server must provide an HTTP or HTTPS interface. The redirect endpoint is available at the path

```
/oauth/redirect
```

File **mediator.ini**

```
interfaces=http://0.0.0.0:30080,https://0.0.0.0:30443
$(include:oauth2.ini)
```

File **oauth2.ini**

```
[auth-oauth2]
clientID=12345-abcd-6789-1234-123456789
clientSecret=qwertzuioplkjhgfdsayxcvbnm
configURI=https://login.microsoftonline.com/c4cc84aa-3413-47c6-bd6e-
c38019596fbf/v2.0/.well-known/openid-configuration
redirectURI=https://exampleserver:30443/oauth/redirect
loginFinishedURI=https://exampleserver:8815/oauth/userAccount
userNameKey=preferred_username
createAccounts=true
```

The configuration is contained in the category [auth-oauth2]. The values are

Configuration	Required	Description
clientID	yes	OAuth Client ID
clientSecret	yes	OAuth Client secret
configURI	yes (*)	URI of an OpenID connect configuration endpoint.This URI is used by the mediator to get information about the openid configuration.

redirectURI	yes	Public redirect endpoint of the mediator server. This is usually http(s)://SERVERNAME:SERVER_PORT/oauth/redirect (SERVERNAME and SERVER_PORT must be replaced with actual values).This is invoked from the authentication service and adresses the mediator. Pay attention to possible sub-pathing, e.g. if the mediator is reachable at `https://server/mediator`
loginFinishedURI	no	URI of the graphs REST endpoint for handling the user data. This is usually http(s)://SERVERNAME:REST_PORT/oauth/login-finished (SERVERNAME and REST_PORT must be replaced with actual values).It is possible to use the macro {volume}, which is replace by the name of the accessed Knowledge Graph.This URI is invoked by the mediator and targets a REST endpoint of the graph the user is logging on to. This is only required, if — after login — data from the authentication token should be used to fill a user topic.
userNameKey	no	Name of the token property that contains the user name, e.g. preferred_username (which is also the default value)
createAccounts	no	Boolean value that defines if new accounts should be created in the Knowledge Graph for authorized users.If false, then users can only login if an account has already been created for them. The default value is false.
scopes	no	Comma separated list of additional requested scopes. The following scopes will always be requested and do not need to be configued: openid, email, profile, offline_access

If no OpenID connect configuration endpoint (configURI) is given, then the following settings must be configured:

Configuration	Description
jwksEndpoint	URI of an endpoint that returns the public keys (JSON Web Key Sets), e.g. `https://login.microsoftonline.com/c4cc84aa-3413-47c6-bd6e-c38019596fbf/discovery/v2.0/keys`

Configuration	Description
tokenEndpoint	URI of the token endpoint, e.g. `https://login.microsoftonline.com/c4cc84aa-3413-47c6-bd6e-c38019596fbf/oauth2/v2.0/token`
authorizationEndpoint	URI of the authorization endpoint, e.g. `https://login.microsoftonline.com/c4cc84aa-3413-47c6-bd6e-c38019596fbf/oauth2/v2.0/authorize`

4.7.3.2.1. Konfiguration von HTML-Seiten

The HTML page displayed in the web browser after an (un)successful login can be customized by defining templates in the **auth-oauth2** section

```
[auth-oauth2]
htmlTemplates=oauth2-html-authorized-de,oauth2-html-authorized-en,oauth2-html-unauthorized-de,oauth2-html-unauthorized-en
; Addition configuration ommitted

[oauth2-html-authorized-de]
state=authorized
languages=de
file=html\authorized-de.html

[oauth2-html-authorized-en]
state=authorized
languages=*
file=html\authorized-en.html

[oauth2-html-unauthorized-de]
state=unauthorized
file=html\unauthorized-de.html

[oauth2-html-unauthorized-en]
state=unauthorized
file=html\unauthorized-en.html
```

htmlTemplates is a comma-separated list of unique section names. Each section can have the following entries:

Section	Description
state	must be `authorized` or `unauthorized` Default value: `authorized`
languages	comma-separated list of languages, * can be used to match any language.Default value: *
file	HTML file. The file must self-contained, no additional files are included.
redirect	URI that is opened via a redirect (307) instead of showing a built-in HTML file. Only used when **file** is not specified.

4.7.3.3. Konfiguration von OAuth für einen Knowledge-Graph

OAuth can be configured for a single Knowledge Graph in the Knowledge-Builder. This can be done by administrators only. To configure OAuth, open the settings, and select **OAuth** on the **System** tab. The settings are equal to the server configuration described above.

If a configuration is present, it has precedence over the configuration of the server.

4.7.3.4. Konfiguration des OAuth REST Endpunktes

The server only creates basic user accounts when registering new users. All additional steps must be performed by a REST endpoint provided by a bridge service. To simplyfiy the setup, add the software component OAuth login in the Admin-Tool. This will create a basic setup:

- A mapping **rest.oauth.userMapping** that maps the user data to objects of the Knowledge Graph
- A script **rest.oauth.postUserAccount** that uses the mapping to create user objects.
- An endpoint **/oauth/userAccount** which calls the script
- An OAuth configuration skeleton. This is incomplete and must be adjusted to the server setup (e.g. set host names)

4.8. Installation von i-views

4.8.1. Allgemeine Information

Für den Betrieb von i-views ist es unerheblich, ob die Prozesse auf realer Hardware (Server, Workstation, Notebook, …) oder in einer virtualisierten Umgebung (VMWare, VirtualBox, docker, …) laufen.

Der Betrieb von i-views erfordert keine besonderen Berechtigungen innerhalb des Betriebssystems. Dadurch können die Prozesse der Produktteile von beliebigen Konten des Betriebssystems gestartet werden. Der Betrieb mit Systemkonten ist aber möglich, i-views arbeitet immer nur in den konfigurierten Datenbereichen und die Produktteile kommunizieren ausschließlich via TCP/IP.

Ohne weitere Konfiguration lesen und schreiben die Produktteile Daten und Log-Dateien in oder unterhalb ihres Arbeitsverzeichnisses (d.h. dort sind Schreibrechte für das ausführende Konto notwendig).

Der Betrieb der Prozesse in einem nicht lokal verfügbaren Verzeichnis (z.B. einer Netzwerkfreigabe) wird nicht empfohlen, da dies abgesehen von schwacher I/O Performance in bestimmten Situationen zu Problemen führen kann.

Alle i-views-Produktteile laufen als eine Kombination von virtueller Maschine (VM) und sogenanntem Image. Die Image-Dateien enthalten den Code für den jeweiligen Produktteil und sind unabhängig vom Betriebssystem (und damit zwischen Betriebssystem austauschbar). Die mitgelieferten virtuellen Maschinen übernehmen die Abstraktion vom konkreten Betriebssystem. Für Windows bietet i-views Executables mit einer Kombination von VM und Image an.

Alle Produktteile kommunizieren untereinander via TCP/IP, so dass sie prinzipiell auch auf mehrere Maschinen verteilt werden können. Diese Verteilung ist nicht Teil dieses Dokuments. Falls die von den Diensten benutzten TCP/IP-Ports auch von außerhalb der jeweiligen Maschine erreicht werden sollen, müssen diese Ports über die Firewall-Mechanismen des jeweiligen Betriebssystems freigegeben werden.

Die Konfiguration der Produktteile findet über INI-Dateien statt. Einige Komponenten erwarten eine solche Datei, ohne die sie nur eine Fehlermeldung in die Log-Datei schreiben, bevor der Prozess beendet wird. Wird beim Aufruf keine INI-Datei explizit genannt, sucht der jeweilige Prozess im seinem Arbeitsverzeichnis nach einer INI-Datei, die zum Typ des Images passt (also z.B. mediator.ini für einen Mediator). Der Dateiname des Image oder Executables ist dabei unerheblich. Die individuelle Konfiguration der Produktteile ist nicht Teil dieses Dokuments.

Alle Produktteile können über die Kommandozeile oder durch Start aus dem Dateimanager im Vordergrund gestartet werden. Die Prozesse werden beim Schließen des Fensters oder durch die Eingabe von Strg-C/Cmd-C beendet.

Komponente / Feature	Beschreibung	Benötigt für…
Core	Kernumgebung	Benötigt für Core

Komponente / Feature	Beschreibung	Benötigt für…
Core (De-)Kompression	Kompressionswerkzeuge	Benötigt für Core
Core mit grafischer Oberfläche	i-views-Werkzeuge mit grafischer Oberfläche	Benötigt für Core mit UI
Mediator-Prüfsummen	Beschleunigte Berechnung von Prüfsummen für Knowledge Graph Inhalte	Optional
TLS Verbindungen	Verwendung von TLS Funktionen des Betriebssystems (5.5 und höher)	Benötigt für das Feature
Bildskalierung	Beschleunigte Skalierung von Pixel-Bildern	Optional
Graph-Editor Alpha-Kanal	Icons mit transparenten Bereichen im Graph Editor schöner anzeigen	Optional für Core mit UI
JNI	Java Native Interface zur Anbindung von Java Software (z.B. Lucene)	Benötigt für das Feature
Verschlüsselung	Kryptographisch abgesicherte Verbindungen (z.B. HTTPS); Verschlüsselte Passwörter für Kommandozeilen-Werkzeuge	Benötigt für das Feature
Verbesserte Text Darstellung	Texte mit besserem Anti-Aliasing und Kerning Anzeigen im UI	Optional für Core mit UI
MySQL-Anbindung	Anbindung von MySQL Datenbanken	Benötigt für das Feature
ODBC-Anbindung	Anbindung von ODBC Datenquellen	Benötigt für das Feature
Oracle-Anbindung	Anbindung von Oracle Datenbanken	Benötigt für das Feature
SQLite-Anbindung	Anbindung von SQLite Datenbanken	Benötigt für das Feature und BlobService
Erweiterte Reguläre Ausdrücke	Erweiterung der standardmäßig vorhandenen regulären Ausdrücke	Benötigt für das Feature

Komponente / Feature	Beschreibung	Benötigt für…
Koordinatentransformation	Umrechnung von geographischen Geometrie-Attributen von einem Raumbezugssystem in ein anderes; Berechnung geodätischer Distanzen	Benötigt für das Feature
Relationsberechnung für Geometrien	Berechnung von Relationen des Dimensionally Extended 9-Intersection Model mit erhöhter Performance	Optional für das Feature

i-views wird als 64-bit Software ausgeliefert. Beim Einbinden externer Softwarebibliotheken ist darauf zu achten, dass diese ebenfalls in 64-bit-Architektur vorliegen müssen. Das Einbinden von 32-bit-Bibliotheken ist nicht möglich.

4.8.2. Betriebssysteme

i-views unterstützt verschiedene Betriebssysteme. Dieses Kapitel beschreibt die Installation unter voll unterstützten Betriebssystemen. Am Ende finden sich Hinweise zur Installation unter weiteren möglichen Betriebssystemen, die aber nicht offiziell unterstützt werden.

Bei Bedarf können alle in i-views verwalteten Daten auf eine andere Systemumgebung (Hardware, Betriebssystem, …) ohne Konvertierung migriert werden. Falls die Produktteile über mehrere Rechner verteilt betrieben werden, dürfen auch diese jeweils mit unterschiedlichen Betriebssystemen laufen (z.B. ein Mix von Windows und Linux). Auch für den Zugriff von Clients auf die Server gibt es keine Beschränkungen bei den verwendeten Betriebssystemen.

4.8.2.1. Microsoft Windows

Unter Windows werden die Produktteile üblicherweise als EXE-Dateien ausgeliefert. Diese bestehen intern aus einer Kombination von virtueller Maschine und Image. Bei Bedarf können statt der EXE-Dateien auch unter Windows getrennte VM-Executables und Images verwendet werden.

Alle i-views-Produktteile, die den Betrieb als Dienst unterstützen, können unter Windows durch folgende (mit Administratorrechten gestartete) Kommandozeile als Dienst installiert werden:

```
PRODUKTTEIL.exe -installAsService DIENSTNAME [DIENST1 ...]
[DIENSTPARAMETER ...]
```

Dieser Aufruf legt nur die Dienstkonfiguration an und beendet sich dann sofort wieder. Falls Dienste von anderen Diensten abhängen, können optional mehrere Dienstnamen DIENST1 bis DIENSTn angegeben werden, die vorher von Windows gestartet sein sollen. Spezielle Parameter für den als Dienst laufenden Produktteil werden am Schluss angegeben.

In der Windows-Dienst-Konfiguration (typischerweise services.msc) erscheint der Dienst danach unter dem Namen DIENSTNAME und kann dort bearbeitet werden, um z.B. das Konto, die Start-Art oder die Beschreibung anzupassen.

Die De-Registrierung erfolgt entweder über das Windows-Werkzeug `sc delete DIENSTNAME` oder über:

```
PRODUKTTEIL.exe -deinstallService DIENSTNAME
```

Alternativ kann zur Dienst-Installation natürlich das Windows-eigene Werkzeug sc.exe eingesetzt werden.

Funktion	Bibliothek	Zu installieren
Core	-	In Windows enthalten
Core (De-)Kompression	-	In Windows enthalten
Core mit grafischer Oberfläche	-	In Windows enthalten
Mediator-Prüfsummen	coastbinary	coastbinary.dll: mitgeliefert, neben mediator.exe
TLS Verbindungen	tlsPlugin	tlsPlugin.dll: mitgeliefert, neben allen ausführbaren Dateien
Bildskalierung	GDIPlus	In Windows enthalten
Graph-Editor Alpha-Kanal	Cairo	libcairo-2.dll: von gtk.org
JNI	Java Virtual Machine	libjvm.dll wird bei der Installation einer Java Runtime (jre) oder eines Development Kits (jdk) mitgeliefert, z.B. unter java.com
Verschlüsselung	Windows nativ und OpenSSL	bcrypt.dll: in Windows enthalten libeay32.dll: via openssl.org
Verbesserte Text Darstellung	-	In Windows enthalten
MySQL-Anbindung	MySQL Client Bibliotheken	libmysql.dll: via mysql.com
ODBC-Anbindung	ODBC	In Windows enthalten bzw. als Feature nachinstallierbar

Funktion	Bibliothek	Zu installieren
Oracle-Anbindung	Windows Advanced Services Oracle Client Bibliotheken	advapi32.dll: Windows Feature Advanced Services, meist vorhanden. oci.dll: Von oracle.com die zur DB passende Version installieren.
SQLite-Anbindung	SQLite	sqlite3.dll: von sqlite.org
Erweiterte Reguläre Ausdrücke	Boost-Regex	boost_regex.dll, msvcp140.dll, vcruntime140.dll: von boost.org
Koordinatentransformation	PROJ	proj_9.dll: entweder von proj.org oder durch Nutzung einer Distribution wie OSGeo4W
Relationsberechnung für Geometrien	GEOS	geos.dll, geos_c.dll: entweder von libgeos.org oder durch Nutzung einer Distribution wie OSGeo4W

Unter Windows werden teilweise Bibliotheken der Microsoft Visual C++ Runtime benötigt. Je nach Art und Alter der Windows-Version, kann es notwendig sein, diese zusätzlich zu installieren.

Der empfohlene Weg ist, das Redistributable von der offiziellen Microsoft Seite Visual C++ Redistributable zu installieren. Hierbei sollte auf die zum Betriebssystem passende Sprache geachtet werden. Auf diese Weise werden nicht nur über den Windows-Update Mechanismus Aktualisierungen zur Verfügung gestellt, sondern man vermeidet auch mehrfache Kopien der Bibliothek.

Alternativ kann man die Bibliotheken msvcp140.dll und/oder vcruntime140.dll auch zu den jeweiligen Executables in das Verzeichnis legen.

4.8.2.2. macOS

Einige Komponenten sind bereits in macOS enthalten, andere kann man z.B. über MacPorts installieren, oder eben von den jeweiligen Projektseiten.

Feature	OS Bibliothek	Name der Datei
Core	LibC	libc.dylib, mit macOS enthalten als libSystem.dylib
Core (De-)Kompression	LibZ	In macOS enhalten
Core mit grafischer Oberfläche	X11 Umgebung	In macOS enhalten

Feature	OS Bibliothek	Name der Datei
Mediator Prüfsummen	coastbinary	Prüfsummen werden intern berechnet, die externe Bibliothek gibt es nicht für macOS
TLS connections	tlsPlugin	tlsPlugin.dylib: mitgeliefert mit der Applikation
Bildskalierung	FreeImage	Verfügbar beim FreeImage Projekt
Graph-Editor Alpha-Kanal	Cairo	libcairo.2.dylib, via Cairo Project oder MacPorts
JNI	Java Virtual Machine	
Verschlüsselung	OpenSSL	libcrypto.1.0.dylib
Verbesserte Text Darstellung	-	In macOS enthalten
MySQL-Anbindung	MySQL	libmysqlclient.dylib: von der MySQL Webseite
ODBC-Anbindung	diverse, z.B. iODBC	libiodbc.dylib: z.B. von iodbc.org
Oracle-Anbindung	Oracle	libclntsh.dylib: von Oracle die zur DB passende Version installieren.
SQLite-Anbindung	SQLite	libsqlite3.dylib: von sqlite.org
Erweiterte Reguläre Ausdrücke	Boost-Regex	libboost_regex.dylib: von boost.org
Koordinatentransformation	PROJ	libproj.dylib: von proj.org
Relationsberechnung für Geometrien	GEOS	libgeos.dylib, libgeos_c.dylib: von libgeos.org

4.8.2.3. Linux / Unix

i-views läuft unter verschiedenen Unix-artigen Betriebssystemen, wobei die Mehrzahl der Installationen unter diversen Varianten von Linux laufen.

Für i-views werden (noch) keine Repositories zur Installation und automatischem Upgrade der Software angeboten. Die Software wird aktuell als Archiv geliefert, das an den bevorzugten Ort kopiert werden kann.

Die virtuellen Maschinen benötigen zum Start einige wenige Bibliotheken. Diese dürften auf den meisten Systemen out-of-the-box installiert sein, auf Minimalinstallationen können diese fehlen. Für bestimmte Features verwenden die i-views-Komponenten externe Bibliotheken, die dann aber erst bei Verwendung des Features angesprochen und nachgeladen werden. Falls die notwendigen

Bibliotheken nicht gefunden werden, wird dieser Umstand in den Log-Dateien genannt. Hierbei kann es vorkommen, dass in den Log-Dateien die Haupt-Bibliothek als nicht verfügbar genannt wird, diese aber vorhanden ist, jedoch abhängige Bibliotheken fehlen. In diesem Fall hilft meist ein `ldd BIBLIOTHEKSDATEI` um vom System nicht gefundene abhängige Bibliotheken zu identifizieren.

4.8.2.3.1. Debian/Ubuntu (x86/amd64)

Debian/Ubuntu verwenden `apt` als Paketmanager. Es wird empfohlen, wo immer möglich Standard-Paket-Quellen zu verwenden, die aktuelle Patches liefern. Die Pakete werden mit `apt install PAKETNAME` installiert; man kann natürlich auch erweiterte Paketmanager wie aptitude oder synaptic verwenden.

Feature	OS Bibliothek	Paketname / Bibliothek
Core	GLibC	libc
Core (De-)Kompression	Kompressionsutilities	zlib1g
Core mit grafischer Oberfläche	X11 Umgebung	libx11-6
Mediator Prüfsummen	coastbinary.so	coastbinary.so: mitgeliefert, im Verzeichnis des mediators
TLS Verbindungen	tlsPlugin	tlsPlugin.so: mitgeliefert, für jede ausführbare Datei
Bildskalierung	FreeImage	libfreeimage3
Graph-Editor Alpha-Kanal	Cairo	libcairo2
JNI	Java Virtual Machine	libjvm.so: wird bei der Installation einer Java Runtime (jre) oder eines Development Kits (jdk) mitgeliefert, z.B. unter java.com
Verschlüsselung	OpenSSL	libssl1.0.0
Verbesserte Text Darstellung	XFT	libxft2
MySQL-Anbindung	MySQL oder MariaDB	libmysqlclient20 oder libmariadb-client-lgpl-dev-compat
ODBC-Anbindung	iODBC	libiodbc2
Oracle-Anbindung	Oracle	libclntsh.so: von oracle.com die zur DB passende Version installieren.
SQLite-Anbindung	SQLite	libsqlite3-0
Erweiterte Reguläre Ausdrücke	Boost-Regex	libboost-regex1.58.0
Koordinatentransformation	PROJ	libproj25 / libproj.so.25

Feature	OS Bibliothek	Paketname / Bibliothek
Relationsberechnung für Geometrien	GEOS	libgeos-c1v5 (Debian), libgeos-c1t64 (Ubuntu) / libgeos.so, libgeos_c.so

Unter Debian kann man Pakete, die ein bestimmtes Feature enthalten, mit `apt-cache search 'NAME'` finden.

Ob ein Paket wirklich die gewünschten Dateien enthält, kann man mit `apt-file list 'PAKET'` (aus dem Paket apt-file) überprüfen. Den Paketnamen zu einer Datei kann man mit `apt-file search 'DATEI'` suchen.

Alternativ kann eine fehlende Datei über die Webseiten der jeweiligen Distribution gefunden werden, also z.B. https://www.debian.org/distrib/packages#search_contents bzw. http://packages.ubuntu.com/ gesucht werden.

4.8.2.3.2. RedHat/Fedora/CentOS (x86/amd64)

RedHat-basierte Distributionen verwenden meist `yum` oder `dnf` als Paketmanager. Es wird empfohlen, wo immer möglich Standard-Paket-Quellen zu verwenden, die aktuelle Patches liefern. Die Pakete werden mit `yum install PAKETNAME` oder `dnf install PAKETNAME` installiert.

Feature	OS Bibliothek	Paketname / Bilbiothek
Core	GLibC	glibc
Core (De-)Kompression	LibZ	zlib
Core mit grafischer Oberfläche	X11 Umgebung	libX11
Mediator Prüfsummen	coastbinary	coastbinary.so: mitgeliefert, im Verzeichnis der VM
TLS Verbindungen	tlsPlugin	tlsPlugin.so: mitgeliefert neben der VM
Bildskalierung	FreeImage	freeimage
Graph-Editor Alpha-Kanal	Cairo	cairo
JNI	Java Virtual Machine	libjvm.so: wird bei der Installation einer Java Runtime (jre) oder eines Development Kits (jdk) mitgeliefert, z.B. unter java.com
Verschlüsselung	OpenSSL	openssl
Verbesserte Text Darstellung	XFT	libXft
MySQL-Anbindung	MariaDB	mariadb-libs

Feature	OS Bibliothek	Paketname / Bilbiothek
ODBC-Anbindung	iODBC	libiodbc
Oracle-Anbindung	Oracle	libclntsh.so: von oracle.com die zur DB passende Version installieren.
SQLite-Anbindung	SQLite	sqlite
Erweiterte Reguläre Ausdrücke	Boost-Regex	boost-regex
Koordinatentransformation	PROJ	proj / libproj.so.25
Relationsberechnung für Geometrien	GEOS	geos / libgeos.so, libgeos_c.so

In RPM-basierten Distributionen kann man Pakete, die ein bestimmtes Feature enthalten- mit `yum search 'NAME'` finden. Ob ein Paket wirklich die gewünschten Dateien enthält kann man mit `repoquery -l 'PAKET'` (aus dem Paket `yum-utils`) überprüfen.

Alternativ kann eine fehlende Datei über https://rpmfind.net/linux/rpm2html/search.php gesucht werden.

4.8.2.3.3. SUSE Linux Enterprise (SLES)

SUSE Distributionen verwenden meist `zypper` als Paketmanager. Es wird empfohlen, wo immer möglich Standard-Paket-Quellen zu verwenden, die aktuelle Patches liefern. Die Pakete werden mit `zypper install PAKETNAME` installiert.

Feature	OS Bibliothek	Paketname / Bilbiothek
Core	GLibC	glibc
Core (De-)Kompression	LibZ	libz1
Core mit grafischer Oberfläche	X11 Umgebung	libX11-6
Mediator Prüfsummen	coastbinary	coastbinary.so: mitgeliefert, im Verzeichnis der VM
TLS Verbindungen	tlsPlugin	tlsPlugin.so: mitgeliefert, im Verzeichnis der VM
Bildskalierung	FreeImage	libfreeimage3
Graph-Editor Alpha-Kanal	Cairo	libcairo2
JNI	Java Virtual Machine	libjvm.so: wird bei der Installation einer Java Runtime (jre) oder eines Development Kits (jdk) mitgeliefert, z.B. unter java.com
Verschlüsselung	OpenSSL	libopenssl1_0_0

Feature	OS Bibliothek	Paketname / Bilbiothek
Verbesserte Text Darstellung	XFT	libXft2
MySQL-Anbindung	MariaDB	libmysqlclient18
ODBC-Anbindung	iODBC	
Oracle-Anbindung	Oracle	libclntsh.so: von Oracle die zur DB passende Version installieren.
SQLite-Anbindung	SQLite	libsqlite3-0
Erweiterte Reguläre Ausdrücke	Boost-Regex	libboost_regex1_54_0
Koordinatentransformation	PROJ	libproj25 / libproj.so.25
Relationsberechnung für Geometrien	GEOS	libgeos_c1 / libgeos.so, libgeos_c.so

4.8.3. Einrichten der Dienste

Das Starten der Produktteile hängt von der Ausführungsumgebung ab. Für Linux Distributionen bestimmt das dort verwendete "init"-Systeme die notwendige Konfiguration. Für die i-views-Produktteile gibt es aktuell Beispieldateien für systemd-basierte Systeme.

Andere init-Systeme wie z.B. Upstart (wie CentOS 6.5) oder LaunchDaemon (Mac OS X) werden aktuell nicht angeboten, da sie bisher nur unvollständig oder nicht ausreichend generisch vorliegen. Da i-views nicht von einer speziellen Unterstützung durch das Betriebssystem abhängig ist, sollte es auf ähnlichen Plattformen einfach eingesetzt werden können.

Für Umgebungen, die auf Containern basieren, werden vom Build-System Standard Container-Images erstellt, die direkt verwendbar sein sollten. Hier werden Beispielkonfigurationen angeboten für:

- docker-compose
- kubernetes

Andere Container Laufzeit Umgebungen wie docker, podman oder openshift können von diesen Vorlagen abgeleitet werden.

4.8.3.1. systemd

Für den normalen Betrieb unter Distributionen, die systemd als init-System verwenden, können die folgenden Konfigurationsdateien als Vorlage verwendet werden:

`/etc/systemd/system/iviews-mediator.service`:

```
[Unit]
```

```
Description=iviews mediator
After=network.target nss-lookup.target

[Service]
User=iviews
WorkingDirectory=/opt/iviews/mediator
ExecStart=/opt/iviews/vw/vwlinux86 -noherald
-=/opt/iviews/mediator/mediator.im
Restart=on-failure

[Install]
WantedBy=default.target
```

/etc/systemd/system/iviews-bridge-rest.service:

```
[Unit]
Description=iviews bridge rest
After=network.target nss-lookup.target iviews-mediator.service
Wants=iviews-mediator.service

[Service]
User=iviews
WorkingDirectory=/opt/iviews/bridge
ExecStart=/opt/iviews/vw/vwlinux86 -noherald
-=/opt/iviews/bridge/bridge.im -ini bridge-rest.ini
Restart=on-failure

[Install]
WantedBy=default.target
```

/etc/systemd/system/iviews-jobclient.service:

```
[Unit]
Description=iviews jobclient
After=network.target nss-lookup.target iviews-mediator.service
Wants=iviews-mediator.service

[Service]
User=iviews
WorkingDirectory=/opt/iviews/jobclient
ExecStart=/opt/iviews/vw/vwlinux86 -noherald
-=/opt/iviews/jobclient/jobclient.im
Restart=on-failure
```

```
[Install]
WantedBy=default.target
```

Sollen Dienste (Services) zu einer Installation zusammengefasst werden, z.B. wenn man mehrere Projekte parallel betreibt und diese getrennt starten und stoppen können will, ist es sinnvoll, ein projektspezifisches target zu definieren:

`/etc/systemd/system/iviews.target`:

```
[Unit]
Description=iviews
After=network.target nss-lookup.target
Wants=iviews-mediator.service iviews-bridge-rest.service iviews-
jobclient.service

[Install]
WantedBy=default.target
```

In den Dienst-Konfigurationen kann man dann die [Install] Sektionen entfernen und dafür in der [Unit] Sektion den Eintrag

```
PartOf=iviews.target
```

hinzufügen. Beim Starten und Stoppen der Unit iviews.target werden dann alle Teile gestartet bzw gestoppt, man kann aber auch Teildienste manuell neu starten, ohne das iviews.target zu beeinflussen.

Anzupassen sind natürlich die Pfade der Verzeichnisse und Namen der Services an die konkrete Installation. Nach dem Editieren der Dateien muss man via systemctl die Dienste noch registrieren und starten.

Bei Bedarf können die Abhängigkeiten der Dienste auch stärker oder anders ausgedrückt werden. Details hierzu finden sich auf systemd Homepage oder natürlich in den man-pages der jeweiligen Distribution.

Cheat Sheet zur Verwaltung der Dienste:

Aufgabe	Kommando
Einen Dienst registrieren	`systemctl enable SERVICE`

Aufgabe	Kommando
Einen Dienst deregistrieren	`systemctl disable SERVICE`
Einen Dienst starten	`systemctl start SERVICE`
Einen Dienst beenden	`systemctl stop SERVICE`
Den Dienststatus abfragen	`systemctl status SERVICE`
Editieren einer Service Datei	`systemctl edit --full SERVICE`
Nach dem manuellen Editieren systemctl aktualisieren	`systemctl daemon-reload`

4.8.3.2. docker-compose

Das vorgestelle Beispiel bietet eine Konfigurationsdatei für docker-compose, die einen Mediator, eine Bridge und einen jobclient started. Es ist ebenfalls eine env-Datei enthalten, in der notwendige Kennwörter für die Installation hinterlegt sind. Das compose-file kann in Versionierungssystemen wie git gespeichert werden. Die env-Datei sollte im git keine echten Kennwörter enthalten.

In den Dateien sind Werte in eckigen Klammern (z.B. "<knowledge-graph-name>") enthalten, die als Platzhalter für Werte im Projekt dienen. Identische Platzhalter signalisieren identische Werte innerhalb der compose-Datei. Sie sind beabsichtigt nicht Teil der env-Datei, damit man nur eine solche verwenden muss (docker-compose verwendet im Standard ".env") und damit alle relevanten Angaben in einer Konfigurationsdatei enthalten sind.

Die Container-Images verwenden keine INI-Dateien, sondern erwarten die Konfigurationsangaben in Umgebungsvariablen mit IV_ Präfix. Die i-views Werkzeuge unterstützen bei der Umwandlung existierender INI-Dateien zu Umgebungsvariablen, wenn sie mit dem Parameter -iniToEnv aufgerufen werden.

Die Konfigurationsdateien befinde sich im Anhang.

4.8.3.3. kubernetes

i-views kann in kubernetes Umgebungen ausgeführt werden. Der Mediator benötigt dazu einen zuverlässigen Storage-Treiber, nur die Speicherung auf Dateisystemen unterstützt wird.

In der Beispiieldatei finden sich Werte in spitzen Klammern (z.B. "<knowledge-graph-name>"). Diese Platzhalter sind durch konkrete Werte des Projekts zu befüllen. Identische Platzhalter sollten

mit dem selben Wert befüllt werden. Die Konfiguration mit einer Datei kann natürlich in mehrere yaml-Dateien aufgeteilt werden, wenn die Verwaltung der Installation mit einer Datei je Ressource stattfinden soll.

Zu beachten:

- Fast alle genannten Werte in der Konfiguration sind abhängig vom Projekt, der kubernetes Umgebung und Vorgaben für diese.
- Speicher per NFS bereit zu stellen hat sich bisher als sehr unzuverlässig erwiesen.
- Prozesse mit permanentem Speicher (insbesondere der Mediator) sollten nicht zwischen Knoten verschoben werden (siehe PodDisruptionBudget, maxUnavailable: 0).
- Nicht alle k8s Object Typen sind enthalten, die benötigt werden, um auf Dienste zuzugreifen. Z.B. fehlt eine Ingress Konfiguration, weil diese stark von der jeweiligen Infrastruktur im k8s Cluster abhängig sind.

Die Konfigurationsdatei findet sich im Anhangskapitel.

4.8.4. Typische Anforderungen

4.8.4.1. Proxy und Load Balancing

4.8.4.1.1. Load-Balancing über mehrere gleichartige REST-Dienste

Bei REST-Services bietet es sich an, durch mehrere Diensterbringer einen höheren Durchsatz an parallelen Requests zu erreichen. i-views selbst ermöglicht mit einer Bridge mit LoadBalancer-Konfiguration das Starten und Überwachen mehrerer REST-Bridges. In der Konfigurationsdatei der Bridge werden dann z.B. folgende Einstellungen vorgenommen:

```
[KLoadBalancer]
hostname=localhost
port=5001
vm=/opt/iviews/vm/vwlinux86
directory=.
image=bridge.im
configNames=REST
autoRestart=true

[REST]
bridgeClientClassName=KWeb.KHTTPRestBridge
inifile=bridge_rest.ini
bridgeLogfile=bridge_rest_<id>.log
ports=5002-5005
```

Diese Konfiguration startet vier REST-Bridges auf den Ports 5002 bis 5005.

Allerdings unterstützt diese LoadBalancer-Bridge nicht selbst das LoadBalancing. Hierfür werden üblicherweise externe Pakete wie Nginx oder Apache-HTTPD eingesetzt.

Nginx

Nginx kann mit der proxy_pass Direktive Anfragen an einen bestimmten Pfad intern an verschiedene Backends weiterleiten. Dazu erweitert man die Konfiguration von nginx um folgende Zeilen:

```
upstream iviews-rest-bridges {
   server 127.0.0.1:5002;
   server 127.0.0.1:5003;
   server 127.0.0.1:5004;
   server 127.0.0.1:5005;
}

server {
   ...

   location ^ ~ /myRestPath/ {
      proxy_pass http://iviews-rest-bridges/;
   }
}
```

Apache HTTPD

In Apache HTTPD können verschiedene Proxy Module aktiviert werden, die u.a. Load Balancing anbieten. Eine Konfiguration für obige Bridge könnte folgendermaßen aussehen:

```
ProxyRequests off
<Proxy balancer://iviews>
   BalancerMember  http://127.0.0.1:5002  disablereuse=On
   BalancerMember  http://127.0.0.1:5003  disablereuse=On
   BalancerMember  http://127.0.0.1:5004  disablereuse=On
   BalancerMember  http://127.0.0.1:5005  disablereuse=On
</Proxy>
ProxyPass         /myRestPath/  balancer://iviews/  lbmethod=bybusyness
ProxyPassReverse  /myRestPath/  balancer://iviews/  lbmethod=bybusyness
```

(verwendet die Module mod_proxy, mod_proxy_balancer, mod_lbmethod_bybusyness).

haproxy

haproxy ist im Gegensatz zu nginx oder apache nur ein reiner Proxy-Dienst. haproxy wird

typischerweise konfiguriert in

/etc/haproxy/haproxy.cfg:

```
frontend         main  *:5000
default_backend  rest_bridge

backend rest_bridge
  balance  first
  server   rest_bridge1 127.0.0.1:5002 check maxconn 1 weight 100
  server   rest_bridge2 127.0.0.1:5003 check maxconn 1 weight 99
  server   rest_bridge3 127.0.0.1:5004 check maxconn 1 weight 98
  server   rest_bridge4 127.0.0.1:5005 check maxconn 1 weight 97
```

4.8.4.1.2. Microsoft Windows Server

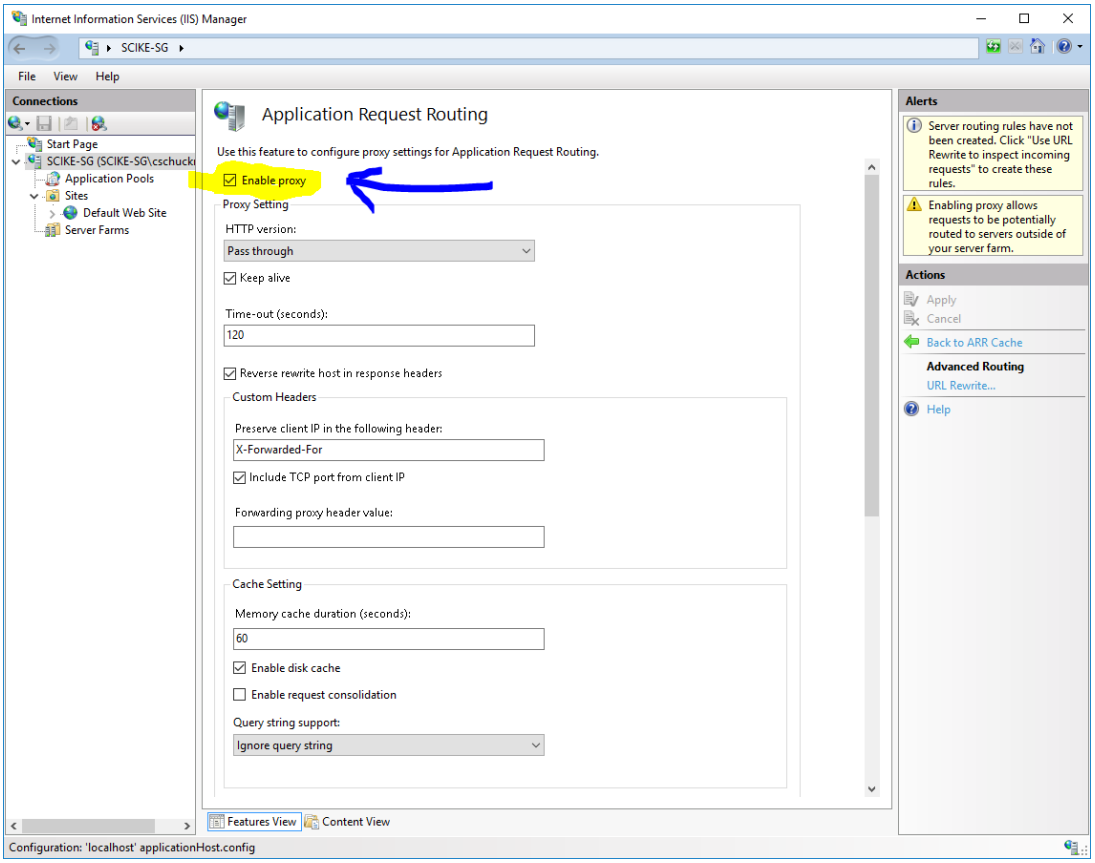
Über die Internet information Services (IIS) können i-views-Dienste von außerhalb des Servers verfügbar gemacht werden. Der Vorteil besteht darin, dass keine Öffnungen der Firewall vorgenommen werden müssen und dass die sichere Kommunikation (HTTPS) samt Zertifikaten an einer zentralen Stelle administriert werden kann.

Folgende Module müssen dazu installiert sein:

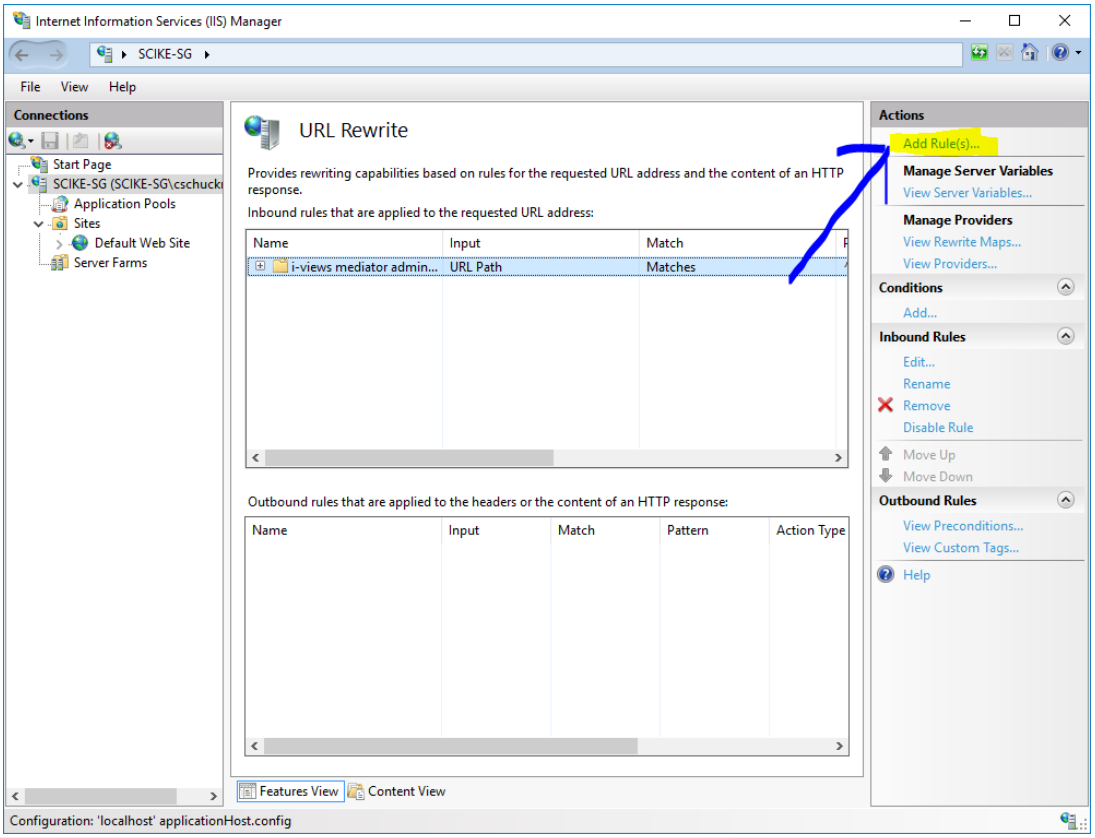
- ApplicationRequestRouting (Achtung: Aktivierung des Moduls nicht vergessen!)
- RewriteModule (Anzeigename ist: "Url Rewrite")
- WebSocketModule (wenn der mediator erreichbar sein soll)

Sind die Module installiert, dann können Umleitungen wie folgt definiert werden:

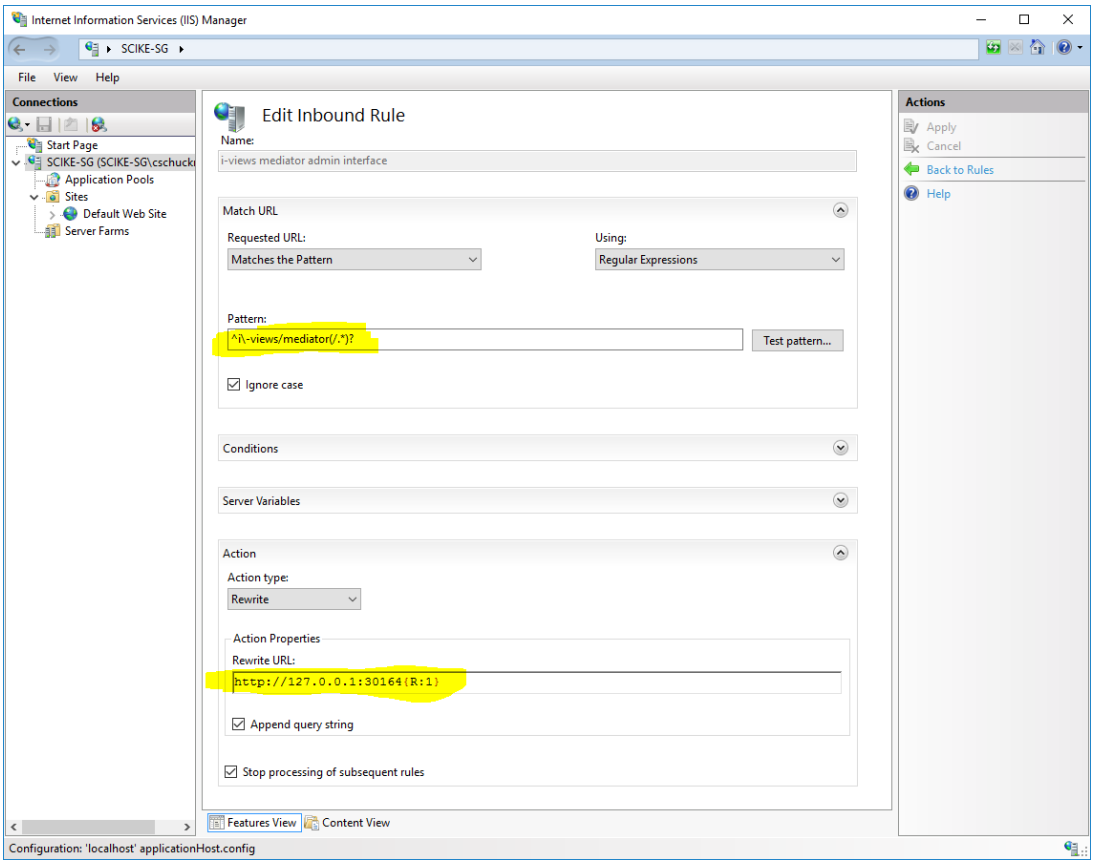
1. Sicherstellen, dass Application Request Routing aktiviert ist. (Server im Baum links anwählen, dann das Icon für "Application Request Routing" doppelt klicken)



2. URL-Rewrite Rule hinzufügen (Blank rule) (Server im Baum links anwählen, dann das Icon für "URL Rewrite" doppelt klicken)



3. Regel konfigurieren (das folgende Beispiel gilt für einen mediator)



Anmerkung: Innerhalb des Servers sollte per HTTP weitergeleitet werden, auch wenn die eingehende Kommunikation HTTPS ist. Das ist schneller und sicherheitstechnisch unbedenklich.

Fehlersuche

Wenn die Requests nicht so ankommen, wie man sich das vorstellt, sollte man die Logging-Funktionalität des IIS verwenden:

1. Links im Baum Server anwählen
2. Icon "Failed Request Tracing Rules" doppelklicken
3. Regel anlegen (z.B. alle Requests mit Return Code 400-999 protokollieren)
4. Links im Baum Web-Site anwählen
5. Rechts "Configure Failed Request Tracing" anklicken
6. Im Dialog "enable" auswählen

Die Logs erscheinen dann im Ordner "C:\\inetpub\\logs\\FailedReqLogFiles".

4.8.4.2. Firewall-Konfiguration

i-views-Prozesse kommunizieren untereinander ausschließlich über TCP/IP. Die konkret verwendeten Ports hängen von der eingesetzten Software-Version und natürlich der lokalen

Konfiguration ab.

Je nach eingesetztem OS und Distribution findet die Konfiguration unterschiedlich statt, daher kann hier keine komplette Dokumentation aller Firewall-Utilities vorgestellt werden. Z.B. unter Linux gibt es diverse Frontends und Konfigurationsweisen, welche aber alle die unterliegende iptables- bzw. nftables-Implementierung im Kernel steuern.

4.8.4.2.1. firewalld

firewalld, wie es z.B. unter CentOS7 eingesetzt wird, verwendet zur Konfiguration Zonen und Dienste.

Firewalld verwendet XML-Konfigurationsdateien, in denen man z.B. für eine Installation die komplette Konfiguration der Firewall-Regeln in einer Datei eintragen kann. Die Dateien werden in /etc/firewalld/services/NAME.xml abgelegt.

/etc/firewalld/services/iviews.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<service>
  <short>iviews-example</short>
  <description>i-views example installation</description>
  <!-- mediator -->
  <port protocol="tcp" port="30063"/>
  <!-- rest-bridge -->
  <port protocol="tcp" port="8815"/>
</service>
```

Eine neue Konfiguration kann man dann dauerhaft (--permanent) mit folgenden Befehlen für die öffentliche Zone (public) aktivieren:

```
firewall-cmd --permanent --zone=public --add-service=iviews
firewall-cmd --reload
```

Alternativ kann man auch die entsprechende Zonendatei manuell erweitern.

/etc/firewalld/zones/public.xml:

```
<service name="iviews"/>
```

5. Anhang

5.1. docker-compose Konfiguration

\.env

```
MEDIATOR_PASSWORD=<mediator-password>
AUTH_TOKEN=<system-account>
```

docker-compose.yaml

```
version: '3'
services:
  mediator:
    image: container-registry.example.com/iviews-mediator:<image-tag-1>
    environment:
      IV_PASSWORD: "${MEDIATOR_PASSWORD:?MEDIATOR_PASSWORD not
configured}"
      IV_SCHEDULED_JOBS: "job-bu,job-gc"
      IV_JOB_BU_VOLUME_PATTERN: "<knowledge-graph-name>"
      IV_JOB_BU_BACKUP_INTERVAL: "1"
      IV_JOB_BU_BACKUP_TIME: "22:22"
      IV_JOB_BU_BACKUPS_TO_KEEP: "5"
      IV_JOB_GC_VOLUME_PATTERN: "<knowledge-graph-name>"
      IV_JOB_GC_GARBAGE_COLLECT_INTERVAL: "1"
      IV_JOB_GC_GARBAGE_COLLECT_TIME: "22:33"
    volumes:
      - "mediator-volumes:/mediator/volumes"
      - "mediator-backup:/mediator/backup"
    ports:
      - "30000:30000"
      - "30001:30001"
    restart: unless-stopped
  jobclient:
    image: container-registry.example.com/iviews-jobclient:<image-tag-2>
    environment:
      IV_HOST: "mediator:30001"
      IV_VOLUME: "<knowledge-graph-name"
      IV_AUTHENTICATION: "${AUTH_TOKEN:?AUTH_TOKEN not configured}"
      IV_JOB_POOLS: "script,index"
    depends_on:
      - mediator
    deploy:
```

```
      replicas: 1
    restart: unless-stopped
  bridge:
    image: container-registry.example.com/iviews-bridge:<image-tag-3>
    environment:
      IV_HOST: "mediator:30001"
      IV_VOLUME: "<knowledge-graph-name>"
      IV_AUTHENTICATION: "${AUTH_TOKEN:?AUTH_TOKEN not configured}"
    depends_on:
      - mediator
      - jobclient
    ports:
      - "8815:8815"
    deploy:
      replicas: 1
    restart: unless-stopped
volumes:
  mediator-volumes:
  mediator-backup:
```

5.2. kubernetes Konfiguration

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mediator
  labels:
    app: mediator
spec:
  selector:
    matchLabels:
      app: mediator
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: mediator
    spec:
      containers:
      - name: mediator
        image: container-registry.example.com/iviews-mediator:<image-tag-
1>
        ports:
```

```
        - containerPort: 30000
        - containerPort: 30001
        env:
        - name: IV_PASSWORD
          valueFrom:
            secretKeyRef:
              name: iviews-secret
              key: MEDIATOR_PASSWORD
        - name: IV_INTERFACES
          value: "http://0.0.0.0:30000,cnp://0.0.0.0:30001"
        - name: IV_SCHEDULED_JOBS
          value: "job-bu,job-gc"
        - name: IV_JOB_BU_VOLUME_PATTERN
          valueFrom:
            configMapKeyRef:
              name: iviews-config
              key: VOLUME
        - name: IV_JOB_BU_BACKUP_INTERVAL
          value: "1"
        - name: IV_JOB_BU_BACKUP_TIME
          value: "22:22"
        - name: IV_JOB_BU_BACKUPS_TO_KEEP
          value: "1"
        - name: IV_JOB_GC_VOLUME_PATTERN
          valueFrom:
            configMapKeyRef:
              name: iviews-config
              key: VOLUME
        - name: IV_JOB_GC_GARBAGE_COLLECT_INTERVAL
          value: "1"
        - name: IV_JOB_GC_GARBAGE_COLLECT_TIME
          value: "22:33"
        volumeMounts:
        - mountPath: /mediator/volumes
          name: mediator-volumes
        - mountPath: /mediator/backup
          name: mediator-backup
      volumes:
      - name: mediator-volumes
        persistentVolumeClaim:
          claimName: mediator-volumes
      - name: mediator-backup
        persistentVolumeClaim:
          claimName: mediator-backup
      securityContext:
        runAsUser: 10000
```

```
        runAsGroup: 10000
        fsGroup: 10000
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mediator-volumes
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 20Gi
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mediator-backup
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 100Gi
---
apiVersion: v1
kind: Service
metadata:
  name: mediator
  labels:
    app: mediator
spec:
  selector:
    app: mediator
  ports:
  - name: mediator-http
    port: 30000
  - name: mediator-cnp
    port: 30001
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: mediator
spec:
  policyTypes:
```

```
  - Ingress
  podSelector:
    matchLabels:
      app: mediator
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: jobclient
    - podSelector:
        matchLabels:
          app: bridge
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: jobclient
  labels:
    app: jobclient
spec:
  selector:
    matchLabels:
      app: jobclient
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: jobclient
    spec:
      containers:
      - name: jobclient
        image: container-registry.example.com/iviews-mediator:<image-tag-
2>
        imagePullPolicy: IfNotPresent
        env:
        - name: IV_HOST
          value: "mediator:30001"
        - name: IV_VOLUME
          valueFrom:
            configMapKeyRef:
              name: iviews-config
              key: VOLUME
        - name: IV_AUTHENTICATION
          valueFrom:
            secretKeyRef:
```

```
              name: iviews-secret
              key: AUTHENTICATION
        - name: IV_JOB_POOLS
          value: script,index,query
      securityContext:
        runAsUser: 10000
        runAsGroup: 10000
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: bridge
  labels:
    app: bridge
spec:
  selector:
    matchLabels:
      app: bridge
  strategy:
    type: Recreate
  replicas: 1
  template:
    metadata:
      labels:
        app: bridge
    spec:
      containers:
      - name: bridge
        image: container-registry.example.com/iviews-bridge:<image-tag-3>
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 8815
        env:
        - name: IV_HOST
          value: "mediator:30001"
        - name: IV_VOLUME
          valueFrom:
            configMapKeyRef:
              name: iviews-config
              key: VOLUME
        - name: IV_AUTHENTICATION
          valueFrom:
            secretKeyRef:
              name: iviews-secret
              key: AUTHENTICATION
      securityContext:
```

```
        runAsUser: 10000
        runAsGroup: 10000
---
apiVersion: v1
kind: Service
metadata:
  name: bridge
  labels:
    app: bridge
spec:
  selector:
    app: bridge
  ports:
  - name: bridge
    port: 8815
---
apiVersion: v1
kind: Secret
metadata:
  name: iviews-secret
type: Opaque
data:
  MEDIATOR_PASSWORD: <encoded-mediator-password>
  AUTHENTICATION: <encoded-system-account>
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: iviews-config
data:
  VOLUME: <knowledge-graph-name>
```