



Release Notes i-views 6.1

Contents

1. i-views 6.1	1
1.1. New features and extensions	1
1.1.1. Language Component	1
1.1.2. Knowledge-Builder as a Web Application	1
1.1.3. Task Component	1
1.1.4. Custom Components	1
1.1.5. View Configuration and View Config Mapper	2
1.1.6. RDF Import	3
1.1.7. GQL Query Language	3
1.1.8. Other improvements	5
1.1.9. Incompatibilities	6
1.1.10. Removed components	7
1.1.11. Discontinued components	7
2. i-views 6.1.2	8
2.1. New features and extensions	8
2.1.1. Model Context Protocol component	8
2.1.2. Variant model component	8
2.1.3. iiRDS component	8
2.1.4. Custom Components	8
2.1.5. Other improvements	9
2.2. Incompatibilities	9
2.2.1. JavaScript API: iterators as function arguments	9

1. i-views 6.1

1.1. New features and extensions

1.1.1. Language Component

The Language Component provides the schema and functionality needed to configure and use languages as objects in the Knowledge Graph. Starting with version 6.1, this component is an integral part of i-views and enables the selection of languages in the display in both the Knowledge Builder and the View Config Mapper. The previous configuration via attributes is automatically converted during the upgrade to 6.1 and will then function as before.

1.1.2. Knowledge-BUILDER as a Web Application

The Knowledge Builder can now also be implemented as a View Configuration Mapper web application. In doing so, you can largely use the familiar view configuration elements and techniques that are employed in every View Configuration Mapper web application.

A fully configured Knowledge Builder as a web application (abbreviated as "Web KB") is available as a preconfigured custom component. This Web KB can be imported into any knowledge graph and is then immediately ready for use.

1.1.3. Task Component

The Task Component provides the schema and functionality needed to represent tasks executed by a job client as objects in the knowledge graph. This enables the creation of new tasks and provides an overview of task processing within the configured web frontend.

1.1.4. Custom Components

1.1.4.1. Assignment

- Registry objects that only exist as an attribute value without having a registry key are now implicitly assigned to the same component as the element that has the attribute.
- New option to mark relations that are not counted as part of the same component as their source, meaning they will not be exported with it.

1.1.4.2. Tools

- Added a new tool to evaluate the current state of the assignment and analyze issues like cyclic dependencies, multi assignment, layering issues and more.
- The component view of the graph editor
 - distinguishes between unassigned elements and elements of a software component.

- highlights layering errors and shows their reason on hover.
- When the assign button of the assignment tool is disabled, it has a tooltip with the reason.
- Added the option to select all sub-elements until the first assigned one to the assignment tool.
- The tool to define relations as dependencies now also shows relations that are defined as a dependency by software components.

1.1.4.3. Transfer

- The version of a component can be updated when exporting it.
- Folders now also export the information, in which other folders they are contained.
- Relations that lead to a requirement are always exported as a reference and will be overwritten on import. Relations that lead to a dependent are not exported and will not be removed on import.

1.1.4.4. Miscellaneous

- Option for components to store all new instances of assigned types in the same cluster.
- The attribute 'Select elements based on prefix/URI/relation' was replaced by the attribute 'Select elements solely based on selection objects'.

1.1.5. View Configuration and View Config Mapper

1.1.5.1. Appearance & Theming

- Instead of numeric values, width and height configurations are now specified as strings, so that in addition to pixel values, units such as **em**, **vw**, or keywords like **auto** and **fit-content** can be used.
- The default theme (**vcmLight**) can be overridden via the **themes** options configuration. By specifying a theme named **vcmLight** with a custom color or variable definition, the appearance of the entire application can be customized without requiring custom CSS.
- Individual panels can now be assigned their own Vuetify theme, defined via the backend configuration. This allows different panels within an application to be visually distinguished from one another.
- Menus (toolbars, menu bars, list menus) and actions now support an **appearance** configuration in the **Style.extra** field. The appearance of actions can be controlled uniformly via **density**, **size**, **variant**, and **color**.
- Two new dialog templates make it easier to create standard dialogs with headers and footers.
- The spacing between nested panels has been removed. Only content panels continue to receive inner padding, making it easier to style panels to the full browser height and equip them with custom scrollbars.

1.1.5.2. Properties & Form Fields

- In read and edit mode, it can be configured which languages are displayed and whether the translation list is shown collapsed or expanded. In this context, the frontend display has also been improved.
- For input fields with multiple sub-components (e.g. FlexTime, intervals), autosave is now triggered more reliably.
- All Boolean input and display components (checkbox, switch, form) now receive a CSS class reflecting the current state (`no-value`, `true`, `false`). This makes targeted styling via custom CSS easier.
- The display of relation targets in edit fields has been revised and is now more readable and less visually intrusive.
- A new form field for uploading binary files (blobs) is now available. This allows users to upload files directly via forms.

1.1.5.3. Table

- Three new CSS helper classes have been introduced for tables to align the content of table cells: `vcm-table-cell-position-start`, `vcm-table-cell-position-center`, and `vcm-table-cell-position-end`. The classes can be applied to the entire table, individual rows, individual columns, or any combination thereof.

1.1.5.4. Net Navigator

- The dialog for selecting nodes when expanding a graph node is now displayed as a modal dialog instead of a dropdown menu. Nodes are grouped by relation, sorted alphabetically, and can be filtered using a search field.

1.1.5.5. Integration

- The application is now published as an internal npm package `@i-views/vcm-vue` and can thus be included as a library in customization projects.
- Stores, facades, and TypeScript types are now accessible via the standard import of the library, simplifying integration into external applications.
- `npm init @i-views/vcm-vue@latest` provides an interactive CLI tool that generates a fully preconfigured View Config Mapper customization project.

1.1.6. RDF Import

The RDF import now also supports the Turtle syntax, including quoted triples (part of RDF-star).

1.1.7. GQL Query Language

i-views 6.1 introduces a new query language for the knowledge graph: the Graph Query Language (GQL). GQL is an international ISO standard for querying graph databases, modelled after the widely

used Cypher language.

Unlike the graphically built structured query, GQL queries are written as text. The result of a GQL query is always a table — comparable to an Excel spreadsheet or an SQL query response. This makes GQL particularly well suited for evaluations, reports, and data export. As a text-based format, GQL is also well suited for querying the knowledge graph from external systems — including AI agents that can independently formulate and evaluate queries.

1.1.7.1. GQL features and unique selling points

GQL queries describe patterns in the graph, but unlike structured queries they always return a structured result table instead of an unordered set of objects. Each column of the table corresponds to an explicitly named return field — for example a plain attribute value, a computed expression, or a referenced object. This enables capabilities that are not available or only awkwardly achievable with structured queries:

- Aggregation and grouping — functions such as **COUNT**, **SUM**, **AVG**, **MIN**, and **MAX** consolidate values across multiple rows; **GROUP BY** evaluates them per group, for example the number of products per manufacturer.
- Boolean expressions — conditions can be freely combined using **AND**, **OR**, and **NOT**.
- Arithmetic and value processing — mathematical operators as well as string, date, and list functions are available in return expressions and conditions, for example to calculate a discounted price directly in the query.
- Sorting — results can be sorted by any field in ascending or descending order, including fields that are not part of the output.
- Paged output — **LIMIT** and **OFFSET** allow result sets to be split up, which can be used for list views with pagination.
- Variable-length paths — edges can be quantified to search for paths spanning a variable number of hops, for example "all persons reachable within two to four degrees of acquaintance".
- Set operations — multiple sub-queries can be combined using **UNION**, **INTERSECT**, or **EXCEPT**.
- Nested queries — other stored searches (GQL, structured queries, full-text searches) can be called as sub-procedures and their results processed further.

1.1.7.2. i-views-specific extensions

The GQL implementation in i-views accounts for the particularities of the knowledge graph data model:

- Multilingual attribute values are automatically returned in the matching application language. Language qualifiers can be used to explicitly request specific translations.
- Multi-valued attributes are split into separate result rows, analogously to multi-valued relations.

- All knowledge graph elements — including relations and attributes — can appear as nodes in a query pattern, making meta-properties naturally accessible.
- In addition to the standard GQL operators, the **IV_COMPARE** function provides extended comparisons, including wildcard searches, regular expressions, and geodetic distance comparisons.
- System relations are directly usable in queries via predefined labels (prefix **iv_**).

1.1.7.3. Executing and integrating GQL queries

GQL queries can be run in several ways:

- GQL Workbench — interactive interface in the Knowledge Builder under *Tools > GQL Workbench* for developing and testing queries
- Registered searches — GQL queries can be saved as folder elements or registered searches, just like other search types
- REST API — queries can be called via HTTP using a configurable built-in resource; results can be returned as JSON, CSV, or Excel
- JavaScript API — **\$k.GQLQuery** allows GQL queries to be executed programmatically and the result table to be evaluated row by row

1.1.8. Other improvements

- The new system relation "has target" allows traversal from a relation to its target.

1.1.8.1. JavaScript API

- Functions of the **JavaScript API** that were tagged as **experimental** were revised
 - Many functions are now officially supported
 - Some experimental functions without known usage were removed
 - Problematic functions were marked as **deprecated**
- **Type.topType()** : Returns the top type of the user-definable part of the knowledge graph

1.1.8.2. Improvements in the Knowledge Builder

- The detail view of a graph element shows a unified banner layout.



- On the left-hand side the elements icon, its kind (which shows type and extensions for objects), the menu to interact with the element and its name are shown.
- On the right hand side, additional information such as the currently assigned component is shown.

- The contrast for some colors was also improved.
- Tooltips are displayed for as long as the cursor hovers the element.
- Ctrl + click on an arrow in a tree view expands the whole branch.
- Buttons with keyboard focus are highlighted.
- The highlighting of selected tabs is now much more distinct and easier to see.
- Added a list of static hotkeys to the settings page where custom hotkeys can be defined.
- The IDs of relations can now be copied from the graph editor.
- In the Graph Editor, you can swap the set of selected nodes with the set of unselected nodes.
- New default icon for supplemental concepts.
- The references-dialog now also shows references as domain and supertype.
- Decision-makers in the rights configuration are highlighted when added to make them easier to distinguish from one another.
- The entries in various context menus have been revised to provide users with a more consistent experience.
- When mapping data sources, warnings are displayed if elements of the data source have not been mapped. To avoid cluttering the mapping tree, a separate list of all warnings is provided.
- Relational targets without names are displayed as <Object of type xy> instead of <No name>.
- The text of error message dialogs can be copied to make it easier to forward error messages.
- New queries or scripts can be created on the fly from the JavaScript editor by inserting a reference to a query or script that does not yet exist.
- Table contents are automatically updated when you trigger a search. Previously, this required an additional click on the "Refresh" button.
- Conversion to all units of the same quantity is shown in the unit configuration view.
- "Show uses" for type operators
- Code editor settings (show line numbers, light/dark mode)

1.1.8.3. Improvements to Structured Query

- Numerous improvements to improve stability and performance
- Index warnings for structure queries are now indexing suggestions and are based on the in-depth analyses of the index report. Although calculating the recommendations takes longer, they are more accurate than the previous warnings. To avoid unnecessarily delaying the display of a structured query, the indexing recommendations may be configured to be calculated only upon explicit manual request.

1.1.9. Incompatibilities

During the upgrade, configurations are automatically adjusted as far as possible. In some cases this

is not possible:

1.1.9.1. Changing the source or target of a relation

Operations modifying a relations source or target are specially handled in versions before 6.1. The unchanged side of the relation was not handled by access rights or triggers.

From 6.1 onward the full relation is handled as deleted and recreated. This way triggers can be formulated consistently without special treatment.

The previous behavior can be manually recreated by checking the "relocate"-flag in a script-filter on the access on the object. By filtering such access, the previous behavior can be achieved.

```
function accessMatch(access) {
  return (access.relocate != true)
}
```

1.1.9.2. Timeouts

HTTP and SMTP connections no longer have built-in timeouts. If timeouts are required, they must be configured (for REST services, via the corresponding resource). This is particularly important to note when operating a web frontend using the View Config Mapper.

1.1.10. Removed components

1.1.11. Discontinued components

The following components of the software have been discontinued and may be removed in future releases.

- VCM-material: Frontends that are based on material and use either individual styling adjustments or views must be adapted when switching to Vue. Will be removed in version 6.2.
- Lucene full text index: The Lucene connection is to be replaced by the existing Elastic connection. This is easier to scale and no longer requires the technically complex connection with the Java VM.
 - ViewConfig rendering in JavaScript: the functions for creating pseudo-tables as JavaScript objects use an old internal interface that was used in the communication of the backend with the frontend. Therefore, the result often has to be post-processed in the JavaScript code.
 - KScript: This script variant can no longer be created, but is still supported when migrating from old versions. In principle, JavaScript is the designated successor. However, as there is currently no automatic translator from KScript to JavaScript, the switch to JavaScript must be made manually in projects.
 - KPath: developed as a sub-language in the KScript environment, it can now only be used in

search pipelines. In principle, it is possible to replace it with JavaScript, but this has not yet been implemented.

2. i-views 6.1.2

2.1. New features and extensions

2.1.1. Model Context Protocol component

The [Model Context Protocol](#) (MCP) is an open standard that allows AI agents and large language model (LLM) tools to interact with external systems through a well-defined interface. The new Model Context Protocol component acts as an MCP server, exposing knowledge graph data and operations as tools that AI agents can discover and invoke.

Tools are not predefined — instead, you configure graph-specific tools in the Knowledge Builder under **TECHNICAL > Model Context Protocol**. Each tool wraps a particular piece of knowledge graph functionality — a query, an export, an import, a script, or a view configuration — and makes it available to any MCP-compatible AI client. Tools are exposed via a configurable MCP service and can be integrated directly by common AI agent frameworks and development tools.

2.1.2. Variant model component

The new variant model component provides schema and functionality for modelling objects as variants of other objects. A variant of an object "inherits" all properties of that object that it does not override itself. Inherited properties are made available through dedicated system relations and script functions.

The supplemental type *Tabulation* additionally enables the creation of tabular overviews of variant characteristics. A tabulation lists all possible concrete variants of a variant tree as rows of a table, with each column representing a configured property.

2.1.3. iiRDS component

The new iiRDS component provides the framework for importing iiRDS packages into the knowledge graph.

2.1.4. Custom Components

- The write protection of components is now stronger and more expansive within the Knowledge Builder.
- Import and Export now also show their current speed and the already elapsed time.
- New relation for `__` objects to specify a type which has all of its objects assigned to the component without needing to be assigned to the same component itself.
- View configuration trees in the Knowledge Builder can now be colored to represent the

assignment of their elements.

- The tree view of view configuration elements in the KB now also has a submenu for custom components, which has additional options to assign whole areas of the configuration at once.

2.1.5. Other improvements

- The new system relation *Identity* connects every knowledge graph element with itself. It is useful wherever a tool expects a configured relation for structuring results — for example, a JSON export can use this relation to build nested JSON structures based on a single graph element.
- The GQL query language now includes the extension function `IV_NAME`, which allows the name of a node to be accessed directly in a query expression.
- The JavaScript API now provides `$k.GQLQueryResult.prototype.toArray()` as a convenience method for processing GQL results as an array.
- Coloring of view configuration trees by component assignment in the Knowledge Builder is now a configurable setting and is disabled by default.
- The document creation component now supports connecting to a dedicated TeX Live conversion service as a new converter option for LaTeX documents.
- Configurable REST services now support OpenID Connect as an authentication method.
- Supplemental types can now be chained: the schema an object gains from a supplemental type may itself include supplemental relations to further supplemental types.

2.2. Incompatibilities

2.2.1. JavaScript API: iterators as function arguments

Most JavaScript API functions that previously expected an array argument now also accept iterators and iterable objects. Conversely, passing an object that is neither an array nor an iterator — which was previously silently coerced to an empty array — now throws an error (`'... is not iterable'`). Scripts that pass such arguments must be updated.