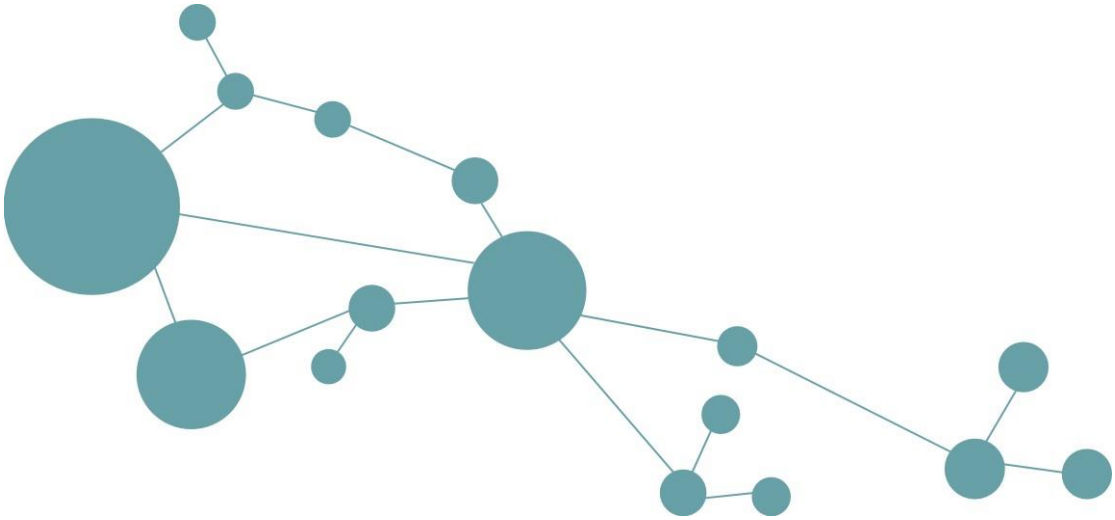


i-views-Plugin für JetBrains-Entwicklungsumgebungen

i·views

Contents

1 Introduction

Empolis Intelligent Views stellt das i-views Plugin für *JetBrains*-Entwicklungsumgebungen zur Verfügung, um Entwickler in ihrer Arbeit mit *i-views* zu unterstützen. Das Plugin ermöglicht eine Synchronisation von Skripten zwischen der Entwicklungsumgebung und einer *i-views*-Datenbank. Außerdem erweitert es die Sprachunterstützung der Entwicklungsumgebung mit den in *i-views* integrierten, eigenen Programmiersprachen.

2 Installation

2.1 Compatibility with JetBrains products

Das Plugin ist mit allen IDE-Produkten ab Version 2020.2 von *JetBrains* kompatibel, die auf der *IntelliJ*-Plattform basieren.

Konkret werden die folgenden Produkte und Versionen unterstützt:

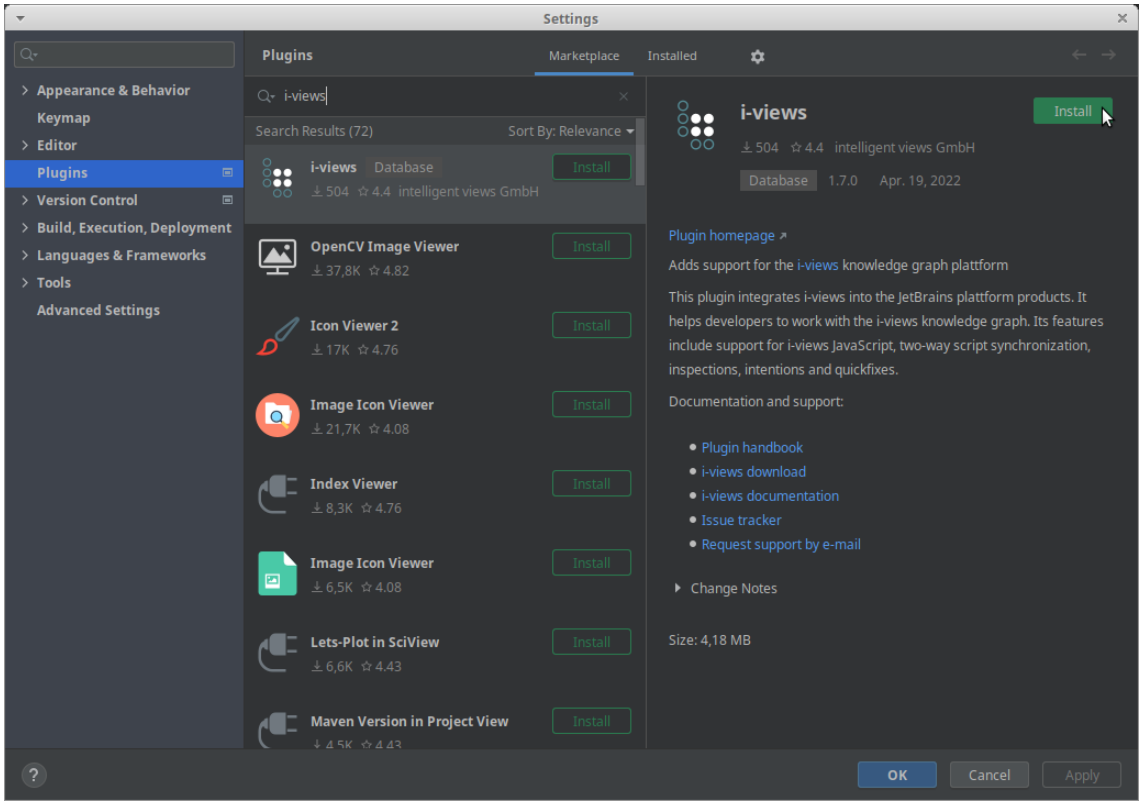
IDE	**Versionen**	**Einschränkungen**
IntelliJ Ultimate	ab 2020.2	
IntelliJ Community	ab 2020.2	Keine Unterstützung für i-views JavaScript
WebStorm	ab 2020.2	
PhpStorm	ab 2020.2	
RubyMine	ab 2020.2	
AppCode	ab 2020.2	
PyCharm Professional	ab 2020.2	
CLion	ab 2020.2	Keine Unterstützung für i-views JavaScript

2.2 Plugin installation in your development environment

Die Erweiterung ist im offiziellen JetBrains Marketplace verfügbar.

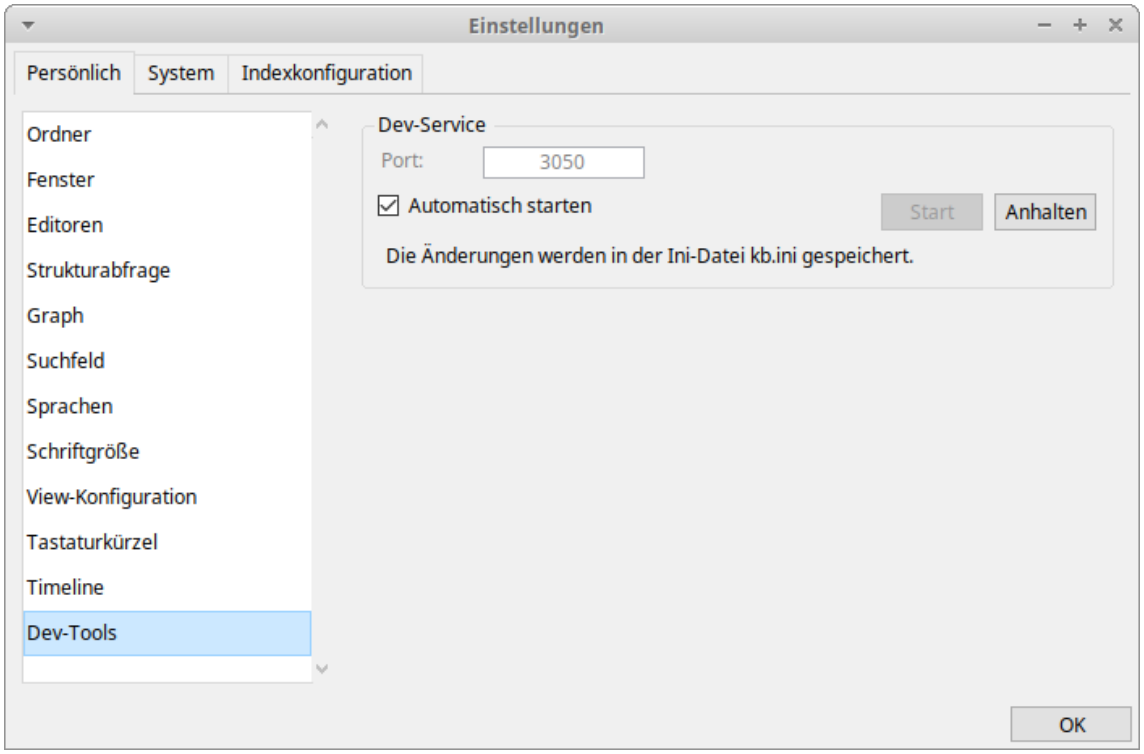
Das Plugin wird wie andere Erweiterungen über die Plugin-Verwaltung der Entwicklungsumgebung installiert:

- Einstellungen öffnen (**File ->** **Settings** oder **Ctrl+Alt+S**)
- Plugins auswählen
- Im Marktplatz nach **i-views** suchen
- Über die Schaltfläche **Install** oberhalb der Informationen kann das Plugin installiert werden

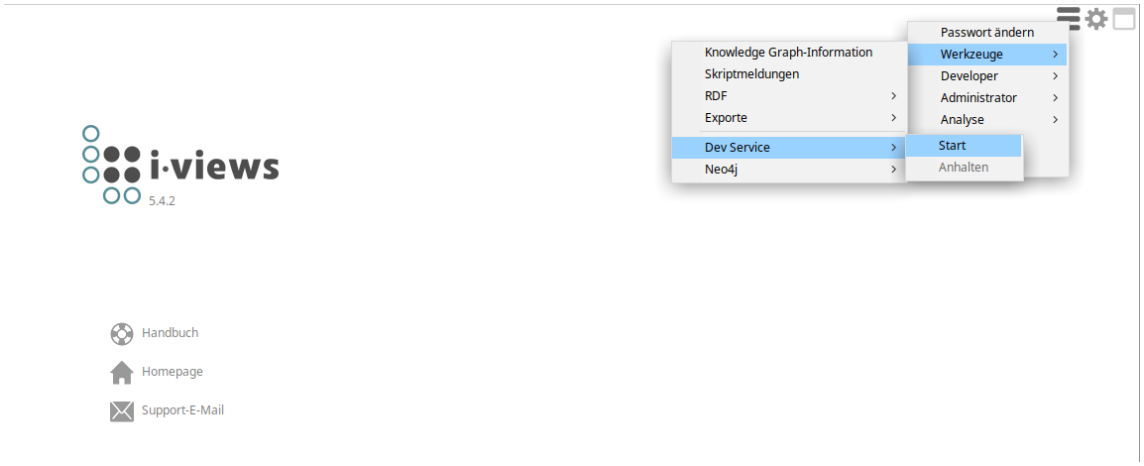


2.3 Configuration of the synchronization service

Um den Synchronisationsdienst verwenden zu können, muss er zunächst in den *Knowledge-Builder* Einstellungen konfiguriert werden:



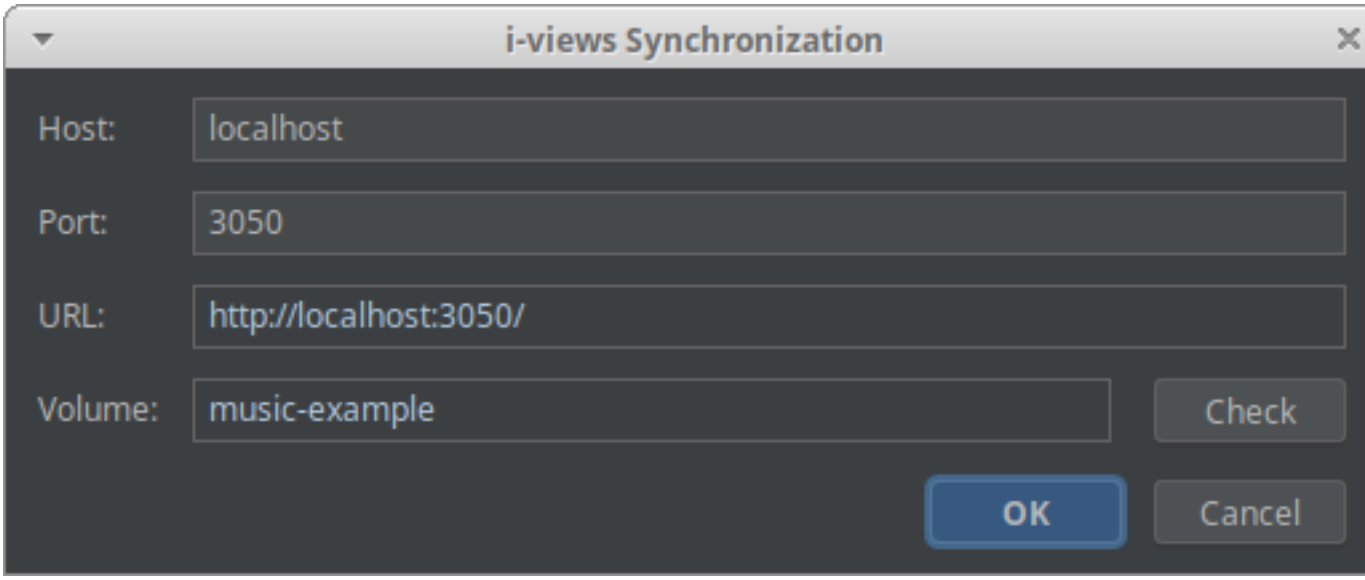
Danach kann der Dienst unter **Aktionen** -> **Werkzeuge** -> **Dev Service** -> **Start** aktiviert und unter **Aktionen** -> **Werkzeuge** -> **Dev Service** -> **Anhalten** wieder deaktiviert werden.



2.4 Configuration in your IDE

In der Entwicklungsumgebung wird über **File** -> **New** -> **Project...** ein neues Projekt angelegt. In der entstandenen Verzeichnisstruktur wird mittels **File** -> **New** -> **Directory** ein Verzeichnis erzeugt, in das die zu synchronisierenden Skripte abgelegt werden.

Zur Kommunikation mit dem Synchronisationsdienst muss das Plugin konfiguriert werden. Dazu wird das Synchronisationsverzeichnis angeklickt und über den Menüpunkt **Synchronize with i-views...** die Synchronisation aufgerufen. Der Menüpunkt kann mittels Rechtsklick auf das Synchronisationsverzeichnis, über das Menü **File** oder per Tastenkombination *Alt+Umschalt+K* erreicht werden. Es erscheint ein Konfigurationsdialog:



Dort wird die Verbindung zum *Knowledge-Builder* konfiguriert. Notwendig sind der Name des Rechners, auf dem sich der *Knowledge-Builder* befindet (**Host**), sowie die Portnummer, über die der im *Knowledge-Builder* gestartete Synchronisationsdienst erreichbar ist (**Port**). Beide Einträge werden automatisch zu einer Netzadresse (**URL**) zusammengebaut, über die der Synchronisationsdient angesprochen wird.

Per Klick auf **Check** lässt sich die Konfiguration prüfen. Wenn eine Datenbank über den konfigurierten Weg gefunden werden konnte, wird ihr Name im Feld **Volume** angezeigt. Ein Klick auf **OK** speichert die Konfiguration und startet die Synchronisation. **Cancel** bricht die Synchronisation ab.

Die im Konfigurationsdialog getroffenen Einstellungen werden im Projekt in der Datei *i-views.ini* gespeichert und können dort nachträglich geändert werden. Entsprechend taucht der Konfigurationsdialog bei zukünftigen Synchronisationsaufrufen nicht mehr auf.

3 Usage

3.1 Synchronization between IDE and Knowledge-Builder

3.1.1 General information

Die Synchronisation zwischen Entwicklungsumgebung und *i-views*-Datenbank erfolgt über eine HTTP-Schnittstelle, den sogenannten *Dev Service*. Über sie können öffentlich registrierte Skripte aus der Datenbank in die Entwicklungsumgebung übernommen, dort bearbeitet, wie andere Daten auch unter Versionskontrolle gehalten und in geänderter Form wieder in die Datenbank zurückübertragen werden. Abweichende Skriptversionen zwischen der Datenbank und der Entwicklungsumgebung können dabei zusammengeführt und Konflikte können aufgelöst werden.

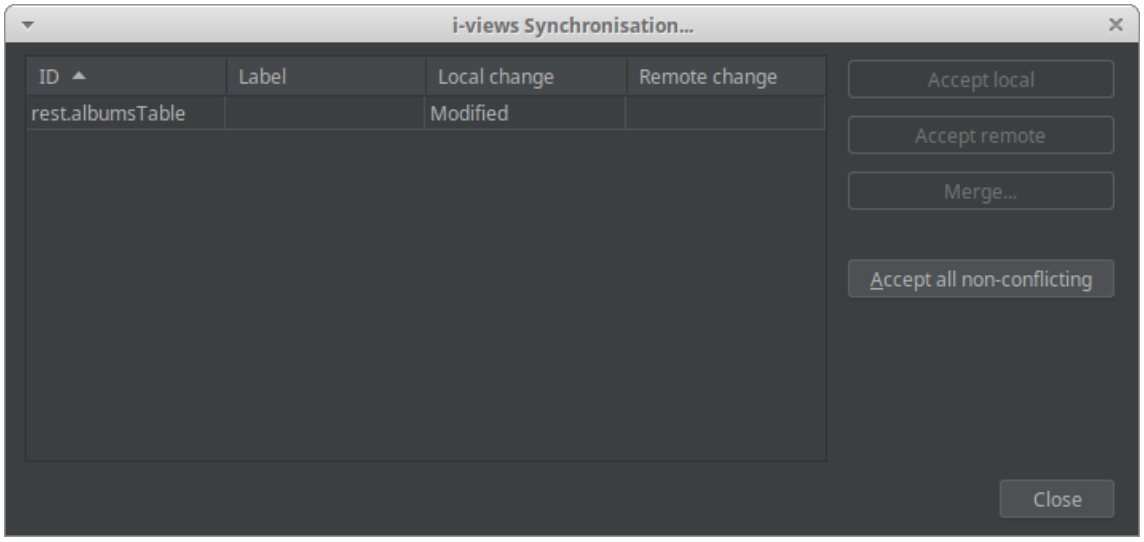
Die Synchronisation der beiden Datenbestände, d.h. der Entwicklungsumgebung einerseits und der *i-views*-Datenbank andererseits, verläuft nicht kontinuierlich, sondern punktuell per Aufruf. Sollen also nach einer Synchronisation Änderungen an Skripten in einem Datenbestand in den anderen Datenbestand übernommen werden, ist ein erneuter Aufruf der Synchronisation notwendig.

Nach jeder Synchronisation werden Metadaten aus der *i-views*-Datenbank, beispielsweise über die verfügbaren Suchen, Skripte und Elementtypen, in die Entwicklungsumgebung übertragen. Die meisten Funktionen des Plugins greifen im Zuge ihres Betriebs auf diese Metadaten und nicht auf die *i-views*-Datenbank zu. Dies ermöglicht eine nur leicht beschränkte Arbeit mit der Entwicklungsumgebung auch dann, wenn eine Synchronisation mit der *i-views*-Datenbank gerade nicht möglich ist.

3.1.2 Sequence of a synchronisation

1. Die zu verwendende *i-views*-Datenbank wird im *Knowledge-Builder* geöffnet.
2. Der Synchronisationsdienst (*Dev Service*) wird im *Knowledge-Builder* gestartet.
3. Das zur Synchronisation gewählte Verzeichnis wird in der Entwicklungsumgebung angewählt.
4. Die Synchronisation wird über *Synchronize with i-views...* aufgerufen.
5. Im Synchronisationsdialog werden Skripte in den beiden Datenbeständen miteinander harmonisiert. Sind beide Datenbestände identisch, erscheint anstelle des Synchronisationsdialogs eine entsprechende Meldung.

3.1.3 The synchronisation dialog



Der linke Bereich stellt alle von der Synchronisation erfassten Skripte mit ihrem Registrierungsschlüssel (*ID*) und, soweit vorhanden, mit ihrem Namen (*Label*) dar. Für jeweils beide Skriptversionen, in der Entwicklungsumgebung (*Local change*) und in der *i-views*-Datenbank (*Remote change*), jedes Skripts deutet er außerdem einen etwaig vorhandenen Unterschied zwischen ihrem aktuellen und ihrem vorangegangenen Zustand aus. Mögliche Zustandswechsel sind neu (*New*), gelöscht (*Deleted*) und verändert (*Modified*).

Per Klick auf einen Spaltenkopf lassen sich die Tabellenzeilen nach der angeklickten Spalte sortieren. Als Voreinstellung ist die Liste nach dem Registrierungsschlüssel der Skripte geordnet.

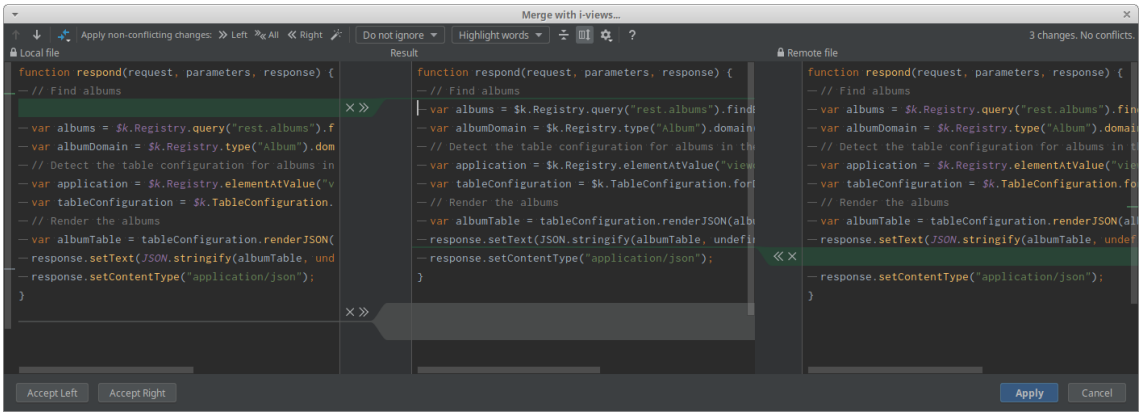
Im rechten Bereich können Operationen für die im linken Bereich erfassten Skripte ausgelöst werden. Dazu werden die gewünschten Skripte vorher in der Liste markiert. Folgende Operationen sind möglich:

Accept local: Die in der Entwicklungsumgebung vorhandene Skriptversion wird in die *i-views*-

Datenbank übernommen.

Accept remote: Die in der *i-views*-Datenbank vorhandene Skriptversion wird in die Entwicklungsumgebung übernommen.

Merge: Beide Skriptversionen werden nebeneinander geöffnet und dienen als Vorlagen für eine bearbeitbare dritte, mittig platzierte Skriptversion.



Über **Apply** wird diese dritte Version in beide Datenbestände übernommen, d.h. danach sind die Skriptversionen in der Entwicklungsumgebung und in der *i-views*-Datenbank zueinander identisch. **Accept left** und **Accept right** hingegen führen zur alleinigen Übernahme der linken beziehungsweise rechten Skriptversion. **Abort** bricht die Gegenüberstellung ohne Änderungen ab.

Accept all non-conflicting: Alle Skriptversionen von allen Skripten, die nur in einem Datenbestand geändert wurden oder deren beide Datenbestände inhaltlich auf die exakt gleiche Weise verändert wurden, d.h. zwischen deren Skriptversionen kein Konflikt besteht, werden in den jeweils anderen Datenbestand übernommen. Zur Vermeidung eines unbeabsichtigten Datenverlustes funktioniert diese Operation nicht für Skripte, deren Skriptversion in der Entwicklungsumgebung gelöscht wurde.

Aufgerufene Operationen werden sofort durchgeführt und bei Erfolg aus der Liste entfernt. Sobald alle Einträge verarbeitet wurden, wird der Dialog beendet. Alternativ kann der Dialog jederzeit mittels **Close** geschlossen werden.

Nachfolgend ein Überblick über die jeweiligen Konsequenzen aller Operationen für alle Zustandswechselkombinationen:

Zustand in der Entwicklungsumgebung	**Zustand in der Datenbank**	**Ergebnis von "Accept local"**	**Ergebnis von "Accept remote"**	**Ergebnis von "Merge"**	**Ergebnis von "Accept all non-conflicting"**
Created	-	Das Skript wird in der Datenbank angelegt.	Das Skript wird in der Entwicklungsumgebung gelöscht.	-	Das Skript wird in der Datenbank angelegt.

-	Created	Das Skript wird in der Datenbank gelöscht.	Das Skript wird in der Entwicklungsumgebung angelegt.	-	Das Skript wird in der Datenbank angelegt.
Created	Created	Die Skriptversion aus der Entwicklungsumgebung überschreibt die Skriptversion in der Datenbank.	Die Skriptversion aus der Datenbank überschreibt die Skriptversion in der Entwicklungsumgebung.	Vergleich der beiden Skriptversionen in einem eigenen Fenster.	-
Modified	-	Die Skriptversion aus der Entwicklungsumgebung überschreibt die Skriptversion in der Datenbank.	Die Skriptversion aus der Datenbank überschreibt die Skriptversion in der Entwicklungsumgebung.	Vergleich der beiden Skriptversionen in einem eigenen Fenster.	Die Skriptversion aus der Entwicklungsumgebung überschreibt die Skriptversion in der Datenbank.
-	Modified	Die Skriptversion aus der Entwicklungsumgebung überschreibt die Skriptversion in der Datenbank.	Die Skriptversion aus der Datenbank überschreibt die Skriptversion in der Entwicklungsumgebung.	Vergleich der beiden Skriptversionen in einem eigenen Fenster.	Die Skriptversion aus der Datenbank überschreibt die Skriptversion in der Entwicklungsumgebung.
Modified	Modified	Die Skriptversion aus der Entwicklungsumgebung überschreibt die Skriptversion in der Datenbank.	Die Skriptversion aus der Datenbank überschreibt die Skriptversion in der Entwicklungsumgebung.	Vergleich der beiden Skriptversionen in einem eigenen Fenster.	-
Deleted	-	Das Skript wird in der Datenbank gelöscht.	Das Skript wird in der Entwicklungsumgebung angelegt.	-	-
-	Deleted	Das Skript wird in der Datenbank angelegt.	Das Skript wird in der Entwicklungsumgebung gelöscht.	-	Das Skript wird in der Entwicklungsumgebung gelöscht.
Deleted	Deleted	Kein Effekt.	Kein Effekt.	-	Kein Effekt.
Modified	Deleted	Das veränderte Skript wird in der Datenbank angelegt.	Das veränderte Skript wird in der Entwicklungsumgebung gelöscht.	-	-

Deleted	Modified	Das veränderte Skript wird in der Datenbank gelöscht.	Das veränderte Skript wird in der Entwicklungsumgebung angelegt.	-	-

3.1.4 The synchronisation directory

Nach der ersten Synchronisation werden im Synchronisationsverzeichnis die Unterverzeichnisse *data* und *scripts* sowie die Datei *i-views.ini* angelegt.

3.1.4.1 The subdirectory data

Im Verzeichnis *data* werden Daten aus der *i-views*-Datenbank abgelegt, auf die bestimmte Funktionen des Plugins zugreifen. Nach jeder Synchronisation werden diese Daten aktualisiert.

queries.json Enthält die Liste aller öffentlich registrierten Suchen. Für jede Suche werden der Registrierungsschlüssel, die Bezeichnung, der Suchtyp sowie die verfügbaren Parameter und deren Datentypen angegeben.

scripts.json Enthält die Liste aller öffentlich registrieren Skripte. Für jedes Skript werden der Registrierungsschlüssel, die Bezeichnung, die Programmiersprache, der zuletzt synchronisierte Inhalt sowie eine Prüfsumme angegeben.

version.json Enthält die Version des *Knowledge-Builders*, mit dem zuletzt synchronisiert wurde.

viewconfigApplications.json Enthält die Liste aller verfügbaren *ViewConfig*-Anwendungen. Für jede Anwendung werden der interne Name und die Bezeichnung angegeben.

viewconfigTables.json Enthält die Liste aller verfügbaren *ViewConfig*-Tabellen. Für jede Tabelle werden die ID, die Bezeichnung und die Liste der zugehörigen ViewConfig-Anwendungen angegeben.

volumeTypes.json Enthält die Liste aller Elementtypen, die einen definierten internen Namen haben. Für jeden Typ werden der interne Name, die Bezeichnung, die Elementtypart, die Objekt-ID sowie die aktuellen Konfigurationseigenschaften angegeben.

3.1.4.2 The subdirectory scripts

Im Verzeichnis *scripts* werden alle von der Synchronisation erfassten Skripte abgelegt. Deren genaue Position hängt von der Struktur ihrer Registrierungsschlüssel ab. Mehr dazu im Kapitel über Namens- und Positionssysteme für Skripte.

3.1.4.3 The configuration file i-views.ini

In dieser Datei *i-views.ini* werden die für zukünftige Synchronisationen notwendigen Einstellungen gespeichert:

- **volume**: Der Name der *i-views*-Datenbank, mit der synchronisiert werden darf. Falls kein Name gesetzt wird, ist eine Synchronisation mit beliebigen *i-views*-Datenbanken möglich. Zur Vermeidung eines Datenverlusts wegen unbeabsichtigter Löschungen ist indes von einer derartigen Freigabe abzuraten.
- **host**: Der Name des Rechners, auf dem sich der *Knowledge-Builder* befindet, mit dem synchronisiert werden soll. Im Regelfall ist dies *localhost*, d.h. der eigene Rechner.
- **port**: Der Port, über den der Synchronisationsdienst erreicht werden soll. Der Standardwert ist *3050*.
- **dir**: Der relative Pfad der Verzeichnisse *data* und *scripts* relativ zur Position der Datei *i-views.ini*. Im Regelfall ist der eingetragene Wert leer, d.h. die beiden Verzeichnisse werden als direkte Unterverzeichnisse des Verzeichnisses angelegt, in dem die Datei *i-views.ini* liegt.

Beispiel einer gewöhnlichen Konfiguration:

```
volume = music example
host = localhost
port = 3050
dir =
```

Beispiel einer komplizierteren Konfiguration:

```
volume = music example
host = virtualMachine1
port = 8000
dir = i-views/daten
```

3.2 Working with scripts

3.2.1 Naming of scripts

3.2.1.1 Within the k-infinity database

In der *i-views*-Datenbank ist die Skriptverwaltung im linken Navigationsfenster unter dem Eintrag **Technik** -> **Registrierte Objekte** -> **Skripte** erreichbar. Über die entsprechenden Schaltflächen in der oberen Menüleiste können dort Skripte neu angelegt, bearbeitet oder gelöscht werden.

Neben der Identifikation dient der Registrierungsschlüssel eines Skripts außerdem als Ordnungswerkzeug. Jeder darin enthaltene Punkt (.) trennt die links von diesem Punkt stehenden Zeichen bis zum nächsten Punkt oder bis zum ersten Zeichen des Registrierungsschlüssels ab und erzeugt daraus eine gleichnamige Strukturordner unterhalb der Rubrik Skripte im linken Navigationsfenster. Werden auf diese Weise mehrere Ordner in einem Registrierungsschlüssel definiert, werden sie ineinander geschachtelt. Die Hierarchie der Ordner verläuft in Leserichtung des Registrierungsschlüssels absteigend. Ein Skript mit dem Registrierungsschlüssel *rest.artists.artistsList* wird folglich in der Ordnerstruktur *Skripte/rest/artists/* abgelegt.

3.2.1.2 Within the IDE

In der Entwicklungsumgebung werden Skripte als lokale Dateien mit der Dateiendung **.kjs* in eine Verzeichnisstruktur eingehängt. Sie befinden sich in einem beliebig anlegbaren Verzeichnis unterhalb des Verzeichnisses *scripts*. *scripts* ist wiederum ein direktes Unterverzeichnis des Synchronisationsverzeichnisses. Allgemein ergibt sich für jedes Skript der folgende Verzeichnispfad: *Projektverzeichnis/.../Synchronisationsverzeichnis/scripts/.../Skriptdatei*.

JavaScript-Dateien lassen sich über **File** -> **New** -> **i-views JavaScript** erzeugen. Diese und weitere Operationen zur Skriptverwaltung sind alternativ über das Kontextmenü der Projektansicht zugänglich.

3.2.1.3 Mapping of both systems during a synchronization

Die Namens- und Positionssysteme beider Datenbestände werden bei der Synchronisation aufeinander abgebildet.

Bei einer Übernahme aus der *i-views*-Datenbank in die Entwicklungsumgebung wird die im Registrierungsschlüssel kodifizierte Ordnerstruktur in Verzeichnisse umgewandelt, das Segment nach dem letzten Punkt wird zum Dateinamen, als Dateiendung wird *.kjs* gesetzt.

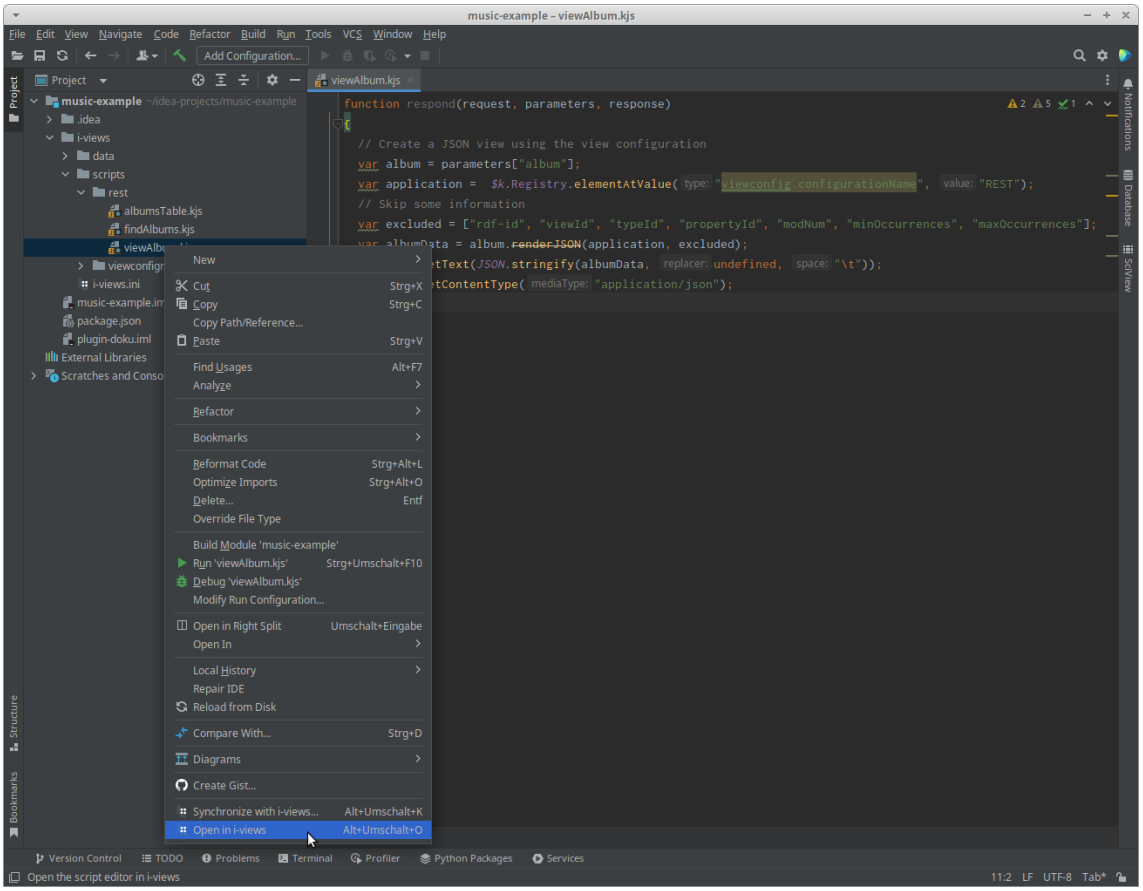
Bei einer Übernahme aus der Entwicklungsumgebung in die *i-views*-Datenbank wird die Verzeichnisstruktur samt Dateiname ohne Dateiendung unterhalb des *scripts*-Verzeichnisses in den Registrierungsschlüssel geschrieben.

Beispielsweise entspricht der Registrierungsschlüssel *rest.viewAlbum* eines Skriptes des Typs *JavaScript* in der *i-views*-Datenbank dem Skriptdateipfad *scripts/rest/viewAlbum.kjs* in der Entwicklungsumgebung.

Hinweis: Umbenennungen von Skripten werden aktuell nicht unterstützt. Wird zum Beispiel das Skript *rest.artists* in *rest.artistsList* umbenannt, so stellt die nächste Synchronisierung das Skript *rest.artists* als gelöscht und das Skript *rest.artistsList* als neu an.

3.2.2 Opening a script in the Knowledge-Builder

Per Menüpunkt **Open in i-views** im Kontextmenü der Skriptdatei oder über die Tastenkombination *Umschalt+Alt+O* ist das Öffnen eines Skriptes im *Knowledge-Builder* aus der Entwicklungsumgebung heraus möglich. Werden mehrere Skriptdateien über *Strg+Mausklick* ausgewählt, können diese parallel geöffnet werden.



Nach Aufruf der Operation öffnet sich im *Knowledge-Builder* für jedes gewählte Skript ein neues Fenster, in dem das Skript bearbeitet werden kann. Alternativ kann dieses Fenster auch über einen Klick auf das Skript in der Skriptverwaltung des *Knowledge-Builders* erreicht werden.

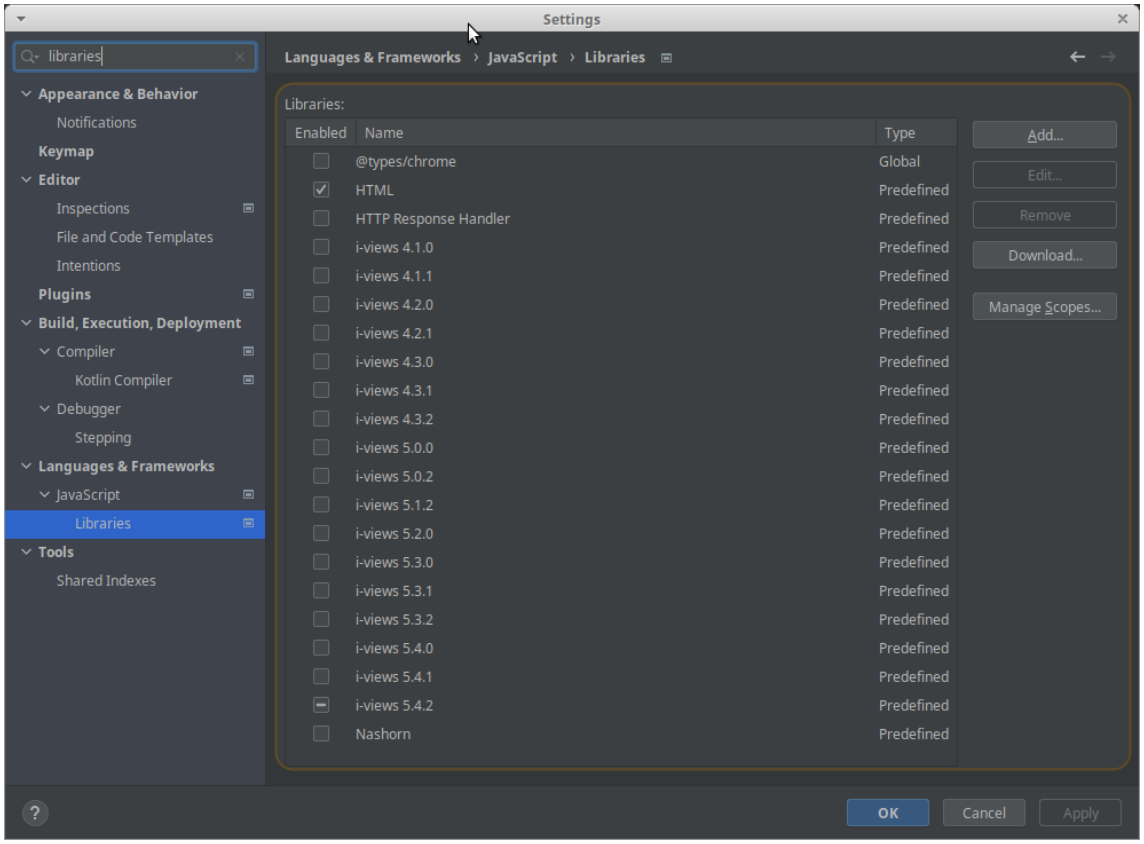
Diese Operation ist nur bei aktiviertem Synchronisationsdienst und nur für Skripte verfügbar, die synchronisiert sind, d.h. die jeweils in beiden Datenbeständen identisch sind. Sie kann von Vorteil sein, um in der Entwicklungsumgebung bearbeitete und synchronisierte Skripte nicht erneut in der möglicherweise langen Skriptliste der Skriptverwaltung im *Knowledge-Builder* ausfindig machen zu müssen, bevor sie dort weiter bearbeitet werden können.

3.2.3 Convinience features for k-infinity

3.2.3.1 Versioning

i-views unterstützt *ECMAScript* in der Version 5.1 und bietet vordefinierte Funktionen, um auf die *i-views*-Datenbank zugreifen zu können. Der Umfang dieser Funktionen unterscheidet sich je nach Version der eingesetzten *i-views*-Software (ab Version 4.1.0).

Im Plugin sind die vordefinierten Funktionen in Form von Bibliotheken enthalten. Über die Installation werden sie in die Entwicklungsumgebung eingebunden und können über den Menüpfad **File** -> **Settings...** -> **Languages & Frameworks** -> **JavaScript** -> **Libraries** verwaltet werden.

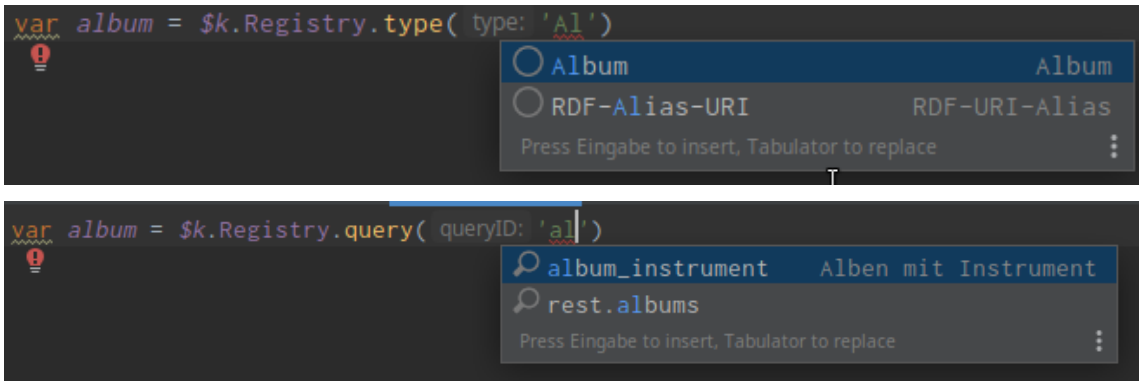


Eine Synchronisation mit der *i-views*-Datenbank aktualisiert die Verknüpfung des Projekts mit der richtigen *JavaScript*-Bibliothek, abhängig von der aufgefundenen *i-views*-Version.

3.2.3.2 Autocompletion

Das Plugin ermöglicht im Texteditor die automatische Vervollständigung interner Namen und Registrierungsschlüssel, die als Argumente einer Funktion übergeben werden. Dies geschieht in Form anklickbarer Vorschläge aus einer Liste.

Die Autovervollständigung lässt sich über die Tastenkombination *Strg+Leertaste* aktivieren. Sie funktioniert nur innerhalb von einfachen oder doppelten Anführungszeichen.



Bei der Ermittlung der Vorschläge werden auch die jeweilige Methode sowie die Position innerhalb der Parameterliste der jeweiligen Methode berücksichtigt, so dass nur Vorschläge erscheinen, die an den jeweiligen Positionen tatsächlich verwendet werden dürfen. Daher werden zum Beispiel für

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

alle verfügbaren internen Namen vorgeschlagen. Für einen Aufruf von

```
$k.Registry.type("example").attributeValue("...")
```

werden jedoch nur die internen Namen vorgeschlagen, die zu Attributen gehören.

Die Vorschläge für Registrierungsschlüssel werden aktuell für Suchen

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

sowie für Abbildungen von Datenquellen

```
$k.Registry.mapping("...")
```

gezeigt.

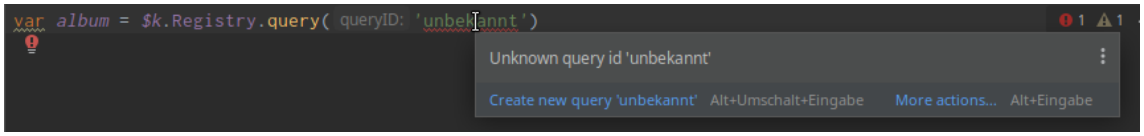
Des Weiteren sind der Entwicklungsumgebung alle in den mitgelieferten JavaScript-Bibliotheken definierten Klassen und Methoden bekannt. Beim Schreiben im Texteditor erfolgen automatisch Vervollständigungsvorschläge in Form einer anklickbaren Liste.

3.2.3.3 Inspections

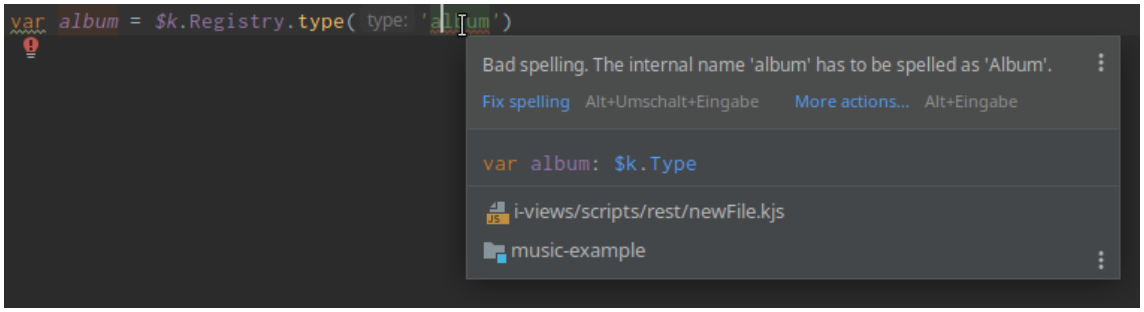
Das Plugin überprüft in Skripten verwendete interne Namen und Registrierungsschlüssel automatisch auf ihre Gültigkeit. Ungültige Verwendungen werden über Warnungen signalisiert, die in der Fußzeile der Entwicklungsumgebung und per Tooltip angezeigt werden. Als Prüfvorlage dient ein in der Entwicklungsumgebung gespeicherter Index, der bei jeder Synchronisation aktualisiert wird. In der Folge funktioniert die Namen- und Schlüsselprüfung auch dann, wenn der Synchronisationsdienst gerade nicht aktiv ist.

Folgende Warnungen sind definiert:

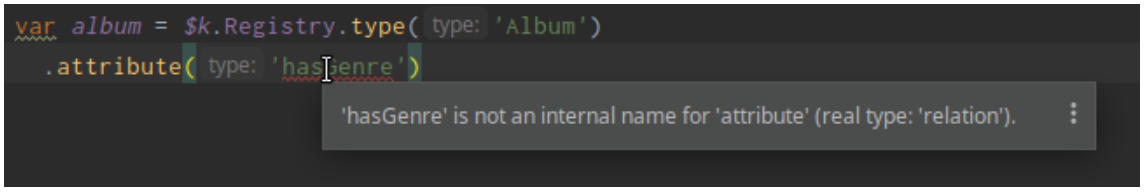
Es wird ein unbekannter interner Name oder Registrierungsschlüssel verwendet.



Ein interner Name oder Registrierungsschlüssel ist zwar bekannt, weist jedoch eine fehlerhafte Groß-/Kleinschreibung auf.



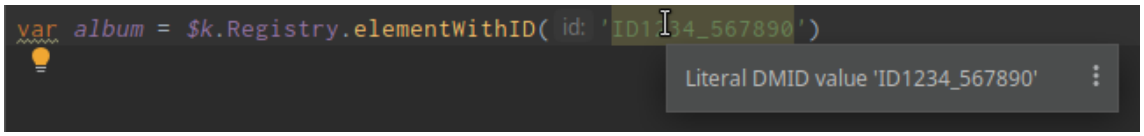
Ein interner Name existiert zwar, aber er ist an ein Element des falschen Typs gebunden.



Ein interner Name eines Attributtyps, der als Argument der Methode *elementAtValue()* übergeben wird, ist nicht mit einem konfigurierten Eindeutigkeitsindex ausgestattet. Eine derartige Indexierung ist sinnvoll, um die Suche nach Attributen des gewählten Attributtyps zu beschleunigen und um vorab zu garantieren, dass jeder Attributwert des gewählten Attributtyps nur einmal existiert.

Eine Prüfung, ob beim Aufruf einer bekannten Suche Argumente für alle notwendigen Parameter übergeben werden, wird zur Zeit nicht unterstützt.

Falls eine technische ID verwendet wird (eine sogenannte DMID), so wird eine Warnung angezeigt. IDs können sich prinzipiell ändern und sollten daher nicht in Skripten verwendet werden. Besser ist hier, die Funktion "elementAtValue" zu verwenden.

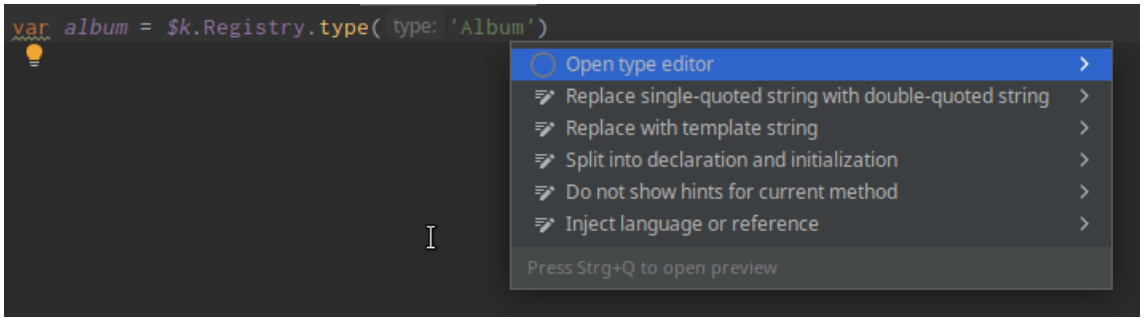


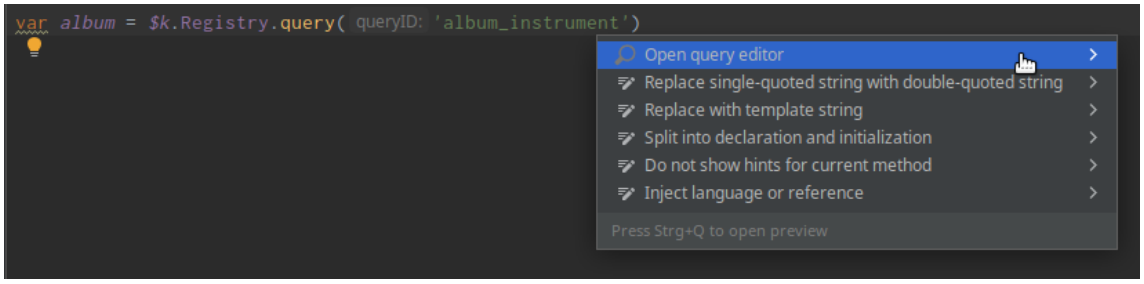
3.2.3.4 Intention actions

Über die in der Entwicklungsumgebung integrierten *Intention Actions* lassen sich im Texteditor spezifische Operationen für interne Namen und Registrierungsschlüssel auslösen. Die Operationen können aus einer Liste ausgewählt werden, die per Tastenkombination *Alt+Eingabe* aufgerufen wird. Die Nutzung dieser Operationen setzt einen aktivierten Synchronisationsdienst voraus.

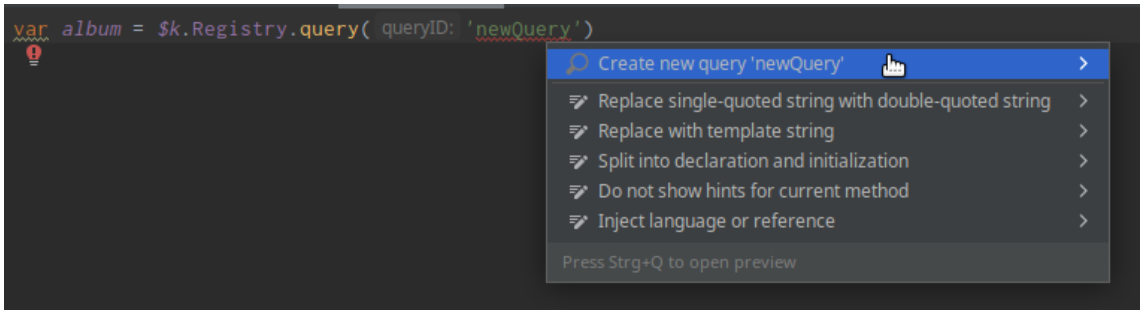
Folgende Operationen werden unterstützt:

Als korrekt erkannte interne Namen und Registrierungsschlüssel lassen sich direkt im *Knowledge-Builder* öffnen. Nach Aufruf der Operation *Open type editor* beziehungsweise *Open query editor* bzw. *Open mapping editor* wird der damit verbundene Elementtyp respektive die damit verbundene Suche oder Abbildung einer Datenquelle in einem neuen Fenster im *Knowledge-Builder* geöffnet. Falls das Fenster bereits geöffnet war, wird es in den Vordergrund geholt.





Wenn ein nicht bekannter Registrierungsschlüssel verwendet wird, kann für diesen über die Operation *Create new query* in der *i-views*-Datenbank eine Struktursuche angelegt werden.

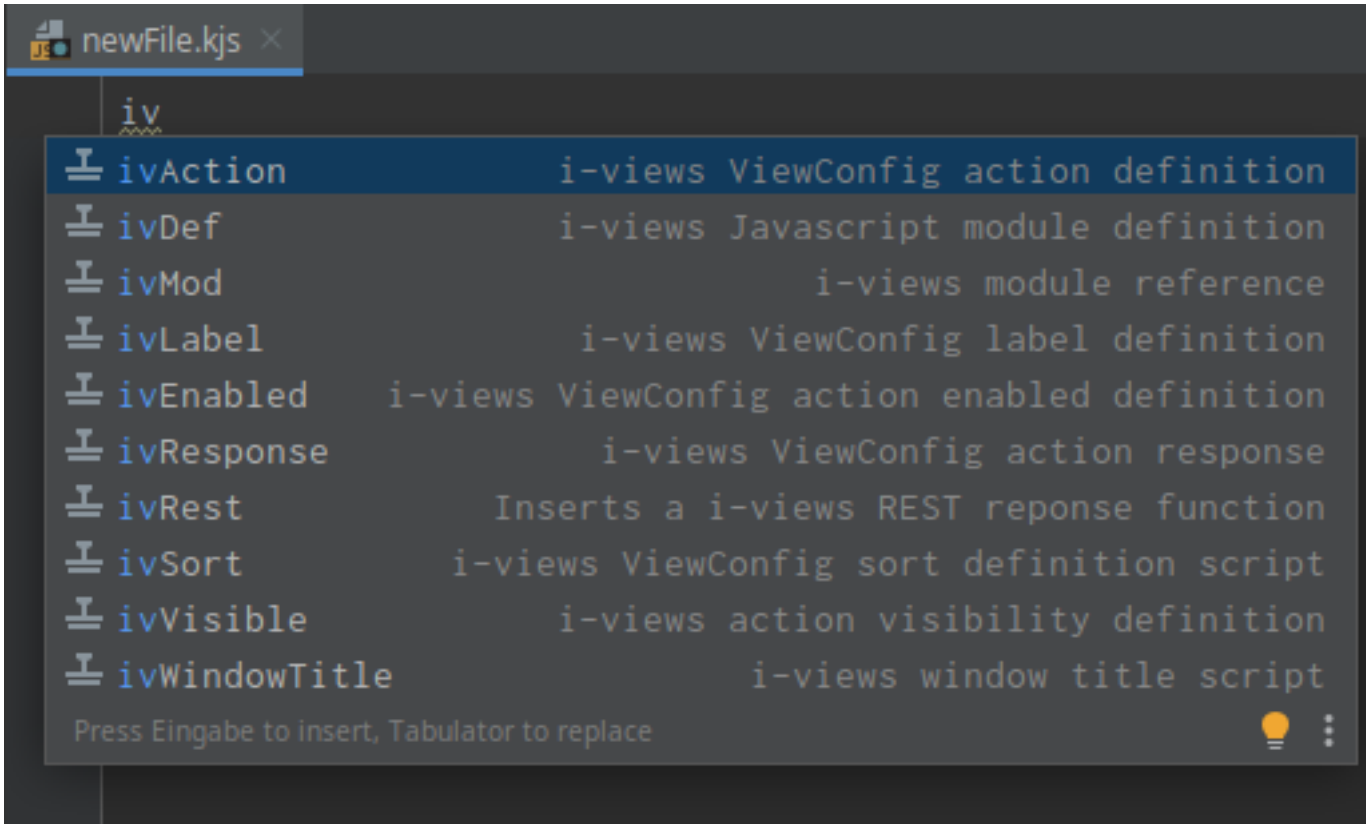


Die Struktursuche wird im Anschluss in einem neuen Fenster der *i-views*-Datenbank geöffnet.

Für *i-views*-spezifische Elemente wird aktuell kein *Refactoring* unterstützt. Das Umbenennen von internen Namen, Registrierungsschlüsseln, Skripten usw. muss manuell durchgeführt werden.

3.2.3.5 Live templates

Das Plugin bringt eine Reihe vordefinierter Funktionsrümpfe mit. Beim Tippen im Texteditor klappt ein Auswahlmenü aus, in dem diese ausgewählt werden können. Die vom Plugin unterstützen Live-Templates haben alle das Prefix *iv*.



Folgende Funktionsrümpfe sind vordefiniert:

- *ivAction*: Funktionsrumpf, um eine *ViewConfig*-Aktion zu verarbeiten.
- *ivDef*: Funktionsrumpf, um ein *JavaScript*-Modul zu definieren.
- *ivMod*: Funktionsrumpf, um ein *JavaScript*-Modul zu referenzieren.
- *ivLabel*: Funktionsrumpf, um die Bezeichnung eines *ViewConfig*-Elementes zu bestimmen.
- *ivEnabled*: Funktionsrumpf, um eine *ViewConfig*-Aktion als aktiviert oder deaktiviert zu markieren.
- *ivResponse*: Funktionsrumpf, um die Anwort auf eine *ViewConfig*-Aktion zu bestimmen.
- *ivRest*: Funktionsrumpf, um einen *REST*-Request zu verarbeiten.
- *ivSort*: Funktionsrumpf, um *ViewConfig*-Elemente zu sortieren.
- *ivVisible*: Funktionsrumpf, um die Sichtbarkeit einer *ViewConfig*-Aktion zu bestimmen.
- *ivWindowTitle*: Funktionsrumpf, um den Fenstertitel des *Knowledge-Builders* zu bestimmen.

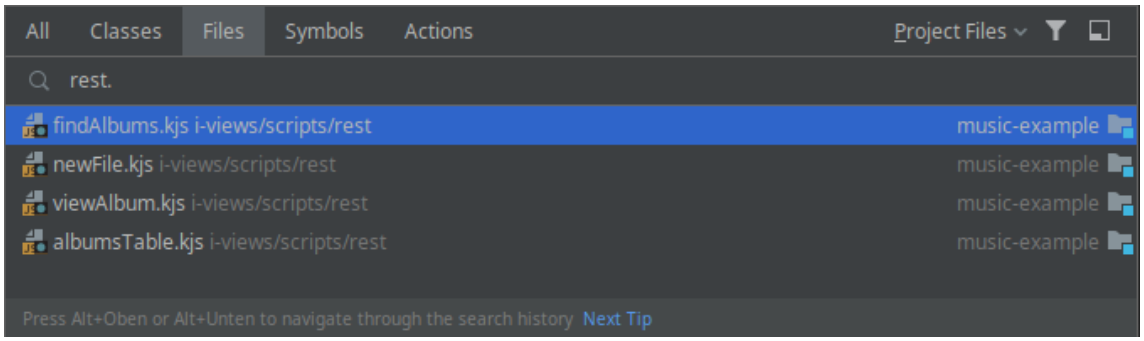
3.3 Navigation

Das Plugin bietet eine vereinfache Navigation zu i-views Elementen. Im Folgenden werden die unterschiedlichen Wege erläutert, mit denen Elemente innerhalb der IDE oder im Knowledge-Builder geöffnet werden können.

3.3.1 Go to file...

Die Funktion "Go to file..." (im Menü unter Navigate > File...) der IDE ist eine einfache Möglichkeit, über die Eingabe eines Dateinamens oder eines Teils alle passenden Dateien im aktuellen Projekt zu sehen. Nach Auswahl eines Vorschlags wird diese zum Bearbeiten geöffnet.

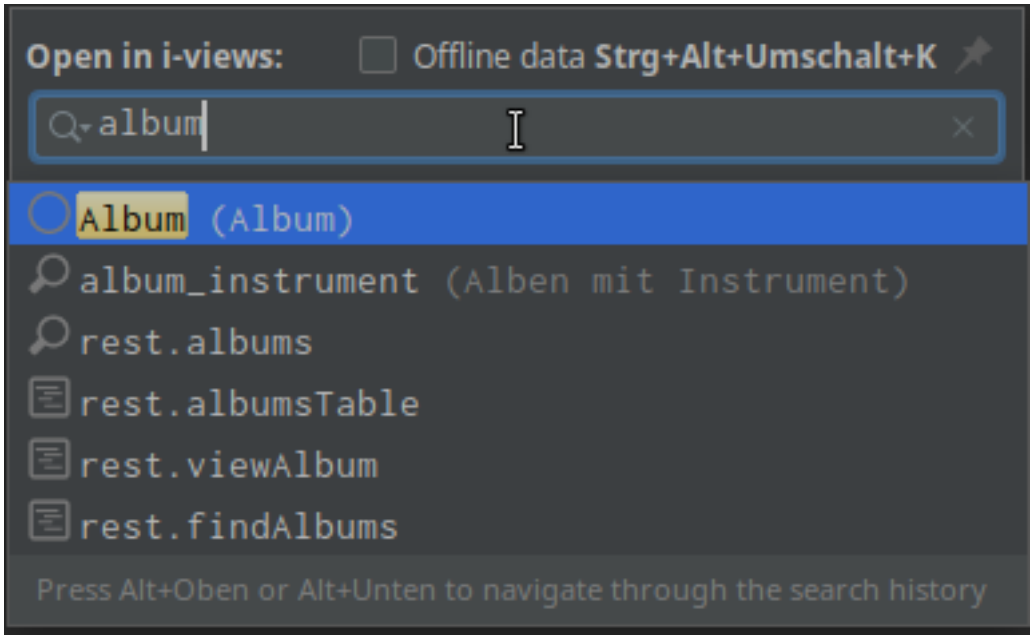
Das Plugin schlägt hier die Registrierungsschlüssel der lokal verfügbaren i-views Skripte vor. Es ist somit also möglich, einen Registrierungsschlüssel einzugeben und alle passenden Skripte als Vorschläge zu sehen.



3.3.2 Open in i-views...

Das Plugin bietet die Möglichkeit, beliebige Elemente über ihren Namen aufzufinden und im Knowledge-Builder zu öffnen.

Im Menü ist diese Funktion unter Navigate > Open i-views element ... zu erreichen. Wählt man diese aus, so erscheint ein Dialog ähnlich zu "Go to file..." oder "Go to class...". Im Eingabebereich kann Text eingegeben werden, mit dem allen passenden i-views Elemente gesucht und vorgeschlagen werden.



Nach Eingabe erscheinen die Vorschläge passender Elemente. Unterstützt werden:

- Skripte
- Suchen
- Typen
- Abbildungen von Datenquellen
- Viewconfig Anwendungen
- Viewconfig Tabellen
- DMID Identifikatoren, bspw. ID123_456 .
- Interne Namen Identifikatoren, bspw. ID~internerName
- sog. Element-at-Value Identifikatoren, bspw. AV~internerName~gesuchterWert. So kann direkt ein Element geöffnet werden, das über einen eindeutigen Attributwert identifizierbar ist.

Es werden nur Vorschläge angezeigt, die in einem aktuell gestarteten Knowledge-Builder geöffnet werden können. Falls kein Knowledge-Builder erreichbar ist, so werden nur Skriptdateien angezeigt (diese Funktion ist auch über die Auswahl von "Offline data" erreichbar).

Nach der Auswahl eines Vorschlags über die Eingabetaste wird das ausgewählte Elememt geöffnet.

3.3.3 Modules

Das Plugin unterstützt die Navigation zu Modulen.

Referenzen auf Module innerhalb von $k.module, $k.require und $k.define werden unterstützt. Die dort verwendeten Zeichenketten sind navigierbar (durch Go To > Declaration bzw. Strg+B oder Strg+Klick),