



FirstSpirit™

Unlock Your Content

FirstSpirit™ Manual for Developers (Basics) FirstSpirit™ Version 5.0

Version	1.16
Status	RELEASED
Date	2013-05-15
Department	FS-Core
Copyright	2013 e-Spirit AG

File name DEVB50EN_FirstSpirit_DeveloperDocumentationBasics

e-Spirit AG

Barcelonaweg 14
44269 Dortmund | Germany

T +49 231 . 477 77-0
F +49 231 . 477 77-499

info@e-spirit.com
www.e-spirit.com

e-Spirit

Table of contents

1	Introduction.....	9
1.1	Topics of this documentation.....	9
1.2	Classification in the complete documentation	11
1.3	General terms.....	12
1.3.1	Templates.....	12
1.3.2	New input components	13
1.3.3	Content Store	14
1.3.4	Workflows	15
1.3.5	Integrated preview	17
1.3.6	Content Highlighting & EasyEdit.....	19
1.3.7	Centralized error correction and system reporting.....	20
2	FirstSpirit JavaClient template store	21
2.1	General.....	21
2.2	General template store context menus	22
2.2.1	New.....	23
2.2.2	Editing on/off.....	26
2.2.3	Reverting changes	27
2.2.4	Cut	27
2.2.5	Copy	28
2.2.6	Paste	29
2.2.7	Rename.....	30



2.2.8	Delete.....	31
2.3	Special template store context menus.....	34
2.3.1	Update	34
2.3.2	Export.....	35
2.3.3	Import.....	38
2.3.4	Restoring deleted objects	43
2.3.5	Edit externally.....	45
2.4	Template store administrative context menus.....	47
2.4.1	Version history.....	47
2.4.2	Starting a workflow.....	48
2.4.3	Running a script.....	48
2.4.4	Search in templates.....	48
2.4.5	Tools – Change permissions	48
2.4.6	Tools – Delete write protection.....	49
2.4.7	Tools – Select/remove preview graphic	49
2.4.8	Tools – Display properties	50
2.4.9	Tools – Display uses	52
2.4.10	Tools – Apply template changes	52
2.4.11	Tools – Cancel editing	53
2.4.12	Tools – Change reference name	53
2.4.13	Tools – Show dependencies.....	53
2.4.14	Tools – Create copy of this workflow	54
2.5	Page templates.....	55
2.5.1	Preview tab.....	56
2.5.2	Properties tab	57
2.5.3	Form tab.....	59



- 2.5.4 Template sets tab 60
- 2.5.5 Rules tab 61
- 2.5.6 Snippet tab 62
- 2.6 Section templates..... 63
 - 2.6.1 Preview, Properties, Form, Template sets, Rules and Snippet tabs 64
- 2.7 Format templates 65
 - 2.7.1 Properties tab 66
 - 2.7.2 Template sets tab..... 68
- 2.8 Style templates 69
 - 2.8.1 Introduction: Inline tables 69
 - 2.8.2 Creating a style template 70
 - 2.8.3 Form area of a style template..... 71
 - 2.8.4 Preassigning layout attributes 74
 - 2.8.5 Presentation channel of a style template 75
 - 2.8.6 Linking with standard table format templates 75
 - 2.8.7 Examples 77
- 2.9 Table format templates 80
 - 2.9.1 Creating and editing display rules 82
 - 2.9.2 Evaluation order..... 86
 - 2.9.3 Inserting an inline table in the DOM editor 87
- 2.10 Link templates 88
 - 2.10.1 Standard link types 89
 - 2.10.2 Generic link editors 89
- 2.11 Scripts..... 91
 - 2.11.1 Properties tab 92
 - 2.11.2 Form tab..... 94



2.11.3	Template sets tab.....	95
2.12	Database schemata	96
2.12.1	New: Create schema.....	97
2.12.2	New: Creating a schema from a database	101
2.12.3	The FirstSpirit schema editor.....	104
2.12.4	Table templates	111
2.12.5	Queries	114
2.13	Workflows	118
3	Content sources in FirstSpirit	119
3.1	Terms.....	120
3.2	Standard layer.....	121
3.3	DBA layer.....	122
3.4	Content sources in FirstSpirit JavaClient.....	124
4	Workflows	126
4.1	Overview.....	127
4.1.1	Task search (filtered overview).....	129
4.1.2	Editing tasks.....	131
4.1.3	Closing tasks	132
4.2	Modeling workflows	133
4.2.1	Creating a workflow	133
4.2.2	Workflow editor tool bar.....	134
4.2.3	Elements of the graphical workflow editor	135
4.2.4	Keyboard shortcuts in the workflow editor.....	137
4.2.5	Operating assistance for the editor.....	138



4.2.6	Rules of modeling	138
4.2.7	Examples for modeling rules	139
4.2.8	Print preview for workflow models.....	140
4.3	Error handling in workflows	142
4.3.1	General error handling.....	142
4.3.2	Error state	142
4.3.3	Example: "Error" workflow	145
4.4	Form support for workflows (form).....	147
4.4.1	Example: "GUI" workflow	149
4.5	Properties of a workflow (configuration).....	150
4.5.1	General properties	150
4.5.2	Display logic for workflows	151
4.5.3	Properties of a state	152
4.5.4	Properties of an activity	156
4.5.5	Properties of a transition	161
4.6	Permission configuration for workflows.....	165
4.6.1	General permission configuration using the template store.....	165
4.6.2	Changing or locking editor preselection	166
4.6.3	Context-dependent permissions for starting a workflow	169
4.6.4	Context-dependent permissions for switching a workflow.....	172
4.6.5	Effects on the permissions configuration	173
4.7	Write protection within workflows	178
4.7.1	General.....	178
4.7.2	Write protection when creating and moving.....	178
4.7.3	Write protection within scripts.....	179
4.8	Use of scripts in workflows.....	180



4.8.1	Automatic activities and scripts.....	180
4.8.2	Manual activities and scripts.....	180
4.8.3	Workflow context.....	181
4.8.4	Example: Output of messages in workflows.....	183
4.8.5	Example: Persistent content within workflows.....	185
4.9	Deleting via a workflow.....	187
4.9.1	Deleting via a workflow in the JavaClient.....	187
4.9.2	Deleting via a workflow in the WebClient.....	189
4.9.3	Permissions configuration.....	190
4.9.4	Example: "Delete" workflow.....	193
4.9.5	Example: "ContentDeleteDemo" workflow.....	196
4.10	Workflows with a complex function.....	199
4.10.1	Example: "RecursiveLock" workflow.....	199
4.10.2	Example: "RecursiveRelease" workflow.....	203
4.11	Multiple workflow selection.....	207
4.11.1	Multiple workflow selection.....	207
4.11.2	Requirements for starting and advancement.....	208
4.11.3	Multiple selection via the task list.....	209
4.11.4	Multiple selection via the "Workflows" overview.....	210
5	Tracking changes via revision-metadata.....	211
5.1	Get revision.....	212
5.2	Determining changes to a revision.....	213
5.2.1	Determining the type of change.....	213
5.2.2	Determining changed elements.....	214
5.3	Changes since the last deployment.....	215



5.4	Changes between two revisions.....	219
6	Server-side release	223
6.1	Default release.....	224
6.2	Specific release	224
6.2.1	Recursive release	227
6.2.2	Dependent release.....	228
6.2.3	Dependent release with recursive release	229
6.2.4	Ensuring accessibility (parent chain).....	231
6.2.5	Ensure accessibility (parent chain) and recursive release	233
6.2.6	Ensure accessibility (parent chain) and dependent release	234
6.2.7	Ensure accessibility (parent chain), recursive and dependent release	236
6.2.8	Order for the release	238
7	Code completion for forms.....	240
7.1.1	Inserting the input component tags.....	240
7.1.2	Inputting tags, parameters and key terms	241



1 Introduction

The objective of this Manual is to describe the implementation of FirstSpirit™ projects from the perspectives of developers. In this context, the structure of the documentation has been chosen to provide an overview of FirstSpirit™ mechanisms and respective applications that is as comprehensive as possible (see Chapter 1.1, page 9).

Some areas, particularly those required for template development, are already documented in detail in the FirstSpirit™ online documentation. For introducing the FirstSpirit™ concept from the perspective of template development, Chapter 1.3 provides an explanation of general terms (starting on page 12).



This document is provided for information purposes only. e-Spirit may change the contents hereof without notice. This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. e-Spirit specifically disclaims any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. The technologies, functionality, services, and processes described herein are subject to change without notice.

1.1 Topics of this documentation

This documentation describes the relevant functions and aspects for template development in FirstSpirit. The structure is based largely on the user interface of FirstSpirit JavaClient.

The FirstSpirit JavaClient Template Store and all the available context menus and editing options are described in **Chapter 2** (see Chapter 2 starting on page 21).

FirstSpirit has efficient mechanisms for connecting databases. Chapter **3** handles the types of layers available in FirstSpirit for connecting databases and lists some general recommendations for handling data sources in FirstSpirit (see Chapter 3, page 119).

Workflows are a sequence of tasks that are processed according to a fixed, defined structure. For example, they can be used to model release processes. **Chapter 4** explains the workflow editor used in FirstSpirit, including all configuration options (see Chapter 4 starting on page 126).



FirstSpirit provides an option for tracking changes via the FirstSpirit Access API. **Chapter 5** describes access to revision metadata using specific API functions. Revision metadata contains information on the type (which changes took place?) and the scope (which elements were changed?) of a change to the project (see Chapter 5, page 211).

In addition to release via a workflow, all objects in FirstSpirit can be released server-side via the Access API. **Chapter 6** shows the methods for defining different release settings for an object (see Chapter 6, page 223).

To support template developers, code completion is being introduced on the Form tab in FirstSpirit Version 5.0. **Chapter 7** explains how, via this code completion, all FirstSpirit input components, as well as the parameters belonging to them, can be shown and inserted with the push of a button.



1.2 Classification in the complete documentation

Some areas, particularly the areas required for template development, are already documented in detail in the FirstSpirit™ online documentation. The classification of developer documentation in the complete documentation is illustrated in Figure 1-1.

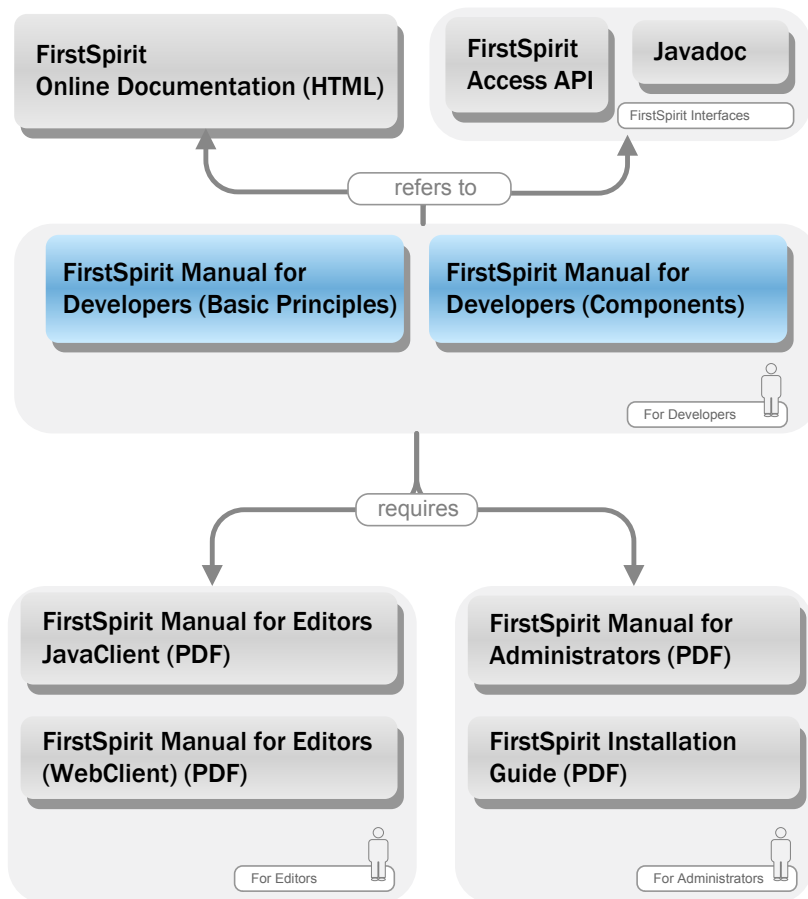


Figure 1-1: Classification of the developer documentation in the complete documentation

At least basic knowledge of the Manual for Editors and the Manual for Administrators is expected for understanding the following chapters. A detailed description of individual template components and the interfaces is given in the FirstSpirit Online Documentation.

Due to its scope, the documentation for developers is divided into this Manual, which explains the basic aspects of template development, and the Developer Manual for Components, which describes special aspects involving the development of modules and components for FirstSpirit.



The Manual for Developers (basics) examines the aspects listed in Chapter 1.1 (page 9).

The Developer Manual for Components examines the following aspects, among others:

- Installing and configuring modules
- The structure of modules and components
- The FirstSpirit GUI Object Model GOM
- The FirstSpirit security architecture
- Example listings

Knowledge of the following areas is also helpful for understanding the chapters that describe expanding and adapting FirstSpirit:

- Programming in Java / BeanShell
- Relational database technology

1.3 General terms

1.3.1 Templates

Templates form the basis for every website. In them, the complete layout of the web page is taken into account (among other things, corporate design and corporate identity). Templates are needed to connect the contents entered in the page store and the media integrated into the media store to the structure stored in the site store for a complete presentation when generating the web page.

The basics of template development are communicated in a detailed, step-by-step-guide in the **FirstSpirit Online Documentation**. The creation of the first templates is explained there based on a simple example. The output language is HTML (see FirstSpirit Online Documentation / Basics chapter / Step by step).

In FirstSpirit, different types of templates are available to the developer:

- **Page templates** create the basic framework of a page. The placement of logos and navigation tools as well as similar, general settings are set in page templates. Moreover, the page templates define the locations where an editor can insert content.
- **Section templates** are used to insert content into this basic framework. Section templates are subdivided into individually specified input windows via which the editor can maintain the



editorial content of the section (in the Page Store).

- The formatting is defined in **Format templates**, which can be used in connection with the DOM editor input element in the Page Store.
- The appearance of links is specified in detail by **Link templates** within a FirstSpirit project. The template developers define all the input fields in which the editors can enter all required content and the appearance of the link on the HTML page.

All template types are updated and administered in the FirstSpirit Template Store.

With FirstSpiritVersion 5.0, line numbers are indicated at many places in the Template Store as a reading aid. They can be shown or hidden in JavaClient using the keyboard shortcut Ctrl + L.

1.3.2 New input components

In the scope of the fundamental revision that began with FirstSpirit Version 4.2 and the consolidation of the input component model (compare with "FirstSpirit Roadmap 2009-2012"), a series of input components that were previously separate were introduced together.

This consolidation affects the following input component groups:

- Single-value input components: link to other FirstSpirit objects, for example, CM_INPUT_FILE, CM_INPUT_PICTURE, CM_INPUT_PAGEREF, etc.
- Set-valued input components: CM_INPUT_CONTENTLIST, CM_INPUT_TABLIST, CM_INPUT_CONTENTAREALIST, CM_INPUT_ILINKLIST

The new generation of input components,officially released with FirstSpirit Version 5.0, is identified with the prefix "FS_" instead of the previous "CMS_INPUT_".



For additional information on the new input components, see the [FirstSpirit Online Documentation](#)¹.

¹ ../Vorlagenentwicklung/Formulare/Eingabekomponenten (new)



1.3.3 Content Store

The Content Store creates and administers heavily structured data inventories that are to be used and updated in FirstSpirit. Such inventories are product catalogs and address lists, for example. These data inventories are subjected to frequent changes. Usually, such data is collected in databases. The Content Store data is saved in a relational database system supported by FirstSpirit.

The separation of layout, content and structure applies for data collection in the Content Store. In order to guarantee this, in the schema area (in the Template Store), the structure of the data and the layout for the data acquisition screen is specified. With this layout, the content is administered in database tables in the Content Store. In the Site Store, these databases can then be inserted into the Website structure.

In a first step, a database schema is created in the Template Store via a graphic editor. This schema can be created either based on an existing database structure and, if needed, adapted in the schema editor, or generated as an empty schema in order to structure it with the aid of the schema editor. In this schema, the tables and relations of a data model are to be mapped. In the table templates, input elements are then defined for the table columns and queries formulated for the data inventory.

In the Content Store, the data inventories are updated by the responsible editor. For this purpose, database tables are created based on the settings in the Template Store and filled with content via the input elements configured.

To illustrate the structured content on a Website, the database table is inserted as a content source on a page of the Page Store. This page then becomes referenced in the Site Store. Settings for the display of data records can be made on this page reference. For example, if only a certain section of the database table is to be displayed, the queries defined in the Template Store can be called up here.

All Content Store menus are described in the "Documentation for Editors (JavaClient)".



For the concept on working with "Schemes, table templates, views of a database", see Chapter 3, page 119.



1.3.4 Workflows

A workflow is a sequence of tasks that are completed in a fixed, predefined structure. The tasks also serve to transfer an object, for example, a page from the page store, from a start state (for example, "Page changed") to an end state (for example, "Changed page checked and released"). Due date deadlines and groups of authorized persons can be defined for the tasks to be run between these states.

The workflows can be shown with a graphic editor in the Template Store. The task of the Workflow Editor is to describe the workflow as abstractly and completely as possible. The graphically created model can also be used subsequently as a basis for user support when carrying out the work process.

The structure (sequence of tasks) and the properties (for example, without context) of a workflow and the definition of authorized persons and groups who may move from one task to the task that follows are defined within the Template Store (see Chapter4, page 126).

An example of a FirstSpirit workflow is the frequently used release process. It is the task of the release process to ensure that a newly created entry made by an editor or a change to the content is subjected to a test before going live. The release process can vary depending on which workflows already are or are to be established in a company.

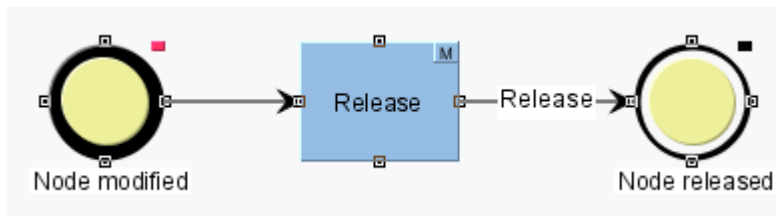


Figure 1-2: Example of a "simple" release

In this example, the editor-in-chief is responsible for inspecting the entries. Deployment is only possible once the editor has checked the changes (see Figure 1-2).



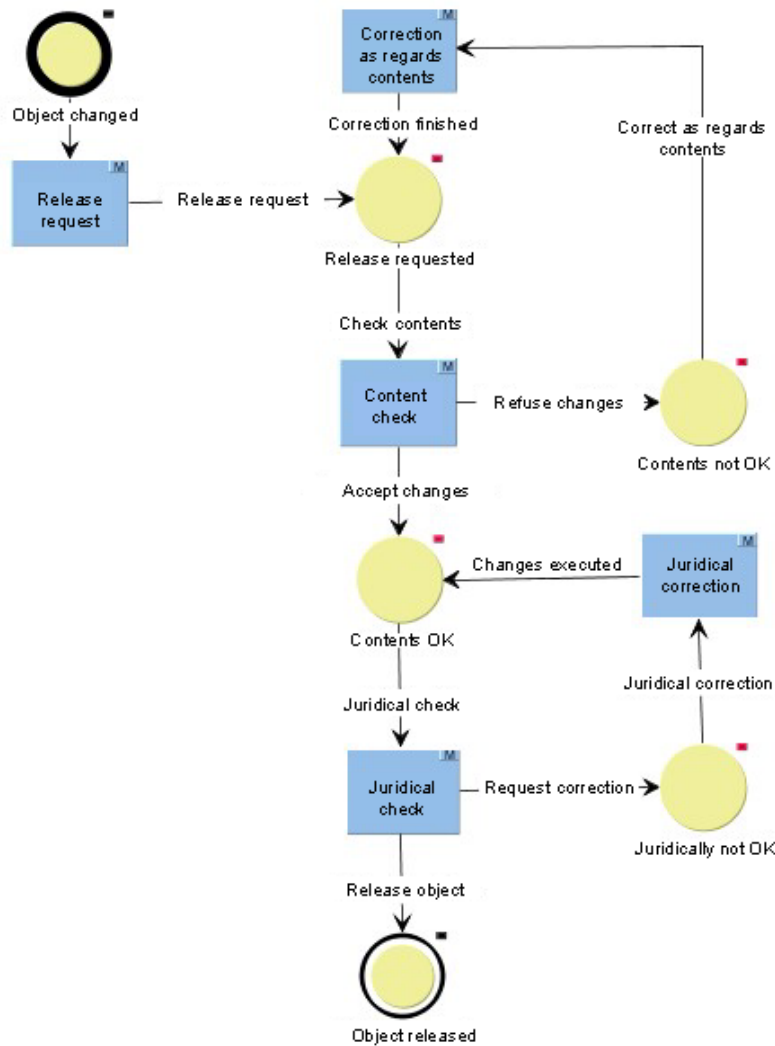


Figure 1-3: Release with "technical and legal" inspection

In this example, the testing of articles specified for deployment is organized into "technical" and "legal" partial steps (see Figure 1-3). These partial steps are normally carried out by different people. In this case, the workflow ensures that the legal inspection is only done after the content inspection, so that necessary content corrections also go through this inspection step. If a correction should be necessary due to legal aspects, the model then assumes that a new content inspection is unnecessary (this could, however, be necessary in application areas with different circumstances).

Important aspects in the use of workflows include:

- Coordination of workflows into "logical" subareas:
 Example: In the media store, only the workflows "B" and "C" are possible in area "A", while the Site Store uses the workflow "E".



- Assigning users/groups and workflows:
Example: Workflow "B" may be carried out only by the user group "G".
- Defining rights in workflows:
Example: The transition to the state "legally inspected" can be carried out only by members of the "Lawyers" group.
- Defining data fields that can (or must) be filled out by the user when going through the workflows:
Example: The person doing the testing inserts a "legal test note" into the corresponding form field.

For further information on generating workflows, see Chapter 4, page 126 ff.

1.3.5 Integrated preview

JavaClient provides a direct WYSIWYG preview with the "Integrated preview" function ("View" / "Show integrated preview" menu). For template development, it can be used to check changes directly in the presentation channel of templates (for example, HTML or XLS-FO) in the preview window because every time the template is saved, the (configurable) preview page is updated.

Moreover an integrated form preview is available within the template store. If a template is selected in the form area, a live view of the template being edited appears in the preview area with the defined fallback values of the input components (see Figure 1-4).

The integrated preview can optionally be shown to the right next to the workspace or in an external window.

All input fields can be edited directly within the integrated template preview. The template must not be blocked for editing.



The editing option just forms an aid for the template developer. With it, you can check directly whether a defined, remote configuration provides the desired results for an input component. However, the content entered is not saved in the Form preview. Default values cannot be defined in the Form preview.



The **default values for the input components** can be defined with the "Default values" button on the "Properties" tab of a template (see Chapter 2.5.2, page 57). The language-dependent default values are shown in the preview area immediately after saving the properties.

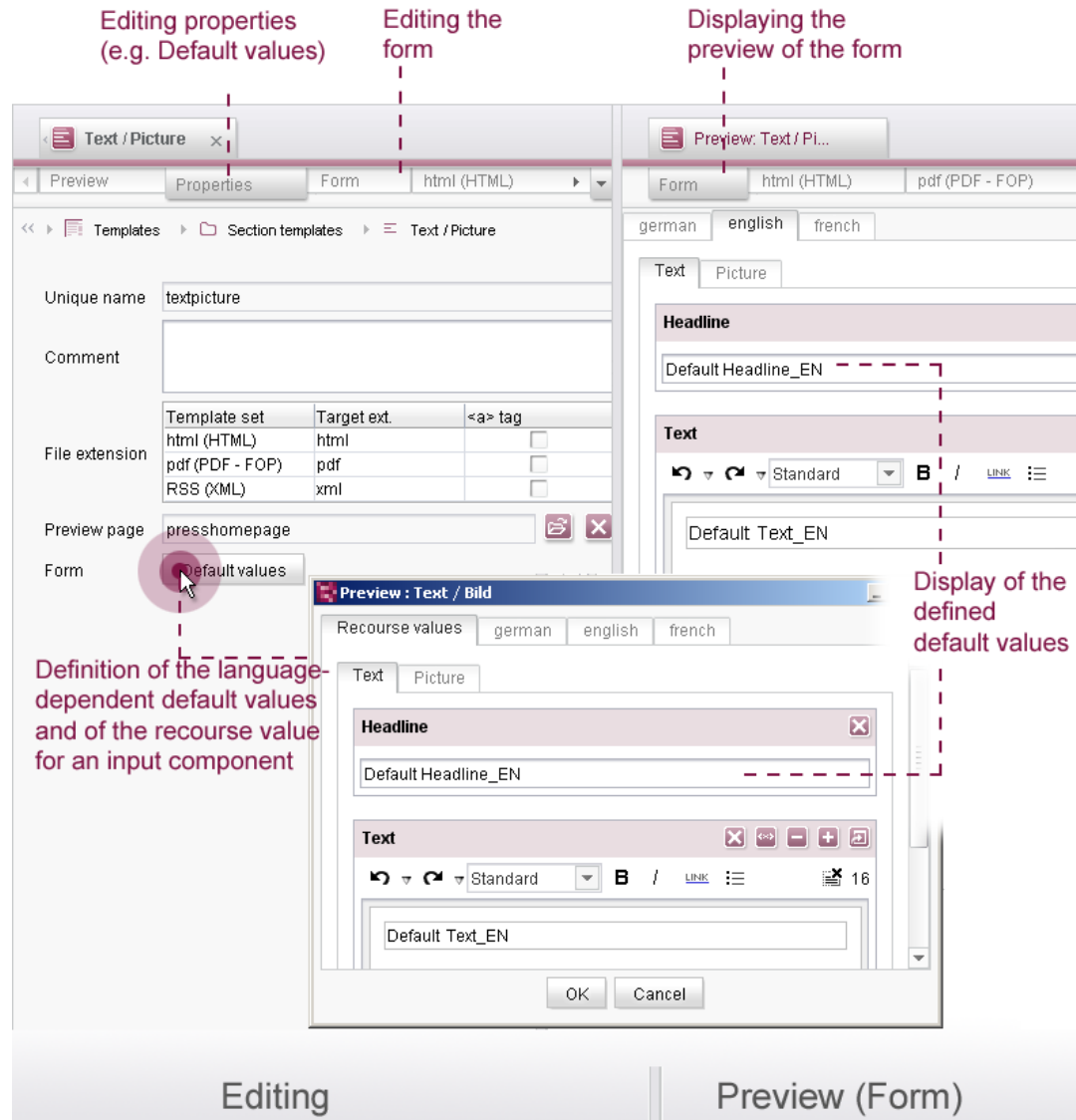


Figure 1-4: Preview of the form area in the template store



1.3.6 Content Highlighting & EasyEdit

In order to make processing editorial content as easy and transparent as possible for the editor, FirstSpirit offers a Content Highlighting function. This functionality links editorial maintenance in FirstSpirit JavaClient to the output in the preview by automatically retrieving content and highlighting it for editing. Content Highlighting functions bidirectionally.

In order to use this function in a project, the templates must first be adapted. The HTML elements that are intended to be highlighted must be able to be referenced uniquely on an HTML page. For this, identifiers known as editor identifiers are used; they can be stored by the template developer within the HTML templates.

Because Content Highlighting was simplified with FirstSpirit Version 5.0, adapting existing projects that still have the old functionality is absolutely necessary. Error-free use of Content Highlighting cannot be ensured without these changes.

The template adaptations that are necessary for using the Content Highlighting function can have an effect on WebClient 5 and are used there for the editorial processing of content. As in JavaClient, the contents in WebClient are also highlighted in color and bordered surrounded by a box. Control elements that allow the editor to edit the bordered contents directly (EasyEdit functionality) are shown in this box.

A detailed description of the format templates and the style sheet is available in the FirstSpirit online documentation².



The technologies used for Content Highlighting functionality are integrated into the HTML source code of a FirstSpirit project more than the functions up to this point. The ability to use Content Highlighting without making changes to a project's HTML cannot be guaranteed. In particular, pixel-specific layouts in conjunction with CHTML can lead to problems since some additional pixels are required in the HTML environment due to the border around the highlighted content.

² FirstSpirit online documentation - Chapter: ../Advanced topics/Content highlighting



1.3.7 Centralized error correction and system reporting

Infrastructure for collecting errors and exceptions is provided. A loader icon is displayed in the bottom left area of JavaClient for this; it continually shows data transmission during editing work. A small exclamation mark is displayed in the icon when an exception occurs. Additional information on the error that occurred can then be requested by clicking on the icon.

For more information on this function, refer to the FirstSpirit Manual for Editors (JavaClient), subchapter "Centralized error collection and system reporting" in Chapter 3 "Menus and icons in FirstSpirit JavaClient".



2 FirstSpirit JavaClient template store

2.1 General



Templates form the basis for every website. The complete website layout is taken into account in these templates (including corporate design and corporate identity). Templates are needed to connect the contents entered in the page store and the media entered in the media store to the structure stored in the site store for a complete presentation when generating the web page.

Different types of templates can be defined and edited in the template store.

- Page templates (see Chapter 2.5, page 55)
- Section templates (see Chapter 2.6, page 63)
- Format templates (see Chapter 2.7, page 65)
- Style templates (see Chapter 2.8, page 69)
- Table format templates (see Chapter 2.9, page 80)
- Link templates (see Chapter 2.10, page 88)

Furthermore, additional editing options are available for:

- Scripts (see Chapter 2.11, page 91)
- Database schemata (see Chapter 2.12, page 96)
- Workflows (see Chapter 2.13, page 118)

Moving using drag-and-drop: Folders and templates can be moved in template management using the mouse for drag-and-drop (indicated by a small square on the mouse pointer).

Copying using drag-and-drop: Furthermore, it is possible to copy folders and templates in the template store by using the mouse for drag-and-drop while holding the Ctrl key (indicated by a small plus on the mouse pointer).





The user has to have the required permissions to use drag-and-drop (moving, copying) for nodes in the template store. Otherwise an error message appears, "This action cannot be performed (insufficient permissions)!"

With FirstSpiritVersion 5.0, line numbers are indicated at many places in the Template Store as a reading aid. They can be shown or hidden in JavaClient using the keyboard shortcut **Ctrl + L**.

2.2 General template store context menus

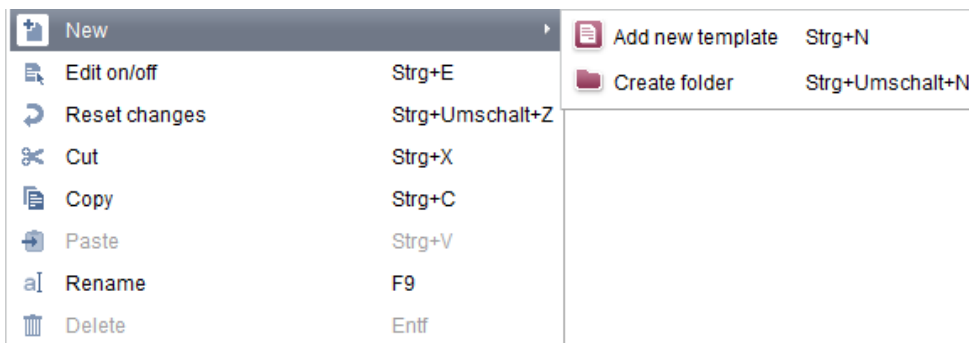


Figure 2-1: General template store context menus

The template store context menus are described in the following chapters:

Context menu structuring (general, specific, administrative):

Context menus are all structured according to the same schema:

- General functions are located in the top area (see this chapter)
- Specific functions for the selected node are located in the middle area (see Chapter 2.3, page 34).
- Functions that are normally only needed by project administrators are located in the bottom area. Normally, these functions cannot be performed by normal users and are disabled for this reason (see Item 3) (see Chapter 2.4, page 47).

Calling up a context menu: To call up a context menu, an object such as a folder or template is highlighted in the tree view on the left half of the screen and then the context menu for that node is opened by right-clicking. Clicking the left mouse button selects the desired menu item.

Disabled menu items are shown in gray. If this is the case, the function is not available to the user. The potential reasons for this are:

- The object is currently being edited by another template developer



- The state of the current object
- The user lacks the permissions to perform a specific action.

2.2.1 New

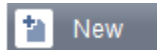


Figure 2-2: New function

New objects can be added to the project using the "New" context menu entry or the "New" icon in the tool bar. The available selection depends on the object type for which the context menu or function was called.

Overall, these objects can be created in the template store:

- Page and section templates
- Folders
- Format templates
- Table format templates
- Style templates
- Link templates
- Scripts
- Schemata
- Schemata from databases
- Table templates
- Queries
- Workflows

The function for creating a new object opens a dialog that has the following structure (see Figure 2-3):

- Display name
- Reference name
- Additional information (if it is necessary)



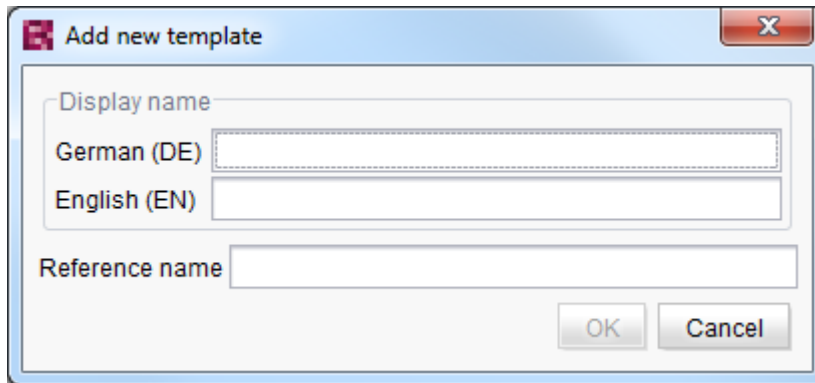


Figure 2-3: Creating a template

Display name: The language-dependent display name can be defined individually for every project language. If language-dependent display of the tree view in JavaClient is activated (View / Preferred display language menu), the display names entered here are displayed depending on the selected project language. In contrast to unique reference names, language-dependent designations can be changed at any time.

Reference name: A unique name for the new template is specified in the "Reference name" field. Upon creation, the field for inputting the reference name is filled the same as the entry for the display name. With that in mind, the display name is applied in the field (the first display name entered if there are multiple languages). Blank spaces and special characters are replaced by underscores in the process. The option for displaying the reference name allows the reference name to be shown in Java Client's tree view (View / Preferred display language/ Show reference names in tree menu). The reference name can be modified by the user during creation. After being created, the reference name can be modified using the "Tools" context menu. Subsequently modifying the reference name is, however, not recommended because otherwise all of the references within the project are lost.

A unique reference to the template can be established using the reference name. In input components (in the form area) for example, the unique reference name is used to establish references to templates (see FirstSpirit online documentation, such as for the CMS_INPUT_DOM input component).

If a reference name that has already been specified in another name space is entered when creating a new object, the name is replaced by a unique name automatically, usually by appending numbers to the name. (Invalid special characters are also replaced automatically.)

E.g.: A *template* page template already exists. A newly created section template called *template* would then be saved under the reference name *template_1* automatically.



The reference name for a template can be determined on the "Properties" tab (see Chapter 2.5.2, page 57).



A reference name for a template should no longer be changed after creation since otherwise all of the references within the project are lost!

OK

Clicking this button attaches the new template to the directory tree and it can be edited further.

Cancel

Clicking this button cancels the operation. A new template is not created.

Special considerations

Folders

Unlike template reference names (see Chapter 2.2.1), folder names do not have to be uniquely specified.

Table templates

A table template has to be created under the schema for each input table in the database model. This table template defines which input components the editor can use later on to add data to the corresponding tables and where this data is to be saved in the database.

In addition to the display and reference names, selecting a table from the database schema (see Figure 2-4) for which the template is to be created is necessary to create a new table template (also see: Chapter 2.12.1, page 97)



The combobox is empty if the schema does not contain any tables. In that case, a table template cannot be created.



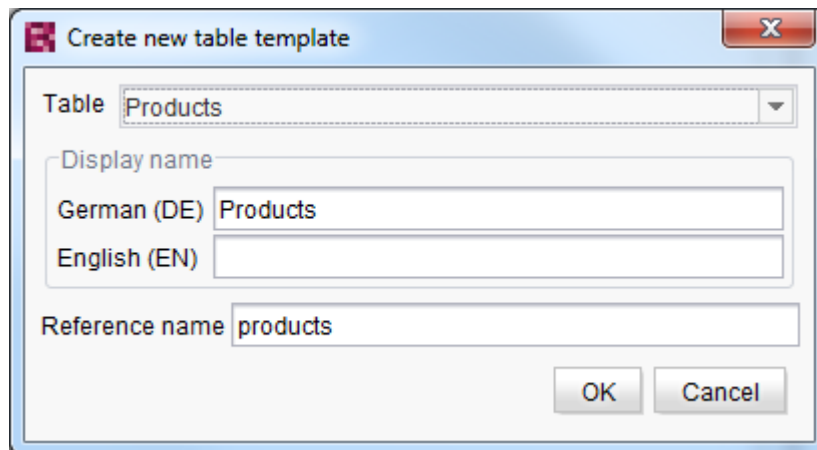


Figure 2-4: Creating a table template

2.2.2 Editing on/off



Figure 2-5: Editing on/off function

You first have to enable editing mode in order to be able to make changes to an object. This prevents another editor from simultaneously editing the same content and prevents conflicts that could result from changing an element at the same time.

New objects can be blocked from being edited using the "Editing on/off" context menu entry on the tool bar.



Editing mode has to be disabled again after the desired changes have been made in order to release the corresponding object for editing for other users. All changes made are saved automatically upon exiting editing mode.



2.2.3 Reverting changes



Figure 2-6: Reverting changes function

This function can be used to undo changes made during the current editing operation that also have *not been saved yet*. A confirmation prompt appears before changes are undone so that content is not deleted accidentally.

The function is only active if editing mode is enabled on the object (see Chapter 2.2.2).

This function can be used for the following template store elements:

- Page and section templates
- Format (exception: system format templates), table and style templates
- Link templates
- Scripts
- Database schemata
- Table templates
- Queries
- Workflows

2.2.4 Cut



Figure 2-7: Cut function

This function can be used to cut an object from the current tree position and store it in the clipboard.



The "Cut" function is only carried out once the cut object is reinserted. If a cut object is not reinserted, it retains its original tree position; i.e. it is not deleted.

These objects can then be inserted at another location using the Paste function (see Chapter 2.2.6).



This function can be used for the following template store elements:

- All template store folders
- Page and section templates
- Format (exception: system format templates and folders that contain system format templates), table format and style templates
- Scripts
- Database schemata
- Table templates
- Queries
- Workflows

2.2.5 Copy



Figure 2-8: Copy function

A copy of the current object is generated and stored in the clipboard using this function. These objects can then be inserted at another location using the Paste function (see Chapter 2.2.6).

This function can be used for the following template store elements:

- All template store folders
- Page and section templates
- Format, table format and style templates
- Scripts
- Database schemata
- Table templates
- Queries
- Workflows

Copying using drag-and-drop: Furthermore, it is possible to copy . folders and . templates in the template store by using the mouse for drag-and-drop while holding the Ctrl key (indicated by a small plus on the mouse pointer).





The user has to have the required permissions to use drag-and-drop (moving, copying) for nodes in the template store. Otherwise an error message appears, "This action cannot be performed (insufficient permissions)!"

2.2.6 Paste



Paste

Strg+V

Figure 2-9: Paste function

The contents of the clipboard are inserted at the current position in the tree structure using this function. Thus, this function is only active if there is data in the clipboard that is allowed to be inserted at the current position.

The clipboard is an area within JavaClient where a wide assortment of objects can be stored (pages, page references, images, data records, sections, individual input components, text, files). It is used as a "collection point" for the editor where materials and content can be centrally and conveniently entered for later work steps.

More detailed information about the clipboard can be found in the release notes for FirstSpirit 5.0.



Objects can only be inserted in the areas intended for the respective objects. It is not possible, for instance, to paste a section template underneath the node for page templates. In this case, the "Paste" entry is disabled.

This function can be used for the following template store elements:

- All template store root nodes
- All template store folders
- Database schemata (only for table templates)



2.2.7 Rename

 Rename F9

Figure 2-10: Rename function

It is possible to modify the language-dependent display name of the current object in the FirstSpirit JavaClient tree structure using this function. Calling this function opens a window with the current display name; the name can then be modified.

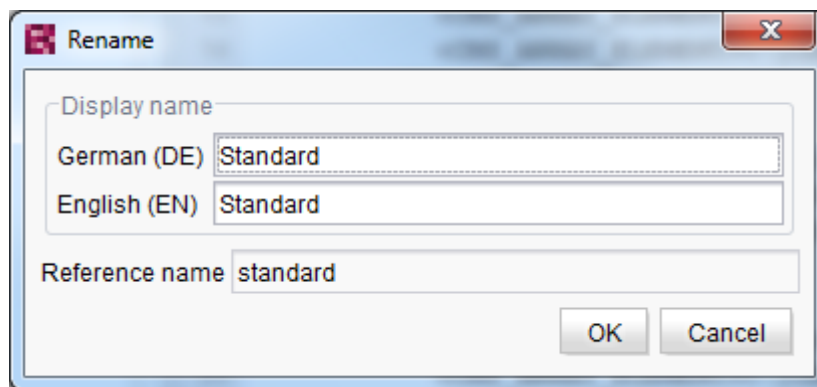


Figure 2-11: Rename



Unique reference names cannot be changed since otherwise the references to this object would be lost (e.g. reference names from section or page templates). In this case, the "Reference name" field is disabled and cannot be edited. Folders do not have unique reference names. Thus, they can be renamed at any time.

This function can be used for the following template store elements:

- All template store folders
- Page and section templates
- Format, table format and style templates
- Link templates
- Scripts
- Database schemata
- Table templates
- Queries
- Workflows



2.2.8 Delete



Figure 2-12: Delete function

It is possible to delete the currently highlighted object or the currently highlighted tree section using this function. A confirmation prompt prevents content from being deleted accidentally.

Project administrators have the option of restoring deleted elements (see Chapter 2.3.4, page 43).

The "Delete objects" function is available for the following elements:

- Page templates
- Section templates (see Chapter 2.2.8.1)
- Format, table format and style templates (system format templates cannot be deleted)
- Link templates
- Scripts
- Workflows
- Database schemata, queries and table templates


The "Delete tree sections" function is available for the following elements:

- All template store folders

For more detailed information on deleting objects and tree sections, see FirstSpirit Manual for Editors, Chapter 3.2.8 "Delete".



2.2.8.1 Deleting objects in use

 *This function is only available to project and server administrators.*

Templates still being used by other objects in the client can also be deleted. In this case, the user is shown a confirmation prompt before individual objects are deleted.

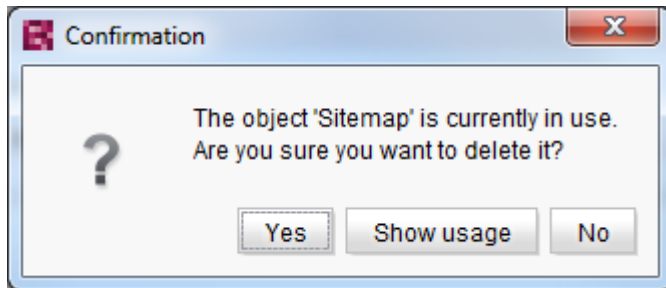
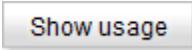


Figure 2-13: Deleting an object in use

 Clicking this button opens the "This object is still being referenced" dialog, which shows the objects still currently using the object slated for deletion (see Figure 2-14).

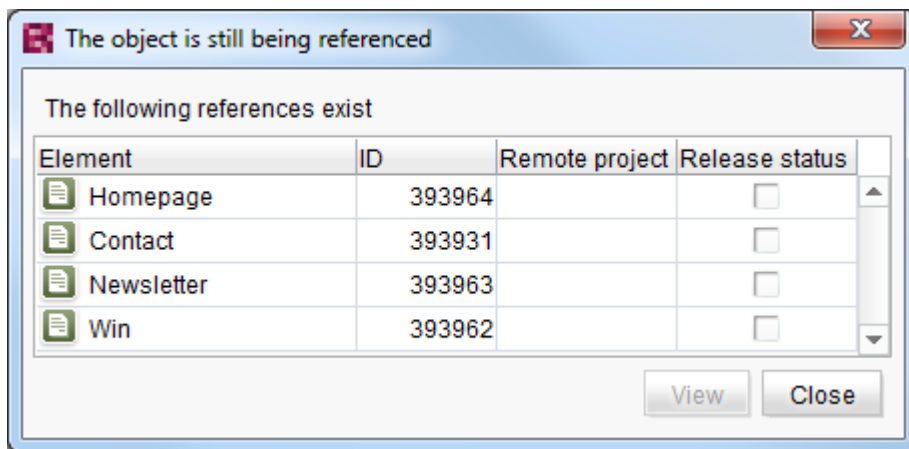



Figure 2-14: This object is still being referenced

 Clicking this button deletes the object slated for deletion despite the uses present in the project.

The reference continues to be shown in the reference graph after deletion. However, the deleted



object is now unknown. The reference is only removed after once again editing the existing object.

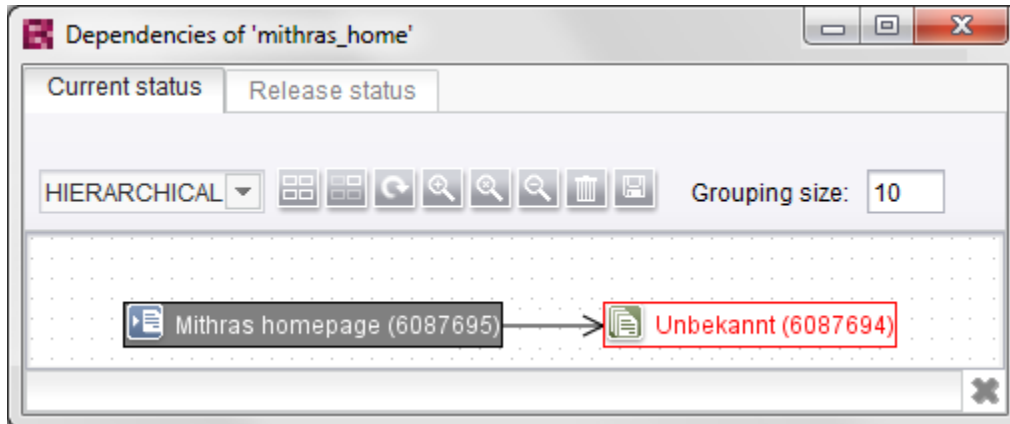


Figure 2-15: Reference graph after deleting an object still in use

Likewise, the user is shown a confirmation prompt before deleting multiple objects where at least one of the objects is still in use. However, the prompt differs from the one for deleting a single object (see Figure 2-16).

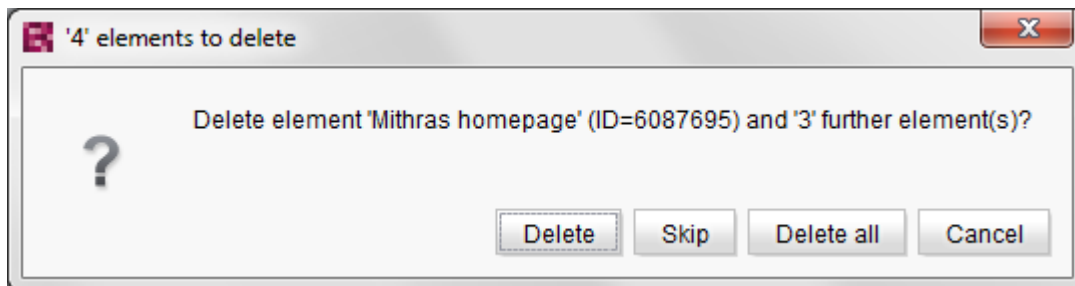


Figure 2-16: Deleting multiple objects in use

In this case, the usage of the corresponding objects cannot be visualized. A distinction in the deletion process is made using the "Delete" and "Delete all" buttons.

Clicking "Delete all" deletes the selection objects as a group after a preceding prompt asking whether the objects in use are to be deleted as well or skipped.

Clicking "Delete" would function the same as the deletion of multiple objects up to this point. In this case, the deletion for each selected object has to be confirmed individually.

If the object to be deleted is a section template being used as a section restriction in the content area of a page template, then this restriction is automatically removed from the list of allowed



section templates for the content area upon deletion of the section template. If, in this context, it is the only section restriction for the content area, all section templates are allowed for this content area after refreshing the page template (also refer to Chapter 2.5.2, page 57).

2.3 Special template store context menus

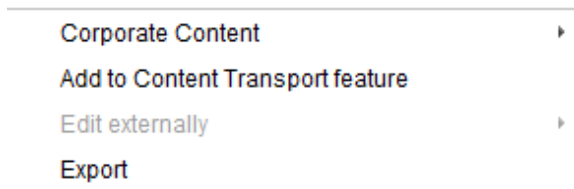


Figure 2-17: Special context menus – Template store (root)

Special functions for the respective object and functions that may be license-dependent are available in the middle area of the context menu. The functions are also heavily dependent on the selected object type or the scope of the license.

2.3.1 Update



Figure 2-18: Update

This function can be used for the following template store elements:

- Template store root nodes

This menu entry can be used to update the template store view. This is necessary if multiple persons are working and making changes to a project at the same time.



This function may not be used if an object is currently being edited and the changes have not yet been saved! Otherwise, the unsaved changes are overwritten by the version on the server and are thereby lost.



2.3.2 Export

Export

Figure 2-19: Export function

Objects from a project can be combined into a compressed zip file and saved to the local file system using the "Export" context menu entry. The export files can then be used to import the exported contents of a project (source project) into other FirstSpirit projects (target project) (see Chapter 2.3.3, page 38). The available selection depends on the object type for which the context menu or function was called.



The "Export" context menu available in FirstSpirit JavaClient is a client-side function and, thus, places heavy demands on the main memory of the client system if large data volumes are involved. Thus, this function should only be used to export small data volumes.

This function can be used for the following template store elements:

- All template store folders (see Chapter 2.3.2.1, page 36)
- Page and section templates (see Chapter 2.3.2.2, page 36)
- Format templates (no system format templates)(see Chapter 2.3.2.2, page 36)
- Style and table format templates (see Chapter 2.3.2.3, page 36)
- Link templates (see Chapter 2.3.2.3, page 36)
- Scripts (see Chapter 2.3.2.3, page 36)
- Database schemata (see Chapter 2.3.2.4, page 36)
- Table templates (see Chapter 2.3.2.5, page 37)
- Queries (see Chapter 2.3.2.5, page 37)
- Workflows (see Chapter 2.3.2.6, page 37)

Upon calling up the context menu, first an export window opens for selecting the desired save location for the export file in the workstation computer's local file system.



2.3.2.1 Exporting folders

Folders can be exported and imported for use in other FirstSpirit projects. The directory structure from the target project is retained when exporting.

2.3.2.2 Exporting templates

Templates can be exported and imported for use in other FirstSpirit projects. If templates are to be exported from a source project and imported into a target project frequently, the use of the license-dependent "Corporate Content" function is recommended since more information is available in the target project for assignment and for the update state.

2.3.2.3 Exporting style and table format templates, link templates and scripts

Exporting style and table format templates as well as link templates and scripts works the same way as exporting templates (see Chapter 2.3.2.2, page 36).



Style and table format templates are closely linked with each other (also see Chapter 2.8 and 2.9 starting from page 69) and should therefore be exported together if at all possible. To accomplish this, it is best to combine them in one folder, which can be exported as described in Chapter 2.3.2.1. However, style templates can also be exported and imported later individually without issue. For table format templates, the style templates used as the standard style template and in the display rules always have to be exported as well.

2.3.2.4 Exporting schemata

Schemata can be exported and imported for use in other FirstSpirit projects (see Chapter 2.3.3.1, page 39).

The following dialog is shown after displaying the export window used to select the desired save location for the export file from the local workstation's directories:



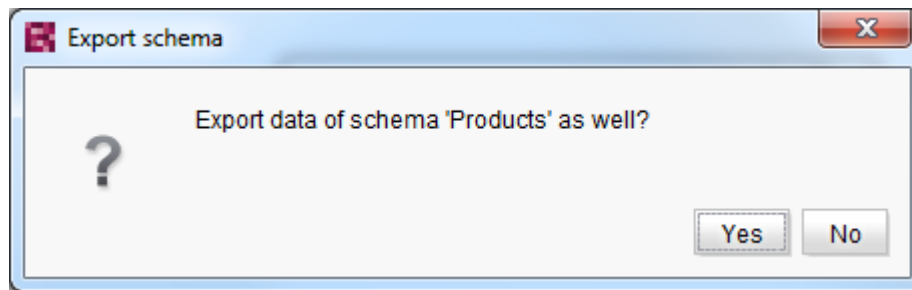


Figure 2-20: Exporting data when exporting a schema

Clicking this button adds the data from the current project's content store's schema to the export file. This data is then available to users of the second project when the export file is imported into another FirstSpirit project.

Clicking this button adds just the schema but not the schema's data from the current project's content store. When the export file is imported into another FirstSpirit project, the schema is available to the users in the second project, but the data from the first project's content store is not.

The table templates associated with the schema are added to the export file automatically.

2.3.2.5 Exporting table templates and queries

If a schema is exported, the associated table templates (and queries) are added to the export file automatically. Table templates (and queries) should always be exported together with the associated schema if at all possible. If this is not possible, they can also be exported individually.

In this case, the template (and/or query) has to be imported in the target project at the appropriate schema node. Otherwise there may be errors in the project since the mapping for the table template no longer matches the schema's tables ("The referenced table 'xy' does not exist").

2.3.2.6 Exporting workflows

Both individual workflows and folders with all included subfolders and workflows can be exported to the computer's file system using this function. This allows workflows to be used at a later point, i.e. in other projects.

To carry out the export, an export window opens where the desired save location for the export



file can be selected from the folders on the local workstation.

If scripts are used within a workflow, they can also be added to the export file. To accomplish this, the desired workflow first has to be selected in the tree view and then the script using Ctrl + click. Both objects are then included in the zip file if the "Export" function is selected using the context menu. However, scripts can also be exported separately at a later time (see Chapter 2.3.2.3, page 36).

2.3.3 Import

A screenshot of a context menu with the word "Import" in blue text.

Figure 2-21: Import function

Objects previously exported from a source project can be added to another FirstSpirit project (target project) using the "Import" context menu entry. To do so, the desired zip file must first be selected from the local workstation's file system and be imported to a suitable position in the target project.

If the imported contents do not fit the context of the target project, they are imported into the correct target project context automatically – to the extent this is possible. In this case, the import is carried out independently of the object where the "Import" context menu was selected. If, for example, a user tries to import a script's export file into the "Workflows" area, the selected script is imported into the target project regardless, but it is placed in the correct "Scripts" area of the target project instead of the "Workflows" area.

This automatic correction does not work in all cases. An error message is displayed instead if the system cannot determine which target project object the import file can be assigned to.

The available selection depends on the object type for which the context menu or function was called.

This function can be used for the following template store elements:

- All template store folders
- Style and table format templates (see Chapter 2.3.3.1, page 39)
- Link templates
- Database schemata (see Chapter 2.3.3.2, page 41)
- Workflows (see Chapter 2.3.3.3, page 43)



Page/section templates and folders, with all included templates and subfolders, exported to the file system can be imported into the selected folder using this function.

To carry out the import, an import window opens where the user can browse for the desired export file in the local workstation's folders.

Assigning template sets in a target project: When importing a template from a source project to a target project, an attempt is made to transfer the contents of the template sets as well.

- In the process, assignment is attempted using the name (*name in the source project to the name in the target project*). This means that if the names of the template sets are identical in the source and target project, the contents are applied in the target project.
- If assignment using the name is not successful because the template sets are labeled differently in the target project, assignment is attempted in the next step using the presentation channel (*name in the source project to the presentation channel in the target project*). In this context, a template set from a source project with the name "HTML" would be assigned to the first located "HTML" presentation channel in the target project (regardless of the name of the channel in the target project).
- If the template set cannot be assigned using the name or the presentation channel, the contents of the template sets cannot be imported into the target project from the source project and have to be created or copied manually, if necessary.

2.3.3.1 Importing style and table format templates

Style and table format templates can be imported from other FirstSpirit projects. To accomplish this, an export file from the desired templates first has to be exported from another FirstSpirit project (see Chapter 2.3.2.3, page 36).

Style templates can be imported without issue. Table format templates should be imported together with the style templates being used (see Chapter 2.8 starting from page 69). If these style templates are not also exported, the table format templates can be exported regardless, but the references to the style templates are lost.

To import style and table format templates, call up the context menu at the "Format templates" root node or a folder below that root node and select the "Import" function. The import dialog appears after selecting the desired export file.



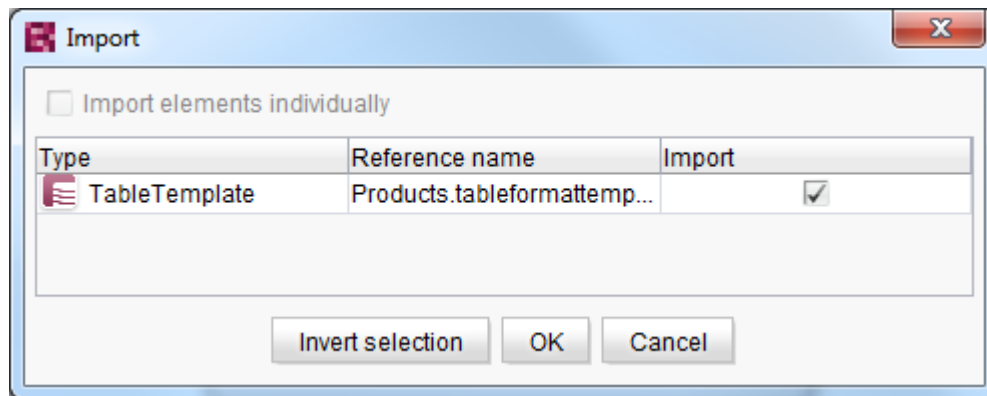


Figure 2-22: Importing a table format template

Import elements individually: This functionality is not available in the template store.

Type: Type of the element contained in the export file.

Reference name: Name of the element contained in the export file.

Import: If this checkbox is enabled, the associated element is imported into the target project; if this checkbox is disabled, the element is not imported.

Invert selection

Clicking this button inverts the selection made in the "Import" column.

OK

Clicking this button confirms the dialog selection and opens the "Select database layer" dialog (see Figure 2-24).

Cancel

Clicking this button cancels the import operation.



2.3.3.2 Importing schemata

Schemata can be imported from other FirstSpirit projects. To accomplish this, an export file from the desired templates first has to be exported from another FirstSpirit project (see Chapter 2.3.2.4, page 36).

The export file from the first FirstSpirit project can now be imported into the second project. To accomplish this, the context menu at the "Database schemata" root node or a folder below that root node is called up and the "Import" function is selected. The import dialog appears after selecting the desired export file.

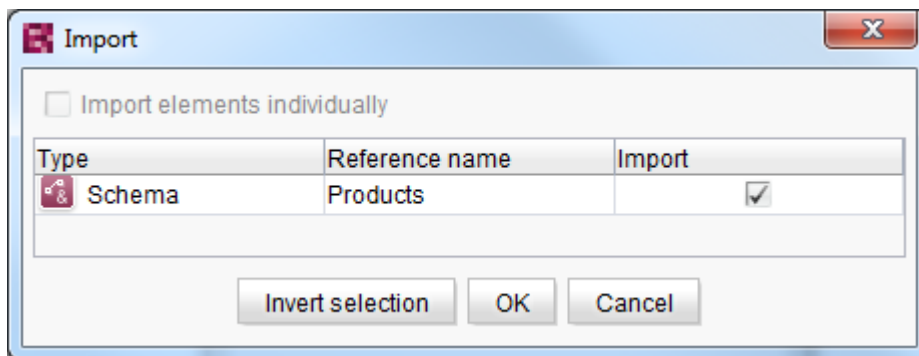


Figure 2-23: Importing a schema with table templates

Import elements individually: This functionality is not available in the template store.

Type: Type of the element contained in the export file.

Name: Name of the element included in the export file.

Import: If this checkbox is enabled, the associated element (and all elements below it, such as table templates) is imported into the target project; if this checkbox is disabled, the element is not imported.

Invert selection

Clicking this button inverts the selection made in the "Import" column.

OK

Clicking this button confirms the dialog selection and opens the "Select database layer" dialog (see Figure 2-24).

Cancel

Clicking this button cancels the import operation.



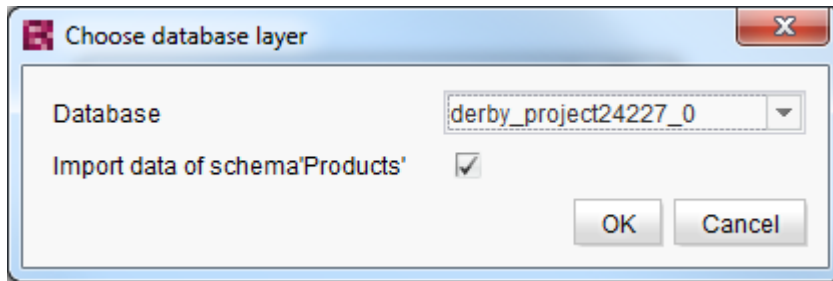


Figure 2-24: Importing a schema

Database: Selection of the desired database layer. All of the layers that have been enabled for the project by the project administrator are displayed in the drop-down list.

Import data from schema 'xy': If this checkbox is enabled, the existing data maintained for this schema in the source project's content store is applied to the target project. Upon creating a table in the target project's content store based on the imported schema information, the structured data maintained up until now is displayed in the target project.

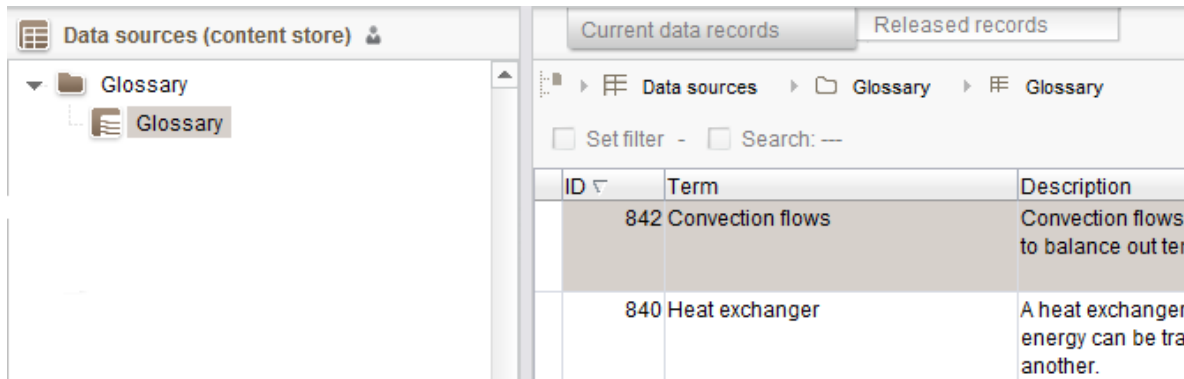


Figure 2-25: Table view of the structured data in the target project

The data can also be modified in the target project; i.e. it is not write-protected.

If only the schema - and not the existing data maintained for this schema in the source project's content store - is to be applied, either the schema has to be exported from the source project without data (see Figure 2-20) or the "Import data from schema 'xy'" checkbox has to be disabled in the target project. The existing data (based on the associated schema) is ignored during importing in both cases. This means it is protected from access from the target project.

If the existing data from the source project is, in fact, to be displayed in the target project but modification to the structured data is to be prevented, write protection for the selected database layer has to be enabled after importing the data.




2.3.3.3 Importing workflows

This function can be called up using the context menu in the Workflows area and at the folder level.

Workflows and folders, with all included subfolders and workflows (and included scripts if applicable), exported to the file system can be imported into the selected folder using this function.

To carry out the import, an import window opens where the user can browse for the desired export file in the local workstation's folders.



The permissions configuration has to be adjusted specifically to the project during importing if necessary (see Chapter 4.6, page 165).

2.3.4 Restoring deleted objects

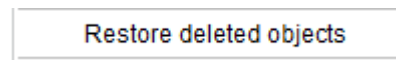


Figure 2-26: Restoring deleted objects

The Restore deleted objects function can be called for page, section, format and link templates as well as scripts at both the root and folder level, as well as at the schema level for database schemata. If an object is accidentally deleted from the tree structure, this function can be used to restore it. A window containing the deleted objects opens after clicking this function.

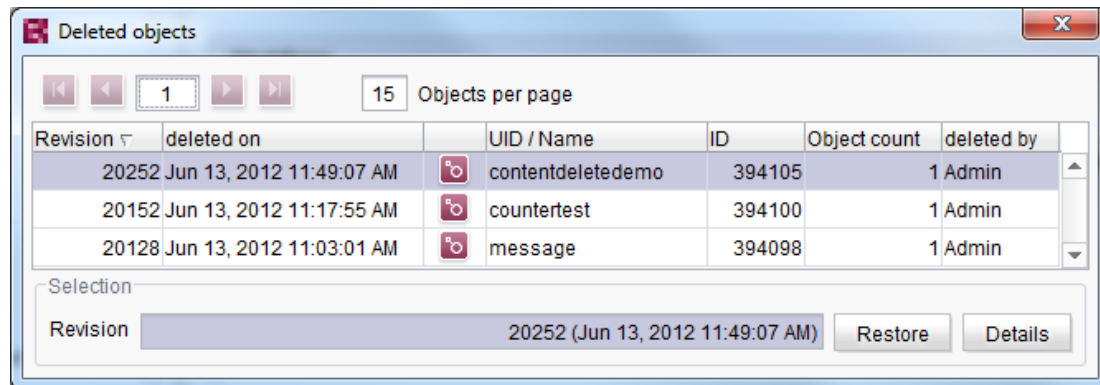


Figure 2-27: Deleted objects



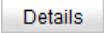
At the root level, all of the objects that have an available backup are displayed, whereas at the folder level only objects that have been located within this folder are displayed. The following information is provided for each object:

Revision: Version number of the deleted object.

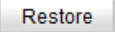
Deleted on: Date and time when the object was deleted.

UID / Name: The reference name of the deleted object.

ID: The unique ID number for the deleted object.

Object count: The number of objects located below the deleted object in the tree structure. These can be displayed in a pop-up window using the  button. These hierarchically lower-level objects are also reinserted if the object is restored.

Deleted by: Name of the user that deleted the object.

Select the specified object and activate the  button to restore the object.

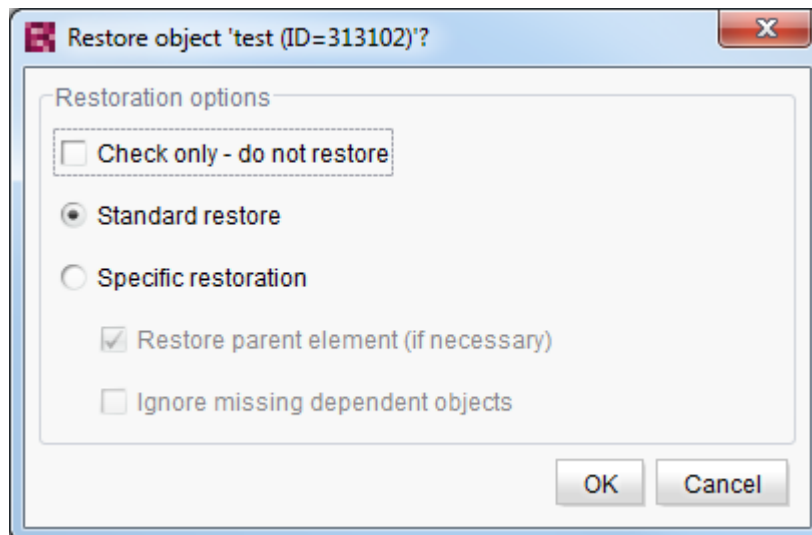


Figure 2-28: Restoring deleted objects

Only check – Do not perform a restore: If this option is selected, whether a restore can be carried out without errors is checked. This simulates the restore, but the deleted object is not restored. Whether or not a restore is possible is then displayed in a pop-up window.

Standard restore: This option is preset by default. If a restore is carried out with this option, the



restore is carried out directly dependent on the object. Therefore, depending on the object, different options can be selected in the "Specific restore" area.

Specific restore: This option can be selected to adapt the standard restore options manually.

Specific restore – Restore parent element (if necessary): If this option is selected, the parent element is also restored if necessary.

Specific restore – Ignore missing dependent objects: If this option is selected, missing references to the selected object are ignored during the restore.



This option is only available to project administrators.

The position where the deleted object is to be inserted can be selected in the next dialog. A subfolder is normally selected as the restore point in this dialog. If the root node of a store is to be selected, the subfolder highlighted in the middle column has to be deselected by holding CTRL and clicking on the subfolder.

2.3.5 Edit externally



Figure 2-29: Edit externally function

This function can be called using the context menu on page templates and section templates. It is divided into multiple areas: All of the **template sets** that have been configured for this project in the server and project configuration are listed; the **Form** and **Rules** areas are also present.

If one of the available editing areas is activated, then the corresponding source file opens in an external editor. An editor should be entered in the user settings of the Global Settings for editing a source file in an external editor. An additional window displaying all of the opened templates also appears.



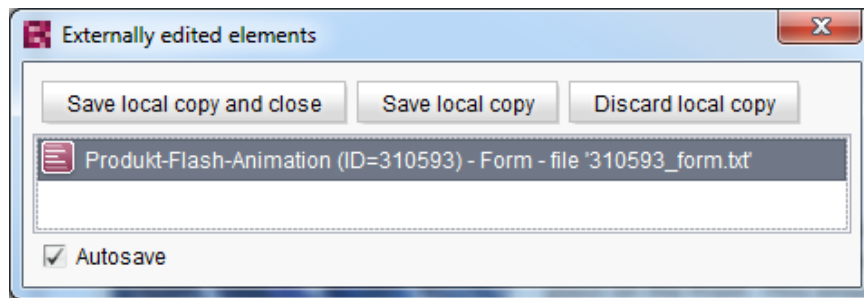


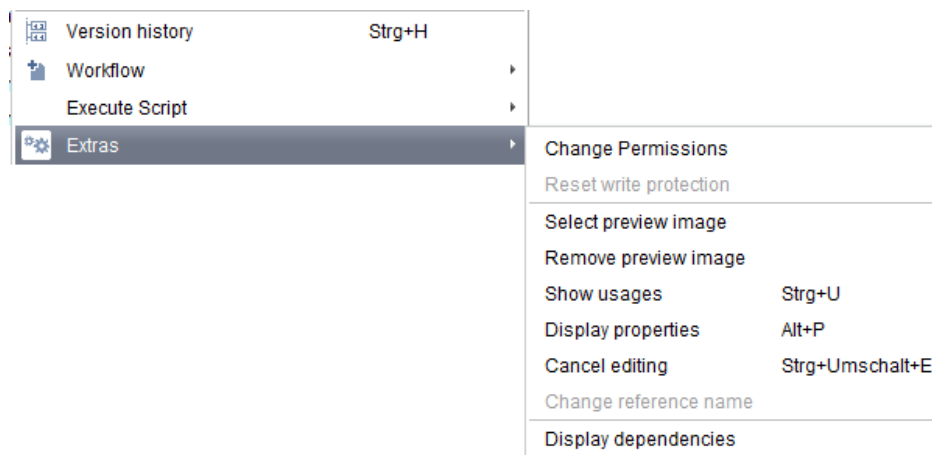
Figure 2-30: Edit externally

Modifications to the source text are saved, after the templates are highlighted, using the "Save and close local copy" or "Save local Copy" buttons. The editor is then ended in the first step. Likewise, unsaved modifications can be reverted using "Discard local copy".

Autosave: If this is checked, then all of the modifications saved in the external editor are also saved in FirstSpirit Client.



2.4 Template store administrative context menus



2.4.1 Version history

The version history for each template store object can be retrieved using this function.

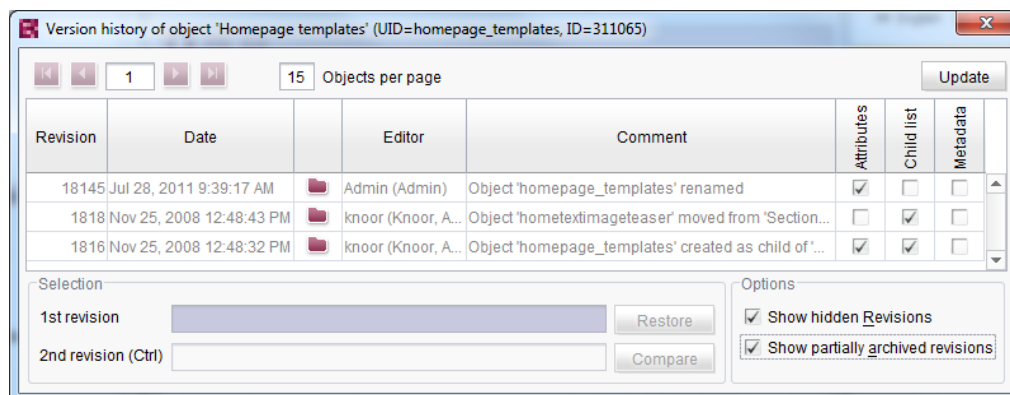


Figure 2-31: Version history for the section template folder

You can find general information on FirstSpirit's version history and on the functions of the dialog in Figure 2-31 in the *FirstSpirit Manual for Editors*, Chapter 11.10.

In addition to the generally available information for a revision (revision, date, editor, comment), the right side of the list display shows which element modification resulted in the allocation of a new revision number (e.g. attribute, child list, preview, presentation channels). This depends on which object the version history was retrieved for.



2.4.2 Starting a workflow

If a workflow is not yet active on the selected object, then all of the workflows that have been defined for this node in the tree structure in the permissions system are listed under this menu item. The required workflow can be started under this menu item.

If a workflow is already active for the selected object, then it can be advanced under this menu item.

Detailed documentation of workflows is located in Chapter 4 starting on page 126 and in *FirstSpirit Manual for Editors*, Chapter 12.

2.4.3 Running a script

All of the scripts that can be called at this position in JavaClient are listed under this menu item. Scripts make it possible to have pre-programmed actions or calculations run. Information on script development in FirstSpirit is located in the *FirstSpirit Online Documentation*.

2.4.4 Search in templates

This function is identical to the "Search in templates" function in the "Search" menu on the FirstSpirit menu bar. Additional information on this search is located in the *FirstSpirit Manual for Editors*, Chapter 3.

2.4.5 Tools – Change permissions

The permissions for the current node in the tree structure are defined using this function. It can be called at every node using the context menu.

The entries in the lists in the areas "Inherited permissions" and "Permissions defined in this object" are automatically displayed and are sorted alphabetically first by groups and then by users.

You can find more detailed documentation on defining permissions in the *FirstSpirit Manual for Editors*, Chapter 13.



2.4.6 Tools – Delete write protection

If write protection is present for the selected node due to an active workflow, the write protection can be removed using this function. (Write protection is indicated in the tree by italics.) You can find detailed information on write protection in workflows in Chapter 4.7 starting on page 178.

2.4.7 Tools – Select/remove preview graphic

This function can be called up using the context menu at the page/section level. The respective object has to be in edit mode for this.

A preview graphic for the Preview tab of the respective object can be selected or an existing one can be deleted using this function. For this purpose, a file window opens where the user can browse for the desired preview graphic in the local workstation's folders. The graphic file has to have the extension "gif", "jpg" or "png".



2.4.8 Tools – Display properties

The properties of an object can be displayed with the following information using this function. The information when using this function can vary depending on the object type.

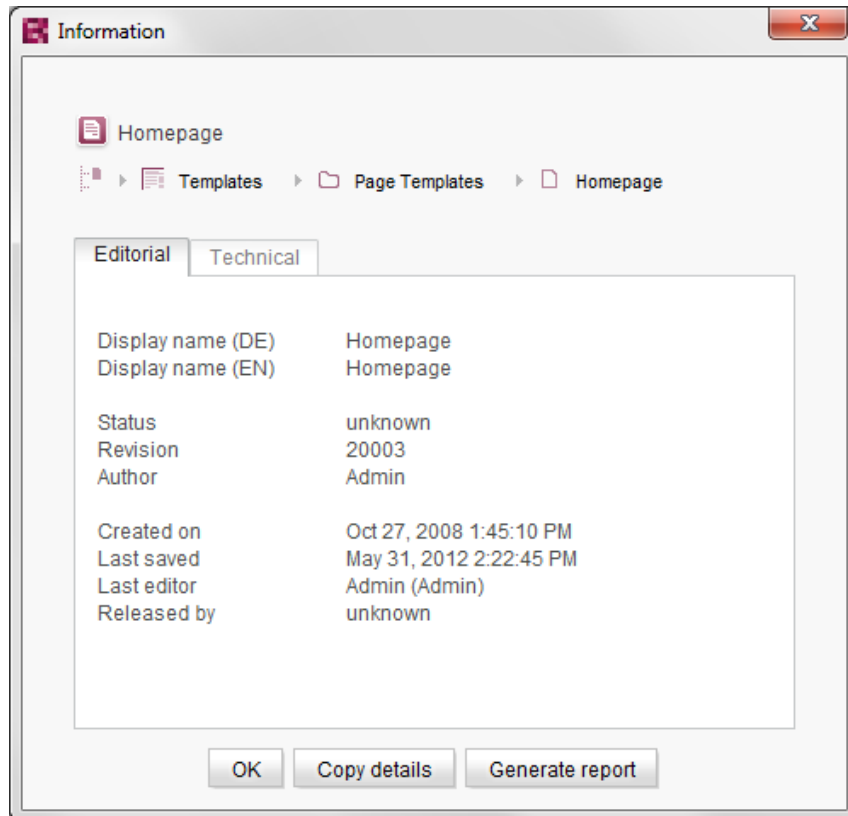


Figure 2-32: Properties of a page template – Editorial

The properties for other objects can be displayed using the path.

Editorial tab

Object properties relevant for editorial work are shown on this tab.

Display name: Object display name (language-dependent)

State: Shows the state (e.g. "Not released", "Released", "Modified (not released)")

Revision: Shows the revision



Author: Name of the user that created the object

Created on: Time when the object was created in JavaClient, with date and time

Last save: Time when the object was last saved, with date and time.

Last editor: Name of the user that edited the object last

Released by: Name of the user that released the object

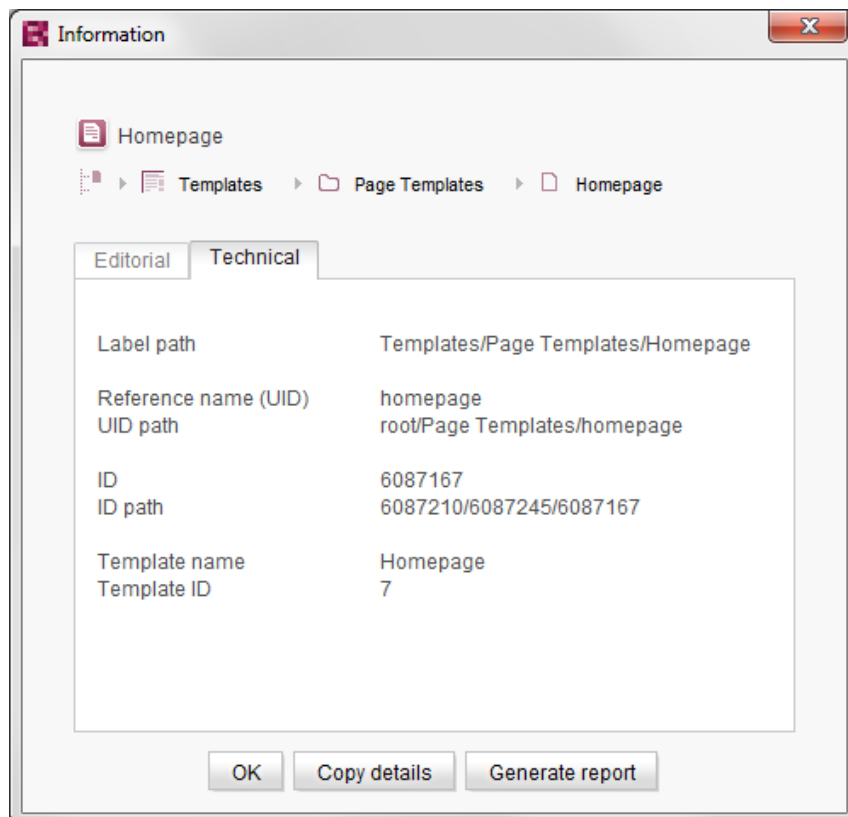


Figure 2-33: Properties of a page template – Technical

Technical tab

Object properties relevant for technical work are shown on this tab.

Label path: Path to the current object (display name)

Reference name (UID): Reference name (UID) for the object



UID path: Path to the current object (reference name)

ID: Object ID

ID path: Path for the current object (IDs)



The path information can also be requested using the keyboard shortcut Ctrl + Shift + Q.

Template name: Display name for the underlying template

Template ID: ID for the underlying template

The "OK" button closes the dialog. All of the information from the dialog can be transferred to the clipboard using the "Copy details" button. The information can be output as an HTML page using the "Generate report" button. An additional comment, such as an error description, can be entered as well.

2.4.9 Tools – Display uses

This functionality can be called using the context menu on page, section, format and table templates as well as scripts.

It can be used to automatically jump to nodes in other stores related to the object where this functionality has been run. If the object is used multiple times, a new window opens displaying all of the nodes (e.g. sections from the page store) based on the current object. Double-clicking one of these entries shows the corresponding node in the directory tree.

2.4.10 Tools – Apply template changes

Changes to the definition of the content areas can be applied to a page template for existing pages using this function.

The functionality is only available for page templates in the template store.

If, for instance, a content area is added to a page template, this change does not automatically affect an existing page. The "Apply template changes" function can be used to update existing pages if a change is made to a page template. The function checks the definition of the content



areas in the page template against the content areas of pages that use this template:

- If content areas are found that are defined in the template but not present on the existing page, these content areas are added to the page.
- If, conversely, content areas are found that are missing in the templates but are present on the page, then:
 - They are removed from the page if they do not contain any sections.
 - They remain on the page if they contain sections.

2.4.11 Tools – Cancel editing

You can use this function to exit edit mode on a node without saving changes that have been made. However, you cannot undo changes that have already been saved using CTRL + S or the save function on the icon bar.

2.4.12 Tools – Change reference name

Object reference names can be modified retroactively using this function.




A warning notice is displayed first since references for the object may still exist that would be invalid once the reference name has been changed. If "Change anyway" is selected, the reference name for the object can be changed in the next dialog.

2.4.13 Tools – Show dependencies

Essential FirstSpirit functions are based on what is known as a project's reference graph. The reference graph for a project is used for recognizing dependencies within the project and thus is an essential component for complex functions, such as server-side release (see Chapter 6, page 223).

The visualization of the reference graph for an object can be requested by project administrators using the "Tools – Show dependencies" context menu. This makes it possible to identify the dependencies of a project even in complex projects.



 Reference graphs for individual data records from the content store are retrieved using the respective data record's context menu.

The tabs indicate the dependencies of the object in the form of incoming and outgoing arrows, both for the current state and the last released state (see Figure 2-34).

The display can be reorganized to a hierarchical view, which is particularly recommended for complex dependencies (see Figure 2-34). It is possible to update directly in the event of changes and to zoom the view using the buttons in the upper area of the window. The view can also be saved as an image for later use.

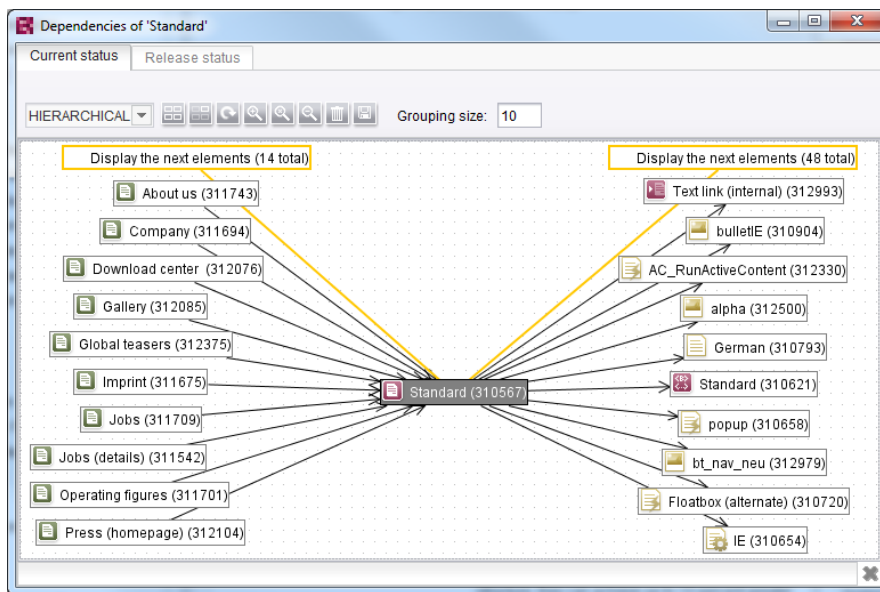


Figure 2-34: Displaying dependencies using the reference graph

2.4.14 Tools – Create copy of this workflow

This function can be called for workflows. It creates a copy of the selected workflow below the "Workflows" node.



2.5 Page templates

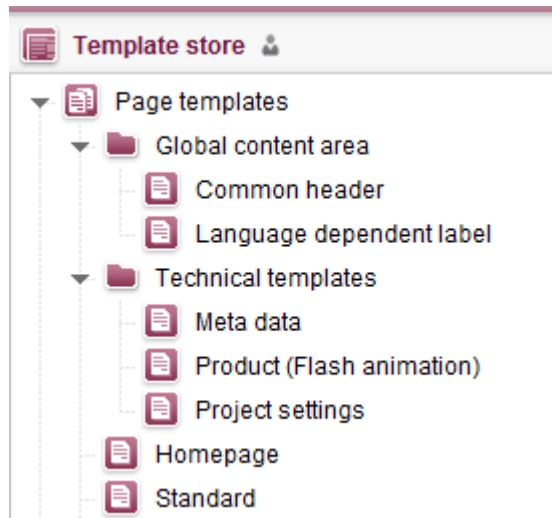





Figure 2-35: Template store tree view – Page templates

Page templates create the basic framework of a page. The placement of logos and navigation tools as well as similar, general settings are set in page templates. Moreover, the page templates define the locations where an editor can insert content.

Tree elements in page templates:

-  Root element of page templates
-  Folder in the page template node
-  Page template

With FirstSpiritVersion 5.0, line numbers are indicated at many places in the Template Store as a reading aid. They can be shown or hidden in JavaClient using the keyboard shortcut Ctrl + L.



2.5.1 Preview tab

To obtain a preview of how the template will look later in a browser, a previously prepared preview graphic (e.g. a screenshot) can be displayed on the "Preview" tab. This allows each user to tell immediately which template has just been highlighted.

A graphic can be added to this tab in three different ways:

1. Locking the template, then clicking "Select preview graphic" from the context menu and selecting a "gif", "jpg" or "png" file.
2. Locking the template, then selecting a "gif", "jpg" or "png" file from the file explorer, dragging it to the preview spot with the mouse and dropping it there.
3. Locking the template, selecting a link-free graphic (while holding the Ctrl key) from a website (only MS IE), dragging it to the Preview tab with the mouse and dropping it there.



Figure 2-36: Page preview – "Preview" tab



2.5.2 Properties tab

The "Properties" tab contains different entries for page and section templates. The following figure shows the properties for a **page template**:

The screenshot shows the 'Properties' tab for a page template named 'homepage'. The interface includes a breadcrumb trail: Templates > Page templates > Homepage. The 'Unique name' field contains 'homepage'. Below it is a 'Comment' field. A table lists 'File extension' and 'Template set' with 'Replaceable' checkboxes and 'Target ext.' values. The 'Preview page' field contains 'mithras_home'. A 'Form' section has a 'Default values' button. At the bottom, there is a checkbox for 'Hide in selection list'.

File extension	Template set	Replaceable	Target ext.
html (HTML)		<input type="checkbox"/>	html
pdf (PDF - FOP)		<input type="checkbox"/>	pdf
	RSS (XML)	<input type="checkbox"/>	xml

Figure 2-37: Page template – "Properties" tab

Unique name: A unique designation that the template is saved under in the file directory is specified in this field (see "Reference name" in Chapter 2.2.1, page 23).

Comment: A comment that describes the page or section template in more detail can be entered here.

File extension – Template set: Name and type of the template sets defined by the project administrator for the current project in the server and project configuration. Deactivated template sets are grayed out and cannot be edited.

File extension – Overwritable: If this is checked, it means that the extensions of a page template specified in one of the following two input fields can be overwritten by a section template.

File extension – Target ext.: The template extension that is to be linked. The extension can be edited by double-clicking in the field.

Preview page: A page from the site store where the template is used can be selected here. This allows changes made to the template to be checked directly in the template store using the preview function.

Hide template in selection list: Activating this option prevents an editor from using this



template when creating a new page.

Form: The "Default values" button that can be used to define default values for the template is available at this spot. The maintenance dialog for defining the defaults opens. The language-dependent default values are shown in the preview area immediately after saving the properties.

Content areas / Section restrictions: You can click on the icons in the top right to add a new content area to the page template, delete an existing content area or resort the list of content areas.

Just like all other templates, content areas can be created language-dependent and thus can be provided with one (or more) language-dependent reference names and one unique reference name.

Content areas		
<input type="checkbox"/> All section templates allowed		
Unique name	Allowed section templates	Content area is active
Content left	Text / Image (marginal teaser), Tag-Cloud, Contact, Press releases teaser	<input checked="" type="checkbox"/>
Content center	Text / Image (homepage teaser), Product flash animation	<input checked="" type="checkbox"/>
Content right	Text / Image (marginal teaser), Tag-Cloud, Contact, Press releases teaser	<input checked="" type="checkbox"/>

Figure 2-38: Defining content areas for a page template

Section restrictions can be defined for the page template by clicking on a content area.

Details

Unique name: Content center

Activate content area for this page template

German English

Display name: mittlerer Bereich

Description:

Allowed section templates

- Text / Image (homepage teaser)
- Product flash animation

Figure 2-39: Defining section restrictions

This is accomplished by allowing or prohibiting the desired section template by adding them to or removing them from a list (for a content area). For the corresponding content areas, this means that only the respective selected section templates are permitted. The templates are added by selecting the content area and clicking the open folder symbol. The templates are deleted by



highlighting the restriction below the page template and then removing it using the delete key on the keyboard or clicking on the trash can icon.

Optionally, all of the section templates for all of the content areas of a page template can also be allowed. This repeals every section restriction for the page template. This is possible using the checkbox "Allow all section templates".



If all of the section restrictions within a page template are deleted, there are no more restrictions for creating sections on the page. The layout of the page may be damaged in process, such as by creating sections unsuited for the page layout.

2.5.3 Form tab

```

1 <CMS_MODULE>
2
3 <CMS_GROUP tabs="top">
4
5 <CMS_GROUP>
6 <LANGINFOS>
7 <LANGINFO lang="" label="Homepage information" description="Fill the homepage components."/>
8 <LANGINFO lang="DE" label="Homepage-Informationen" description="Geben Sie ihre Homepage-Informationen ein."/>
9 </LANGINFOS>
10
11 <CMS_INPUT_TEXT name="pt_headline" hFill="yes" noBreak="yes" singleLine="no" useLanguages="yes">
12 <LANGINFOS>
13 <LANGINFO lang="" label="Headline" description="Insert your headline for that page."/>
14 <LANGINFO lang="DE" label="Überschrift" description="Überschrift der Seite."/>
15 </LANGINFOS>
16 </CMS_INPUT_TEXT>

```

Figure 2-40: Page template – "Form" tab

The "Form" tab shows the GUI.XML file. If a template is locked, changes can be made here directly.

DTD validation is carried out when saving GUI.XML. Incorrect formatting would be disastrous here; GUI.XML cannot be saved in that case. Other errors are just displayed.



If a preview for GUI.XML is requested in the locked state then the changes are saved automatically beforehand. (A new version is not created – that only occurs when unlocking!)

For updating content areas on existing pages (when changing the definition within the page

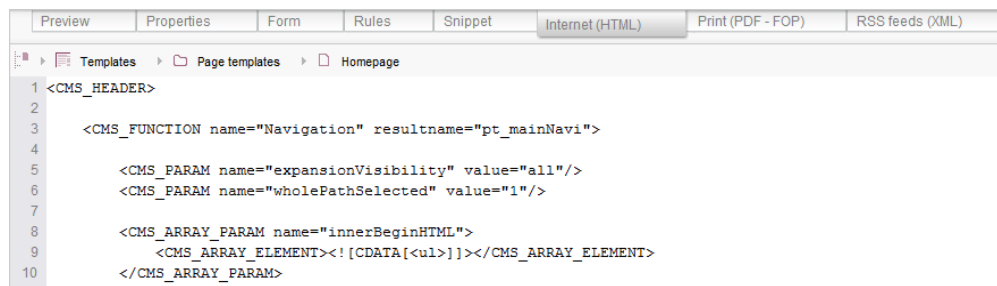


template), see Chapter 2.4.10, page 52.

A listing of all of the available input elements can be found in the "FirstSpirit Online Documentation". The input elements are explained with all of their attributes and a schematic example under the menu item **Template development – Forms**.

The addition of input components on the "Form" tab has been simplified by code completion (see Chapter 7, page 240).

2.5.4 Template sets tab



```
1 <CMS_HEADER>
2
3   <CMS_FUNCTION name="Navigation" resultname="pt_mainNavi">
4
5     <CMS_PARAM name="expansionVisibility" value="all"/>
6     <CMS_PARAM name="wholePathSelected" value="1"/>
7
8     <CMS_ARRAY_PARAM name="innerBeginHTML">
9       <CMS_ARRAY_ELEMENT><![CDATA[<ul>]]></CMS_ARRAY_ELEMENT>
10    </CMS_ARRAY_PARAM>
```

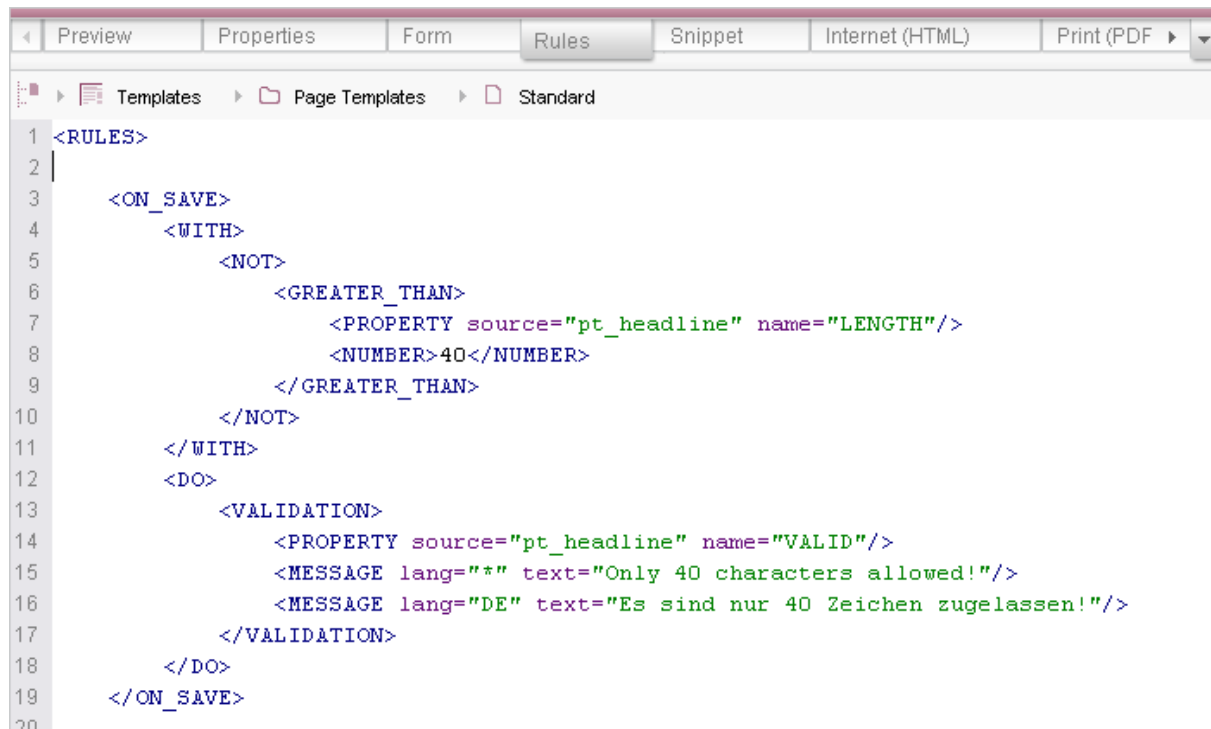
Figure 2-41: Page template – Template sets tab

The "Internet", "Print" and "RSS feed" tabs are template sets that the project administrator created during server and project configuration for this project. The tabs show the source text of the different template sets for the current template. If the template is locked, changes can be made here directly.

If a change has been made to the source text, the formatting of CMS_HEADER is checked for the change when saving. If an error occurs, it is displayed immediately via a new window.



2.5.5 Rules tab



```
1 <RULES>
2 |
3     <ON_SAVE>
4         <WITH>
5             <NOT>
6                 <GREATER_THAN>
7                     <PROPERTY source="pt_headline" name="LENGTH"/>
8                     <NUMBER>40</NUMBER>
9                 </GREATER_THAN>
10            </NOT>
11        </WITH>
12        <DO>
13            <VALIDATION>
14                <PROPERTY source="pt_headline" name="VALID"/>
15                <MESSAGE lang="*" text="Only 40 characters allowed!"/>
16                <MESSAGE lang="DE" text="Es sind nur 40 Zeichen zugelassen!"/>
17            </VALIDATION>
18        </DO>
19    </ON_SAVE>
20
```

A template developer can influence specific elements or properties of a form by defining rules within the (form) template and can create a "dynamic form" this way. The "Rules" tab area is used for defining rules. Here, for instance, an input component that has been defined in the template's "Form" tab area can be linked to a rule.

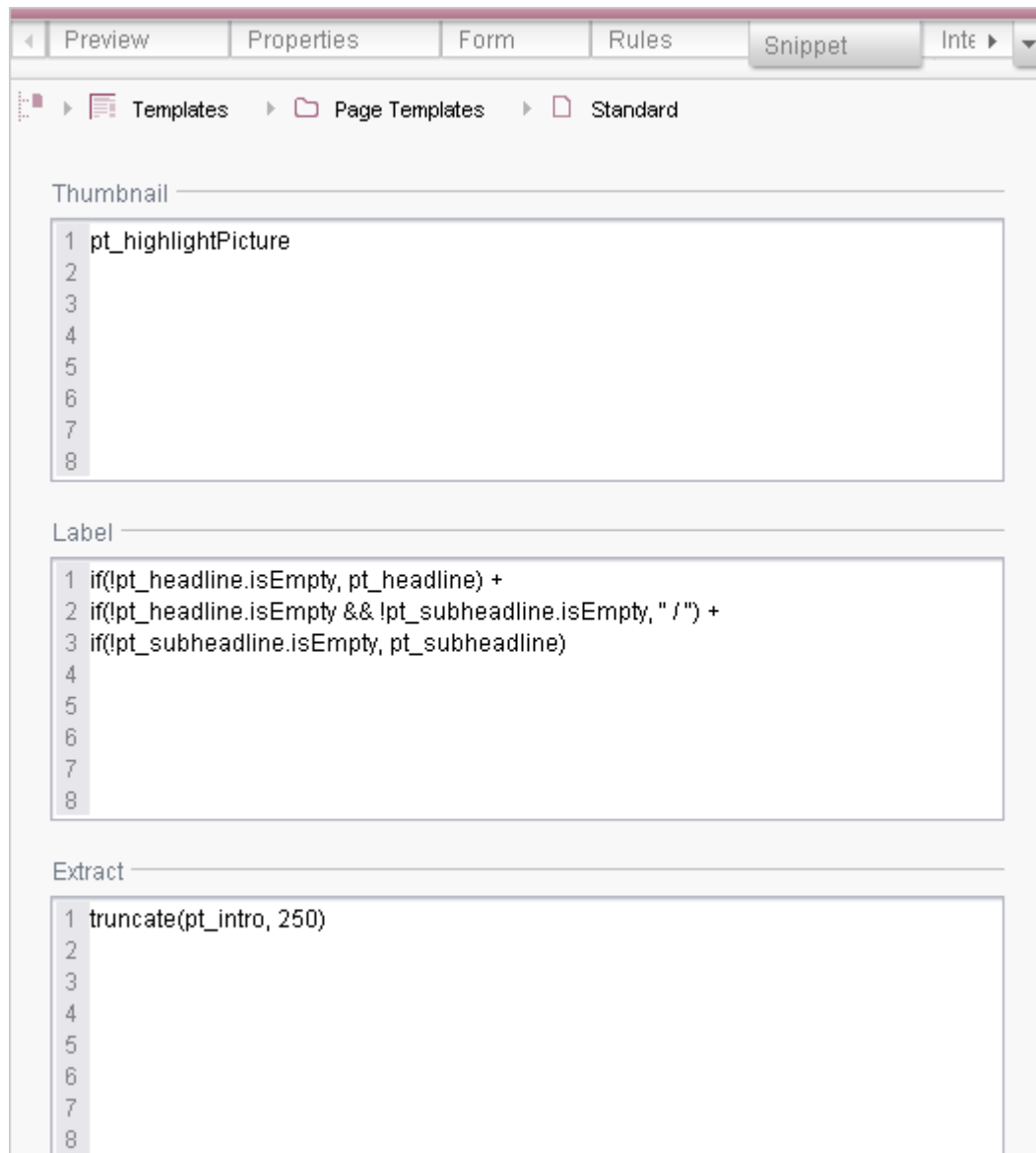
You can find a detailed description of the rules in the [FirstSpirit Online Documentation](#).



Higher level objects, i.e. objects outside the actual form, cannot be included. This means that you cannot influence which templates may be used in which areas of the site store, for instance.



2.5.6 Snippet tab



The way search results are to be displayed based on some template types can be specified via the "Snippet" tab. The variable names of a template's input components are referenced for this. This display is used in both JavaClient and WebClient.

The goal is to display search hits with the following instead of with just the object name:

- an image
- a title
- a text excerpt



This should be information that best represents the respective object. This way, the editor is able to receive a clear presentation of the content of the search hit in order to determine the most relevant hit more easily and to get to the object being searched for more quickly.

In order to be able to adapt the output more strongly to the requirements of the respective projects and editors, multiple input components can also be combined. In addition, methods that can be used with `$CMS_VALUE(...)$` can be implemented. Thus input editorial content can be used for searching depending on the inputs. By default, if an input component specified on the "Snippet" tab is not filled by an editor, the name is shown as the title and the path to the search hit is shown as the text excerpt. The path is shown in WebClient in each case regardless.

You can find a detailed description of the snippet in the FirstSpirit Online Documentation.

2.6 Section templates

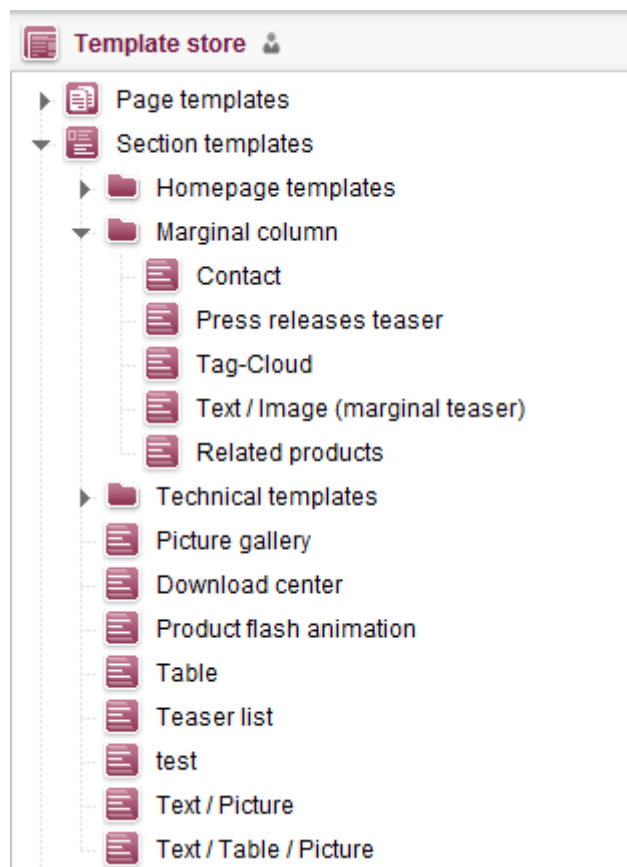


Figure 2-42: Template store tree view – Section templates



Section templates are used to insert content into the basic framework of a page defined by page templates. All of the input elements that are to hold dynamic page content (text, tables, images, data records, etc.) are defined in a section template. Any number of sections can be added in each section area of a page. There are usually also multiple different section templates available for the different types of potential content a page may have.

Tree elements in section templates:



Root element of section templates



Folder within section template node



Section templates

2.6.1 Preview, Properties, Form, Template sets, Rules and Snippet tabs

The Preview, Properties, Form, Rules, Snippet and Template sets tabs of the section templates are identical in function to the page template tabs of the same name and can be edited the same way.

You can find information on the individual tabs in Chapter 2.5.1 (page 56) to Chapter 2.5.6 (page 62).



There is also the option of using HTML anchor links on a page, such as when using extensive page content. This requires that the `<a>` tag be enabled on the section template's Properties tab. Subsequently, the anchor link is generated from the section template's reference name automatically and can then be used within a link template.

You can find a detailed description of using anchor links in the FirstSpirit Online Documentation.



The option "Hide template in selection list" (see Chapter 2.5.2) is not available for section templates.



2.7 Format templates

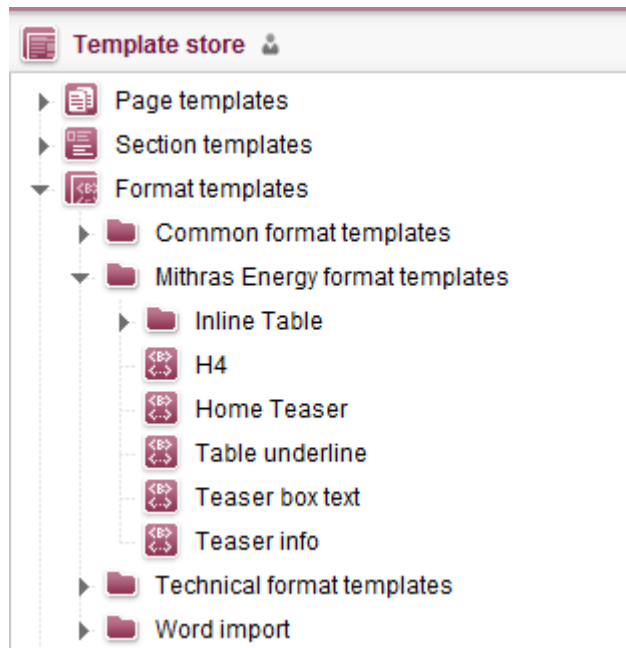


Figure 2-43: Template store tree view – Format templates

Text formatting is defined using format templates. The formatting can then be used in the DOM editor and DOM table input elements. Some format templates are included in the delivery by default, such as format templates for sections ("standard"), line breaks (abbreviated as "br"), "bold", "italic", to display tables (abbreviated as "table", "tr" and "td"), etc. They can be found in the "Common format templates" folder in the "Format templates" area of the Template Store.



These standard format templates are in many ways required for correct operation and must not be deleted.

The format templates

- "Deleted" (abbreviated as "deleted"),
- "Deleted (Block)" (abbreviated as "deleted_block"),
- "Inserted" (abbreviated as "inserted") and
- "Inserted (Block)" (abbreviated as "inserted_block"),

are used to display version comparisons ("Version history"). Changing the properties of these



format templates (color, font size, border, etc.) (see section 2.7.1, page 66) affects how changes are displayed in version comparisons.

It is even possible to change how other standard format templates are displayed in the DOM editor.

In addition to the standard format templates, template developers can create other project-specific format templates.

2.7.1 Properties tab

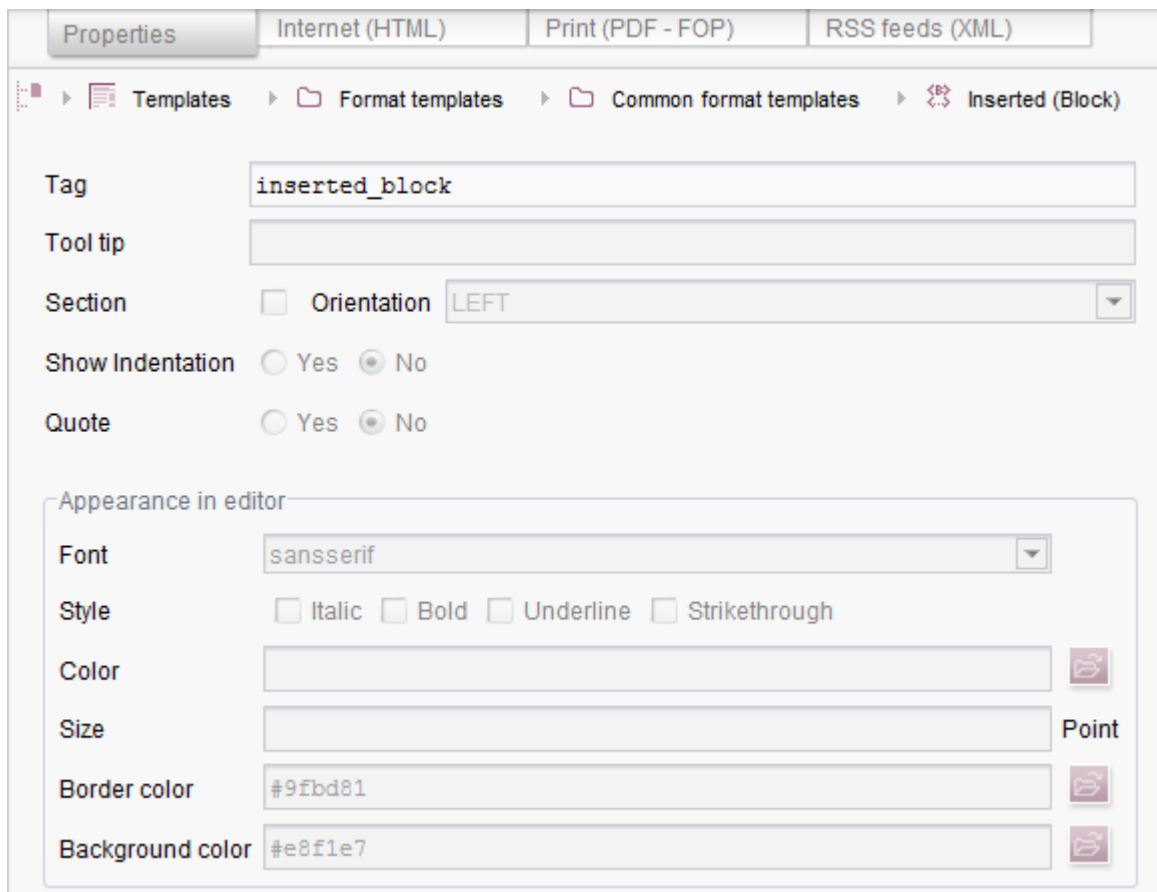


Figure 2-44: Format template – "Properties" tab

The basic properties of a format template are defined on the **Properties** tab page. The individual fields have the following meanings in this context:

Tag: The text entered in this field is needed in the form area of a page and section template to specify valid format templates for the input component. The corresponding xml tag name is



generated from this name, such as for the "Bold" format template (also refer to FirstSpirit Online Documentation – Template syntax / format templates area). The name must be unique and cannot contain any special characters. It is specified automatically according to the format template's unique reference name. In order to ensure the uniqueness, the abbreviation should not be changed manually (see Chapter 2.2.1, page 23).



A unique name or abbreviation of a format template cannot be changed once it has been created since otherwise all references within the project would be lost!

Tool tip: The text entered in this field appears as helpful text when the user hovers over it with the mouse, such as in the DOM editor.

Section: The entire section is formatted if the "Section" checkbox is checked. If the checkbox is unchecked, the formatting is only applied to individual, highlighted characters.

Alignment: If the "Section" checkbox has been checked, the alignment of the text, such as in the DOM editor, can be specified here.

Display indent: Defines how the corresponding formatted text is to be displayed. All of the blank spaces are displayed and the text is no longer automatically wrapped if this option is enabled. Black spaces are shown in HTML notation if this option is disabled.

Quote: Selecting **Yes** applies the complete conversion rules to the individual template sets (the convert part and quote part). Selecting **No** only applies the convert part of the conversion rules to the individual template sets.

Additional formatting only shown in the editor can be defined using the "**Display in editor**" field.

Font: A font used to display the text is selected here. (This font has to be installed on every client computer, otherwise a similar font is used.)

Style: You can select whether the text is to be shown in bold, italics or underlined in the DOM editor here.

Color: The color for the displayed text can be selected here.

Size: The size that text is to be shown at in the DOM editor is specified here. Relative inputs are also possible here (+2, -1, etc.).



Border color: The color for a border within an input component can be selected here.

Background color: The color for the background within an input component can be selected here.

2.7.2 Template sets tab

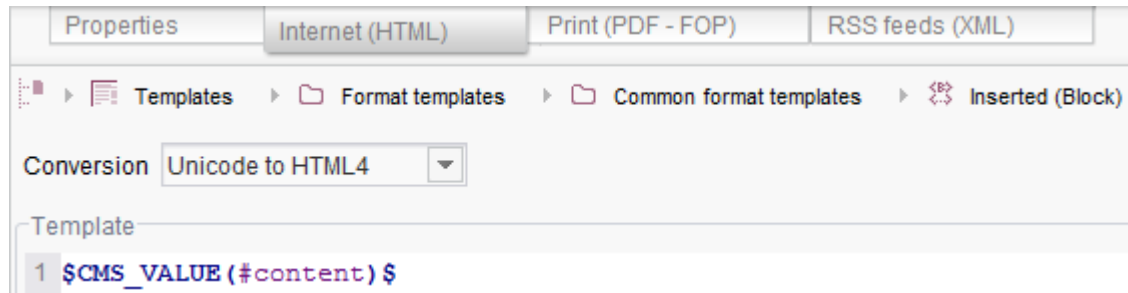


Figure 2-45: Format template – Template sets tab

Conversion: One of the conversion rules configured in the server and project configuration in the server properties can be selected here.

Template: The HTML code that generates the desired formatting for text on the website can be entered here. The `#content` expression represents the text that has been entered in the DOM editor.

A detailed description of the format templates and the conversion rules is available in the FirstSpirit Online Documentation.



The selected conversion rule is applied only when outputting `CMS_INPUT_DOM` or `CMS_INPUT_DOMTABLE` input components via the `#content` system object, e.g. via `$CMS_VALUE(#content)$`.

With FirstSpiritVersion 5.0, line numbers are indicated at many places in the Template Store as a reading aid. They can be shown or hidden in JavaClient using the keyboard shortcut `Ctrl + L`.



2.8 Style templates

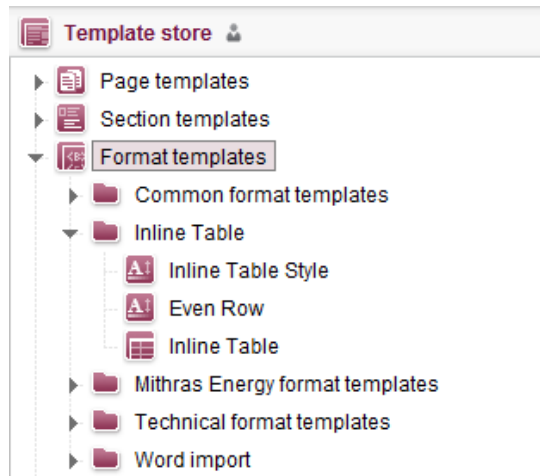


Figure 2-46: Template store tree view – Style templates

2.8.1 Introduction: Inline tables

What are known as inline tables can be integrated into the flow of text using the DOM editor (CMS_INPUT_DOM input component). This allows any number of design possibilities to be made available to the editor down to the cell level.



The table layout is specified by both table format templates (see Chapter 2.9, page 80) and style templates (start from Chapter 2.8.2, page 70). Style templates are used to define table layout features, such as background color, text alignment, font, word wrapping, borders and border spacing.

Each table format template can be assigned precisely one standard style template (for the entire table) and multiple additional style templates for separately displaying individual table cells (see Chapter 2.9.1, page 82). The style templates define the layout of individual table cells, such as the background color ("bgcolor"), the alignment of text in a cell ("align") or the color of the text in a cell ("color").

Therefore, a style template has to be created first in order to use inline tables in the DOM editor (see Chapter 2.8.2, page 70).

Style and table format templates should be combined in a folder (e.g. "Table") for a better overview.



Inline table icons: Folder Table format template Style template

The inline table function is also available in WebEdit.

With FirstSpiritVersion 5.0, line numbers are indicated at many places in the Template Store as a reading aid. They can be shown or hidden in JavaClient using the keyboard shortcut **Ctrl + L**.

2.8.2 Creating a style template

Style templates are created in the "Format templates" area. Select the item **New – Create style template** in the context menu to do this. A reference name for the style template has to be specified in the window that opens. Specifying a display name is optional.

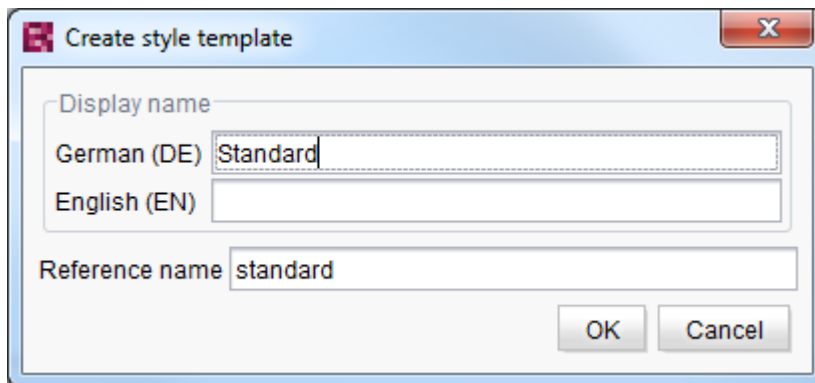



Figure 2-47: New – Create style template

Clicking  creates the new style template. Input components that affect the properties of the layout, such as background color, text alignment, font, word wrapping, borders and border spacing, can be created using a style template's form area (see Chapter 2.8.3, page 71).



2.8.3 Form area of a style template

Unlike other format templates, style templates have a "Form" tab. In the form area of a style template, input components can be created for maintaining layout attributes.



```

1 <CMS_MODULE>
2
3 <CMS_INPUT_COMBOBOX name="align" hFill="yes" singleLine="no" useLanguages="yes">
4   <ENTRIES>
5     <ENTRY value="left">
6       <LANGINFOS>
7         <LANGINFO lang="*" label="left"/>
8         <LANGINFO lang="DE" label="links"/>
9       </LANGINFOS>
10    </ENTRY>

```

Figure 2-48: Form area of a style template

Some specified layout attributes (with reserved identifiers) directly affect the display of the table in the DOM editor.

- `bgcolor`: Determines the background color of a table cell (for an example see Chapter 2.8.7.1, page 77)
- `color`: Determines the font color for text in a table cell (for an example see Chapter 2.8.7.2, page 78)
- `align`: Determines the alignment of text in a table cell (for an example see Chapter 2.8.7.3, page 79)



The specified identifiers cannot be modified. The attributes always have to be specified with `name="Identifier"` in the input component, e.g. `<CMS_INPUT_TEXT name="bgcolor" .../>`

Other additional, freely defined attributes can certainly be maintained using the form area's input component in addition to these standard attributes, e.g. CSS attributes.



Supported input components for maintaining layout attributes:

- `CMS_INPUT_TEXT / CMS_INPUT_TEXTAREA`: Text field for specifying a value, e.g. for the background color.
(for an example see Chapter 2.8.7.1, page 77)
- `CMS_INPUT_COMBOBOX`: Selection from a predefined number of values, e.g. for specifying a background color or alignment
(for an example see Chapter 2.8.7.2, page 78)
- `CMS_INPUT_RADIOBUTTON`: Selection from a predefined number of values, e.g. for specifying a background color or alignment
(for an example see Chapter 2.8.7.3, page 79)
- `CMS_INPUT_NUMBER`: Specifying a numeric value (e.g. value for a cell's background color)
- `FS_BUTTON`: Button for activating a script or executing a class

The addition of input components on the "Form" tab has been simplified by code completion (see Chapter 7, page 240).



The following applies to all input components used in a style template's form area: The components should be defined as language-independent (`useLanguages="no"`). In this case, the language-dependence of the component is covered by the language selection in the DOM editor instance being processed by the editor.



Additional input components for maintaining layout attributes (in style templates) are not supported.

2.8.3.1 Preventing layout editing for editors

Editors can be prevented from performing layout attribute maintenance. Either the attribute `hidden="yes"` or a corresponding rule has to be defined in the input component for this. The attribute `hidden="yes"` causes the input component to be visible only in the template store but not when maintaining the table in the page store. Thus the template developer can use the attribute to prevent an editor from editing the layout and instead to specify defined values for the layout such as the background color for cells (see Chapter 2.8.4, page 74).



If the attribute `hidden="yes"` is defined (for all of a style template's input components), the editor does not have the option of modifying the layout properties of a table cell in the page store. The corresponding "Cell properties" button is inactive in this case.



*If maintenance of layout properties was prevented by using validators, the button is **never** inactive. Even if all of the input components are hidden. Consequently, an empty dialog appears in both clients. This behavior also applies to the dialog for preassigning layout attributes (see Chapter 2.8.4, page 74).*

You can find more detailed information on validators in the FirstSpirit Online Documentation.

If, on the other hand, individual components are "visible" (`hidden="no"`) and others are "hidden" (`hidden="yes"`), then the "Cell properties" button in the DOM editor (in the page store) is active; however, the editor is only shown the "visible" components in the following dialog.

All of the style template components are then displayed only if the template developer has not defined any restrictions.



Figure 2-49: "Cell properties" button in the DOM editor

The editor opens a dialog for editing project-specific layout attributes by clicking the button (see *FirstSpirit Manual for Editors*).



2.8.4 Preassigning layout attributes

Fallback values for the layout attributes can be defined in the template store. The input component is preassigned values (e.g. with a color value) in a separate dialog that can be opened on the "Properties" tab using the corresponding button.

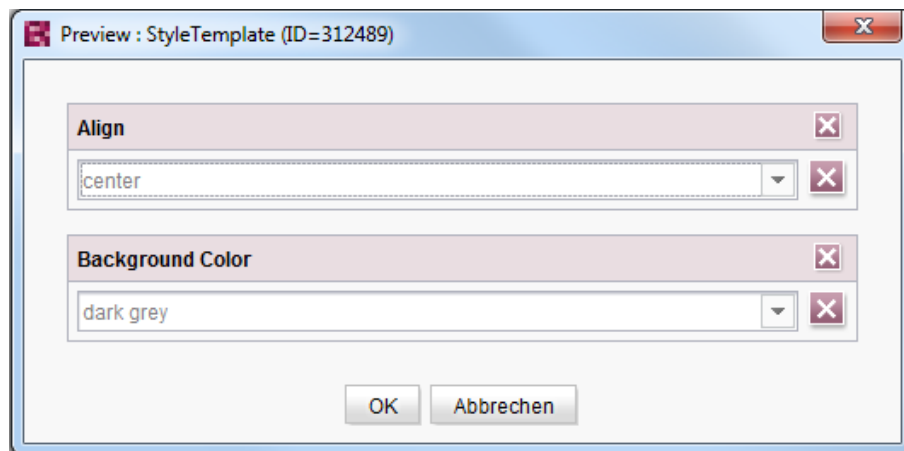


Figure 2-50: Preview style template

In the dialog, a preassignment for the input component can be defined by the template developer. A color value can be selected in the input component for "Background Color", for instance (see Figure 2-50). This color value is applied to all table cells based on the corresponding style template when creating an inline table in the DOM editor.

Depending on the defined value for the `hidden` attribute in the input component's definition, this preassignment can be modified by the editor (see Chapter 2.8.3.1, page 72).

If editing is possible (`hidden="no"`), the editor can overwrite this predefined value when editing a table cell in the DOM editor (see *FirstSpirit Manual for Editors*).



The values in the dialog can only be edited or saved if the template is locked ("Switch to edit mode" button)!



2.8.5 Presentation channel of a style template

Inside a style template's presentation channel (e.g. "HTML"), the values input in the input components can be read back out (see Chapter 2.8.4, page 74).

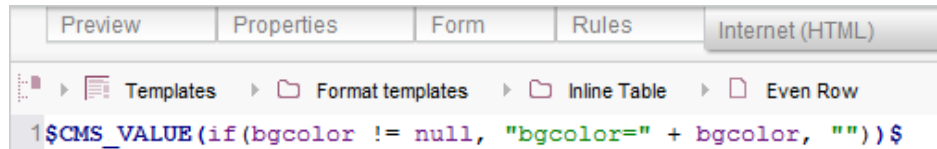


Figure 2-51: A style template's "HTML" presentation channel

The name of the input component has to be output using the instruction `$CMS_VALUE(...)$` for this:

```
$CMS_VALUE(if(bgcolor != null, " bgcolor=" + bgcolor, ""))$
```

or

```
$CMS_IF(!bgcolor.isEmpty)$CMS_VALUE(bgcolor)$CMS_END_IF$
```

Refer to the *FirstSpirit Online Documentation* for more detailed information on outputting variables³.

2.8.6 Linking with standard table format templates

Style template values can be linked to standard format templates for generating (and previewing) tables in a project using the system object `#style`. The standard format templates for tables made available by FirstSpirit are:

- Table (abbreviation: table): Formatting for tables
- Table cell (abbreviation: TD): Formatting for table cells
- Table row (abbreviation: tr): Formatting for table rows

For example, if, in a `TD` standard format template, the system object `#style` is used for instance, values that have been defined by the editor in the "Cell properties" dialog (see

³ FirstSpirit Online Documentation in the template development / variables area



FirstSpirit Manual for Editors) or values that have been predefined for the style template by the template developer (see Chapter 2.8.4, page 74) are taken into account during generation.

Example of the output to the HTML channel of the TD standard format template:

```
<td$CMS_VALUE(#style)$  
$CMS_VALUE(if(#cell.rowspan != 0, " rowspan='" + #cell.rowspan + "'"))$  
$CMS_VALUE(if(#cell.colspan != 0, " colspan='" + #cell.colspan + "'"))$>  
$CMS_VALUE(if(#content.isEmpty, "&nbsp;", #content))$  
</td>
```



For more detailed information about how properties and information from tables and their contents can be accessed, see [FirstSpirit Online Documentation](#), #cell, #content, #table and #tr system objects in the [template development/template syntax/system objects area](#).

The values of layout attributes defined by the editor (or template developer) in the "Cell properties" dialog are now taken into account when generating the table (see Figure 2-52):

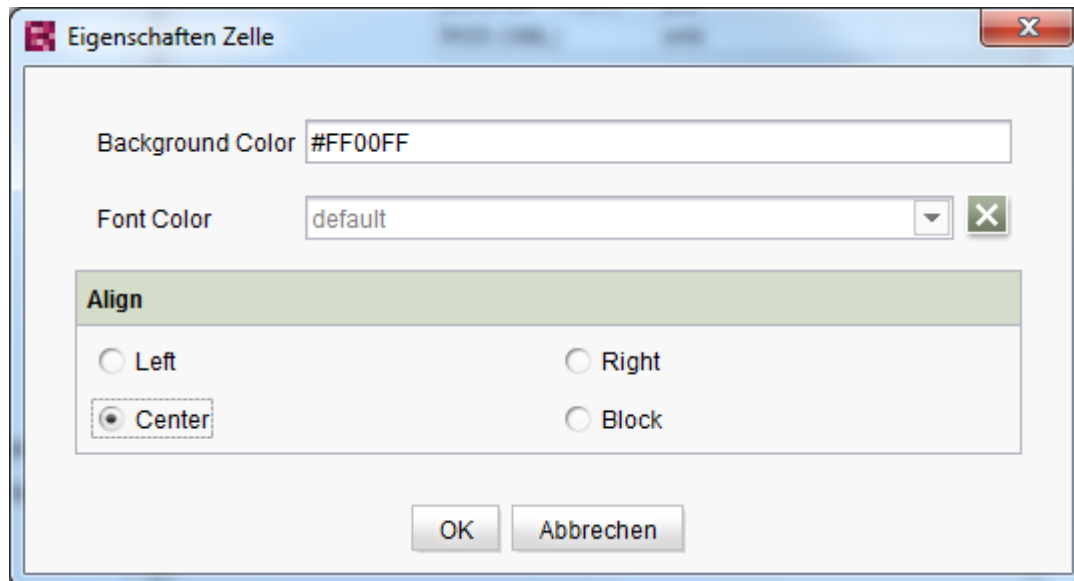


Figure 2-52: Table cell properties



The source text of a table cell is now generated as follows, for instance:

```
<table>
<tr>
<td bgcolor="#ff00ff" align="center" color="#00ddee" rowspan='1'
colspan='1'>This is a text.</td>
..
</tr>
</table>
```

2.8.7 Examples

2.8.7.1 Example: Text input component for entering a background color

Defining a component in the form area:

```
<CMS_MODULE>
  <CMS_INPUT_TEXT name="bgcolor" useLanguages="no">
    <LANGINFOS>
      <LANGINFO lang="*" label="Background color:"/>
    </LANGINFOS>
  </CMS_INPUT_TEXT>
</CMS_MODULE>
```

name="bgcolor": The input component uses the key value "bgcolor" to define a background color. This name cannot be changed since it is a fixed key value.

Inputting a color value using an input component:

A screenshot of a web form input field. The field has a light purple header with the text "Background Color:" and a close button (X) on the right. Below the header is a text input box containing the hexadecimal color code "#FF00FF".

Figure 2-53: Input component for entering a background color

For outputting the value in the style template's presentation channel, see Chapter 2.8.5, page 75.



2.8.7.2 Example: Input component for entering a text color

Defining a component in the form area:

```
<CMS_MODULE>
  <CMS_INPUT_COMBOBOX name="color" useLanguages="no">
    <ENTRIES>
      <ENTRY value="">
        <LANGINFOS>
          <LANGINFO lang="*" label="default"/>
        </LANGINFOS>
      </ENTRY>
      <ENTRY value="#ee00ff">
        <LANGINFOS>
          <LANGINFO lang="*" label="superior"/>
        </LANGINFOS>
      </ENTRY>
      <ENTRY value="#00ddee">
        <LANGINFOS>
          <LANGINFO lang="*" label="lightGrey"/>
        </LANGINFOS>
      </ENTRY>
    </ENTRIES>
    <LANGINFOS>
      <LANGINFO lang="*" label="Font Color"/>
    </LANGINFOS>
  </CMS_INPUT_COMBOBOX>
</CMS_MODULE>
```

Selecting a color value using an input component:

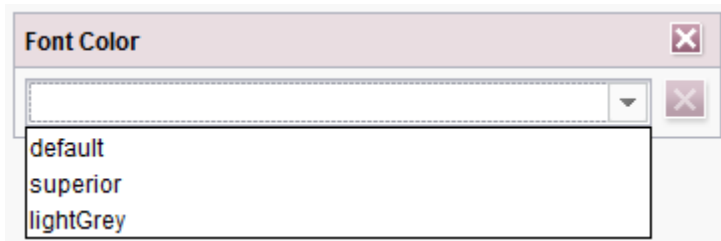


Figure 2-54: Input component for selecting a color value for the text color

For outputting the value in the style template's presentation channel, see Chapter 2.8.5, page 75.



2.8.7.3 Example: Input component for entering text alignment

Defining a component in the form area:

```
<CMS_MODULE>
  <CMS_INPUT_RADIOBUTTON name="align" useLanguages="no">
    <ENTRIES>
      <ENTRY value="">
        <LANGINFOS>
          <LANGINFO lang="" label="Left"/>
        </LANGINFOS>
      </ENTRY>
      <ENTRY value="right">
        <LANGINFOS>
          <LANGINFO lang="" label="Right"/>
        </LANGINFOS>
      </ENTRY>
      <ENTRY value="center">
        <LANGINFOS>
          <LANGINFO lang="" label="Center"/>
        </LANGINFOS>
      </ENTRY>
      <ENTRY value="block">
        <LANGINFOS>
          <LANGINFO lang="" label="Block"/>
        </LANGINFOS>
      </ENTRY>
    </ENTRIES>
    <LANGINFOS>
      <LANGINFO lang="" label="Align:"/>
    </LANGINFOS>
  </CMS_INPUT_RADIOBUTTON>
</CMS_MODULE>
```



Selecting a text alignment using an input component

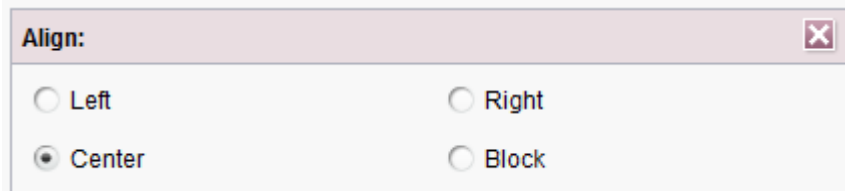


Figure 2-55: Input component for selecting text alignment

For outputting the value in the style template's presentation channel, see Chapter 2.8.5, page 75.

2.9 Table format templates

Table format templates are required for creating what are known as inline tables (see Chapter 2.8.1, page 69). A table format template has to be created in the "Format templates" area for each desired table layout.

Select **New – Create table format template** in the context menu to do this. A reference name for the table format template has to be specified in the window that opens. Specifying a display name is optional.

A detailed description of reference and display names can be found in Chapter 2.2.1, page 23.

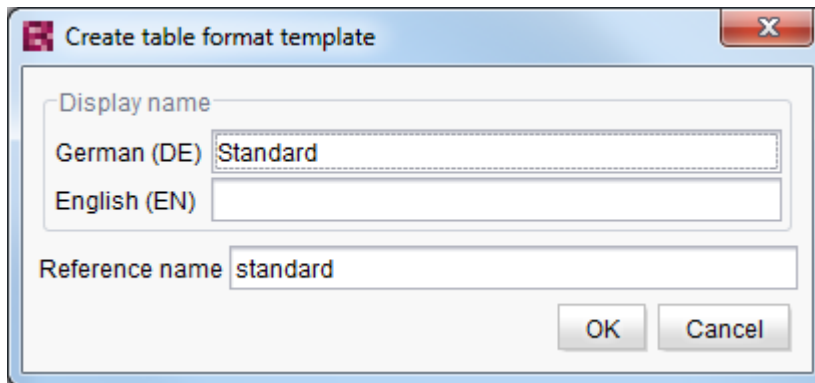


Figure 2-56: New – Create table format template

The "Properties" tab opens once has been clicked. The size of the table can be defined and the style templates created in Chapter 2.8.2 can be assigned here.



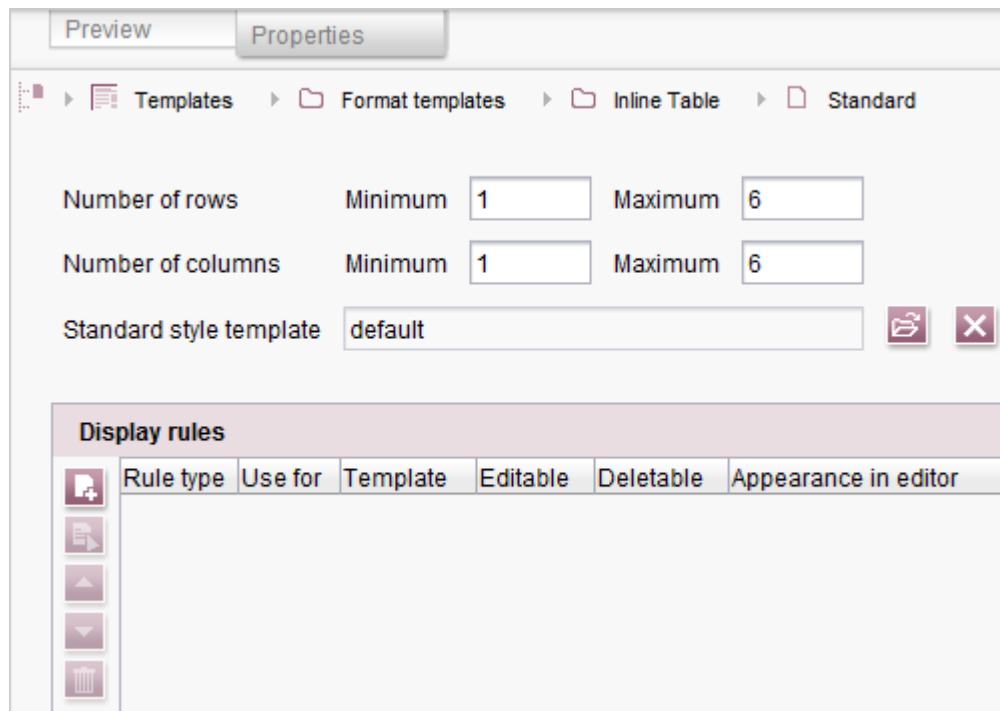



Figure 2-57: Table format template

Number of rows/columns: The **Minimum** and **Maximum** fields are used to define how many rows and columns the table can have at least and at most.



The minimum of rows and columns is by default 2.

If the editor inserts an inline table with this table format template into the DOM editor later on, the table is automatically created with the minimum number of rows and columns specified here. The editor cannot exceed or go below the default values when editing a table. The corresponding buttons are disabled in this case. For instance, if a minimum row count of four rows is defined, the "Delete row" button is disabled in the DOM editor as soon as the table only contains four rows.

Standard style template: The desired style template used as the basis for the table can be selected in this field by clicking the  icon. All of the available style templates are displayed in the window that opens.



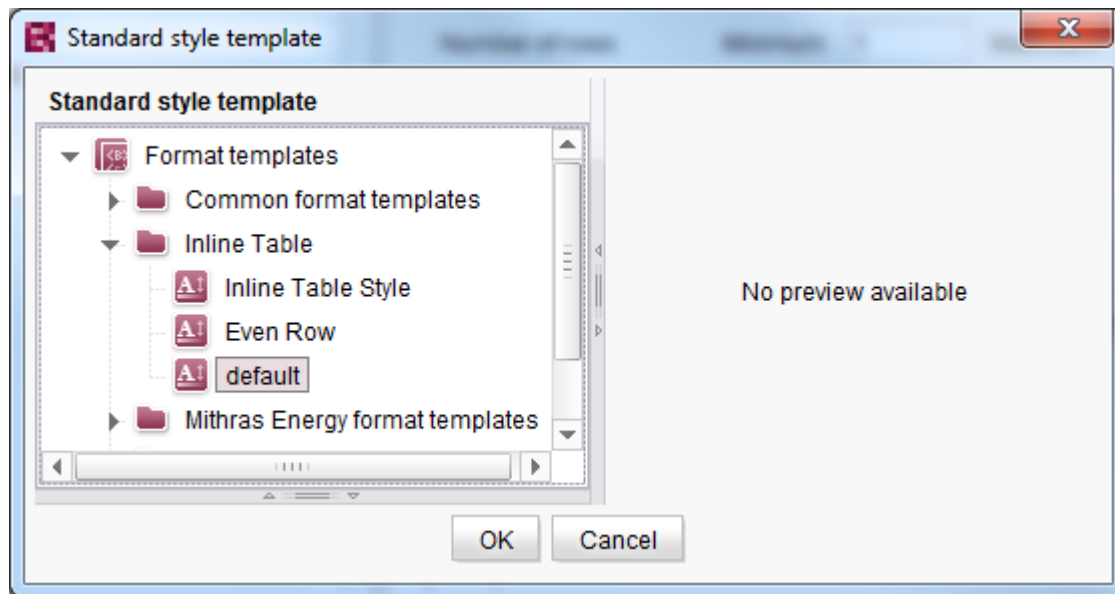



Figure 2-58: Table format template – Standard style template

The desired style template can be selected from the tree structure and the selection can be confirmed with .

2.9.1 Creating and editing display rules

The default style template defined in the table format template applies as the basis for a table's layout (see Figure 2-58). Furthermore, additional layout options for formatting rows, columns and individual cells that overwrite the default style template are available to the editor in the **Display rules** area. These layout options are based on the previously created style templates (see Chapter 2.8.2, page 70).



New display rules are created using the  icon. The following window opens after the icon is clicked:

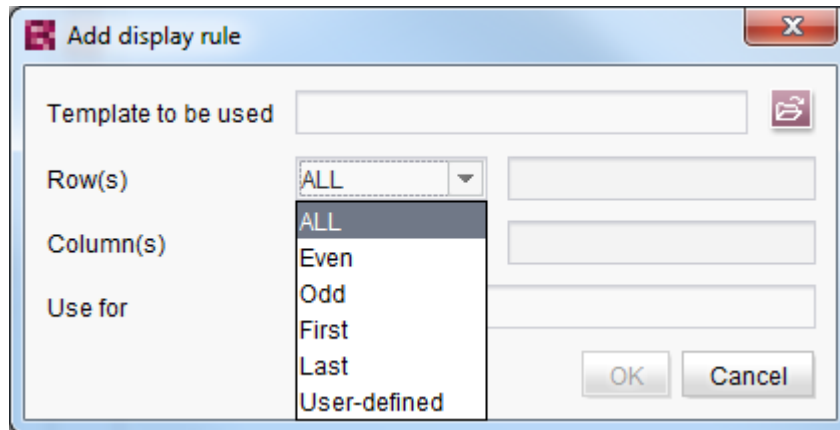



Figure 2-59: Table format template – Display rule

The desired style template to be applied for the display rule can be selected by clicking the  icon. All of the available style templates are displayed in the window that opens.

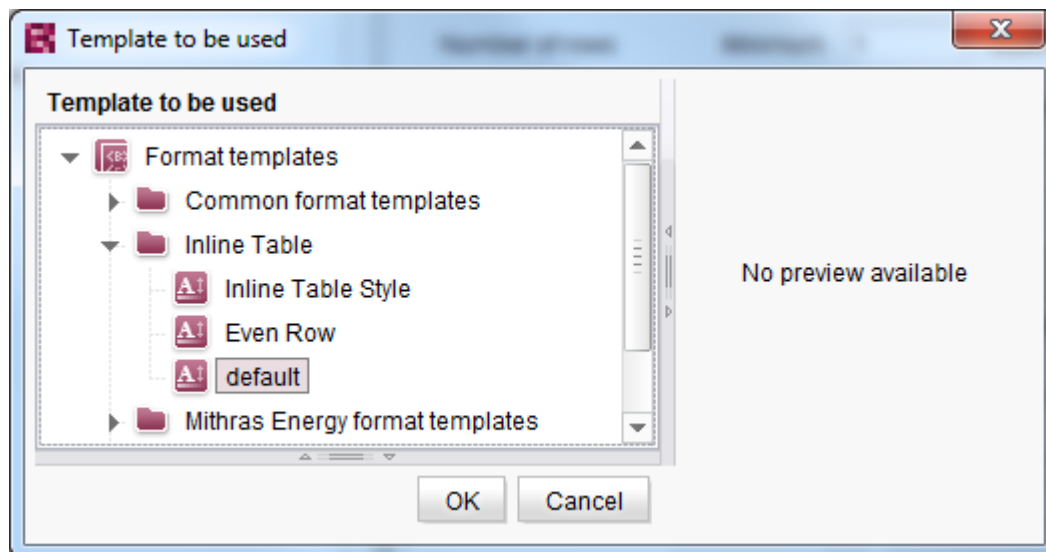



Figure 2-60: Table format template – Style template

The desired style template can be selected from the tree structure and the selection can be confirmed with .

Conditions for applying the rule, and thus applying the selected style template, are defined using



the **Row(s)** and **Column(s)** comboboxes. Both conditions have to be met to apply the rule.

ALL: The display rule applies to all rows or columns without restriction. If the "ALL columns" option were selected, the display rule would only take the restriction defined under the "Rows" option into account and vice versa.



Defining a display rule that affects ALL columns and ALL rows is not possible. A rule of that type would correspond to the default style template.

Even: The display rule applies to even rows or columns (starting with the second row/column).

Odd: The display rule applies to odd rows or columns (starting with the first row/column).

First: The display rule applies to the first row or column.

Last: The display rule applies to the last row or column.

User-defined: The display rule applies to a specific row or column. The number for the specific row/column has to be entered in the field to the right next to the combobox if this option is enabled.

The "Applies to" field continues to show which row(s) and column(s) the selected style template applies to.

Examples:

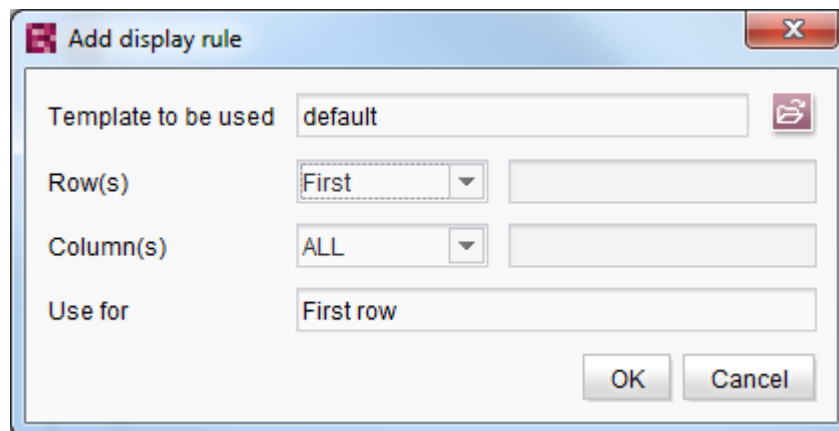


Figure 2-61: Display rules – Example 1



In this example, the "header" style template is applied to the **First row** in **ALL columns**, i.e. to all of the cells in the first row.

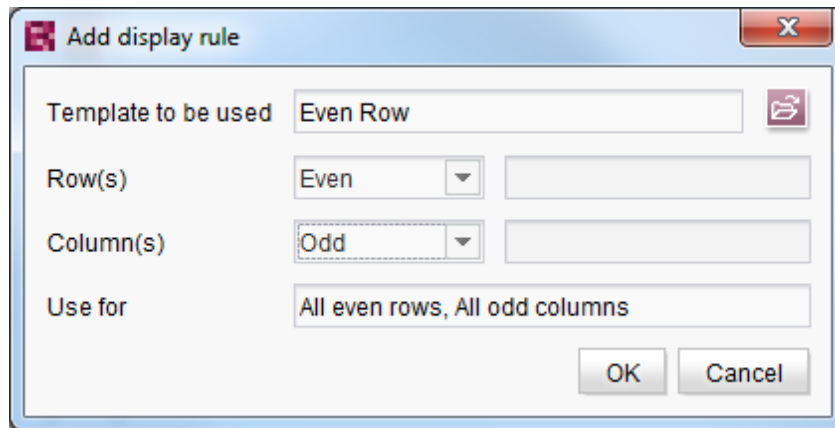



Figure 2-62: Display rules – Example 2

In this example, the "checker" style template is applied to **Even rows** in **Odd columns**, i.e. to all of the cells where both conditions apply.

The settings for the new display rule are saved by clicking . The display rule then appears in the following list:

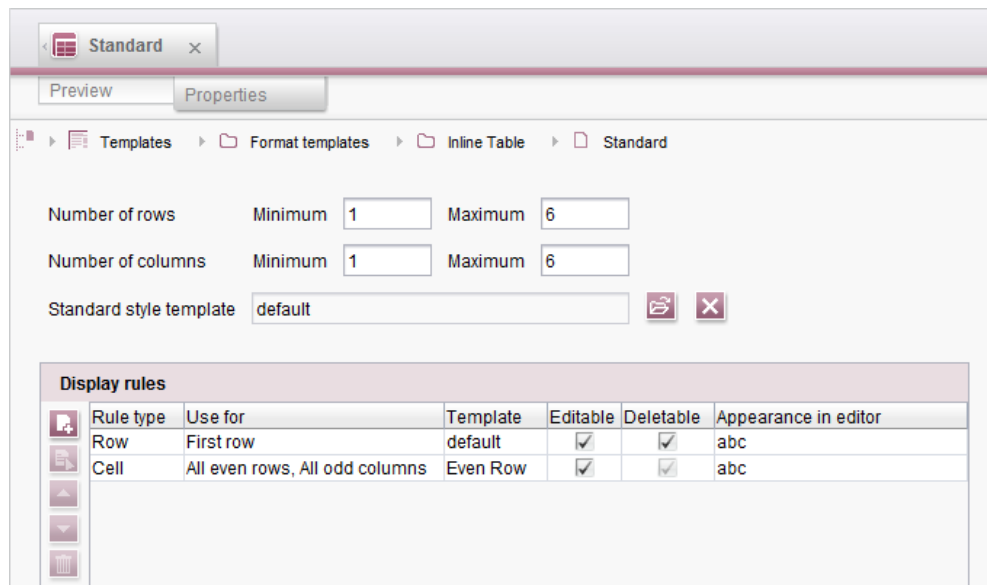


Figure 2-63: List of display rules

Rule type: Specifies whether the rule applies to rows, columns or cells.



Applies to: Specifies which areas of the table the rule applies to.

Template: Specifies which style template is applied.

Editable: This checkbox is checked by default. As a result, the editor can modify the properties of the cell(s) that the display rule applies to. The editor cannot change the properties for the associated cell(s) if this checkbox is unchecked.

Deletable: This checkbox is checked by default if the rule is for rows or columns. The editor can delete the row(s) or column(s) that this display rule applies to. The editor cannot delete the associated row(s) or column(s) if the checkbox is unchecked. The checkbox is checked if the rule applies to a cell. It cannot be unchecked since individual cells cannot be deleted from tables.

Display in the editor: This column shows the background color and the text alignment of the row/column/cell if the corresponding values are defined.



Edit: Clicking this icon (or double-clicking the display rule) opens an already existing display rule for editing.



Up one position: If multiple display rules are present, they can be moved up in the list by one position using this icon.



Down one position: If multiple display rules are present, they can be moved down in the list by one position using this icon.



Delete: Clicking this icon deletes the highlighted display rule.

The width of columns in this list can be changed as needed by clicking on the column line and dragging with the mouse button held down.



If multiple rules are present in the list, they are evaluated from top to bottom.

2.9.2 Evaluation order

The table format templates, style templates and display rules created in chapters 2.8.2 to 2.9.1 contain formatting specifications. These are evaluated as follows:

1. First, the **display rules** in the list (see
2. Figure 2-63: List of display rules) are evaluated top-down.



3. The **standard style template** is used in cells where a display rule does not apply.

2.9.3 Inserting an inline table in the DOM editor

In order to make inline tables available to the editor in the DOM editor, this input component type has to be inserted in the desired section template. To accomplish this, the parameter `table="yes"` has to be added to the input component `CMS_INPUT_DOM`.

Example:

```
<CMS_INPUT_DOM name="st_inlinetable" table="yes">
  <LANGINFOS>
    <LANGINFO lang="*" label="Table"/>
  </LANGINFOS>
</CMS_INPUT_DOM>
```

The input component can appear as follows:

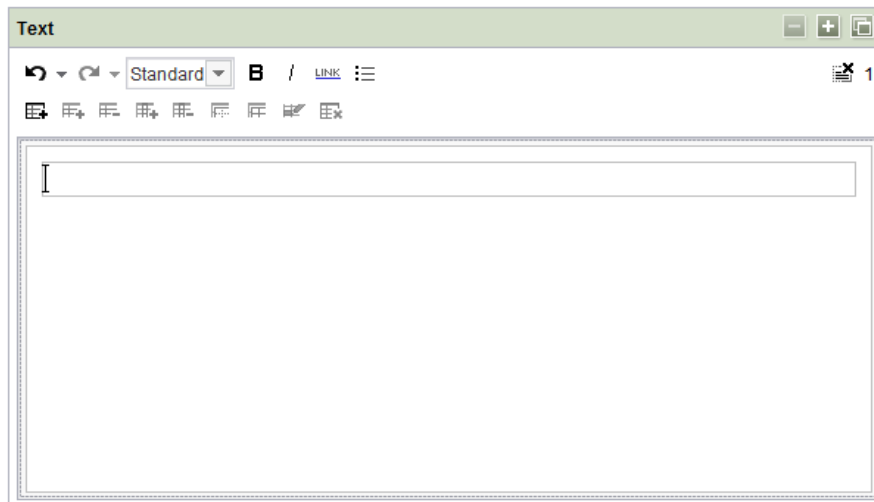


Figure 2-64: DOM editor with inline table



2.10 Link templates

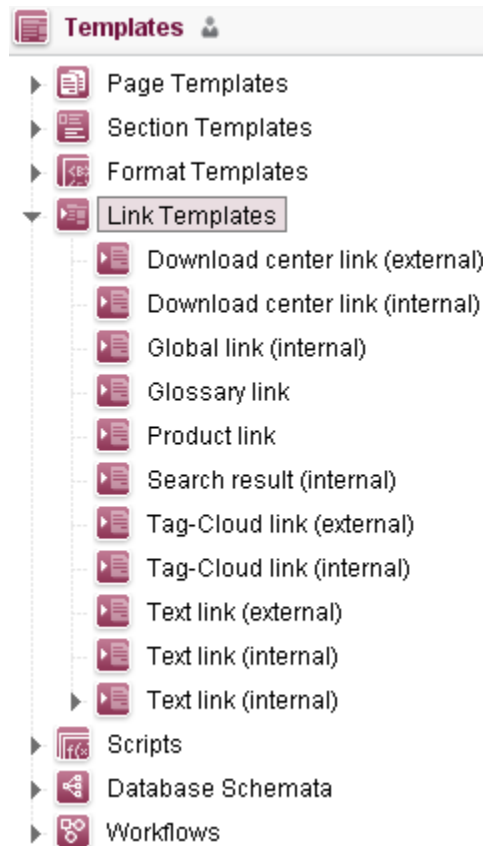


Figure 2-65: Template store tree view – Link templates

Template developers can specify the layout of links within a FirstSpirit project in detail using link templates. Editors input all of the necessary content using an input screen. Which fields can be filled out in the input screen depends on the configuration of link templates. Previously static link editors have been completely converted for generic links in FirstSpirit Version 5.0. As many instances of these links as is necessary can be created below the "Link templates" node. Each instance has to have a unique name.

Different link templates can be defined for different input components in the form area of a page or section template thanks to the option to define multiple instances (link templates). This way, internal links input in the DOM editor input component by the editor are configured and displayed differently than links such as those maintained in the FS_LIST input component.





For more detailed information on link templates see "FirstSpirit Online Documentation".



If you change link templates afterwards, this can result in the fact that the preview of content which has been already entered in the related input masks is not up-to-date. This can be resolved by repeated saving the input masks.

2.10.1 Standard link types

Selecting a link type is no longer necessary since static links are no longer supported in FirstSpirit Version 5.0 and thus only generic links still exist. The corresponding selection in the New dialog has been removed for this reason.

More detailed information on this topic can be found in the release notes for FirstSpirit 5.0.

2.10.2 Generic link editors

The configuration options of link templates have been expanded by the implementation of "generic link editors". Just like with page and section templates, now the configuration is created with the help of input components in the form area. All input options for maintaining links can be mapped using FirstSpirit's regular form syntax in the process.

The addition of input components on the "Form" tab has been simplified by code completion (see Chapter 7, page 240).

As part of implementing generic link editors, link templates can now be structured in folders as well.

The conventional input options for links (in static link editors) can certainly be generated using the new, generic editors as well. Some new input components have been introduced to map all of the functions of previous static link editors to the new generic editors. For instance, making a selection using the "mediaref" field in static link editors was not able to be mapped to previously existing input components. The input components CMS_INPUT_PICTURE and CMS_INPUT_FILE each only support selecting the respective reference type, i.e. either images or files but not both. Therefore, FS_REFERENCE input components were introduced, which

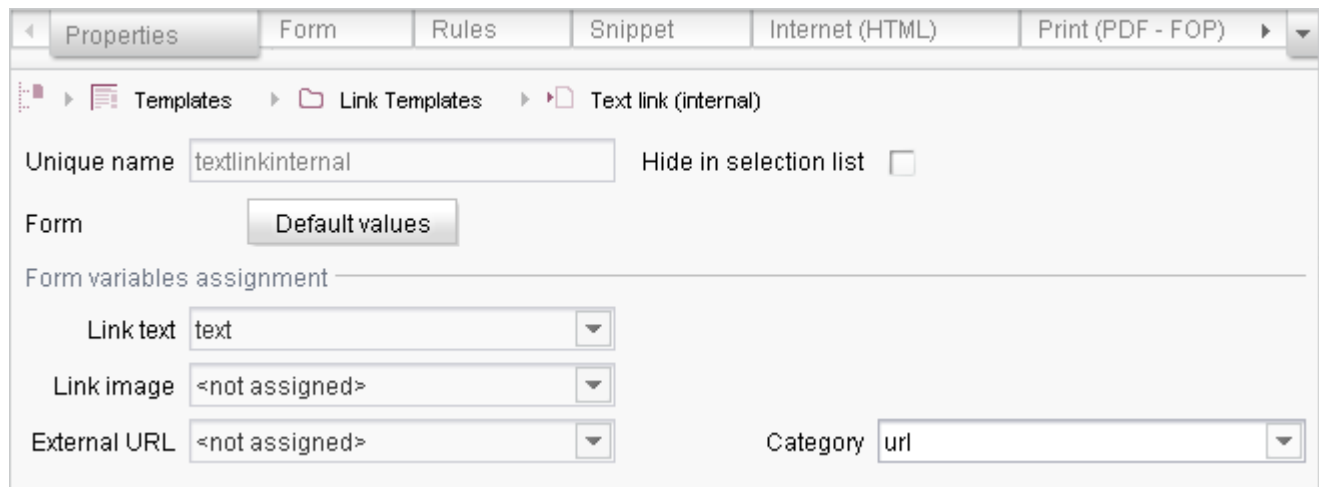


support any number of reference types.

In the same vein, expansions to CMS_INPUT_ OBJECTCHOOSER input components for selecting data records from *one* specific database table and the new FS_DATASET input component for selecting data records from *any number of* database tables were introduced for mapping links to database content.

The distinction between definition (form) and output was removed for generic link editors. Now a new template just has to be created under the "Link templates" root node or under a folder.

The "Properties" tab can appear as follows:



The screenshot shows the 'Properties' tab of a generic link editor. The breadcrumb path is 'Templates > Link Templates > Text link (internal)'. The 'Unique name' field contains 'textlinkinternal' and there is a 'Hide in selection list' checkbox. Below this is a 'Form' section with a 'Default values' button. A horizontal line separates this from the 'Form variables assignment' section. This section contains three dropdown menus: 'Link text' (set to 'text'), 'Link image' (set to '<not assigned>'), and 'External URL' (set to '<not assigned>'). To the right of these is a 'Category' dropdown menu set to 'url'.

Figure 2-66: Generic link – "Properties" tab

Each input component can be used on the "Form" tab.

For more detailed information on generic link editors see "FirstSpirit Online Documentation" – "Link templates" / "Generic link editors" chapter.



2.11 Scripts

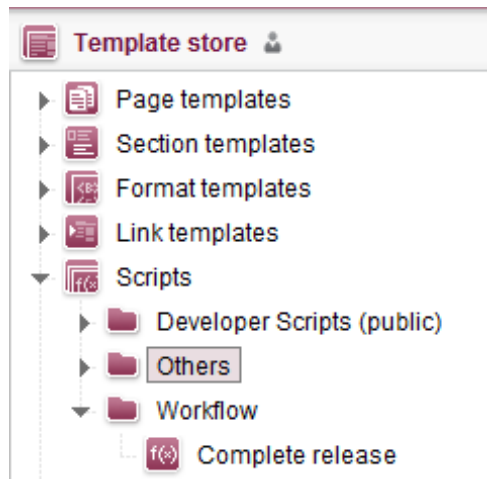


Figure 2-67: Template store tree view – Scripts

Different types of operating workflows can be automated in FirstSpirit using scripts. In this process, a script is used for describing the sequence to be carried out and can make changes to FirstSpirit data structures as needed. Scripts allow functions that are not yet present in FirstSpirit to be implemented quickly. Additional implementation areas include complex migration scenarios and connecting external systems.

BeanShell⁴ is the supported scripting language in FirstSpirit. BeanShell syntax is heavily based on JAVA but offers multiple simplifications, such as dynamic typing for variables and functions instead of static typing, as well as (limited) reflexive access to the program itself and substantial additional functionality.

Scripting with BeanShell provides a high degree of flexibility for template developers. Working with scripts is not a trivial matter, however. Therefore, carefully check whether a corresponding function is already available in FirstSpirit before implementing a script.

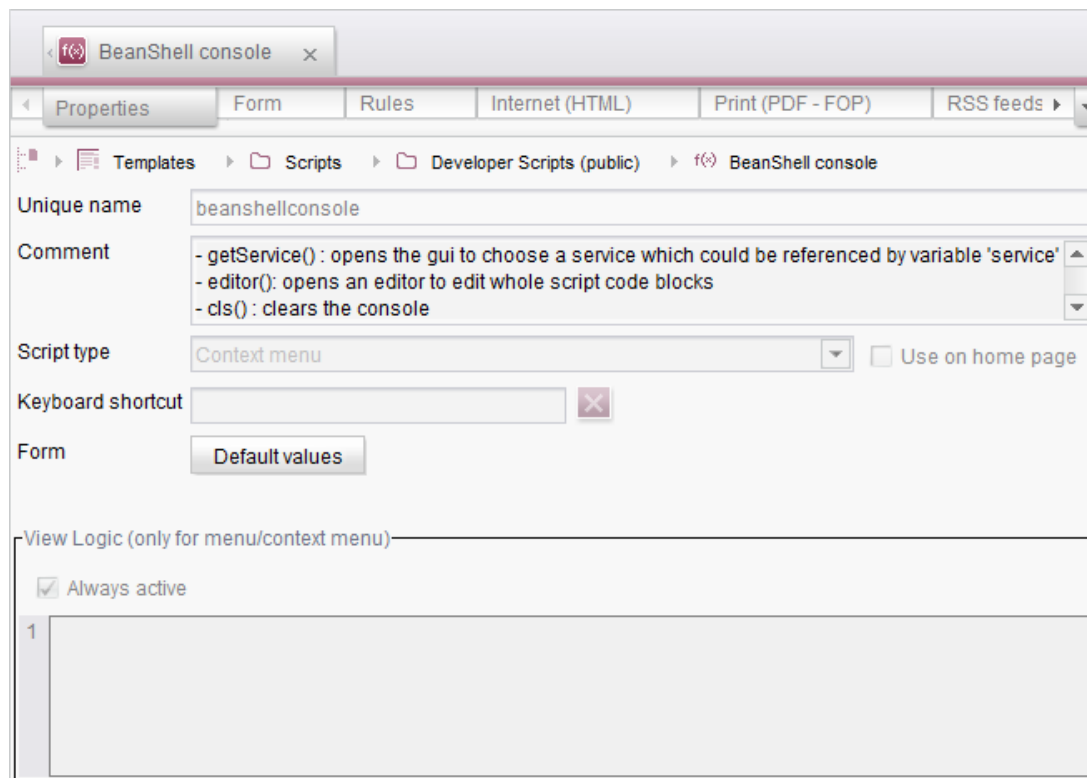
For more detailed information on developing scripts in BeanShell see "FirstSpirit Online Documentation" – "Scripting" chapter.

⁴ Additional information on this scripting language can be found at www.beanshell.org, which also provides a detailed manual (in English).



Examples of using scripts in workflows (see Chapter 4.8, page 180).

2.11.1 Properties tab



The screenshot shows a web-based configuration interface for a 'BeanShell console' script. The interface is titled 'BeanShell console' and has a tabbed navigation system with 'Properties' selected. The breadcrumb trail is: Templates > Scripts > Developer Scripts (public) > BeanShell console. The configuration fields are:

- Unique name:** beanshellconsole
- Comment:** - getService(): opens the gui to choose a service which could be referenced by variable 'service'
- editor(): opens an editor to edit whole script code blocks
- cls(): clears the console
- Script type:** Context menu (dropdown menu) Use on home page
- Keyboard shortcut:** (empty text field) [X]
- Form:** [Default values button]

Below the configuration fields is a section titled 'View Logic (only for menu/context menu)'. It contains a checked checkbox 'Always active' and a large empty text area for logic, with a line number '1' on the left side.

Figure 2-68: Scripts – "Properties" tab

Unique name: Script reference name.

Comment: An optional comment that describes the script in more detail can be entered here.

Script type: The context where the script is to be run can be configured here.

- **Template:** The script can be called and run in a template using `$CMS_RENDER(script:..)$`, e.g. for rendering specific content for the PDF presentation channel:



```
<fo:table table-layout="fixed" width="170mm">
  $CMS_RENDER(script:"fotablecolumns", colWidth:set_cw, colNumbers:set_cn) $
  <fo:table-body>
    $CMS_VALUE (#content) $
  </fo:table-body>
</fo:table>
```

- **Menu:** The script can be run using the "Tools" – "Execute script" menu.




Scripts of "Menu" type are displayed in the "Actions" WebEdit menu.

- **Context menu:** The script can be called and run on a specific element in FirstSpirit JavaClient's tree view using the context menu.
- **Uninterpreted:** The script is not checked against BeanShell syntax when saved. This allows HTML syntax to be saved (such as for displaying list elements). **These scripts should no longer be used.**

The corresponding templates are to be converted to format templates in projects that use uninterpreted scripts. For this reason, it will no longer be possible to save scripts of this type until the scripts have been converted to another script type. This type is no longer available for newly created scripts.

Use on entry page: This option can be enabled for scripts of "Menu" type. Then, depending on the settings in the "Display logic" area (see below), this script is displayed on the project entry page in the "My actions" area and can be run directly when clicked.

Keyboard shortcut: A unique keyboard shortcut can be defined for a script in this field. In this case, the script does not have to be run using the context menu or the "Tools" menu, instead it can be called directly using the defined keyboard shortcut. The cursor has to be inside this field to define a new keyboard shortcut. Then entering the desired key combination using the keyboard is all that is needed. The input is then applied in the input field. Text input is not possible. To change the keyboard shortcut, reposition the cursor in the field and then select the new key combination. Press the  icon to delete the defined keyboard shortcut for the script.





Keyboard shortcuts can only be used for scripts of "Context menu" or "Menu" type.

Display logic (only for menus/context menus): Scripts can be displayed or hidden depending on specific properties using display logic (the same as for workflow display logic, see Chapter 4.5.2, page 151). For instance, a script of "Context menu" script type can only be displayed if the context menu is called on a page reference in the site store:

```
#!/Beanshell
import de.espirit.firstspirit.access.store.sitestore.PageRef;
e = context.getStoreElement();
return (e instanceof PageRef);
```

Always active: The "Always active" checkbox can be unchecked if the display logic is to be disabled. In this case, the script is always displayed, regardless of the display logic. The stored display logic is no longer evaluated, but it remains stored and can be reenabled by unchecking the checkbox.

2.11.2 Form tab

As with page and section templates, individual input components that can be called during a script's runtime can be defined on the "Form" tab. The values of the input components can be returned to the script for processing (the same as for form support in workflows, see Chapter 4.4, page 147).

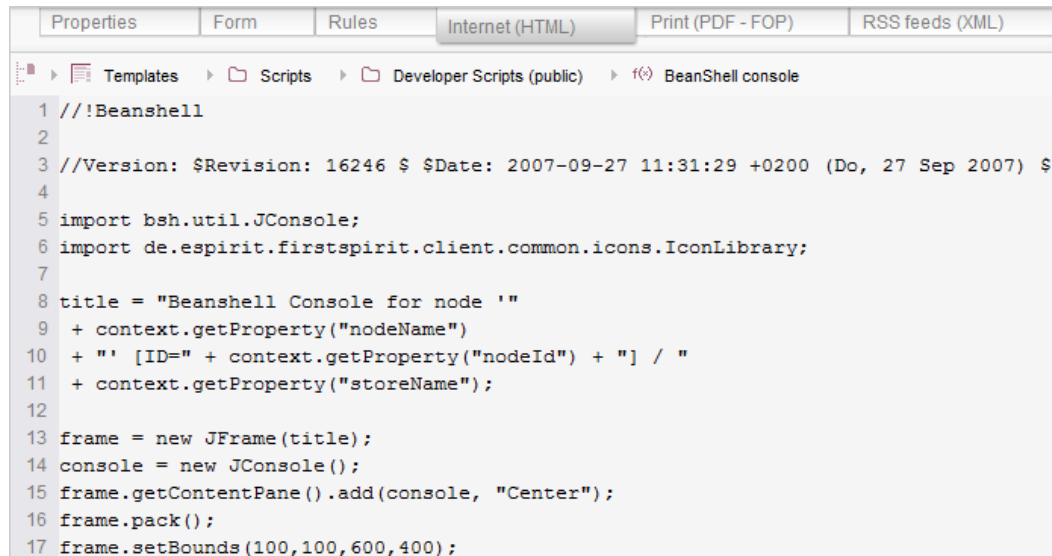
The addition of input components on the "Form" tab has been simplified by code completion (see Chapter 7, page 240).

With FirstSpiritVersion 5.0, line numbers are indicated at many places in the Template Store as a reading aid. They can be shown or hidden in JavaClient using the keyboard shortcut Ctrl + L.

*A listing of all of the available input elements can be found in the "FirstSpirit Online Documentation". The input elements are explained with all of their attributes and a schematic example under the menu item **Template development – Forms**.*



2.11.3 Template sets tab



The screenshot shows a web application interface with a navigation bar at the top containing tabs for 'Properties', 'Form', 'Rules', 'Internet (HTML)', 'Print (PDF - FOP)', and 'RSS feeds (XML)'. Below the navigation bar is a breadcrumb trail: 'Templates > Scripts > Developer Scripts (public) > BeanShell console'. The main content area displays a BeanShell script with line numbers 1 through 17. The script starts with '1 //!Beanshell' and includes comments and code for creating a Java Swing window titled 'Beanshell Console for node ' + context.getProperty("nodeName") + ' [ID=' + context.getProperty("nodeId") + '] / ' + context.getProperty("storeName");'. The code uses 'import bsh.util.JConsole;' and 'import de.espirit.firstspirit.client.common.icons.IconLibrary;'. It then creates a 'JFrame' and a 'JConsole' object, adds the console to the frame's content pane, packs the frame, and sets its bounds to (100, 100, 600, 400).

Figure 2-69: "Scripts" presentation channels

The BeanShell source code is defined in the presentation channels where the script is to be run. Specifying the character string `//! Beanshell` in the first line of a script causes the system to interpret the following source text as BeanShell script.

For examples of script development in workflows see Chapter 4.8, page 180.

For general information on script development in FirstSpirit, see "FirstSpirit Online Documentation".

With FirstSpiritVersion 5.0, line numbers are indicated at many places in the Template Store as a reading aid. They can be shown or hidden in JavaClient using the keyboard shortcut `Ctrl + L`.



2.12 Database schemata

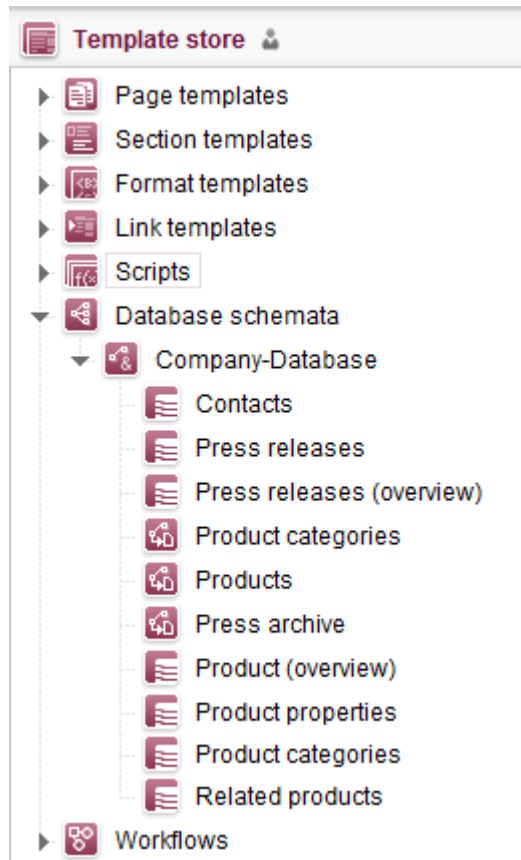



Figure 2-70: Template store tree view – Database schemata

FirstSpirit has high-performance mechanisms for connecting databases (see Chapter 3, page 119).

A graphical schema editor in the template store can be used to create and modify database tables (see Chapter 2.12.1, page 97), define templates for maintaining and displaying data records (see Chapter 2.12.4.1, page 111) and formulate queries for filtering data records (see Chapter 2.12.5.1, page 114). To accomplish this, a database abstraction layer that maps the universal FirstSpirit content type system to the specific database system to be used has been implemented by FirstSpirit (see Chapter 3, page 119).



2.12.1 New: Create schema

 Create new schema Ctrl+N

Which data are saved in which form and how these data relate to each other is defined in a database schema. The graphical editor in FirstSpirit JavaClient can be used to model database tables with the associated columns and relationships between individual tables (see Chapter 2.12.1, page 97).

Creating a new schema in JavaClient generates – in addition to adding a schema node below the "Database schemata" root node – a new database (see Figure 2-71 "Database_1") or a new database schema (see Figure 2-71 "Schema") in the database configured for the associated project. If the project administrator configures the default database for the project, a new database would be created in the default database (Derby) using the "New – Create schema" context menu. (The behavior depends on the configuration of the database layer – see "FirstSpirit Manual for Administrators" for setting "No schema sync".) The editor receives access to the database via the respective tables in FirstSpirit's content store and can import content that is written to the database into the associated tables (if the database was not defined as "write-protected" by the project administrator) (see Chapter 3, page 119).



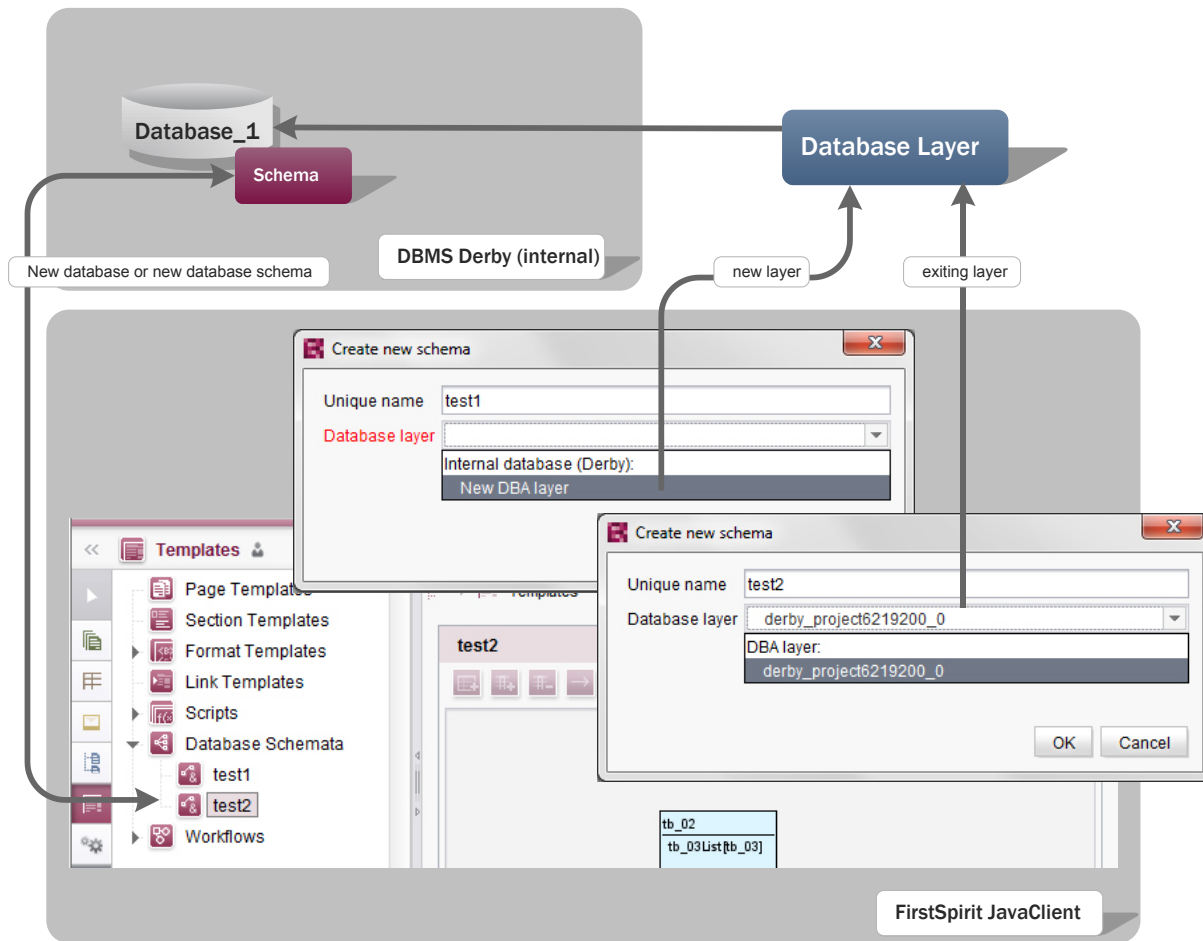



Figure 2-71: Creating a new schema

If a new schema is created for a content source in FirstSpirit, the database where it is to be stored for productive operation and the rights for the DBMS account used by FirstSpirit in productive operation should be decided in advance. Converting the layer types is not readily possible later on (see "FirstSpirit Manual for Administrators"). In case of doubt, a separate standard layer should be created for each FirstSpirit schema (see Chapter 3.2, page 121).

 *We generally recommend carrying out development in an environment matching production operation. In particular, the Derby DBMS contained in FirstSpirit is not suited for production operation and, therefore, should only be used for tests.*

A database layer has to be selected in addition to the (language-dependent) display and reference name to generate a new schema.



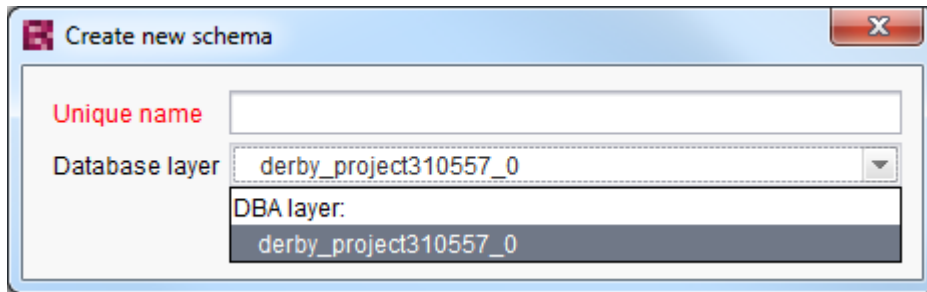


Figure 2-72: Creating a schema

A detailed description for reference and display names can be found in Chapter 2.2.1 on page 23.



The reference name defined for a schema in a FirstSpirit project does not correspond to the physical name of the schema in the database. The physical name is specified automatically based on the database – for instance, according to the following pattern for the default database (Derby): `derby_projectID_schemaID`

Database layer: An existing database layer for a database, where the individual database tables for this schema are to be saved, has to be selected in the "Database layer" field. What are known as "DBA layers" and (optional) "Standard layers" are available for selection in the process:

- **Standard layer:** A standard layer has to be selected when working directly on an existing database schema (see Figure 2-75). This layer can be used in multiple FirstSpirit projects that all write to the same database or read content from the same database. Only one of the respective participating projects should have write permission for the database so that overlap does not occur when using a standard layer (see Chapter 3.2, page 121).
- **DBA layer:** If a DBA layer is selected, a separate schema or database is created for each respective project upon creation (during the first sync). It is impossible to overlap when writing content in this case (see Chapter 3.3, page 122).



Please contact the project or system administrator if you have questions regarding the database.

If there are *no database layers* available, a new one can be created directly using the "Create schema" dialog. The entry "New DBA layer" is then shown instead of a database layer in the



"Database layer" drop-down list (see Figure 2-72). The new layer is generated in addition to the schema or database upon confirming the dialog (see Figure 2-71).

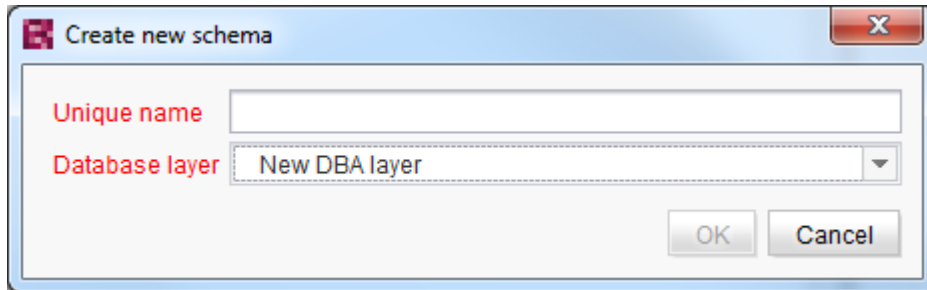
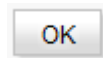
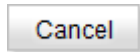


Figure 2-73: Creating a schema – When a layer is not available



Clicking this button connects the new, empty schema in the tree view and a schema or a database is generated in the configured DBMS. The schema can be edited further using the graphical editor.



Clicking this button cancels the operation. A new schema is not created.

A new schema can also be created via importing an export file from another FirstSpirit project (see Chapter 2.3.3.2, page 41).



2.12.2 New: Creating a schema from a database

Create schema from database

An already existing schema can be taken over in the new schema node from a(n external) database using this function.

Instead of generating an empty schema node (see Chapter 2.12.1), a new schema node is created in the FirstSpirit project based on the pre-existing tables and relationships of a database. The new schema is created from a database using the "Generate schema from database" context menu entry.



The structure and contents of an external database may not be changed. In contrast to internal databases, only read access is possible for external databases, not write access. The restrictions "No schema sync" and "Write-protected" have to be enabled by the project administrator for this. In this case, content can be read out of an external schema and generated as a new schema node in FirstSpirit JavaClient. The contents can then be displayed (but not modified) in the content store using table templates.

For more information see "FirstSpirit Manual for Administrators".

Generating a new schema from an existing database in JavaClient inserts a new schema node underneath the "Database schemata" root node. In the process, an attempt is automatically made to transfer the existing tables and content from an existing database or from an existing database schema to the graphical schema editor (*for "Restrictions" see "FirstSpirit Manual for Administrators"*). If the project administrator configures an external Oracle database for the project, a schema based on the external database is generated using the "New – Generate schema from database" context menu. The associated tables are also taken over. Depending on the database configuration, the editor receives read access to the database via the content store and can read out and generate content from the respective tables.



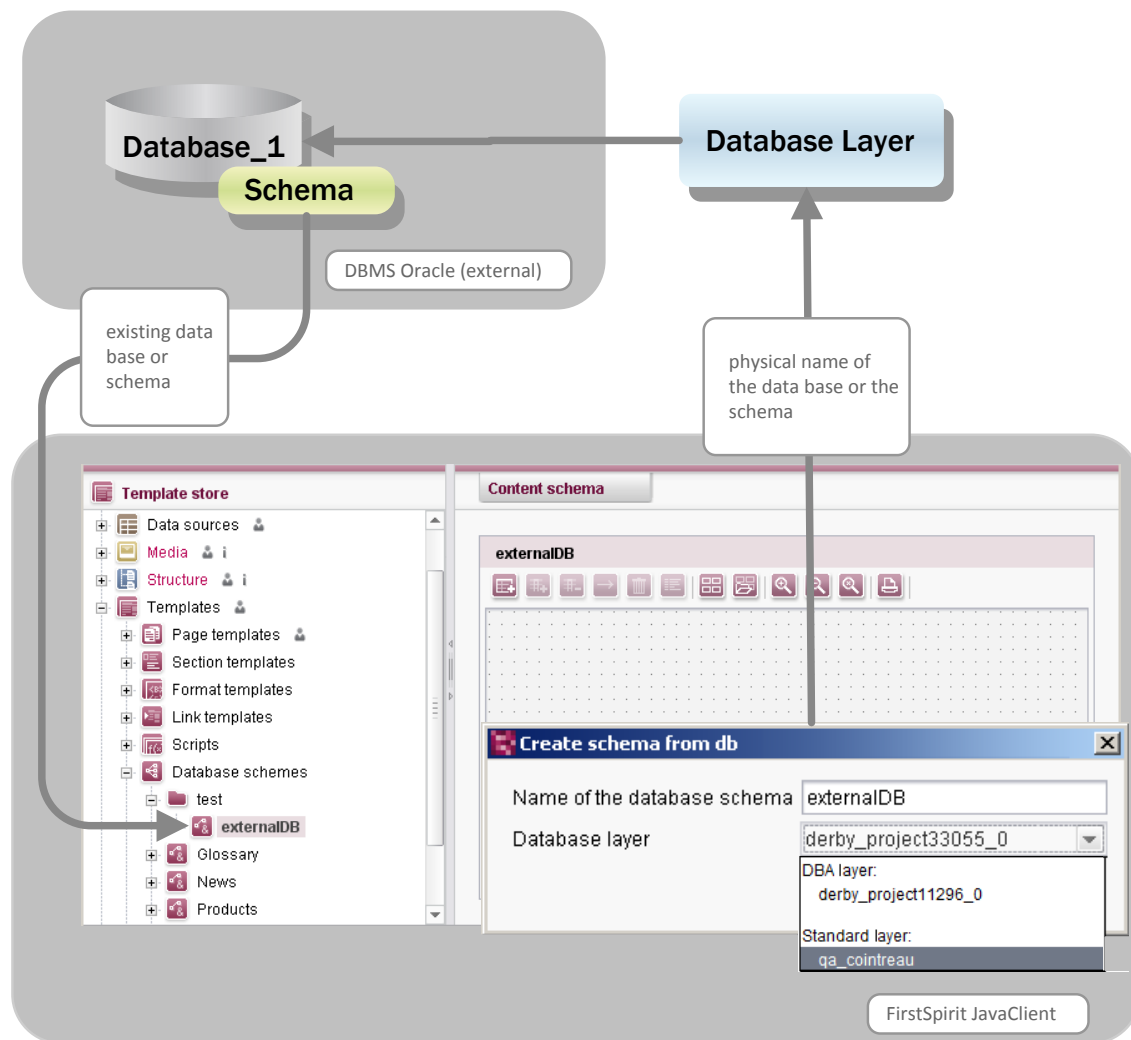


Figure 2-74: Generating a schema from an external database



The following inputs are required to create a new schema:

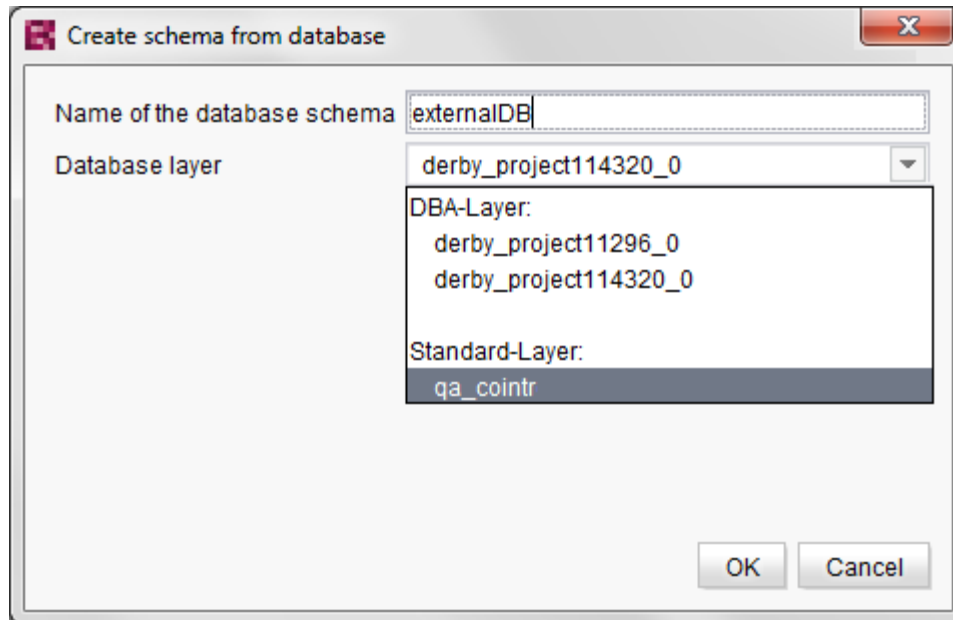


Figure 2-75: Generating a schema from a database

Name of the database schema: The name for the database schema has to be specified in this field. In contrast to creating a new, empty schema, the name is *not* freely selectable. Instead, it has to correspond exactly to the physical name of the database schema (or the database).



If a name is selected that is not available in the respective database, then an empty schema node is created. In this case, no content (e.g. tables) can be taken over from the selected database.

Database layer: An existing database layer has to be selected in the "Database layer" field (see Chapter 2.12.1, page 97).

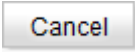


*The option to select an external database is only available if the project administrator configured access to an external database for the project in the project properties.
Please contact the project or system administrator if you have questions regarding the database.*





Clicking this button connects the new schema and the associated tables in the tree view and it can be edited further using the graphical editor.



Clicking this button cancels the operation. A schema is not created.

2.12.3 The FirstSpirit schema editor

A graphical editor for editing a schema in FirstSpirit JavaClient is available on the right side of the window. The editor can be used to create the desired database schema. Depending on the configuration, a schema can access existing database structures or create new table structures in an existing database.

The editor is operated either using the editor's tool bar or using a context menu which can be called up at any position in the editor by right-clicking.

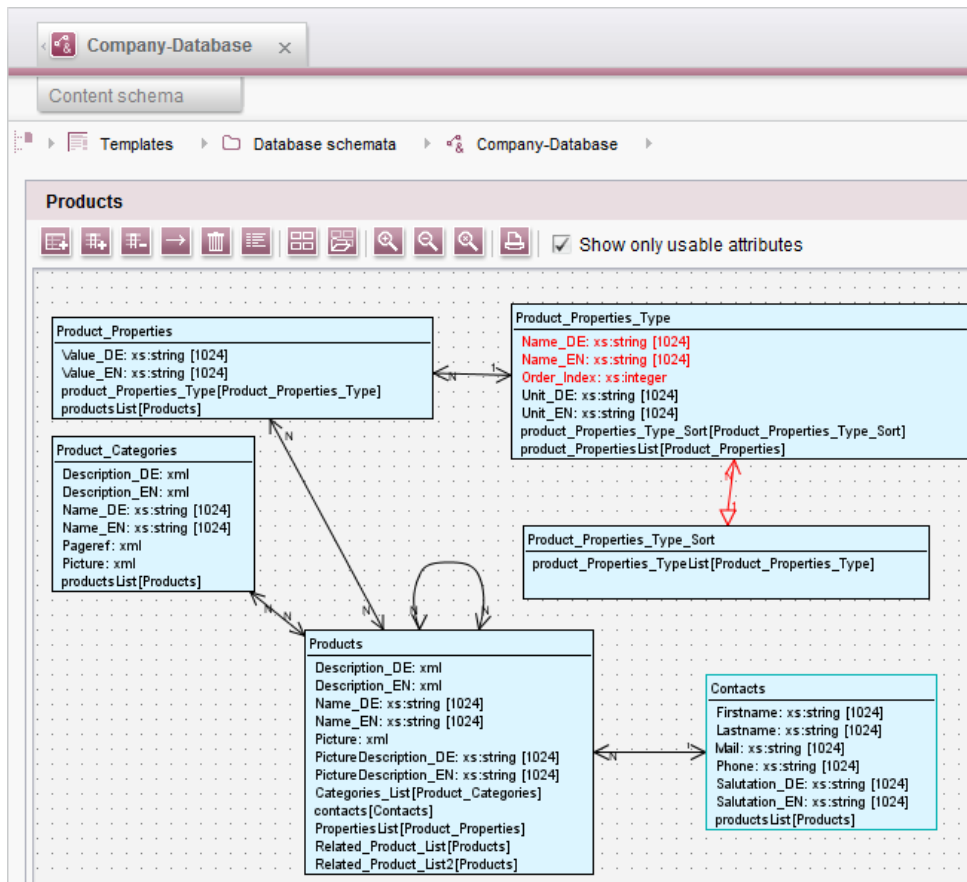


Figure 2-76: Database schema editor



Figure 2-76 shows the database schema of the product database of the "Mithras Energy" test project as an example. Among other aspects, the tables for products (*Products*), product categories (*Product_Categories*), product properties (*Product_Properties*) and contacts (*Contacts*) can be seen. In this instance, products and contacts, for example, are linked via a 1:N relationship, and products and product categories via an M:N relationship.



Create table: with this button, a new table can be inserted in the database schema. The following window opens:

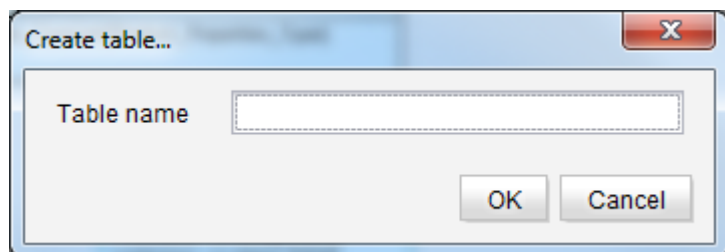


Figure 2-77: Creating a table

Table name: A unique name has to be entered for the database table in this field.



Create column: with this button, a new column is added to the activated table. The following window opens:

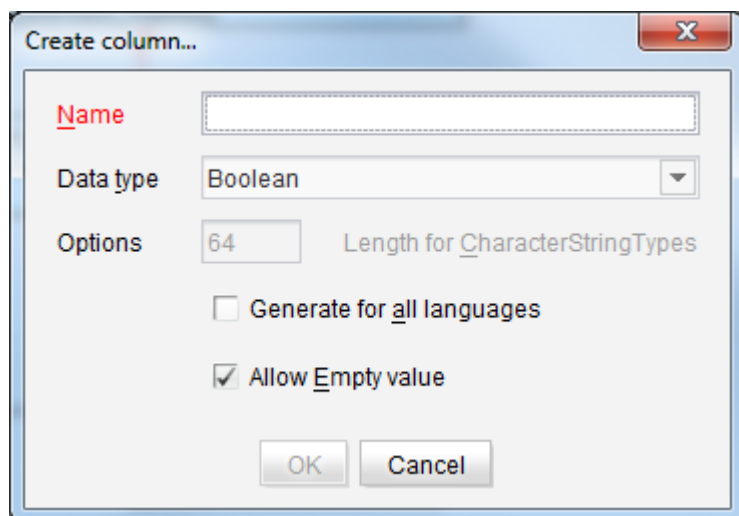


Figure 2-78: Creating a column

Name: The column name has to be entered in this field. As long as this field is empty, *Name* is



shown in red and the new column cannot be saved.

Data type: The desired data type can be selected for the new table column using this combobox.

Boolean: This data type allows two values: `true` or `false`. In the schema editor, this data type receives a `xs: boolean`.

Date: This data type is used for date values. In the schema editor, this data type receives a `xs: date`.

Double: This data type enables entry of floating-point numbers. In the schema editor, this data type receives a `xs: decimal`.

FirstSpirit editor: This data type enables the use of DOM editors. The maximum character length is 65535. In the schema editor, this data type receives a `xml`.

Integer: This data type is used for whole numbers. In the schema editor, this data type receives a `xs: integer`.

Long: This data type is also used for whole numbers, but the value range is larger than the range for the Integer data type. In the schema editor, this data type receives a `xs: long`.

String: This data type is used for character strings. In the schema editor, this data type receives a `xs: string`. The number of maximum characters allowed can be specified for this data type.

Options: The maximum character length for the String column type has to be specified in this field. The respective value is shown in square brackets after `xs: string`.

Generate for all languages: This option enables language-dependent input of the values by the editor. If the checkbox is checked, a single column is generated for each attribute in every language. This makes sense if the attribute terms differ based on language. The columns receive a corresponding language code for each language in the process.

Allow empty values: By activating this option, the editor is allowed to create a new data record without putting a value in this column. If empty values are not allowed (i.e. input is mandatory), the column name is shown written in red in the database schema model.



Remove column: The individual columns of a table of the database schema are removed using this function. The desired column can be selected from the combobox in the following dialog:



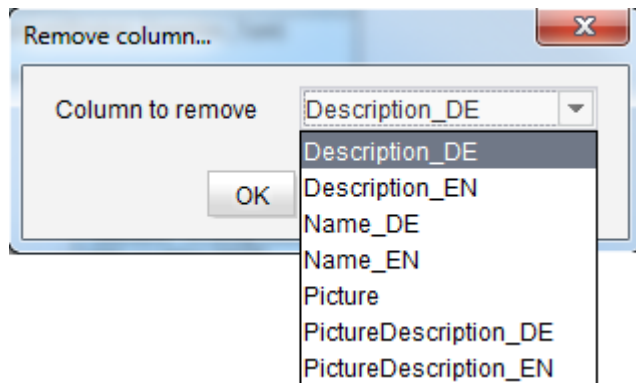



Figure 2-79: Removing a column

 Create foreign key relationships: A relationship can be created between the activated table and another table using this button.

Creating a relationship should be done according to the example in the product database shown in Figure 2-76. We want to create the relationship between the tables named Products and Contacts: A contact person can assign multiple products, they are in other words in a 1:N relationship.

Initially, the Contacts table and (while pressing the shift key) the Products table have to be activated (activated tables change their frame color). If the button *Create relationships* is now selected, the first step of the procedure appears, in which the type of relationship has to be established. The two tables should be in a 1:N relationship.

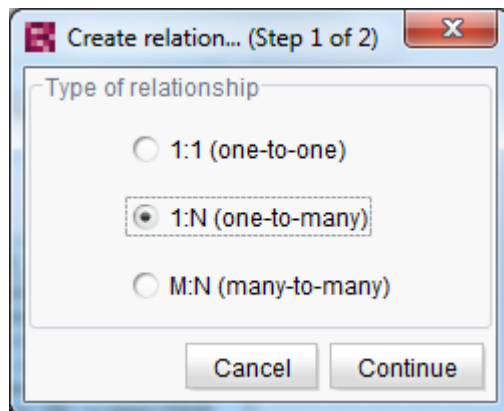


Figure 2-80: Create relationship– Step 1

The second window of the relationship appears as follows (however, the appearance can vary depending on the selection in the first window):



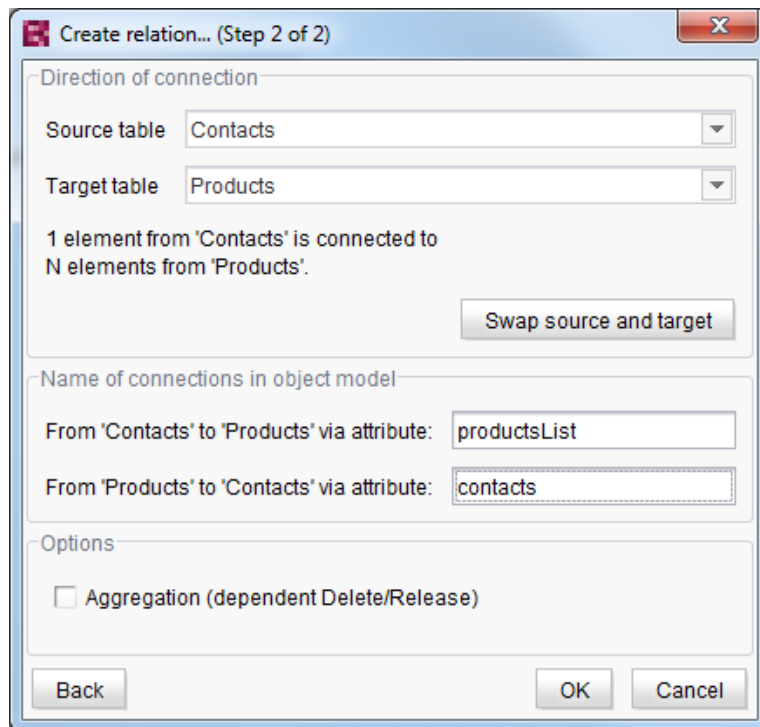





Figure 2-81: Create relationship– Step 2

The direction of a connection is assigned based on the order the tables are activated. As a result, 1 element from the Contacts table is connected to N elements from the Products table. If the tables are accidentally activated in the wrong order, it can be reversed with the buttons *Exchange source and target*. The additional information in this window can usually be taken over the same way the system suggests. The names that are specified in the "Names of connections in object model" area are used during later use to trace data inventories based on their relationships.

 **Delete elements:** The activated table can be deleted from the database schema with this button. It is deleted directly, without a confirmation prompt; unsaved files are lost in the process and cannot be restored.


 **Properties:** The name of the activated table can be shown using this button.

 **Assign automatically:** The tables shown are automatically assigned in the editor by activating this button.


 **Load saved assignments:** Changes to the assignment of the tables in the schema can be




undone using this button. The last saved assignment is again shown.

 Enlarge view: With this button, the elements of the database schema are shown enlarged.

 Shrink view: With this button, the elements from the database schema are shown smaller.

 Normal view: With this button, the elements of the database schema are once again shown at their original size.

 Print: With this button, the database schema can be printed. Next, the following print preview window opens:

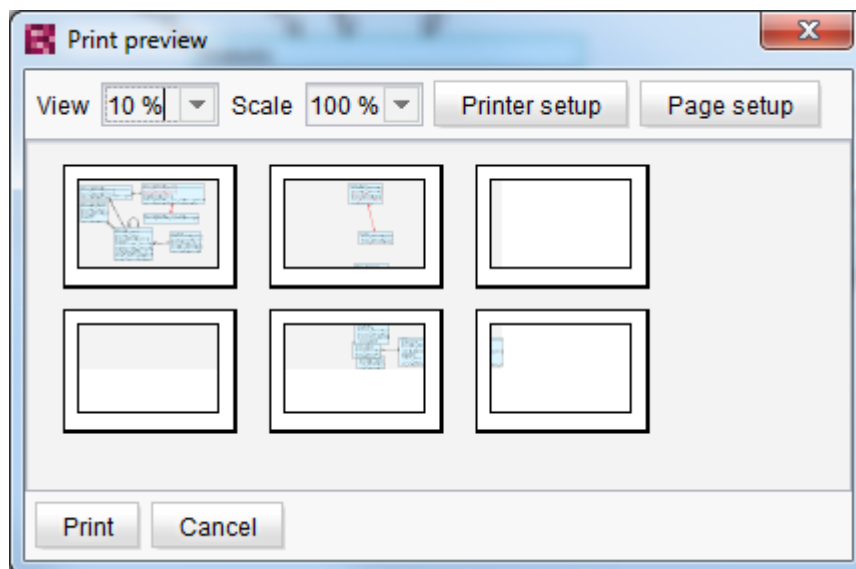


Figure 2-82: Print preview

View: This combobox can be used to affect the preview size of the database schema. Possible zoom levels are 10%, 25%, 50% and 100%.

Scaling: The database schema is printed smaller as needed. Scaling levels 10%, 25%, 50% and 100% are possible.

Set up printer: Opens the dialog for the printer settings.

Set up page: Opens the dialog for the page settings.



The print job is started with the current settings using the  button.

Show only usable attributes

Only show usable attributes: If this is checked, then all of the attributes of a table that the editor can fill with content are hidden.

In addition to functions of the icons, additional functions can be called up using the context menu:

Rename table/column: A new name can be specified for an existing table or for an existing column of a table using this function. A window appears where the desired table or column can be selected and a new name can be specified. When renaming it is important to note that table templates and queries based on this table or column have to be adapted.

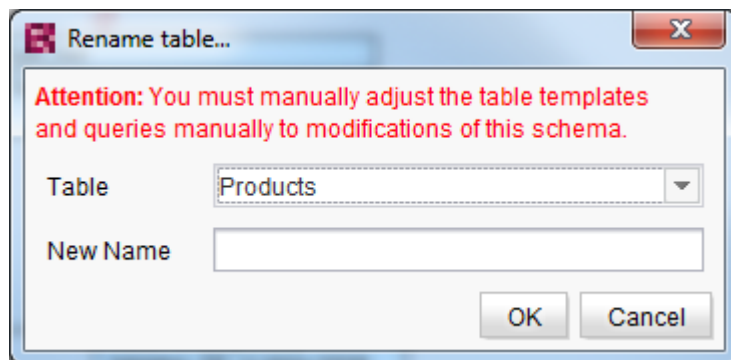


Figure 2-83: Renaming a table

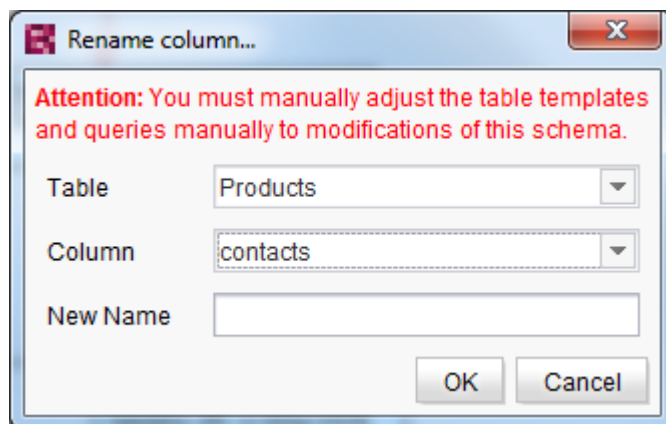


Figure 2-84: Renaming a column



2.12.4 Table templates

A table template has to be created under the schema for each input table in the database model. Which input components the editor can use to import data into the correspond tables is defined in this table template.

2.12.4.1 Table templates – Preview, properties and form tabs

The preview, properties and form tabs for table templates are identical to the tabs for page templates with the same names and can be edited in the same way.

You can find information on the tabs in Chapter 2.5.1 (page 56) to Chapter 2.5.3 (page 59).



The option "Hide template in selection list" (see Chapter 2.5.2) is not available for table templates.

Important for using table templates in WebClient

If a table template is to be able to be used in WebClient, the checkbox "Usable in WebEdit" must be activated on the "Properties" tab. Moreover, an adequate preview page must be set in the tab "Properties" (cf. Chapter 2.5.2) for a correct display of the data records in WebClient.



2.12.4.2 Table templates – Mapping tab

This area defines which input components are used to insert data records into the database tables. Each input component (defined on the Form tab) is assigned a table column in the process.

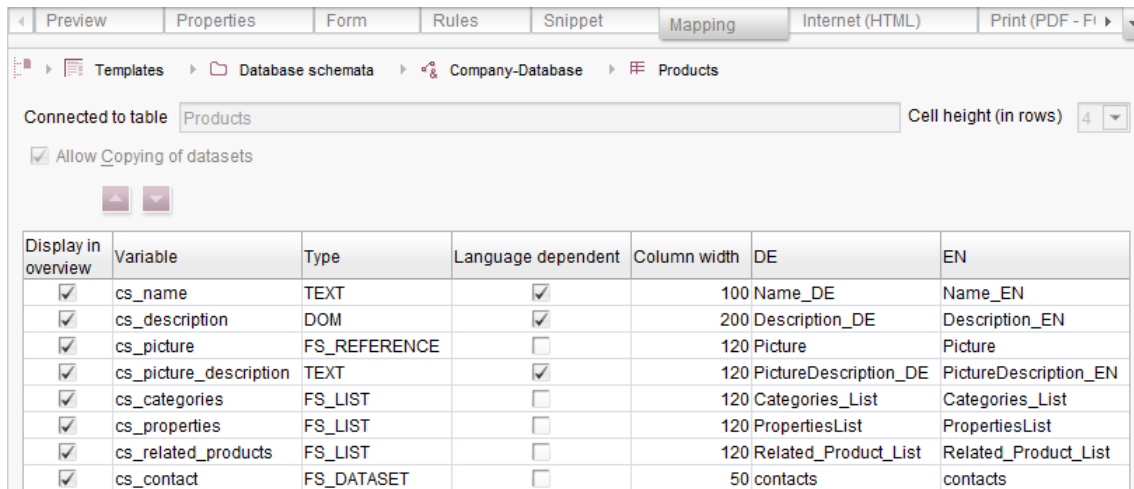



Figure 2-85: Table template – "Mapping" tab

Connected to table: The table that the mapping settings apply to is displayed in this field.

Cell height (in rows): Data records can be displayed over multiple rows in the content source later on (see *FirstSpirit Manual for Editors*, Chapter 5). This combobox can be used to configure how many rows a cell or data record is to have in the data overview (maximum of 10). This could allow image thumbnails to be displayed in the overview, for instance.

Allow data record copying: If this checkbox is checked (default setting), existing data records can be copied by the editor in the associated data source. If the checkbox is unchecked, only "blank" new data records can be created; the  icon is disabled.



Each row of the list corresponds to a column in the data overview of the associated content source. Clicking the icons moves a highlighted row up or down or the associated column in the data overview left or right by one position. This can be used to move more important columns farther to the left. The order can be changed manually by the editor; upon updating the view, however, the order reverts back to the setting on this tab. In contrast, the order selected here has no effect in data entry.

Display in overview: This column can be used to hide table columns from the data overview in



the respective content source by unchecking the checkboxes, e.g. to improve the overview if there are too many columns. In contrast, hiding has no effect in data entry.

Variable: This column contains the name of the variable as it was defined in the table template's form (Chapter 2.12.4.1, page 111).

Type: The type of the input component for the respective variable is specified in this column.

Language-dependent: If the input component on the Form tab is defined as being multi-lingual, the this is shown by a check in this column.

Column width: In this field, the width of the column is specified in pixels as it will be displayed in the content store later.

Language (DE/EN): The table column where the content of the input element is to be transferred to is selected in this field. There is a separate column for each project language. The same table column has to be selected for each language if a language-independent input component is involved. A separate table column where the value is transferred to has to exist for each language for language-dependent input components.

2.12.4.3 Table templates – Template sets tab

The appearance that individual data records are intended to assume later on the website or in other presentation channels imported using this table template is determined using this tab.

The tab for table templates is identical to the tab of the same name for page templates and can be edited in the same way.

For information on the "Template sets" tab see Chapter 2.5.4, page 60.



2.12.5 Queries

Multiple queries can be created for each database schema to limit the number of data records for later output. The conditions a data record has to fulfill to be recorded in the result list is determined in these queries.

2.12.5.1 Query – Conditions tab

The desired filter criteria for a query can be defined on the "Conditions" tab using a graphical editor in what is known as Wizard mode. Multiple rules can be set in the process, which then affect the display of suitable data records on the "Result" tab.

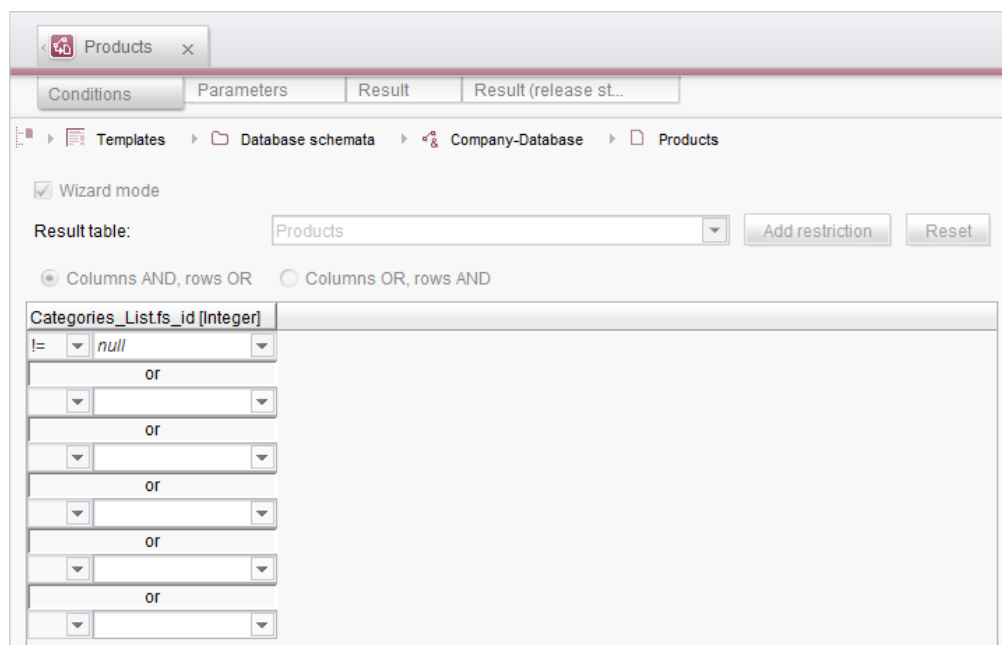


Figure 2-86: Query – "Conditions" tab (Wizard mode) (new Look&Feel)

Wizard mode: If this option is disabled, the source code for the selected query is displayed in an editor and can still be modified as needed. A query can also be programmed directly using this editor.





Tags and parameters that can be used for directly programming queries can be looked up in the FirstSpirit Online Documentation: Section "Query portion (QUERY)" in Chapter "contentSelect function" ("Template development" / "Template syntax" / "Functions" / "In header" / "contentSelect").

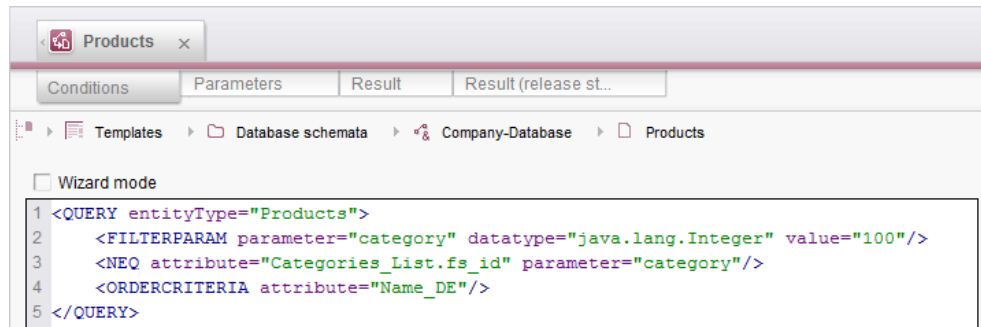


Figure 2-87: "Conditions" tab (no Wizard mode)

If changes that cannot be mapped in Wizard mode are made to the query, then the query is adjusted automatically (in the editor) as soon as Wizard mode is activated again.

Result table: A table from the FirstSpirit schema for restrictions to be made during output can be selected here. This field is deactivated once a selection has been made. The selection can only be removed by "resetting" the entire query (see below).

Add restriction: Activating this button adds a new condition. A window opens where a specific column of the selected result table can be selected as a new reference. The limiting condition can then only be set for this reference. The specific values that have to be met can be specified in the condition column in the lower window area. The desired comparison operator for the condition for this is selected in the field on the left. In the field on the right, either a specific comparison value can be entered or a parameter identifier for the comparison value can be specified. This parameter is then requested each time the query is run; as a result, the specific comparison value only has to be specified when the query is run.

Reset: All of the conditions and query results defined up to this point are deleted. A result table can be selected again. A confirmation prompt appears before changes are undone so that data is not deleted accidentally.

Columns AND, rows OR: If this option is selected, then the intersection of all column results is always output. The individual rows of a column condition are connected by an OR link in this



case.

Columns OR, rows AND: If this option is selected, then all of the combined results from all columns are output; duplicate data records are skipped in the process. The individual rows of a column condition are connected by an AND link in this case.

2.12.5.2 Query – Parameter tab

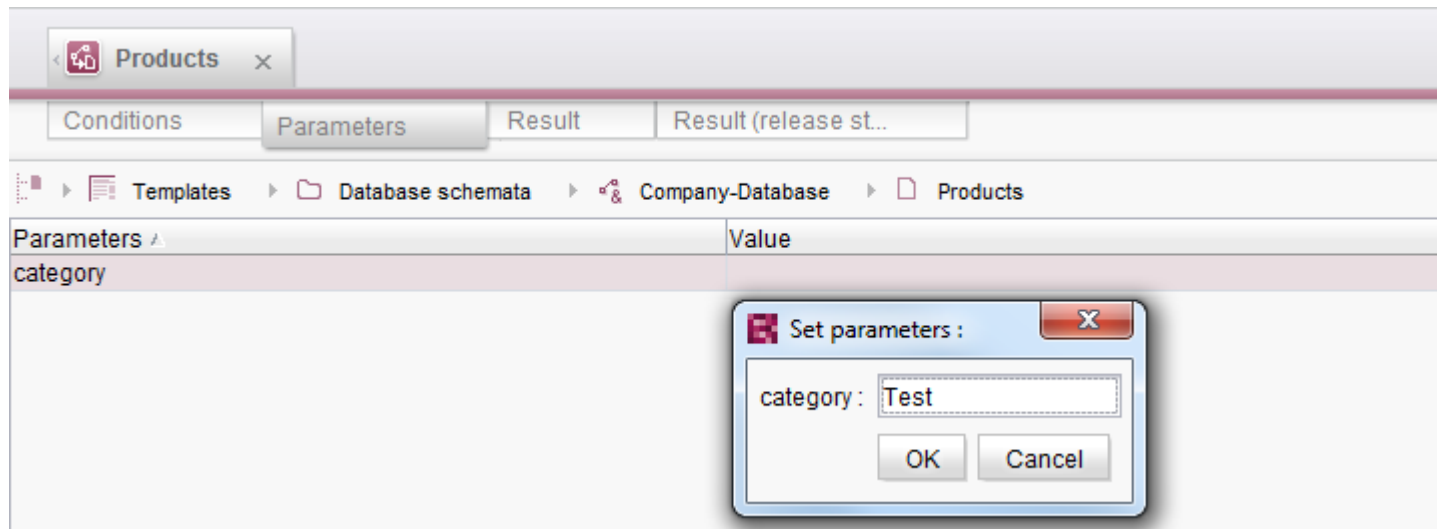


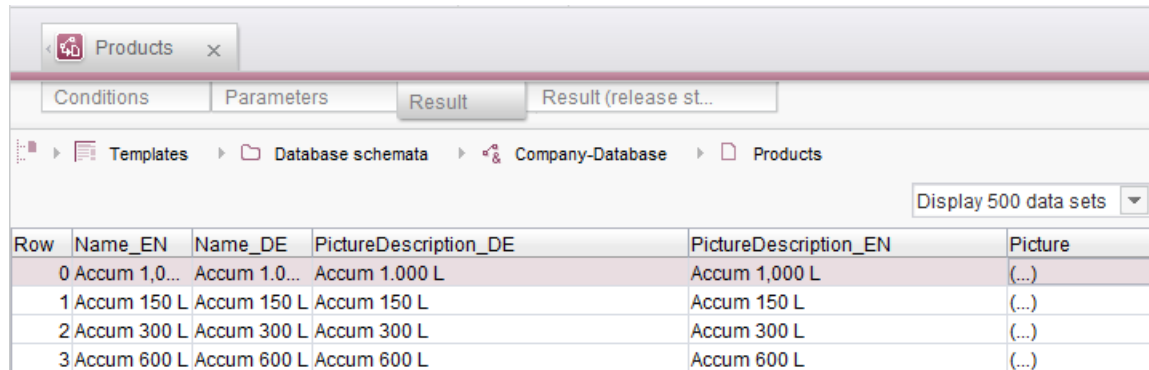
Figure 2-88: Query – "Parameter" tab (new Look&Feel)

All of the parameters that can be used in this query are listed in this area. The parameter can be set as needed even at this point in the **Value** column. This means the corresponding query parameters are assigned values. These values are then used for the query each time it is run.



2.12.5.3 Query – Result tab

The result data records that result from the conditions in the query and query parameter value assignment are output in this area.



Row	Name_EN	Name_DE	PictureDescription_DE	PictureDescription_EN	Picture
0	Accum 1,0...	Accum 1.0...	Accum 1.000 L	Accum 1,000 L	(...)
1	Accum 150 L	Accum 150 L	Accum 150 L	Accum 150 L	(...)
2	Accum 300 L	Accum 300 L	Accum 300 L	Accum 300 L	(...)
3	Accum 600 L	Accum 600 L	Accum 600 L	Accum 600 L	(...)

Figure 2-89: Query – Result tab

2.12.5.4 Query – Result (release) tab

The result data records that result from the conditions and query parameters in the query and are **in the release state** are output in this area. Thus the result quantity can differ from the list from the Result tab.



2.13 Workflows

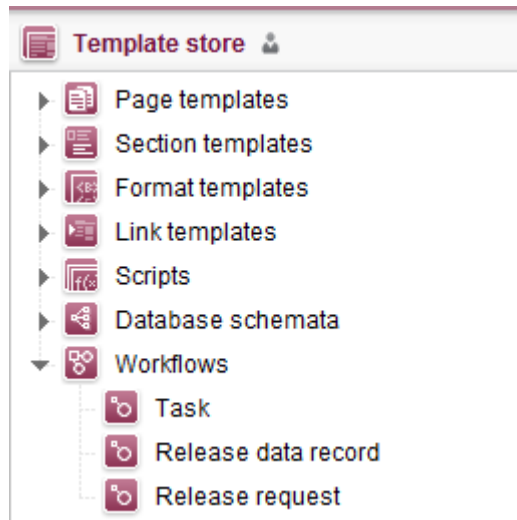


Figure 2-90: Template store tree view – Workflows

A workflow is a sequence of tasks that are completed in a fixed, predefined structure. Due date deadlines and groups of authorized persons can be defined for the respective tasks. Issuing a task and requesting release are workflows integrated into FirstSpirit.

For modeling, configuring and running workflows, see Chapter 4, page 126 ff.



3 Content sources in FirstSpirit

FirstSpirit has efficient mechanisms for connecting databases. The connected databases are identified as content sources in the editing environment. The data records managed in the content sources can be integrated into websites and edited seamlessly in FirstSpirit without leaving the editing environment (see Chapter 3.4, page 124).

A database connection is available for a majority of databases and is run using the JDBC driver provided by the database suppliers. Each database supplier implements a separate internal structure for managing data saved in the database server (DBMS). These internal structures, in conjunction with security and maintenance specifications, have implications on the form and configuration for connecting databases to FirstSpirit.



For more detailed information on connecting databases to FirstSpirit and configuring them, see "FirstSpirit Manual for Administrators", Chapter 4.8 "Database connection".

The following chapter is intended to support template developers with selecting the correct connection and to explain concepts for working with content sources in FirstSpirit JavaClient:

Chapter 3.1 defines the terms in use since, in relation to databases, the terms can have different meanings depending on the context being used.

Chapter 3.2 and 3.3 cover the layer types used in FirstSpirit for connecting databases. The layer type selection has various effects on later operation and requires substantial effort to change and should be considered carefully for these reasons.

The concept of content sources in FirstSpirit JavaClient is described in Chapter 3.4.



3.1 Terms

Many misunderstandings occur when dealing with connecting content sources to FirstSpirit due to the vast assortment of terms, some of which may have multiple meanings. The suppliers of databases to be connected often maintain their own terminology, which can overlap with FirstSpirit's. Therefore, terms used in this document will first be clarified here:

Layer: This term represents a connection configuration to a database management system (DBMS) in FirstSpirit. A layer can be assigned to multiple FirstSpirit schemata (see below) in FirstSpirit.

- **Standard layer:** This layer type includes an explicit definition of the used DB schema in the layer definition. In this case, all of the tables of FirstSpirit schemata that use this layer are saved in the specified DB schema. A FirstSpirit user cannot create any additional new schemata in a FirstSpirit project only assigned standard layers. Only a FirstSpirit administrator can add additional standard layers to a project. A standard layer should always be assigned to precisely one FirstSpirit schema. (see: Chapter 3.2, page 121)
- **DBA layer:** This layer type does not include any explicit definitions of the DB schema to be used. FirstSpirit automatically creates a separate DB schema for each FirstSpirit schema. This allows additional schemata to be created by FirstSpirit users as well. However, to do so, comprehensive DBA permissions (DBA = database administrator) are required for most DBMSs. (see: Chapter 3.3, page 122)


FirstSpirit schema: This term describes the structures and templates of content sources described in FirstSpirit. Thus, FirstSpirit schemata contain both tables and their foreign key relationships as well as templates for generation. The table structure and data records of FirstSpirit schemata are stored in a DBMS within a DB schema (see below). Each FirstSpirit schema is always assigned to precisely just one layer (see Chapter 3.2, page 121 and Chapter 3.3, page 122).

DB schema: This term describes the logical area within a database where tables can be stored (tablespace). Each table in this area has to have a unique name. In DBMS, the term "database" is also frequently used as a technical term. In the layer configuration, FirstSpirit simply calls this a "schema".



3.2 Standard layer

The standard layer (see also: Chapter 3.1, page 120) was used in earlier versions of FirstSpirit under the term "MultiProjectLayer". The mechanism for avoiding conflicts between identical table names in different FirstSpirit schemata is no longer present in the standard layer, however. The DBA layer was introduced as a replacement for the flexible assignment of FirstSpirit schemata to DB schemata (see Chapter 3.3).

 *If the standard layer is assigned to multiple FirstSpirit schemata, a conflict occurs in the FirstSpirit schemata if table names are identical since they are assigned the same table in the DB schema (see Figure 3-1).*

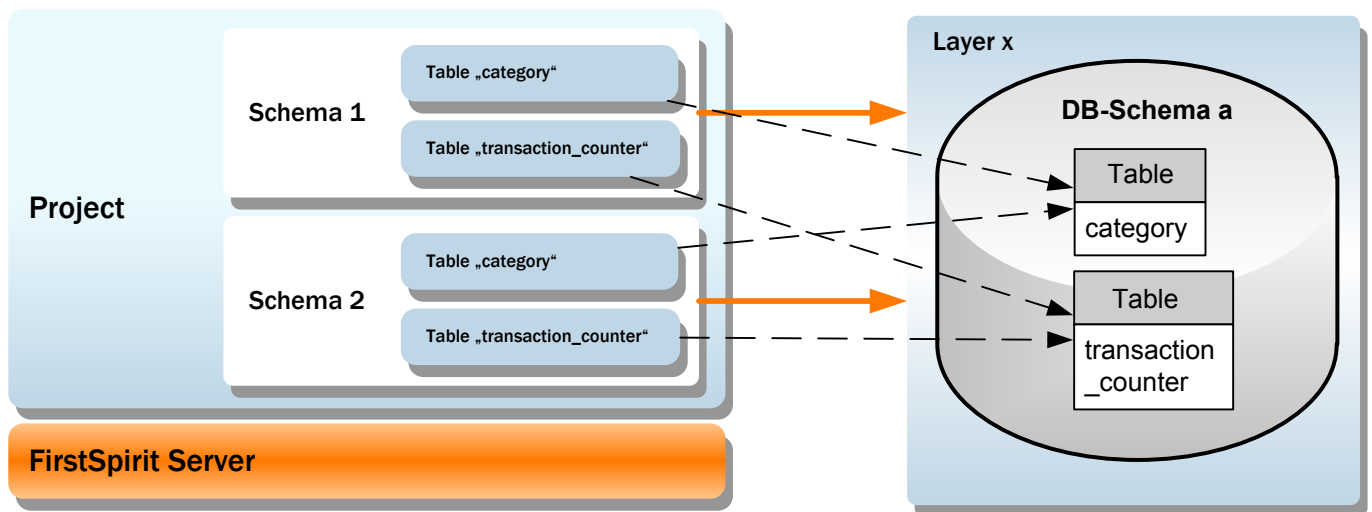


Figure 3-1: Problematic use of a standard layer

The "transaction_counter" system table is a special case in this context; a hidden version of it can be created for each FirstSpirit schema. FirstSpirit tries to resolve the conflict mentioned above by converting the tables into one table.


 *In any case, mixing two FirstSpirit schemata in one DB schema is not recommended. Standard layers should **always** be assigned to just one FirstSpirit schema.*

Figure 3-2 shows the correct use of standard layers. A separate standard layer is created for



each FirstSpirit schema and thus a separate DB schema where the associated tables are stored.

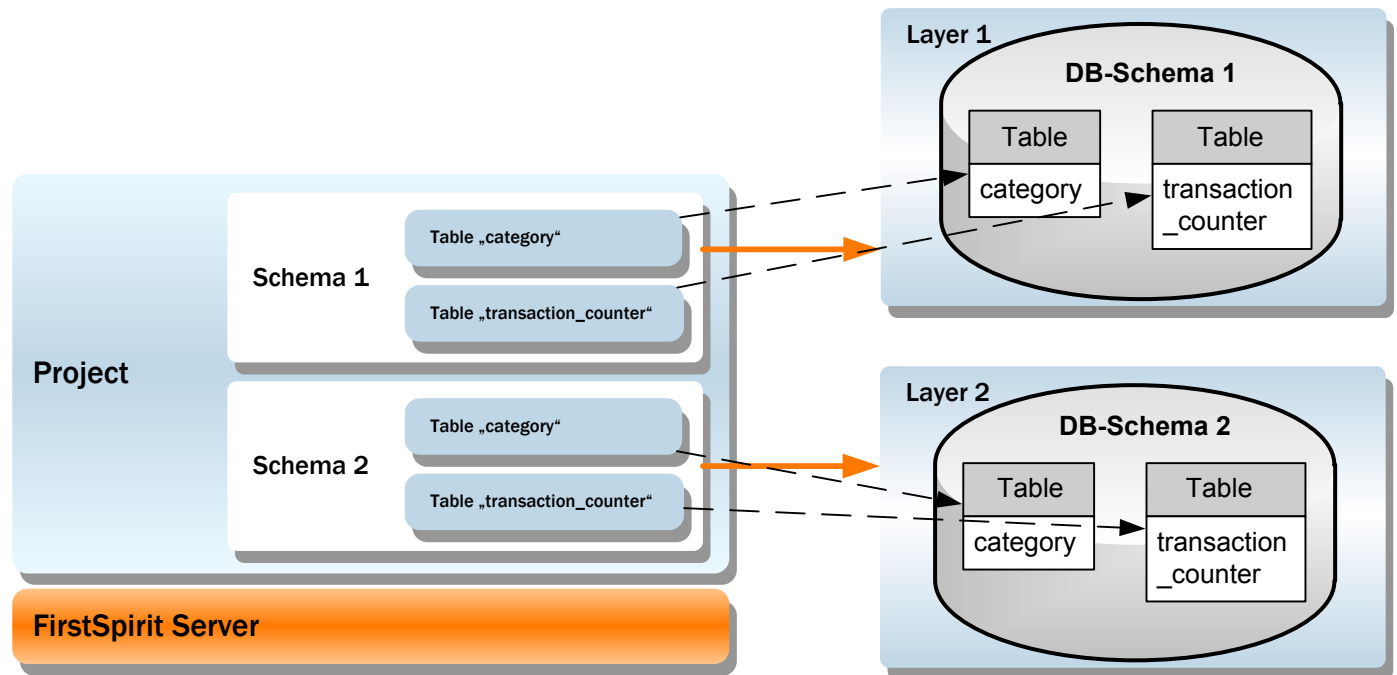


Figure 3-2: Correct use of separated standard layers

3.3 DBA layer

The DBA layer was used in earlier versions of FirstSpirit under the name "SingleProjectLayer". It was introduced for being able to create FirstSpirit schemata in a project even without intervention by a database administrator.

In contrast to the standard layer, no explicit DB schema for saving tables is specified in the layer definition for a DBA layer. FirstSpirit independently creates the DB schemata belonging to the FirstSpirit schemata in the DBMS. The name of the DB schemata is composed of the schema and project ID in the process (see Chapter 2.12.1, page 97).

The user specified in the layer has to have the permissions to create DB schemata in the DBMS to use a DBA layer. In many DBMSs, this is only possible with permissions similar to a database administrator's.



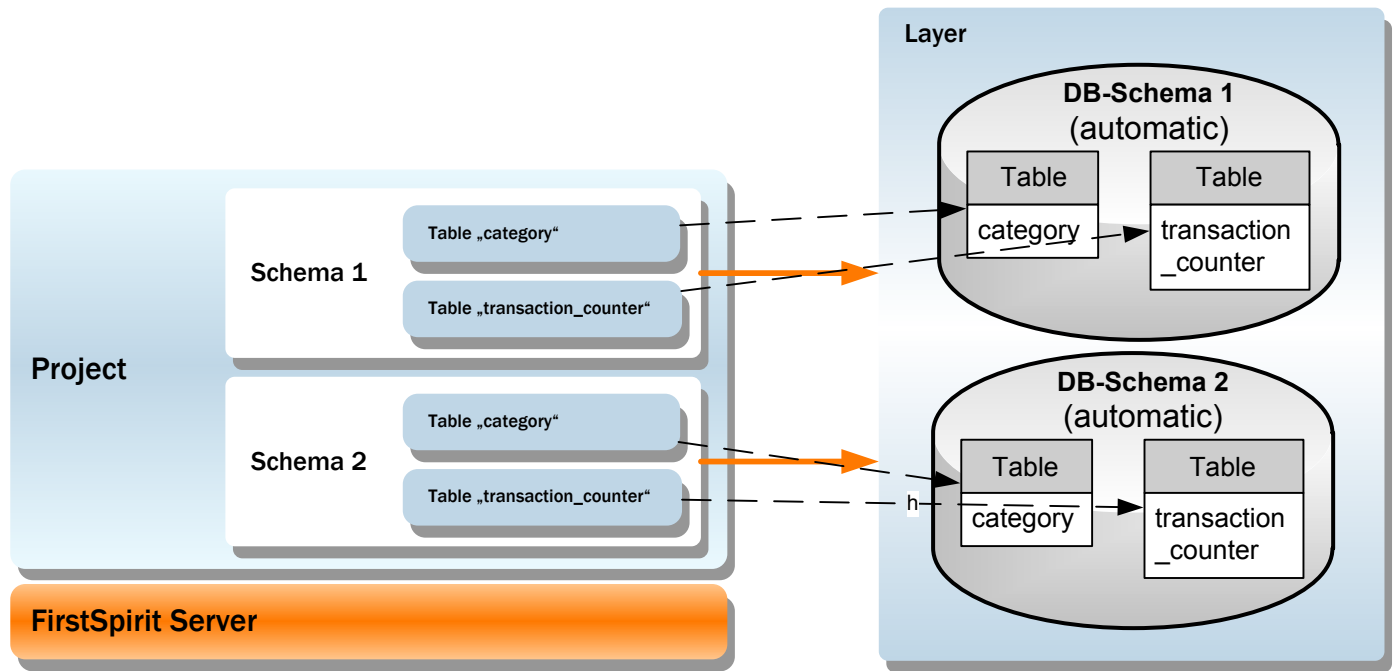


Figure 3-3: DBA layer



3.4 Content sources in FirstSpirit JavaClient

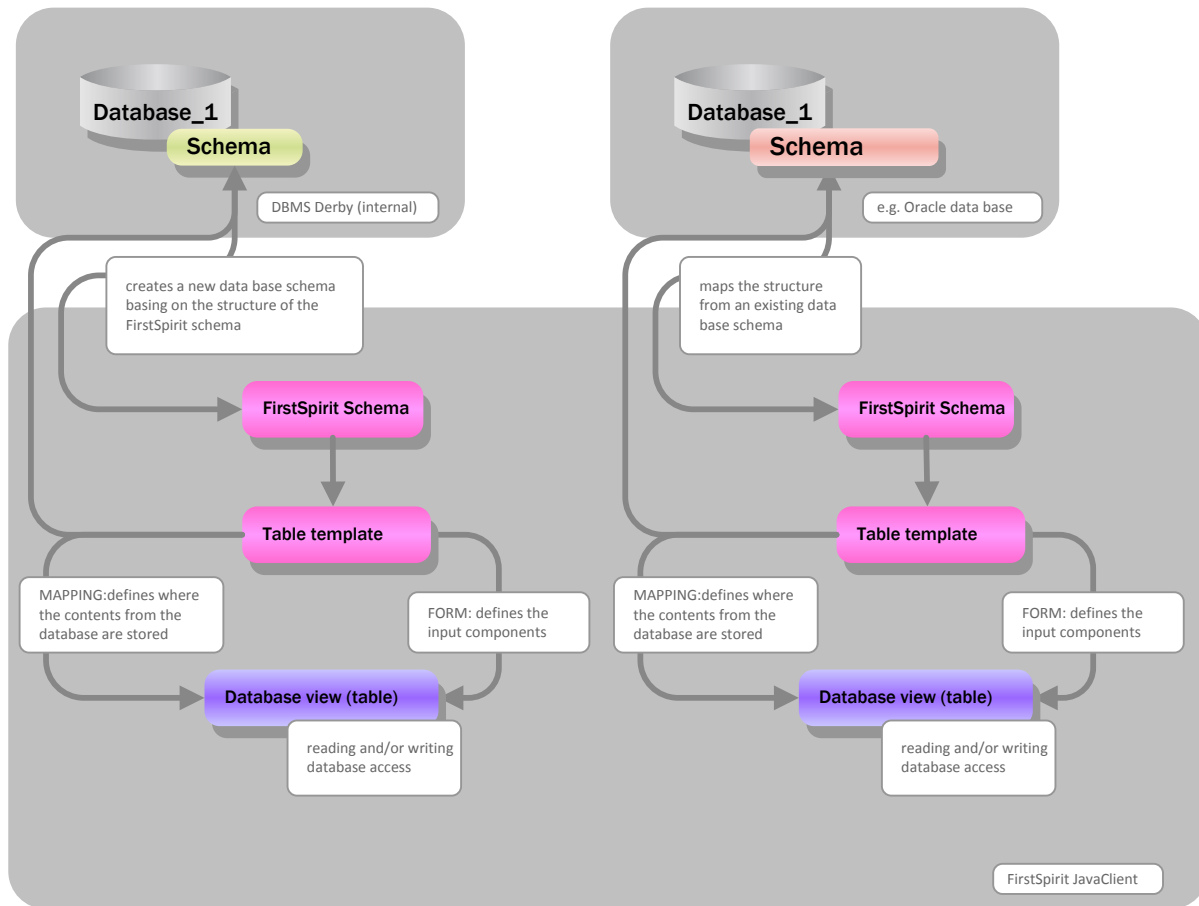


Figure 3-4: Concept – Schemata, table templates, database views

FirstSpirit schema: Either a new, blank database schema (see Chapter 2.12.1, page 97) or a database schema from an existing database (see Chapter 2.12.2, page 101) can be created using FirstSpirit JavaClient.

After a new schema has been created, the required tables can be created in the selected database and their relationships can be set using the graphical editor in FirstSpirit JavaClient (see Chapter 2.12.1, page 97). The columns that are intended to be imported by the editor later on have to be specified for each table. A column with the necessary primary key is generated automatically when the table is created.

Instead of generating a blank schema node (see Chapter 2.12.1, page 97), a new schema node can also be created based on the existing tables and relationships of a FirstSpirit project (see



Chapter 2.12.2, page 101).

Depending on the project administrator's settings for the configured database, the changes to a schema in JavaClient, such as adding a table to the physical database, can be applied ("Sync") or prevented ("No sync").

Table template: A table template can be created (below the schema node) for each table modeled in the schema. Which input elements the editor can use later to import data into corresponding tables and which input elements the editor can use to take over data from a reference table is determined in these table templates (see Chapter 2.12.4.1, page 111). In addition, the assignment of contents maintained via an input component to a database table in the physical database can be established using the "Mapping" tab (see Chapter 2.12.4.2, page 112). Thus mapping defines the save location of the contents in the database. The appearance of the data records for generation in the individual presentation channels can be set using the Template sets tab (see Chapter 2.12.4.3, page 113).

Queries: In addition, queries can be created for each database schema (see Chapter 2.12.5.1, page 114). Restrictions are made in the queries which are used to evaluate the result table. The restrictions made are then taken into account when outputting a table's data records.

View of a database: The editors work from a database's "view" in FirstSpirit's content store. A table with a link to the database table is created for this. The data is displayed in tabular form in this table. Depending on the project administrator's settings for the configured database, editors can either access database content with read-only access and for a task, such as outputting it sorted on a page as the result of a query ("content projection"), or they can also have write access and thus add new content to the database. New data records can be added or existing data records can be modified if write access is granted. The input elements defined in the table template are available to the editor for this (see Chapter 2.12.4.1, page 111).



4 Workflows

A workflow is a sequence of tasks that are completed in a fixed, predefined structure. Due date deadlines and groups of authorized persons can be defined for the respective tasks. Workflows are integrated into FirstSpirit for issuing a task and requesting release.

Project-specific workflows can be created in the template store using a graphical workflow editor (see Chapter 4.2, page 133).

Instances of these workflows can then be started on each element in a FirstSpirit project linked to a specific context or without context using the FirstSpirit menu bar. Each instance of a workflow has to run according to the rules set in the workflow.

An overview of all open or already closed workflows (instances) in a project is located at the "Workflows" root node in the template store (see Chapter 4.1, page 127). The overview also makes it possible to have a filtered view dependent on different search criteria (see Chapter 4.1.1, page 129). Tasks can be edited (see Chapter 4.1.2, page 131) and closed again (see Chapter 4.1.3, page 132) in the overview.

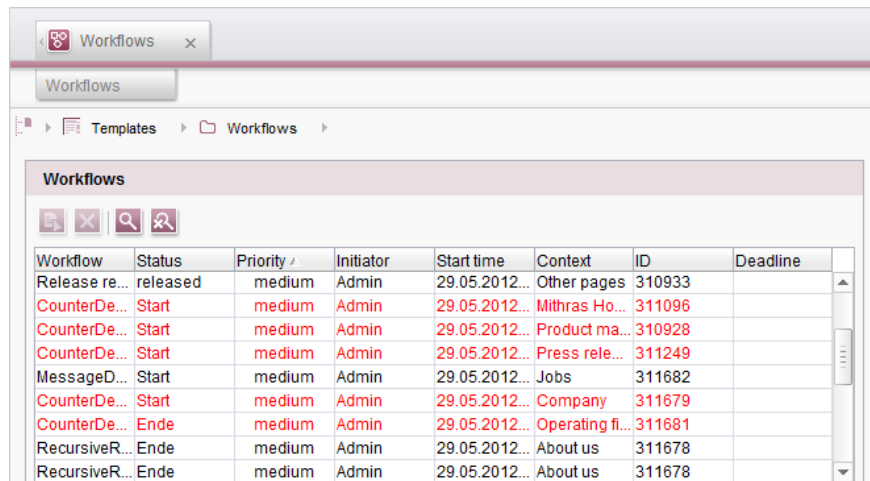


For additional information on starting and advancing workflows, see "FirstSpirit Manual for Editors", Chapter 12 "Workflows in FirstSpirit JavaClient" and "FirstSpirit Manual for Editors (WebClient)".



4.1 Overview

An overview of all open or already closed workflows (instances) in a project is shown at the "Workflows" root node in the template store.



Workflow	Status	Priority	Initiator	Start time	Context	ID	Deadline
Release re...	released	medium	Admin	29.05.2012...	Other pages	310933	
CounterDe...	Start	medium	Admin	29.05.2012...	Mithras Ho...	311096	
CounterDe...	Start	medium	Admin	29.05.2012...	Product ma...	310928	
CounterDe...	Start	medium	Admin	29.05.2012...	Press rele...	311249	
MessageD...	Start	medium	Admin	29.05.2012...	Jobs	311682	
CounterDe...	Start	medium	Admin	29.05.2012...	Company	311679	
CounterDe...	Ende	medium	Admin	29.05.2012...	Operating fi...	311681	
RecursiveR...	Ende	medium	Admin	29.05.2012...	About us	311678	
RecursiveR...	Ende	medium	Admin	29.05.2012...	About us	311678	

Figure 4-1: Workflows overview



Clicking this icon opens the task list for editing a task (see Chapter 4.1.2, page 131).



Clicking this icon closes the highlighted task (see Chapter 4.1.3, page 132).



Clicking this icon opens the "Task search" dialog for defining a task search filter (see Chapter 4.1.1, page 129).



Clicking this icon removes the filtered display and instead displays the complete view of all workflows that are still open (see Chapter 4.1.1, page 129).

The (filtered or unfiltered) workflows are listed in the table. The following information is available for each task here:

Workflow: Name of the workflow that has been started.

State: State that the current instance of the workflow is in.

Priority: The current priority that has been defined for editing the task.

Initiator: Login name of the editor that started the workflow.



Start time: Date and time when the workflow was started.


Context: If the workflow was started on an element, such as a page or media file, that element is displayed. Double-clicking the line changes the context directly to the corresponding element in the tree view.

ID: If the workflow was started on an element, such as a page or media file, the ID of the element is shown. Double-clicking the line changes the context directly to the corresponding element in the tree view.

Deadline: If a deadline is set for the current task it is shown here.

Double-clicking a (context-sensitive) schedule in the table switches the focus in JavaClient directly to the element in the tree view where the schedule was started.

It is possible to select multiple items in the table at once by pressing the SHIFT or CTRL key at the same time.

Sorting by column content: Clicking the respective column header changes the sorting of the entries in the table. The first click on a column header sorts entries in ascending order; clicking again puts them in descending order (based on column content). Sorting is indicated by a  icon after the column header:


Start time
29.05.2012 12:31
29.05.2012 12:28
29.05.2012 12:28
29.05.2012 12:28
29.05.2012 12:26
29.05.2012 12:25
29.05.2012 12:20
29.05.2012 11:47
29.05.2012 11:44

Figure 4-2: Sorting by column content (ascending order)

Clicking a third time removes the sorting.



4.1.1 Task search (filtered overview)

Workflows or tasks can be filtered according to filters such as workflow, element ID, editor, etc. using the "Task search" dialog. Clicking  opens the dialog:

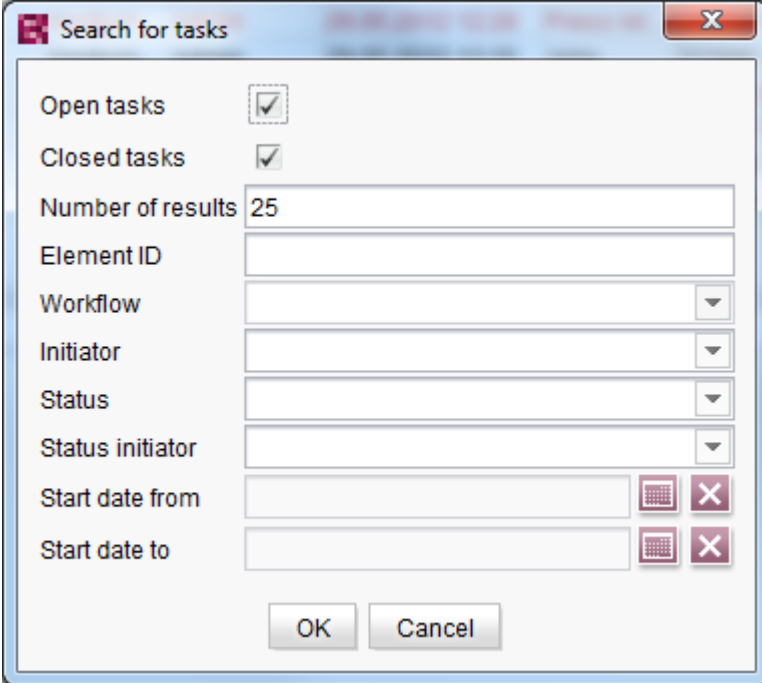


Figure 4-3: "Task search" for filtering the view

Open tasks: All "open tasks" are displayed. Tasks that have not yet reached the end state (of the workflow) are considered open.

Closed tasks: All "closed tasks" are displayed. Tasks that have reached the end state (of the workflow) are considered closed.

Result number: The number of found tasks that match the entered filter criteria; can be limited to a maximum number of results. If more results match the search criteria than the maximum allowed, only the most current results are shown.

Element ID: Unique ID for the object where the workflow was started. An empty field is shown if the workflow does not have context.

Workflow: Name of the workflow that has been started. Either the unique reference name or the language-dependent display name for the workflow is displayed in the view depending on the






"Preferred display language".

Initiator: Login name of the editor that started the workflow. Searching by partial strings is supported here. This means the search result does not have to match the search term exactly. Instead, all of the results that include the search term are displayed.

State: State that the current instance of the workflow is in. Searching by partial strings is supported here. This means the search result does not have to match the search term exactly. Instead, all of the results that include the search term are displayed.


State initiator: Login name of the editor that switched the current instance of the workflow into the current state. Searching by partial strings is supported here. This means the search result does not have to match the search term exactly. Instead, all of the results that include the search term are displayed.

Start time from / Start time to: The date selection component can be opened using the  icon. A date for the start or end period of the search can be specified here. The deciding factor is always the date when the workflow was started. All of the workflows from the currently selected day are searched if only a start date is specified

 Clicking this button filters the tasks by the entered criteria. Clicking the  icon removes the filtered display and instead displays the complete view of all workflows that are still open.



4.1.2 Editing tasks

The task list can be opened by clicking the  icon in the "Overview of workflows" dialog (see Figure 4-1). The task selected in the overview is highlighted in the task list. If the user has the permissions required for switching the workflow, the transitions are shown directly in the lower area of the task list.

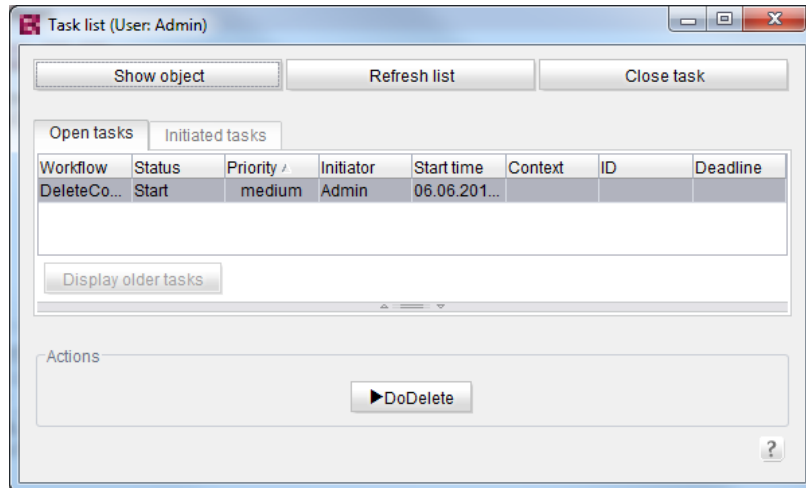


Figure 4-4: Task list


Initially, only 25 tasks are displayed in the task list on the "Open tasks" and "Initiated tasks" tabs for performance reasons. If there are more tasks, they can be displayed using the **Show older tasks** button.

Colored highlighting around a task specifies criteria such as whether the logged-in user is selected as an editor directly or due to his or her group affiliation (red text), whether the logged-in user is not selected as an editor (black text) or whether there is an invalid task (red background).

Invalid tasks can occur due to an object with an active workflow being deleted. These cannot be advanced; instead they can only be closed using the "Close task" button. If the task can be repaired, such as when a deleted object whose workflow still exists is restored, the "Repair task" button is shown in the Actions area. Carrying out this action resets the task, the state color and write protection.

Multiple selections can be made by simultaneously pressing the SHIFT or CTRL key (all of the tasks can be selected using the key combination CTRL + A). If multiple tasks are highlighted in a list, they can be advanced in one processing step (see Chapter 4.11.3, page 209).




The task list can also be opened using the "Tasks" menu or by clicking the  icon on the FirstSpirit tool bar.)



For additional information on the task list see "FirstSpirit Manual for Editors".

4.1.3 Closing tasks

Under certain condition it may be necessary to close an open task even though the end state has not yet been reached. A task can be closed by clicking the  icon in the "Overview of workflows" dialog (see Figure 4-1).

This function corresponds to the "Close task" button available in the task list.

Multiple selections can be made by simultaneously pressing the SHIFT or CTRL key (all of the tasks can be selected using the key combination CTRL + A). If multiple tasks are highlighted in a list, they can be deleted in one processing step.

A confirmation prompt appears before the tasks are deleted.



Closed tasks cannot be restored.



4.2 Modeling workflows

4.2.1 Creating a workflow

New, project-specific workflows are created using the context menu at the "Workflows" root node in the template store or at a folder in that node. Clicking the "Create workflow" entry creates a new workflow in the tree display.



Figure 4-5: Creating using the context menu

A graphical editor for modeling a new workflow opens in the edit window on the right. A start state with a transition to the first workflow activity and an end state are displayed there by default.

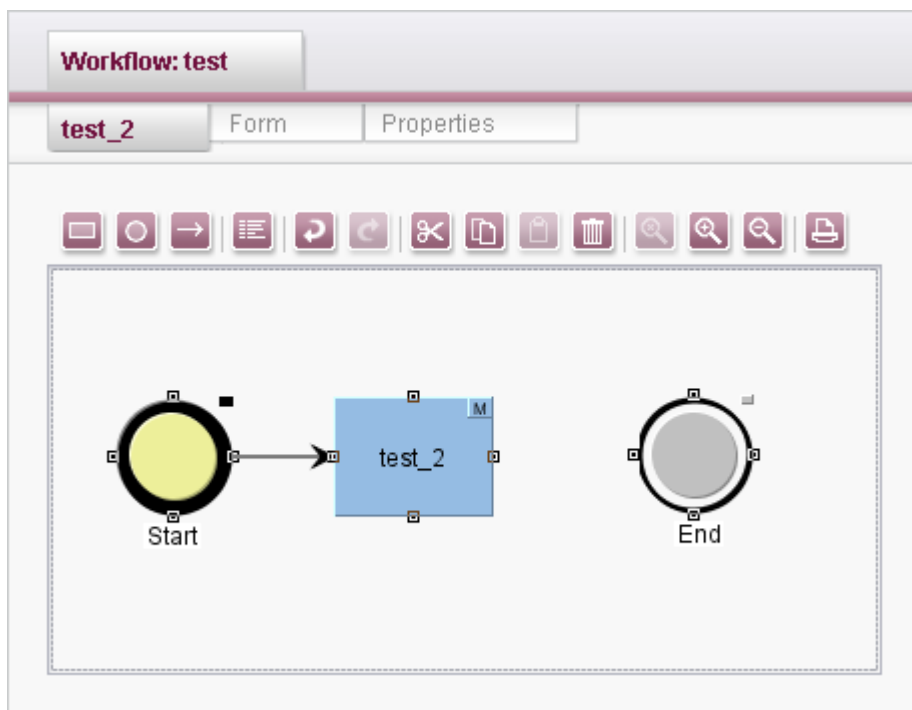


Figure 4-6: Initial state after creating a new workflow

The workflow can now be modeled in the editor by adding additional states, activities and transitions (see Chapter 4.2.3 ff.).




In the process, each workflow has to begin with a start state and end with an end state.


The editor is operated either using the tool bar (see Chapter 4.2.2, page 134) or using a context menu that can be activated at any position in the editor.


4.2.2 Workflow editor tool bar





Figure 4-7: Workflow editor tool bar


 **Creates new activity (A)** Create new activity: A new activity is created in the editor by clicking this icon (or using the keyboard shortcut A) (see Chapter 4.2.3.2, page 136).


 **Creates new status (S)** Create new state: A new state is created by clicking this icon (or using the keyboard shortcut S) (see Chapter 4.2.3.1, page 135).

 **Creates new transition (T)** A new transition is created in the editor by clicking this icon (or using the keyboard shortcut T) (see Chapter 4.2.3.3, page 137).

 **Properties (Alt+Enter)** Modify properties: Clicking this icon opens the Properties window for the activated workflow element.

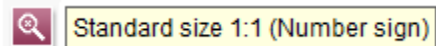
 **Cut (Ctrl+X)** Cut element: Clicking this icon cuts out all of the highlighted workflow editor elements and copies them to the clipboard. (Multiple elements can be highlighted by dragging a box around them with the mouse.)

 **Copy (Ctrl+C)** Copy element: Clicking this icon copies all of the highlighted workflow editor elements to the clipboard.

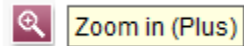
 **Insert (Ctrl+V)** Paste element: Clicking this icon pastes the elements copied to the clipboard into the workflow editor.

 **Delete (Del)** Delete: An element can be removed from the workflow process using this icon.

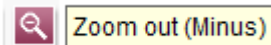




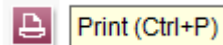
Zoom 1:1: Workflow editor elements can be displayed back at their original size using this icon.



Zoom in: Workflow editor elements can be shown enlarged using this icon.



Zoom out: Workflow editor elements can be shown shrunken using this icon.



Print: It is possible to print a graphic of the workflow using this icon (or using the keyboard shortcut Ctrl + P). A window for print settings opens (see Chapter 4.2.8, page 140).

4.2.3 Elements of the graphical workflow editor

Three different object types are available in the editor. They can be used to model and configure new workflows:

- States or statuses (see Chapter 4.2.3.1, page 135)
- Activities (see Chapter 4.2.3.2, page 136)
- Transitions (see Chapter 4.2.3.3, page 137)

4.2.3.1 State/status

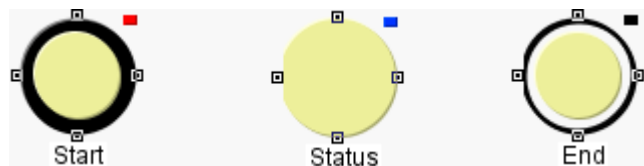
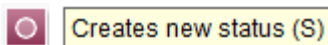


Figure 4-8: States (statuses) in the workflow editor

States, also called statuses, are represented by circles. A state is the result of an (automatic or manual) activity. States specify the state a workflow can be in.



A new state is created in the editor by clicking this icon (or using the keyboard shortcut S). Depending on the configuration, a state can be:

- A start state (only has outgoing transitions)
- An end state (only has incoming transitions)



- A normal state (has incoming or outgoing transitions)

The display of the different types is emphasized by a dark border (for start and end states) (see Figure 4-8).

4.2.3.2 Activity

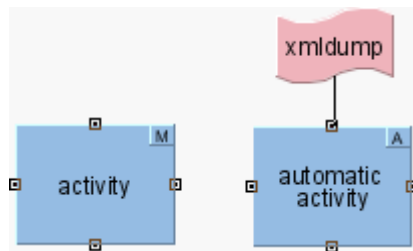


Figure 4-9: Activities in the workflow editor

Activities are represented by rectangles. An activity consists of implementing a task (e.g. "reviewing") and triggering an action (e.g. Clicking the "Approve release" button).

An activity can be run manually by a user or automatically by a script (see Chapter 4.5.4, page 156).

Manual activities are marked in the editor with an "M" in the top right corner (see Figure 4-9 – left activity); automatic activities are marked in the editor with an "A" in the top right corner (see Figure 4-9 – right activity).



Creates new activity (A)

A new activity is created in the editor by clicking this icon (or using the keyboard shortcut A). Depending on the configuration, an activity can be run:

- Manually (by an editor)
- Automatically (by a script)




4.2.3.3 Transition



Figure 4-10: Transition in the workflow editor

Transitions are represented by arrows. Transitions form the connection between an activity and a state. The permissions for a workflow model are defined here. Canceling an action results in the previous state (before switching to the transition) being retained. Canceling does not have to be modeled separately in a workflow.

 **Creates new transition (T)** A new transition is created in the editor by clicking this icon (or using the keyboard shortcut T).

4.2.4 Keyboard shortcuts in the workflow editor

A	Create a new activity.
T	Create a new transition.
S	Create a new state.
Ctrl + P	Request a print preview of the workflow model
Alt + Enter	Open the Properties dialog box for a highlighted element in the workflow model.
Ctrl + Z	Undo
Ctrl + Shift + Z	Restore
Ctrl + X	Cut
Ctrl + C	Copy
Ctrl + V	Paste
Del	Delete



4.2.5 Operating assistance for the editor

Clicking on an element once in the editor selects the element. This element can be moved to a specific spot in the editor by holding down the left mouse button. Incoming and outgoing transitions follow the moved element in the process.

You can use the mouse to draw a box around multiple elements. This allows you to move, cut or copy multiple elements simultaneously.

If a state is selected in the workflow editor, then the **Insert activity** function causes a new activity to be connected to this state automatically by a transition. In the same way, if an activity is selected, the **Insert state** function creates a transition between the activity and the new state.

Transitions are always automatically straight connections from a source to a target. In particular, support points can be inserted at a transition in order to make the representation of loops more clear. The connection between two support points is straight as well, but as many support points as needed can be added.

In order to add a support point, you have to right-click the transition at the desired spot. Right-clicking on a support point removes that support point again.

A support point can be moved to a specific position in the editor by holding down the left mouse button.

4.2.6 Rules of modeling

- Each workflow has precisely one *start state*.
- The start state can follow *precisely one outgoing transition*. Since nothing can be selected in the start state, the first outgoing transition is always taken into account.
- Transitions represent a target-oriented connection between precisely one source and one target element.
- A transition's source and target element can be states or activities but not other transitions.
- Transitions can only ever be between *one* state and *one* activity, never between two states or two activities.
- States and activities can have as many incoming and outgoing transitions as desired (exceptions: start state and end state).
- States and activities should always have names that are unique (to the workflow).



- Transitions *may* have a name; this name *must* then be unique in relation to the starting element.
- Each workflow has a fixed, defined number of end states; at least one *end state* has to be defined.
- An end state may *not have any outgoing* transitions.

4.2.7 Examples for modeling rules

- An activity always follows a state. A state and an activity are connected with "transitions" and require a unique name. A name can be specified for transitions; the name has to be unique in relation to the transition's starting element.

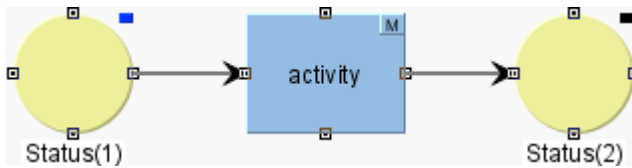


Figure 4-11: State and activities modeling rule

- Multiple activities can result from one state. Likewise, multiple activities can lead to one state.

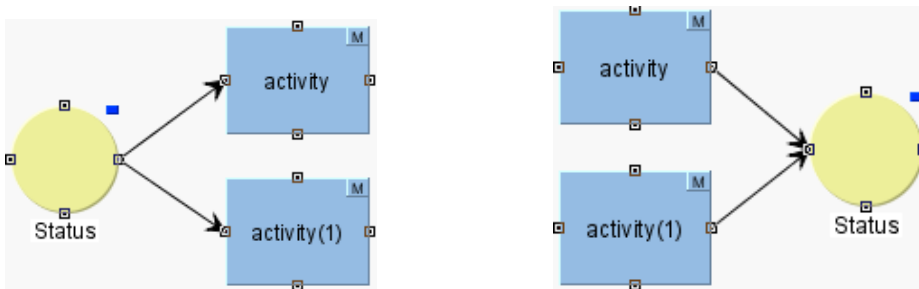


Figure 4-12: One state, multiple activities modeling rule

- One activity can lead to multiple states. Likewise, multiple states can trigger one activity.



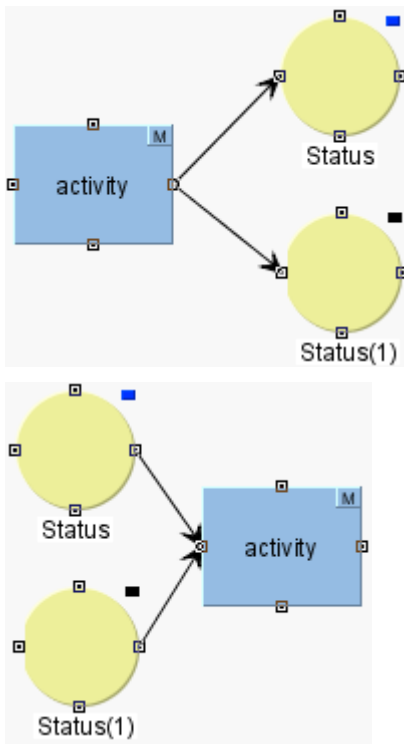


Figure 4-13: Multiple states, one activity model rule

- A script can only be attached to an (automatic) activity where the connecting line is straight.

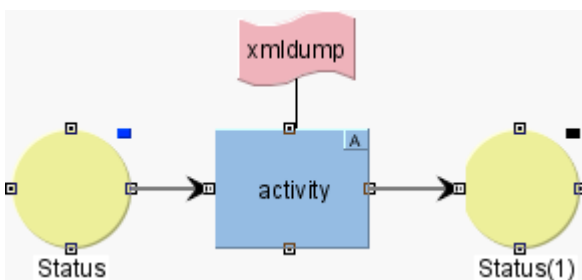


Figure 4-14: Activities and scripts modeling rule

4.2.8 Print preview for workflow models



Print (Ctrl+P)

Clicking the Print button (or using the keyboard shortcut Ctrl + P) requests a print preview of the modeled workflow in the workflow editor (see Chapter 4.2.2, page 134). A window for the print settings opens.



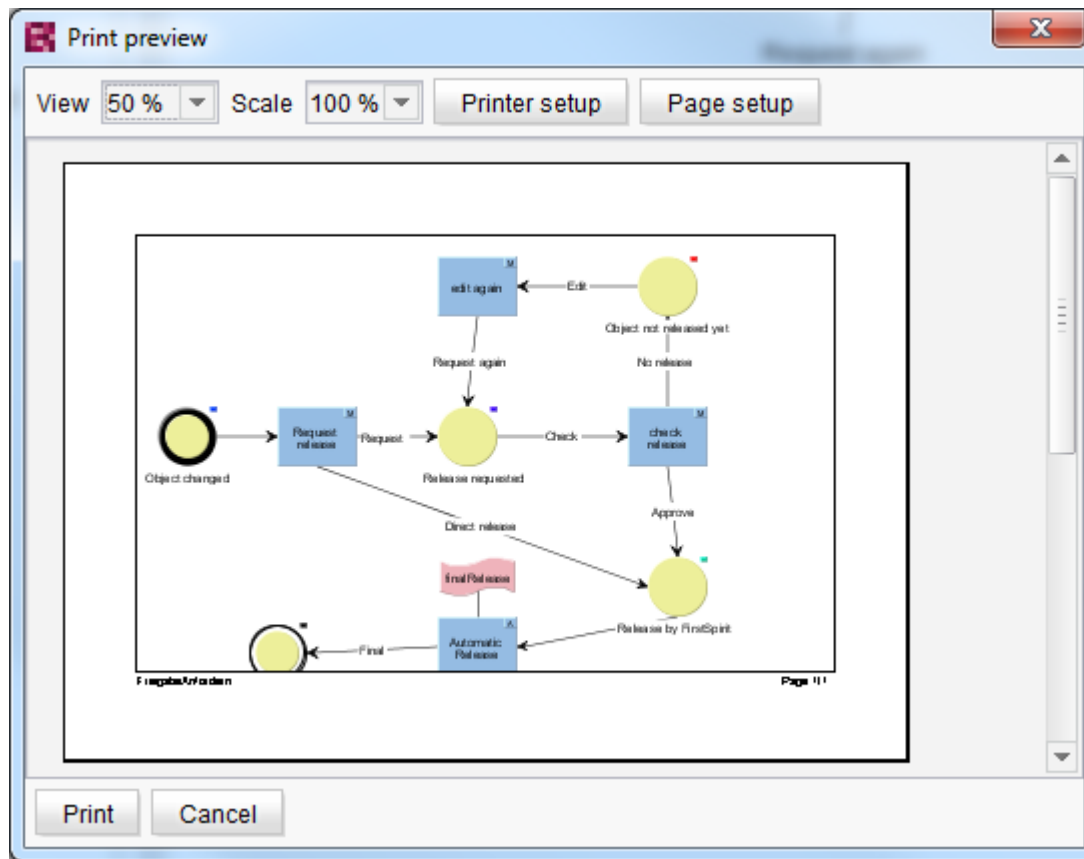


Figure 4-15: Print preview

View: The percentage size of the pages in the preview window can be configured using the combobox.

Scaling: The percentage size of the workflow model can be selected on the print preview page using the combobox. If the display is sized accordingly, multiple preview pages are displayed.

Printer setup

Clicking this button opens a window where print settings can be changed.

Page setup

Clicking this button opens a window where some settings for the printed pages can be changed.

Print

Clicking this button starts the print operation.

Cancel

Clicking this button cancels the print operation.



4.3 Error handling in workflows

4.3.1 General error handling

When starting: If an exception occurs when starting a workflow, because the user does not have permission for switching a workflow's transition for instance, the workflow is not started for the object.

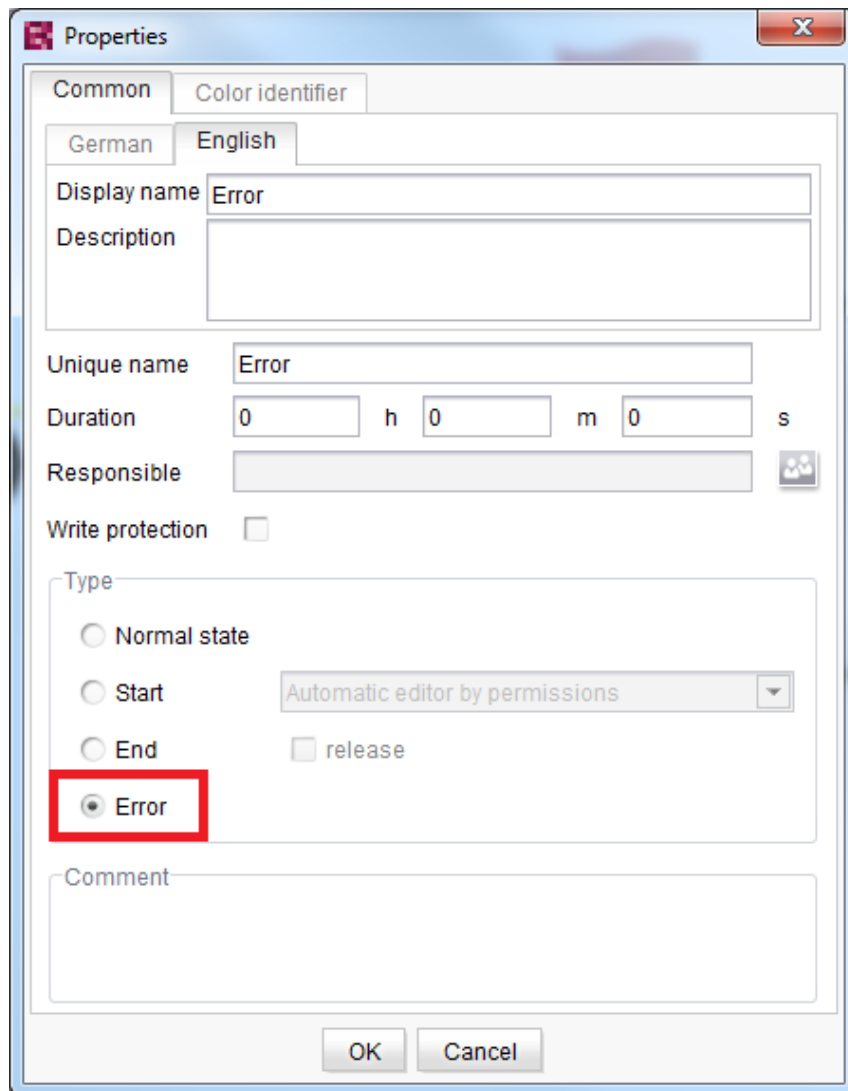
When switching: The situation is different if the workflow has already been started and an error or exception occurs while switching a transition. In this case, the state before switching the transition, i.e. the last "error-free" state, is retained. If an error state is defined in the workflow model, the element will be in the error state after the exception occurs (see Chapter 4.3.2, page 142).

4.3.2 Error state

There are many reasons for why an exception could occur when running a workflow, such as improper configuration in the workflow's model or a script error in an attached script. In order to reliably catch these errors and prevent an instance of the workflow from being in an inconsistent state after switching a transition, an optional error state is available in the modeling for workflows

A normal state is simply added to the model for this. Then the "Error" type has to be activated in the "Properties" dialog for the state:





The screenshot shows a 'Properties' dialog box with two tabs: 'Common' and 'Color identifier'. The 'Common' tab is active. It contains several fields and options:

- Language tabs: 'German' and 'English'.
- Display name: 'Error'.
- Description: (empty text area).
- Unique name: 'Error'.
- Duration: 0 h, 0 m, 0 s.
- Responsible: (empty field with a user icon).
- Write protection: .
- Type section:
 - Normal state
 - Start: Automatic editor by permissions (dropdown menu)
 - End: release
 - Error (highlighted with a red box)
- Comment: (empty text area).
- Buttons: 'OK' and 'Cancel'.

Figure 4-16: Configuring the error state

The state is then shown with a red border in the model.

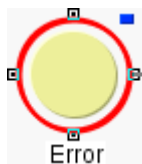


Figure 4-17: Error state in the model

The error state cannot be switched via a transition; i.e. it only has outgoing transitions just like the start state. Error handling within the workflow is modeled using these outgoing transitions (see Figure 4-19).



If an exception occurs at any spot in the workflow, the workflow instance arrives at the error state directly.

The error state catches all exceptions that occur when running the workflow, even exceptions that are not handled in the workflow. Examples of handled and unhandled exceptions are described in Chapter 4.3.3.

After error handling, the workflow can be advanced to the subsequent state (according to the workflow model).

The task list provides an overview of all instances of the workflow that had errors when being run.

The screenshot shows a window titled 'Aufgabenliste (Benutzer: Admin)'. It contains three buttons at the top: 'Objekt anzeigen', 'Liste aktualisieren', and 'Aufgabe schließen'. Below these are two tabs: 'Offene Aufgaben' and 'Initiierte Aufgaben'. The main area displays a table with the following data:

Arbeitsabla...	Status	Priorität	Initiator	Startzeitpu...	Kontext	ID	Termin
ErrorTest	Error1	mittel	Admin	31.07.201...	Über uns	575090	
ErrorTest	Error	mittel	Admin	31.07.201...	Unterneh...	575041	
ErrorTest	Error	mittel	Admin	31.07.201...	Umsetzun...	575691	
Freigabe ...	Freigabe ...	mittel	Admin	31.07.201...	Leere Seite	576396	
Freigabe ...	Freigabe ...	mittel	Admin	31.07.201...	Mithras H...	574442	

Below the table is a button 'Ältere Aufgaben anzeigen'. The lower section shows details for a task:

Bearbeiter:
 31. Juli 2012 - Admin, Manuell
 Status: Start

31. Juli 2012 - Admin, Manuell
 Aktivität: gotoMain
 Status: Error

31. Juli 2012 - Admin, Automatisch
 Aktivität: Error1
 Status: Error1

31. Juli 2012 - Admin, Automatisch
 Aktivität: ShowError
 Status: Error1
 Kommentar: de.espirit.firstspirit.access.script.ExecutionException: Method Invocation context.doTra

At the bottom, there is an 'Aktionen' section with a 'ShowError' button and a help icon.

Figure 4-18: Task list with tasks in the error state



Clicking the table "State" label sorts schedules by their current state.

Each workflow can have just one error state. If a state is defined as the error state even though an error state already exists in the workflow model, the first state is automatically reset to the "Normal" type.

4.3.3 Example: "Error" workflow

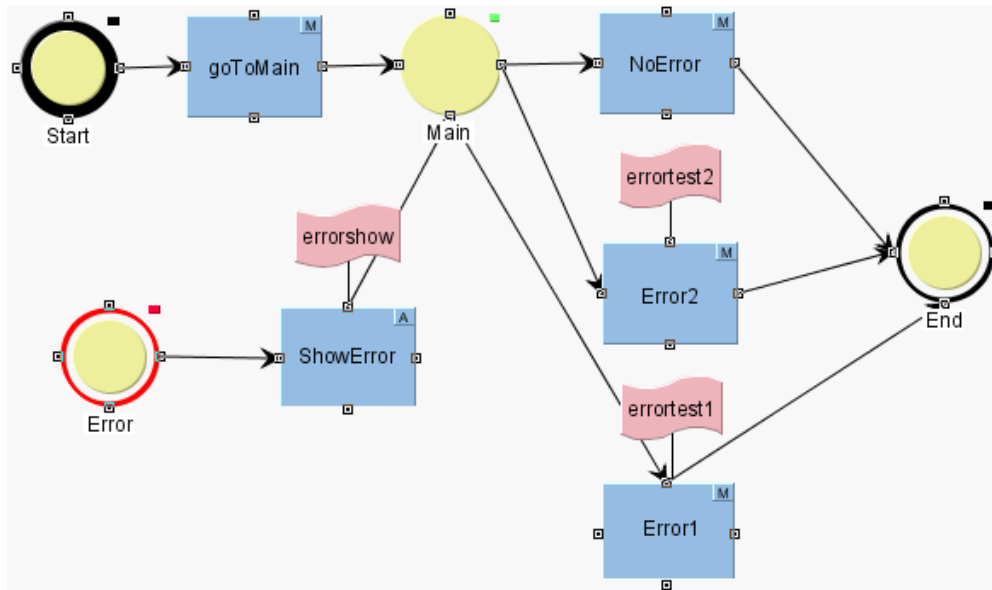


Figure 4-19: "Error" example workflow

The workflow consists of the "errorrest" workflow and the associated scripts "errorshow", "errorrest1" and "errorrest2". The workflow is made available as a compressed zip file for import into the template store ("Workflows" node).

Script errorrest1:

```
#!/Beanshell
throw new IllegalArgumentException("Error test 1");
```

The first script, "errorrest1", throws an unhandled `IllegalStateException`. This exception is not handled in the workflow, but results in the state going to "Error" instead of "End" regardless.

Script errorrest2:

```
#!/Beanshell
context.gotoErrorState("Error test 2",
new IllegalArgumentException("Error test 2"));
```



The second script, "errortest2", shows error handling within a script. The instance of the workflow is switched to the Error state directly when an exception occurs using `context.gotoErrorState(...)`.

Script errors show:

```
import de.espirit.firstspirit.common.gui.*;
import de.espirit.firstspirit.access.*;
import de.espirit.firstspirit.ui.operations.RequestOperation;
import de.espirit.firstspirit.agency.OperationAgent;

errorInfo = context.getTask().getErrorInfo();
if (errorInfo != null) {
    text = new StringBuilder("<html>Error information:<br>");
    text.append("<ul>");
    text.append("<li>Benutzer: " + errorInfo.getUserLogin() + " (" +
        errorInfo.getUserName() + ")");
    text.append("<li>Kommentar: " + errorInfo.getComment());
    text.append("<li>Aktivität: " + errorInfo.getErrorActivity());
    text.append("<li>Error: " + errorInfo.getThrowable());
    text.append("<li>ErrorInfo: " + errorInfo.getErrorInfo());
    text.append("</ul>");
    text = text.toString();
    requestOperation =
        context.requireSpecialist(OperationAgent.TYPE).getOperation(RequestOperation
            .TYPE);
    requestOperation.setKind(RequestOperation.Kind.INFO);
    requestOperation.addOk();
    requestOperation.perform(text);

} else {
    requestOperation =
        context.requireSpecialist(OperationAgent.TYPE).getOperation(RequestOperation
            .TYPE);
    requestOperation.setKind(RequestOperation.Kind.INFO);
    requestOperation.perform("No error information available.");
}
context.doTransition("->Main");
```



The "errorshow" script displays error information via an error dialog. The dialog is called automatically via a workflow as part of error handling when an error occurs. The dialog contains relevant information for resolving errors (e.g. the user that started the workflow, the activity that resulted in the error, etc.):

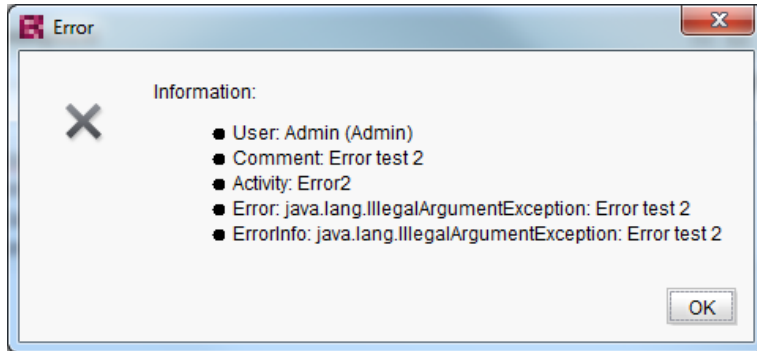


Figure 4-20: Dialog with relevant error information

The instance of the workflow is automatically reset to the "Main" state after displaying the dialog.

4.4 Form support for workflows (form)

Forms for entering content can be used in workflows. The forms are defined on the "Form" tab in a workflow:

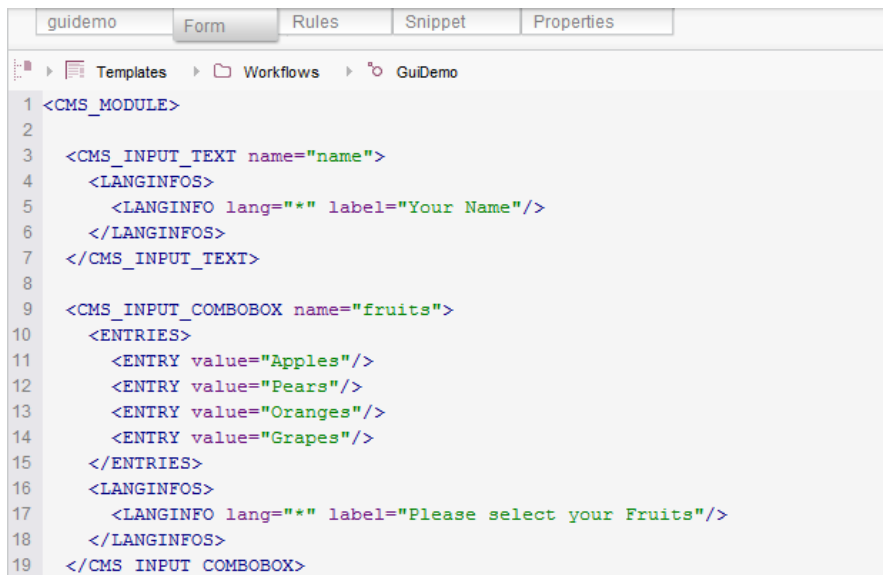


Figure 4-21: Form tab (workflow model)



While running a workflow, the editor can import values via input components that have been defined in the form area.

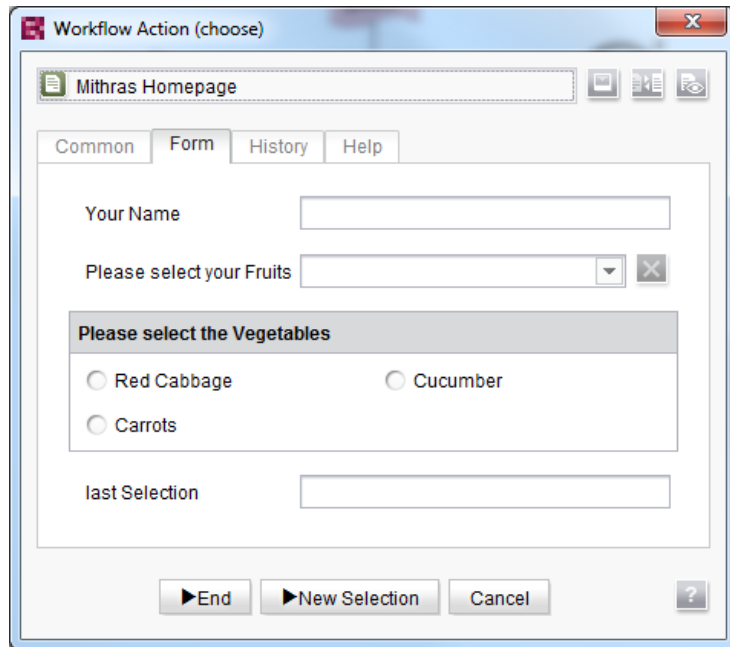


Figure 4-22: Form while running

The saved values can be output in the workflow at a later time.

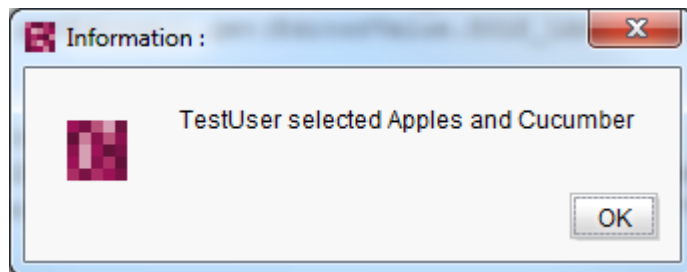


Figure 4-23: Information dialog with form content



4.4.1 Example: "GUI" workflow

In the example workflow, a script called "guitest" for displaying forms is run via the activity.

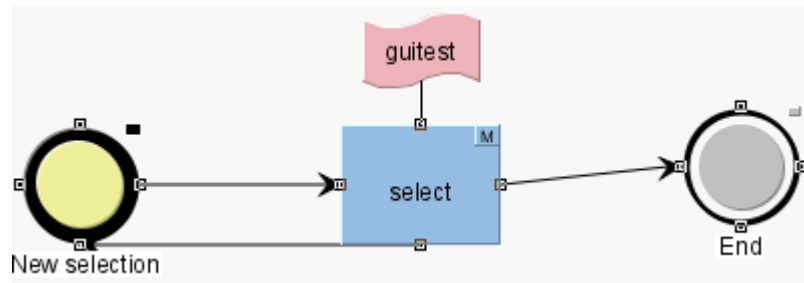


Figure 4-24: "GUI" example workflow

Script "guitest":

```

#!/Beanshell
import de.espirit.firstspirit.common.gui.*;
import de.espirit.firstspirit.access.editor.*;
import de.espirit.firstspirit.ui.operations.RequestOperation;
import de.espirit.firstspirit.agency.OperationAgent;

se = context.getStoreElement();
transition = context.showActionDialog(); data = context.getData(); if (transition !=
null) {
// display selected values
name = data.get("name").getEditor().get(EditorValue.SOLE_LANGUAGE);
obst = data.get("obst").getEditor().get(EditorValue.SOLE_LANGUAGE);
gemuese = data.get("gemuese").getEditor().get(EditorValue.SOLE_LANGUAGE);

// save selected values
lastSelection = data.get("lastSelection").getEditor();
lastSelection.set(EditorValue.SOLE_LANGUAGE, name + ", " + obst + ", " + gemuese);
text = name + " hat " + obst + " und " + gemuese + " ausgewählt";
requestOperation =
context.requireSpecialist(OperationAgent.TYPE).getOperation(RequestOperation.TYPE);
requestOperation.setKind(RequestOperation.Kind.INFO);
requestOperation.addOk();
requestOperation.perform(text);

```

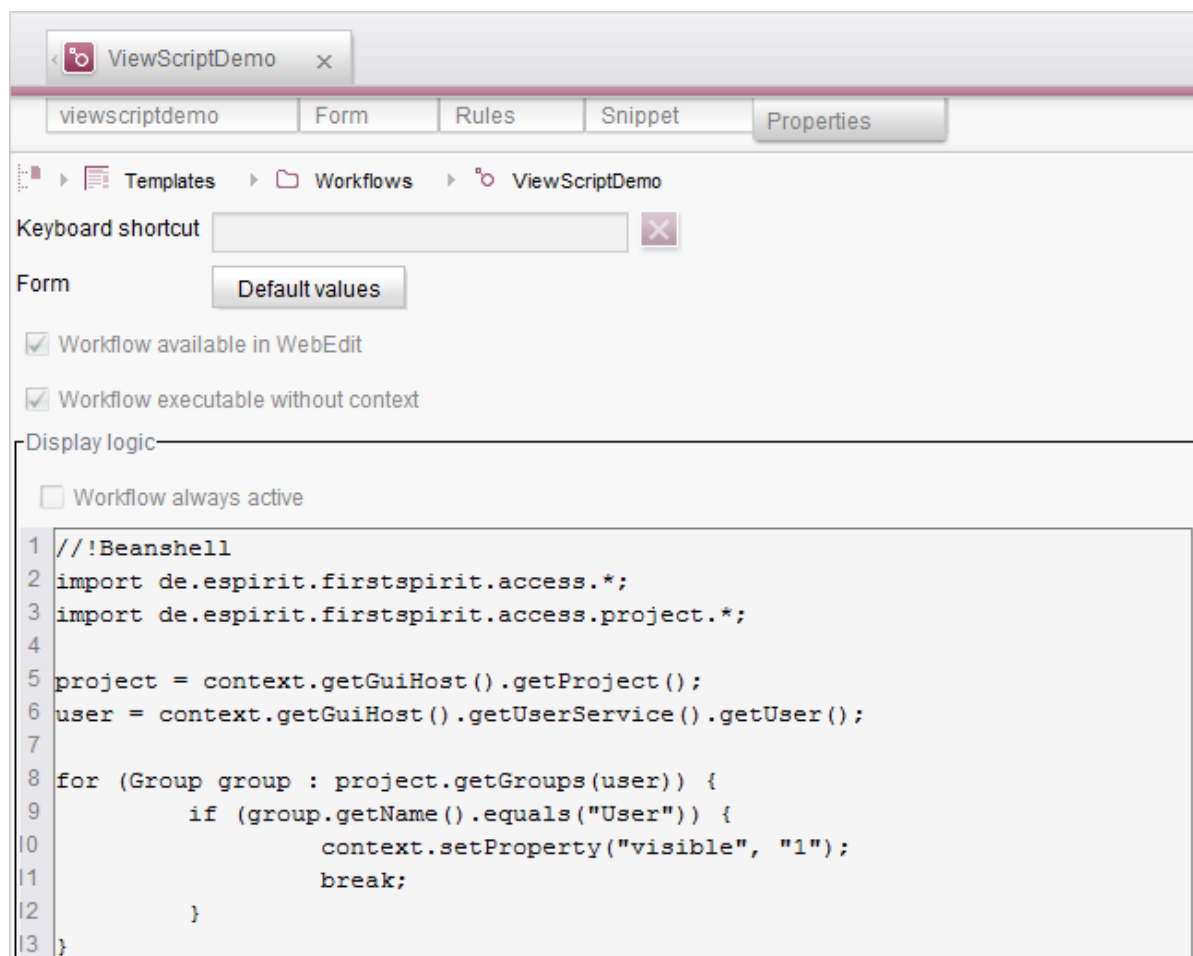


```
// do transition
context.doTransition(transition);

} else {
    requestOperation =
    context.requireSpecialist (OperationAgent.TYPE).getOperation (RequestOperation.TYPE);
    requestOperation.setKind (RequestOperation.Kind.INFO);
    requestOperation.perform ("Sie haben keine Transition ausgewählt.");
}
```

4.5 Properties of a workflow (configuration)

4.5.1 General properties



The screenshot shows the 'Properties' tab for a workflow model named 'ViewScriptDemo'. The interface includes a breadcrumb trail: Templates > Workflows > ViewScriptDemo. The 'Keyboard shortcut' field is empty. Under the 'Form' section, there is a 'Default values' button and two checked checkboxes: 'Workflow available in WebEdit' and 'Workflow executable without context'. The 'Display logic' section has an unchecked checkbox for 'Workflow always active'. Below this, a code editor displays the following Beanshell script:

```
1 //!Beanshell
2 import de.espirit.firstspirit.access.*;
3 import de.espirit.firstspirit.access.project.*;
4
5 project = context.getGuiHost().getProject();
6 user = context.getGuiHost().getUserService().getUser();
7
8 for (Group group : project.getGroups(user)) {
9     if (group.getName().equals("User")) {
10         context.setProperty("visible", "1");
11         break;
12     }
13 }
```

Figure 4-25: Properties tab (workflow model)



Keyboard shortcut: A unique keyboard shortcut can be defined for each workflow in this field. In this case, the workflow does not have to be started or switched using the context menu or the "Tasks" menu, instead it can be called directly using the defined keyboard shortcut. The cursor has to be inside this field to define a new keyboard shortcut. Then entering the desired key combination using the keyboard is all that is needed. The input is then applied in the input field. Text input is not possible. To change the keyboard shortcut, reposition the cursor in the field and then select the new key combination. Press the "Esc" key to delete a defined keyboard shortcut for a workflow.



Keyboard shortcuts can only be used for context-related workflows.

Workflow can be run in in WEBedit: If this checkbox is checked, the workflow can be run in WebClient in addition to JavaClient.

Workflow can be run without context: If this checkbox is checked, the workflow can be started without context relating to one (or more) objects. The standard "task" workflow can be started without any context, for instance.

Display logic: Display logic can be used to display or hide workflows depending on certain properties (see Chapter 4.5.2, page 151).

4.5.2 Display logic for workflows

A workflow can be assigned display logic in the template store on a workflow's "Properties" tab. Display logic can be used to display or hide workflows depending on certain properties. The display logic only relates to starting the workflow (not to visibility in the template store). If the display logic prevents a workflow from starting, such as at a certain time or for a certain group, this workflow is no longer displayed via the context menu (for context-related workflows) or via the "Tasks – Start workflow" menu function (for workflows without context).

Display logic is implemented specific to a project via a BeanShell script. Thus, specific display options can be stored for each workflow.


Possible applications:

- Workflows may only be run during a specific time frame (e.g. only on Monday between 8:00 a.m.– 9:00 p.m.)
- Workflows may only be run by a specific user or a specific group (see Figure 4-25 "Editors")



group). This can also be implemented by configuring the permissions for running a workflow, but that is only possible for context-related workflows. Displaying and hiding workflows without context can be implemented using display logic.

- Workflows may only be displayed for specific elements, e.g. image media. The configuration of permissions for individual elements for running a workflow would be accordingly extensive depending on the number of image media. Therefore, this application is easier to implement using a workflow's display logic.

 Workflow always active

If the display logic is to be deactivated, the "Workflow always active" checkbox can be checked. In this case, the workflow is always displayed, regardless of the display logic. The stored display logic is no longer evaluated, but it remains stored and can be reenabled by unchecking the checkbox.



If this is a context-related workflow, the permission to start the workflow on the element is evaluated in addition to the display logic. If the user does not have permission to start the workflow, the workflow is not displayed regardless of the display logic.

4.5.3 Properties of a state

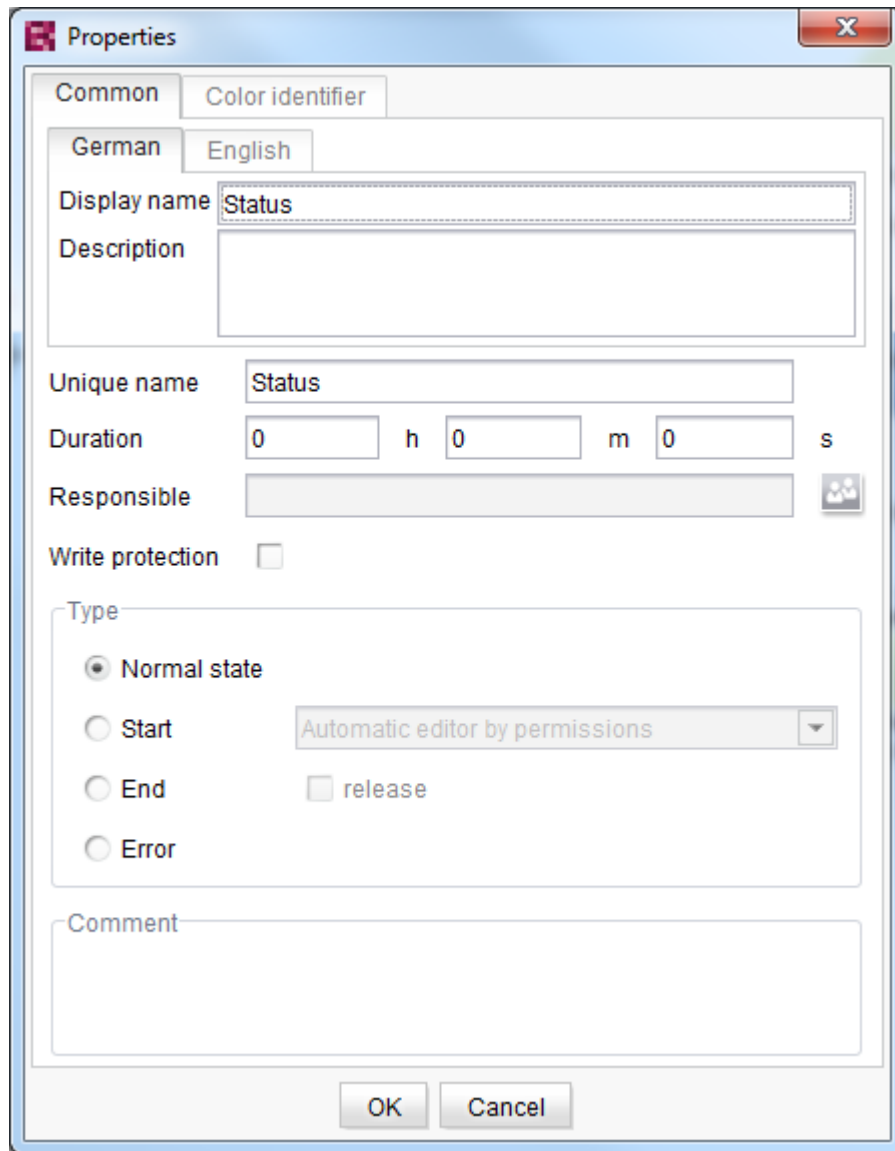


Properties (Alt+Enter)

If a state is highlighted in the workflow editor, the "Properties" window can be retrieved by clicking on the icon, using the context menu, using the key combination Alt + Enter or by double-clicking (see Chapter 4.2.2, page 134). Then settings for the selected element can be made on the "General" and "Color coding" tabs.



4.5.3.1 General tab



The screenshot shows a 'Properties' dialog box with the following fields and options:

- Common** / **Color identifier** (tabs)
- German** / **English** (sub-tabs)
- Display name:** Status
- Description:** (empty text area)
- Unique name:** Status
- Duration:** 0 h 0 m 0 s
- Responsible:** (empty field with user icon)
- Write protection:**
- Type:**
 - Normal state
 - Start: Automatic editor by permissions
 - End: release
 - Error
- Comment:** (empty text area)
- Buttons:** OK, Cancel


Figure 4-26: Properties of a state (general)

Unique name: A unique name has to be specified for the selected state in this field (character limit: <= 40 characters).

Dwelling period: A time frame that a workflow can remain in the current state before a message is sent to the responsible user or group can be specified here.

Responsible party: The responsible users or groups that are to be messaged in the event the



dwelling period is exceeded are listed in this field. Clicking the  symbol opens an additional window where the responsible parties can be selected from a list.



For using the group or user selection see FirstSpirit Manual for Editors, Chapter 13.2.4 "Changing authorized groups/users".

Write protection: If this option is enabled, then edit mode is blocked for the corresponding object while it is in this state (see Chapter 4.7, page 178).

State type: The current state can be defined as a start or end node here. Each workflow requires precisely one **start state** and at least one **end state** (also see Chapter 4.2.3.1, page 135).

- **Normal:** By default, applies to all states that are not a start or end state.
- **Start:** Describes the state of an object where the workflow is started. The start state is used to define the selection of authorized users that are allowed to switch to future states of a running workflow instance (see Chapter 4.6, page 165)
 - Manual editor (for each action)
(see Chapter 4.6.2.1, page 168)
 - Automatic editor using permissions
(see Chapter 4.6.2.2, page 169)
- **End:** Describes a possible state where an object can be after completing the workflow. Whether an object is to be released as soon as it reaches the end state can also be set.

Comment: An explanatory comment for the current state can be provided in this field. This comment is shown as a tool tip in the workflow editor.

Additional language-dependent display names and descriptions can be added using the **Display name** and **Description** fields. This refers to the editing languages (not the project languages). The project administrator defines the editing languages for a project and the editor can then switch between them using the "View – Preferred display language" menu. The display name is used in various places such as workflow dialogs (e.g. labeling the buttons in the transition dialog, on the Help tab and History tab), as entries in the context menu for starting/switching workflows, the description as a tool tip and on the Help tab. The unique name is displayed if a display name is not specified. If a description does not exist, the text from the **Comment** field is displayed.



4.5.3.2 Color coding tab

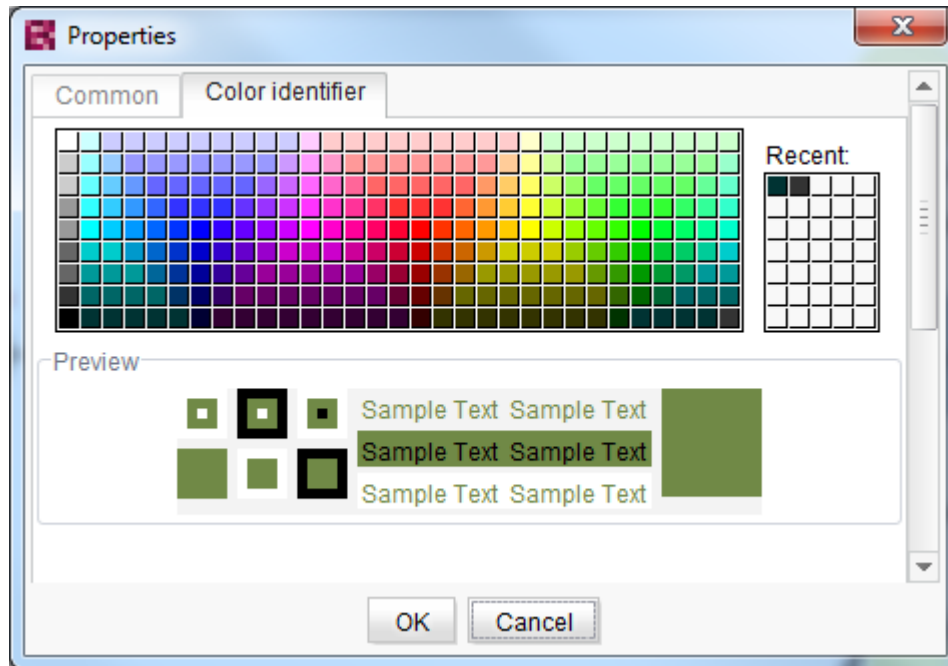


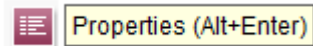
Figure 4-27: Properties of a state (color coding)

The desired color coding for the current state can be selected using the color schema on this tab. The object in the FirstSpirit client's tree structure (where the workflow was started) is highlighted with this color once the instance of the workflow has reached the corresponding state.

In order to make subsequently finding a color that has already selected easier, all of the colors that have already been selected once are listed on the right side of the window area



4.5.4 Properties of an activity



Properties (Alt+Enter)

If an activity is highlighted in the workflow editor, the "Properties" window can be retrieved by clicking on the icon, using the context menu, using the key combination Alt + Enter or by double-clicking (see Chapter 4.2.2, page 134). Settings for the selected element can subsequently be made on the "General" and "E-mail" tabs.

4.5.4.1 General tab

The screenshot shows a 'Properties' dialog box with the following fields and options:

- Common tab selected, E-mail tab also visible.
- German and English sub-tabs.
- Display name: Activity
- Description: (empty)
- Unique name: Activity
- Script: <none>
- Execution: Manual, Automatic
- Comment: (empty)
- Buttons: OK, Cancel

Figure 4-28: Properties of an activity (general)

Unique name: A unique name has to be specified for the selected activity in this field (character limit: <= 40 characters).

Script: A script that is run as soon as the activity is called can be selected using the combobox. If the required activity is to be run using a script, automatic execution has to be selected (see



"Execution").

Execution: Whether an activity is to take place manually via a user or automatically via the system is determined at this point (see Chapter 4.2.3.2, page 136):

- **Manual:** The editor is shown a dialog box that can be used to advance a workflow (instance) when running a *manual activity*.
- **Automatic:** *Automatic activities* do not wait for user interaction and are run as soon as one of the states is reached upstream in the model (i.e. the action is triggered by the system and not by the user). Thus, an automatic action (and a connected script along with it) is run directly after reaching a state. The script can carry out the necessary check and advance the workflow (instance) automatically.

Comment: An optional explanatory comment can be provided in this field.

Additional language-dependent display names and descriptions can be added using the **Display name** and **Description** fields. This refers to the editing languages (not the project languages). The project administrator defines the editing languages for a project and the editor can then switch between them using the "View – Preferred display language" menu. The display name is used in various places such as workflow dialogs (e.g. labeling the buttons in the transition dialog, on the Help tab and History tab), as entries in the context menu for starting/switching workflows, the description as a tool tip and on the Help tab. The unique name is displayed if a display name is not specified. If a description does not exist, the text from the **Comment** field is displayed.



4.5.4.2 E-mail tab

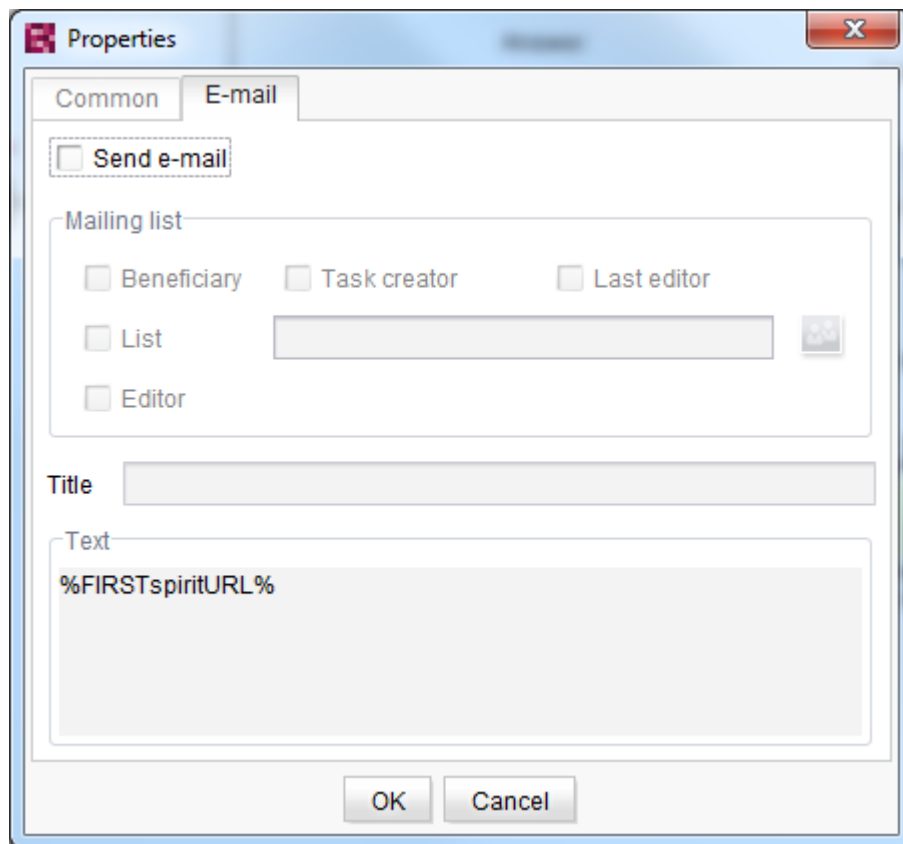



Figure 4-29: Properties of an activity (e-mail)

Send e-mail: If the checkbox is checked, an e-mail is sent to the selected recipients (see "Distributor") as soon as the activity has been carried out.

Distributor: Which persons are to be sent an e-mail can be selected here.

- **Authorized:** Persons authorized to advance the workflow to the subsequent state. These permissions are defined either directly in the workflow model using the permissions for switching the transition (see Chapter 4.5.5.2, page 162) and/or using the permissions for switching a transition on the object where the workflow's instance was started.
- **Task creator:** The user that started the instance of the workflow.
- **Last editor:** The user that switched the instance of the workflow to the current state.
- **List:** Clicking the  symbol opens a window where the desired persons or groups can be



selected from a list.



For using the group or user selection see FirstSpirit Manual for Editors, Chapter 13.2.4 "Changing authorized groups/users".

- **Editor:** The current editor of the workflow.

Title The text for the e-mail subject line is entered in this field.

Text: The message that the recipient is to receive is entered in this field. Here, the following % expressions can be used as placeholders that are replaced by the system automatically:

Placeholders for creating context-specific information:

%FIRSTspiritURL% = HTTP connection mode (default mode)
 %FIRSTspiritSOCKETURL% = SOCKET connection mode
 %PAGESTORE_PREVIEW_URL% = Preview URL for a page from the page store
 %SITESTORE_PREVIEW_URL% = Preview URL for a page reference from the site store
 %WF_NAME% = Name of the workflow
 %CREATOR% = Creator of the workflow (complete name)
 %LAST_USER% = Last editor
 %LAST_COMMENT% = Last comment
 %NEXT_USER% = Next editor
 %PRIORITY% = Priority
 %DATE% = Due date (only if set)
 %HISTORY% = History of the instance of the workflow
 %WEBeditURL% = WebEdit link to the preview of the page

If the %FIRSTspiritURL%, %FIRSTspiritRMIURL% or %FIRSTspiritSOCKETURL% placeholders are specified in the "Text" field, a link (that links to the corresponding node in the project) is created in the sent e-mail, e.g. for %FIRSTspiritURL%:

http://myServer:9999/start/FIRSTspirit.inlp?app=client&project=QS_akt&name=vorlage_1&type=Page&id=4394331&host=myServer&port=9999&mode=HTTP

or for %PAGESTORE_PREVIEW_URL%:

<http://myServer.espirit.com:9999/fs5preview/preview/4238727/page/DE/current/4238731/4394331>



Additional context-specific information for the respective instance of the workflow can be generated using the other placeholders, e.g. %HISTORY%:

```
16. April 2012 - Admin, Manuell  
Aktivität: Freigabe anfordern  
Status: Freigabe angefordert  
Kommentar: UserB : Freigabe erteilen bitte
```

In addition to the JavaClient-URL (%FIRSTspiritURL%), a link to a preview page in WebClient can be transmitted in the text (%WEBeditURL%), e.g.:

<http://myServer:9999/fs5webedit/?project=476656&store=pagestore&element=477196>

If a placeholder cannot be resolved because information is not available in the selected context, it is replaced by the appropriate information:

- German (DE): <in aktuellem Kontext nicht verfügbar>
- English (EN): <not available in current context>



Placeholder replacement only works if the JNLP servlet is installed on the system.

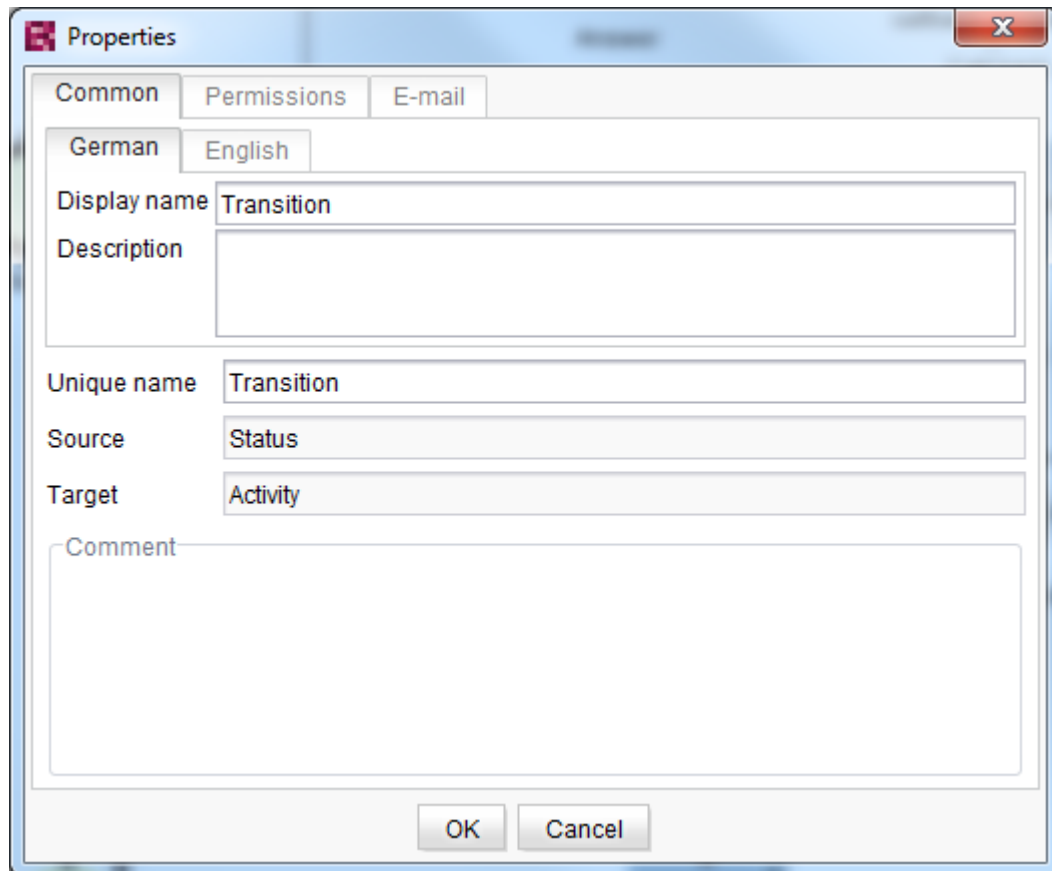


For more detailed information on the JNLP servlet see FirstSpirit Manual for Administrators, Chapter 4.3.1.2 "Area: Server"



4.5.5 Properties of a transition

4.5.5.1 General tab



The screenshot shows a 'Properties' dialog box with the following fields and values:

- Language tabs: German, English
- Display name: Transition
- Description: (empty)
- Unique name: Transition
- Source: Status
- Target: Activity
- Comment: (empty)

Figure 4-30: Properties of a transition (general)

Unique name: A name for the selected transition can be specified in this field. This name has to be unique in relation to its source (character limit: <= 40 characters).

Source: The source that the transition starts from is displayed in this field automatically.

Target: The target that the transition points to is displayed in this field automatically.

Comment: An explanatory comment for the current transition can be provided in this field.

Additional language-dependent display names and descriptions can be added using the **Display name** and **Description** fields. This refers to the editing languages (not the project languages). The project administrator defines the editing languages for a project and the editor can then



switch between them using the "View – Preferred display language" menu. The display name is used in various places such as workflow dialogs (e.g. labeling the buttons in the transition dialog, on the Help tab and History tab), as entries in the context menu for starting/switching workflows, the description as a tool tip and on the Help tab. The unique name is displayed if a display name is not specified. If a description does not exist, the text from the **Comment** field is displayed.

4.5.5.2 Permissions tab

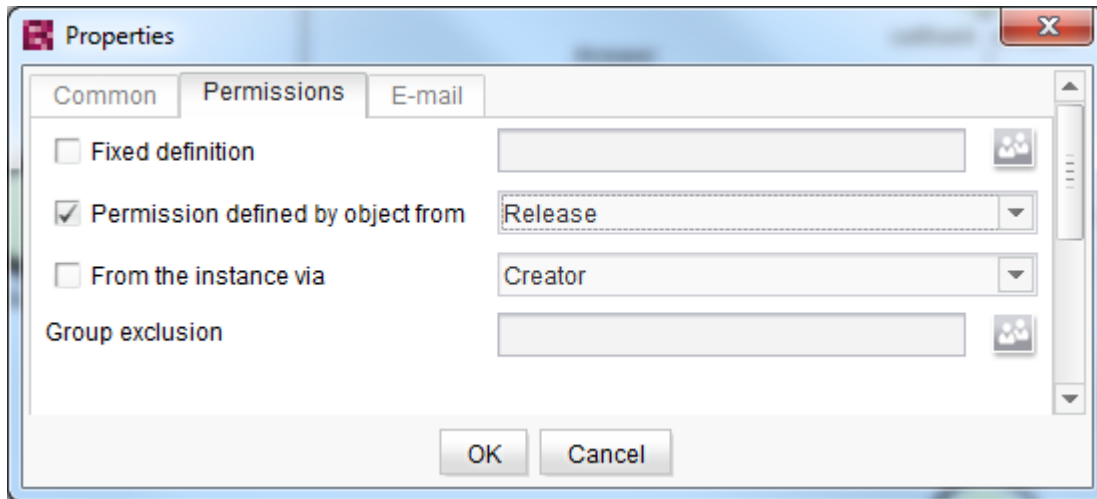


Figure 4-31: Properties of a transition (permissions)

Firmly defined: If this option is selected, the authorized users for this transition are firmly defined. The responsible users and/or groups that are allowed to switch this transition are listed in the field. Clicking the symbol after this field opens another field where the responsible parties can be selected from a list of project groups or users.

From the object: If this option is selected, then authorized users are derived from the permission definition in the FirstSpirit Client's tree structure. Which permission a user has to have for the object being considered in order to be allowed to carry out this transition can be selected in the field.

From the instance: If this option is selected, then authorized users are derived from the running instance of the workflow. The creator of the instance or the last editor can be selected in the field. The "Last editor of the target action" option is only available for outgoing transitions of a state and can only be used if the workflow contains a loop so that an activity can be passed through multiple times, e.g.:



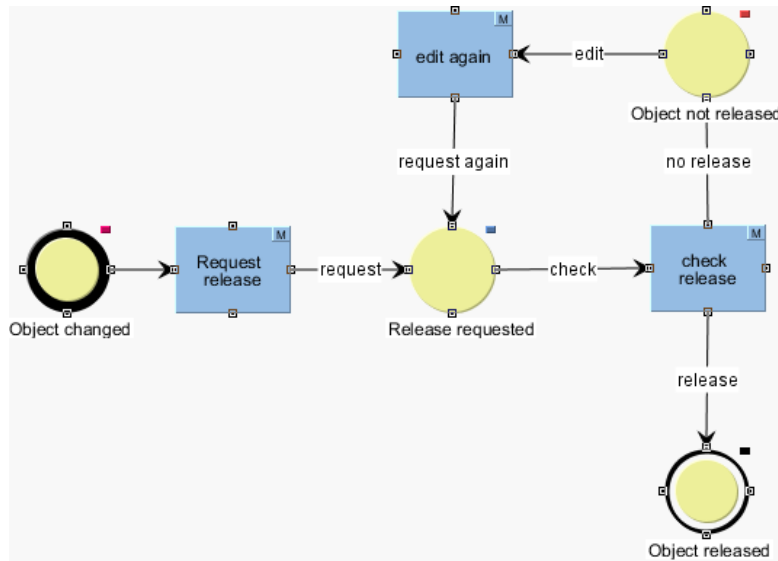


Figure 4-32: Standard workflow release

The "Review release" activity shown in Figure 4-32 would be a target action in this case, i.e. an activity that a state points to. If the "Last editor of the target action" option were selected for the "Review" transition, only a user that has already carried out this transition once before can carry out the respective transition.

Group connection: Groups that are not to appear in the "Next editor" of a "Workflow action" workflow dialog can be selected here.

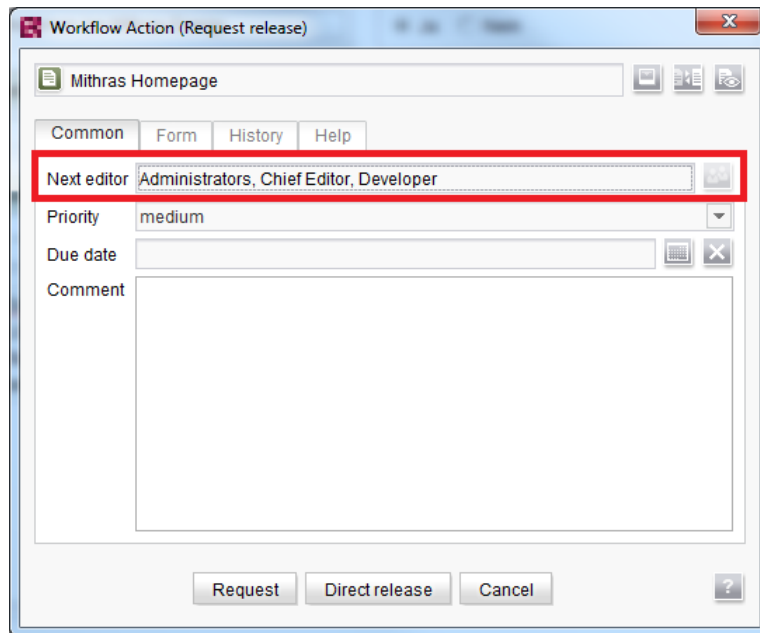



Figure 4-33: "Next editor" preselection



The groups selected using the "Group connection" function are in the dialog shown above (Figure 4-33) but remain selectable using the  icon regardless. In addition, the selection under "Group connection" affects the transmission of e-mails.

4.5.5.3 E-mail tab

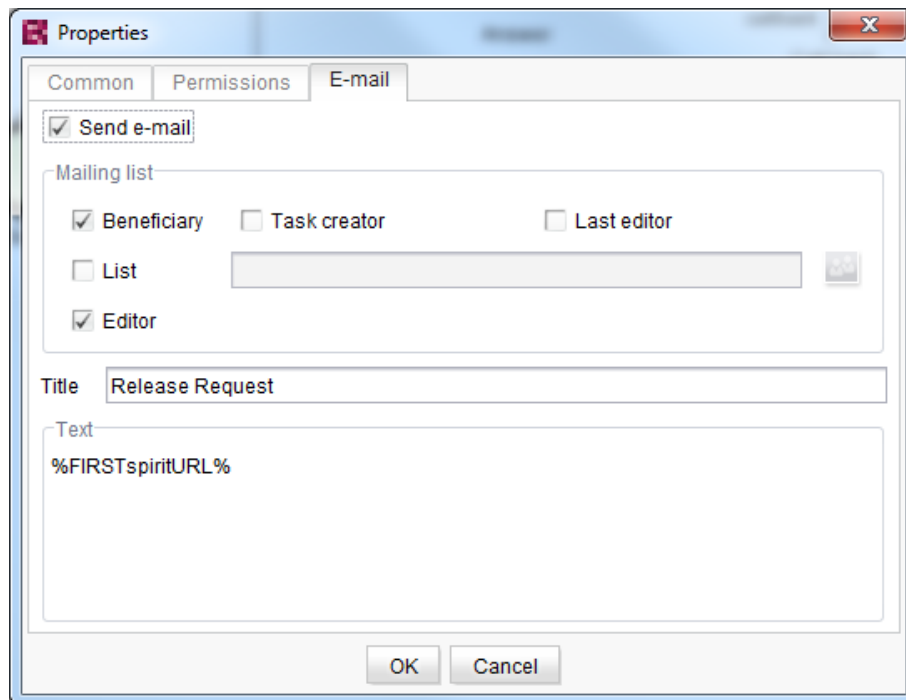


Figure 4-34: Properties of a transition (e-mail)

Send e-mail: Activating this option sends an e-mail to selected recipients as soon as this transition has been carried out.

Sending e-mail and placeholder replacement works the same as the description of sending e-mail in Chapter 4.5.4.2, page 158.



4.6 Permission configuration for workflows

Permissions for running workflows are a special type of editorial permissions that only relate to workflows in a project.

The permission configuration can be defined context-dependent directly on an object where the instance of a workflow is started or it can be defined to be generally applicable in the workflow model in the template store:

- *General permission configuration* for starting and switching a workflow in the template store (for all instances) (see Chapter 4.6.1, page 165)
- *Context-dependent permission assignment* for starting a workflow on individual objects, subtrees and stores (for individual instances – depending on the object where the workflow is started) (see Chapter 4.6.3, page 169)
- *Context-dependent permission assignment* for switching individual transitions of a workflow ("special permissions") on objects, subtrees or stores (for individual instances – depending on the object where the workflow is started) (see Chapter 4.6.4, page 172)

In addition to actual permission configuration, the authorized editors of a workflow (instance) can be limited by the content editor (when editing an activity) if this has been configured by the workflow's template developer (see Chapter 4.6.2, page 166).

The effects of permission definition in JavaClient are described in Chapter 4.6.5 (page 173 ff.) using an example.

4.6.1 General permission configuration using the template store

Permissions for starting or switching a workflow are configured in the template store using permission assignment to individual transitions. This ensures that each individual activity can only be carried out by authorized users. The Properties dialog opens when double-clicking a transition in a workflow's model. The permissions for switching a transition can be assigned on the "Permissions" tab (see Chapter 4.5.5.2, page 162).

Overwriting transition permissions: The permissions defined in the workflow model are evaluated for all instances of the workflow. Generally applicable permission configurations can be defined for the workflow this way. These permissions, however, can be overwritten for (context-dependent) workflows. It is possible to overwrite transition permissions for individual objects, subtrees or stores using the "Permission assignment" dialog for the respective objects (see



Chapter 4.6.3, page 169 and Chapter 4.6.4, page 172).

Context-dependent transition permissions: In addition to the firmly defined permissions of a group or user for switching a transition, transition permissions can also be assigned in the template store based on context. In this case, "From the instance" transition permissions have to be selected (see Chapter 4.5.5.2, page 162). If the "Last editor" is selected here, for instance, then only the editor that switched the instance of the workflow into the current state automatically receives permission for switching the transition.

Linking to editorial permissions: In addition to the option of determining permissions based on context from the instance of the workflow (see above), the editorial rights can also be linked to transition permissions (also based on context). In this case, the "From the object" transition permissions have to be selected (see Chapter 4.5.5.2, page 162). If, for instance, the "Release" editorial permission is selected here, then only the editor that has the "Release" permission for the object where the instance of the workflow was started automatically receives permission for switching the transition.

4.6.2 Changing or locking editor preselection

The preselection of authorized "Editors" is displayed in the activity dialog in the "Editors" field for the content editor running the workflow. "Editors" are all of the groups or users that have permission for switching the workflow's future transitions. Permissions that have been defined for outgoing transitions of the future state are taken into account in the process (see Chapter 4.6.1, page 165).

Example (workflow model):

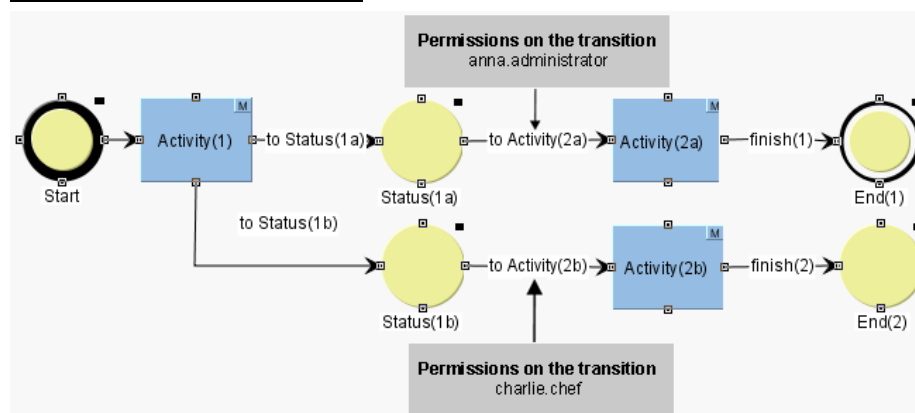


Figure 4-35: Workflow model with permissions configuration (with grey background)



Example - Description (see Figure 4-35):

- The permissions for the transition "To activity (2a)" have been firmly defined for the user "anna.administrator".
- The permissions for the transition "To activity (2b)" have been firmly defined for the user "charlie.chef".

The content editor now starts the workflow. The "Activity(1)" activity dialog opens (see Figure 4-36). The editor can select between the two states "State(1a)" and "State(1b)". The future editors of the workflow are listed in the "Editors" field automatically. After advancing the current "Activity(1)", the workflow is in either "State(1a)" or "State(1b)". Therefore, future "editors" can only be groups or editors that have permissions for the outgoing transitions of these two states. Thus, in the example, "editors" that have permissions for switching "To activity(2a)" transitions and for switching "To activity (2b)" transitions are shown.

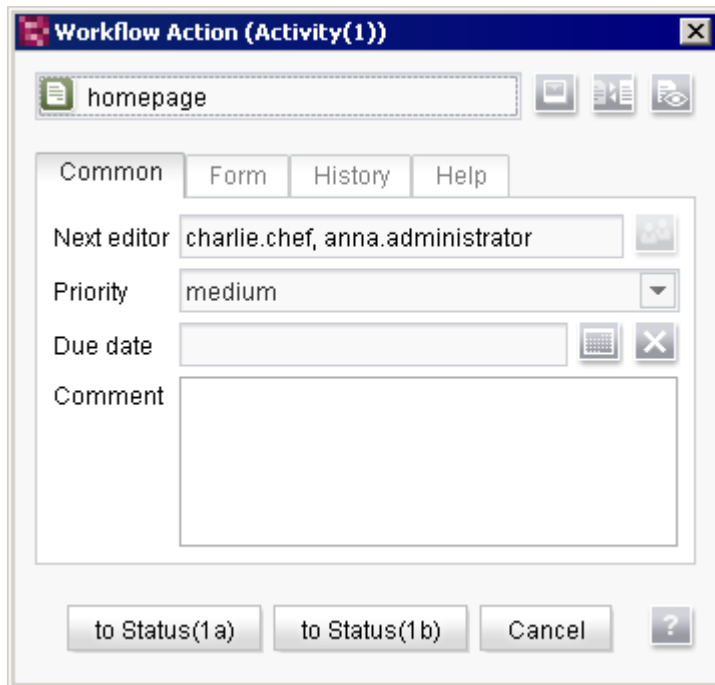


Figure 4-36: Example – "Activity(1)" workflow action

This preselection of potential future editors can be modified by the content editor. The configuration dialog for the start state has to be opened by the template developer at the workflow's start state in the template store for this (see Chapter 4.5.3.1, page 153).

A choice can be made between two options on the "General" tab.



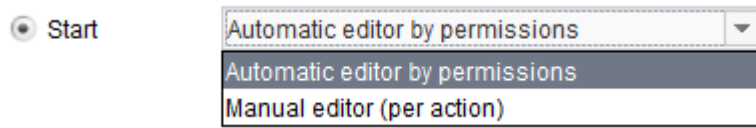


Figure 4-37: Permissions configuration for the start state

- Manual editor (for each action)
(see Chapter 4.6.2.1, page 168)
- Automatic editor using permissions
(see Chapter 4.6.2.2, page 169)

4.6.2.1 Manual editor (for each action)

Permissions defined in the workflow (for outgoing transitions of the future states) are evaluated in the "Editors" field. If the option "Manual editor (for each action)" is selected, these editors can be *modified* by the content editor. The button for selecting groups or users in the "Workflow action" dialog, which is displayed when starting or switching the workflow (instance), is then *active*.

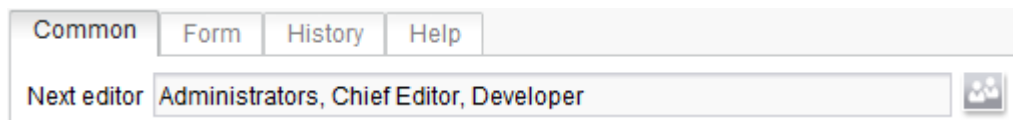




Figure 4-38: Workflow action – Manual editor using permissions

Clicking the  button opens the dialog with all of the authorized editors for selecting groups and users. This selection can then be limited to specific users by the editor.

 *The content editor is only able to limit the editors. If the list of editors is to be expanded, then permissions for the transitions of the workflow model have to be adjusted by the template developer for this.*



4.6.2.2 Automatic editor using permissions

Permissions defined in the workflow (for outgoing transitions of the future states) are evaluated in the "Editors" field. If the option "Automatic editor using permissions" is selected, these editors *cannot be modified* by the content editor. The button for selecting groups or users in the "Workflow action" dialog, which is displayed when starting or switching the workflow (instance), is then *inactive*.

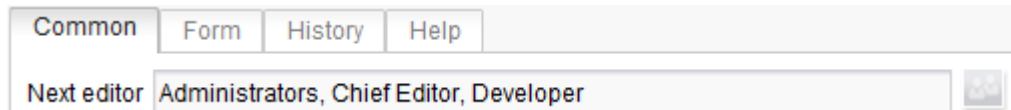


Figure 4-39: Workflow action – Automatic editor using permissions

4.6.3 Context-dependent permissions for starting a workflow

Permission is assigned based on context in FirstSpirit JavaClient. All areas of the project can be assigned editorial permissions for specific groups or users here. Detailed permission assignment for each object is possible while doing so, i.e. for a single page in the page store, for instance. These permissions can be inherited hierarchically within individual stores.

The permissions for running workflows are assigned using the "Permission assignment" dialog, the same way as for editorial permissions for groups and users. The "Permission assignment" dialog is opened using the "Tools – Modify permissions" context menu on the desired object in the JavaClient tree structure. There is a "Workflow permissions" tab in addition to general "permission assignment" of editorial permissions:



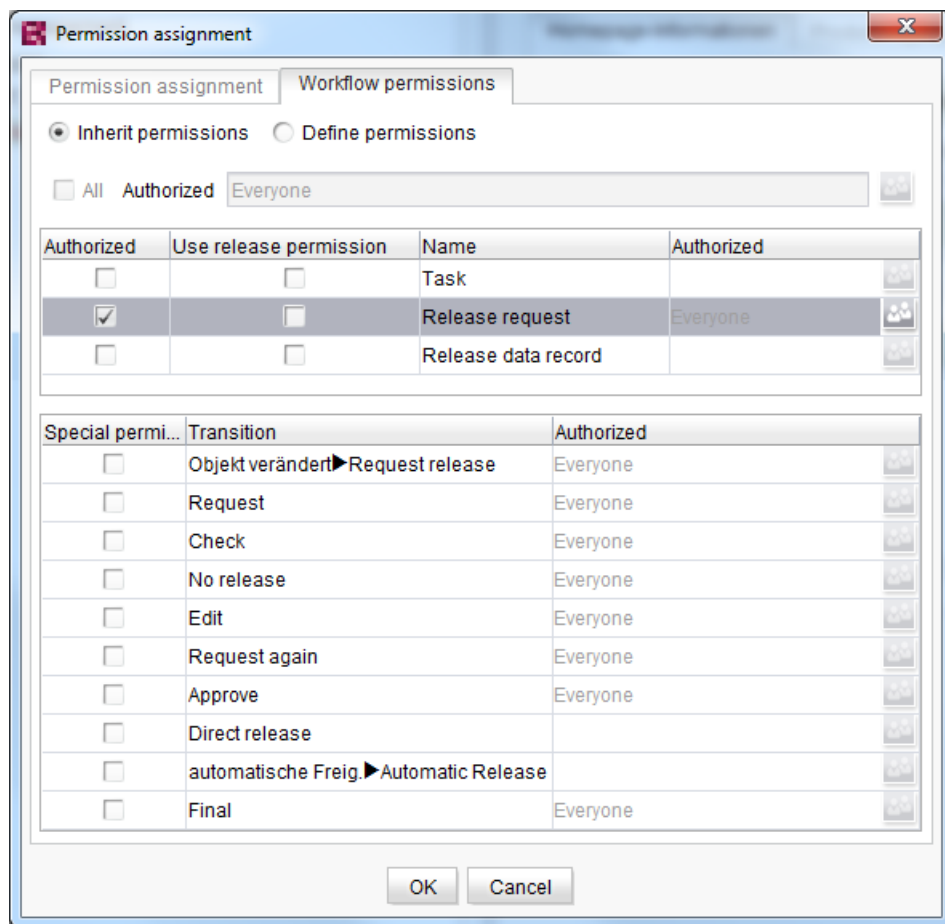


Figure 4-40: Context-dependent workflow permissions

Inherit permissions: The "Inherit permissions" radio button is selected by default (exception: root node) This setting causes the permissions from "Workflow permissions" to be inherited from a higher level node.

Define permissions: If the "Define permissions" radio button is activated, permissions for workflows can be defined on the node. The window for taking over inherited permissions opens in the first step.




Figure 4-41: Taking over inherited permissions?



If the permissions are taken over from a higher level node, then the inherited permissions are taken over in the table of workflows. If, on the other hand, the dialog is confirmed with "No", the workflow permissions are reset. The table view now active independent of the selection, this means the user can define his or her own permissions.

All: If the checkbox "All" is checked, all workflows on the current node and all hierarchically subordinate nodes of the tree structure can be started by the "authorized" user. The two tables underneath cannot be edited in this case and the settings they contain are of no significance. If the checkbox is not checked, the settings have to be set for each workflow individually.

Authorized: In this field, all users and/or groups that may call up a workflow on the current node are listed. Upon clicking the  icon, the "Select groups/users" window opens. All groups and users of the project are listed. Authorized groups and individual users can be selected using the window.

All of a project's workflows are listed in the top table (see Figure 4-40). If only selected workflows are to be permitted for a subtree, then a list of workflows that can be started by selected users can be created when defining permissions. A different user can be specified for each workflow in the process.

The input options of this table are only active if the "All" checkbox is unchecked. In this case, the permission for starting a workflow can be granted or prohibited for individual workflows.




Authorized	Use release permission	Name	Authorized
<input checked="" type="checkbox"/>	<input type="checkbox"/>	Task	Everyone 
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Release request	Everyone 
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Release data record	Administrators 

Figure 4-42: Context-dependent permissions for starting individual workflows


Allowed: If the "Allowed" checkbox is checked, all of the authorized users (see "Authorized" column) are allowed to start the workflow. The authorization is granted for the current node and all hierarchically subordinate nodes of the tree structure.

Use release permissions: If the "Use release permissions" checkbox is checked, the release permissions defined on the "Permission assignment" tab are evaluated for each user. Caution: Inconsistencies can occur when defining permissions if this checkbox is not checked. A conflict could result if, for instance, a user does *not* have permission for releasing for a specific object but is placed in the "Request release" default workflow as an authorized user. Even in this sort of case, the system would prohibit release, but the behavior (no release) is not obvious to the user since the workflow can be run through as defined up until the "Grant release" status. If, however, the "Use release permissions" checkbox is checked, then the user's release permissions are



evaluated for each of the workflow's transitions. If inconsistencies between editorial permissions (no permission for releasing) and permissions in the workflow (e.g. granting release) are determined, these transitions are hidden for the "unauthorized" user. In this case, the user can in fact start "Request release", i.e. start the workflow, but the user can no longer advance to the "Object released" state. The transition necessary to do so is hidden.

Name: The name of the workflow is in this column.

Authorized: All of the users and/or groups that are allowed to start a workflow on the current node are listed in this field. Upon clicking the  icon, the "Select groups/users" window opens. All groups and users of the project are listed. Authorized groups and individual users can be selected using the window.



For more detailed information on editorial permissions see FirstSpirit Manual for Editors, Chapter 13.

4.6.4 Context-dependent permissions for switching a workflow

The existing context-dependent permission assignments from Chapter 4.6.3 (page 169 ff.) only refer to permission for starting a workflow. However, what are known as context-dependent "special permissions" can be defined for the individual transitions.

If a specific activity is to be carried out by another user in an individual node, this can be defined using context-dependent special permissions. First, the desired workflow has to be selected in the upper table for this (see Figure 4-42). All of the transitions of the highlighted workflow are then listed in the lower table for defining special permissions (see Figure 4-43). An authorized user can then be specified for the desired workflow transition.



If permissions for switching a transition have already been defined in the template store using a workflow model (see Chapter 4.6.1, page 165), this definition (context-dependent "special permissions") overwrites the existing authorizations (from the workflow model).




Special permi...	Transition	Authorized
<input type="checkbox"/>	Objekt verändert ▶ Request release	Everyone
<input type="checkbox"/>	Request	Everyone
<input type="checkbox"/>	Check	Everyone
<input checked="" type="checkbox"/>	No release	Administrators
<input type="checkbox"/>	Edit	Everyone
<input type="checkbox"/>	Request again	Everyone
<input type="checkbox"/>	Approve	Everyone
<input checked="" type="checkbox"/>	Direct release	Chief Editor

Figure 4-43: Context-dependent special permissions for switching a transition

Special permissions: If the checkmark in this column is set, then the permissions assigned in the workflow for this transition on this node are ignored. Instead, the permissions that were listed on this location in the permissions column apply for this transition.

Transition: In this column, the names of the transitions are listed. If no name was assigned in the workflow for a transition, the name of the source and the target of the transition appear here.

Authorized: In this field, all users and/or groups that may run this transition are listed. The transition permissions listed here are taken over from the workflow model (see Chapter 4.5.5.2, page 162), but upon activating the "Special permissions" checkbox, they are overwritten (default setting: Group "Everyone").

Upon clicking the  icon, the "Select groups/users" window opens. All groups and users of the project are listed. Via the window, a selection of the authorized groups and users is made.

4.6.5 Effects on the permissions configuration

Transition permissions are either defined generally via the workflow model (see Chapter 4.6.1, page 165) or in context-dependent form for individual objects or partial trees (see Chapter 4.6.3, page 169 and Chapter 4.6.4, page 172).

The effects are identical for both permissions definitions:

Transitions *which lead to an activity* authorize the user to call up and carry out these activities via the context menu.

Transitions which lead to a state authorize the user to switch these states in the activity dialog.



Example (workflow model):

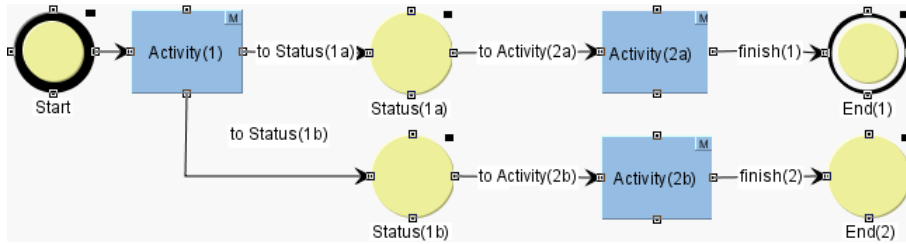


Figure 4-44: Example workflow model

Example permissions definition (defined via the workflow model):

Special permi...	Transition	Authorized
<input type="checkbox"/>	Start▶Activity(1)	Editor
<input type="checkbox"/>	to Status(1 a)	Editor
<input type="checkbox"/>	to Activity(2a)	Everyone
<input type="checkbox"/>	to Status(1 b)	Administrators
<input type="checkbox"/>	to Activity(2b)	Chief Editor
<input type="checkbox"/>	finish(1)	Everyone
<input type="checkbox"/>	finish(2)	Chief Editor

Figure 4-45 Example permissions definition via the model

Example: Effects of the transition permissions:

1. Starting the workflow via the context menu: The Editors group can open the context menu and start the workflow:



Figure 4-46: Starting the workflow via the context menu

2. The "Activities(1)" dialog provides the option to advance the workflow into "State(1a)" or "State(1b)". The button to advance "State (1a)" is only shown to the "Editors" group (see Figure 4-47); the button to advance to "State (1b)" is only shown to the "Administrators" group (see Figure 4-48).



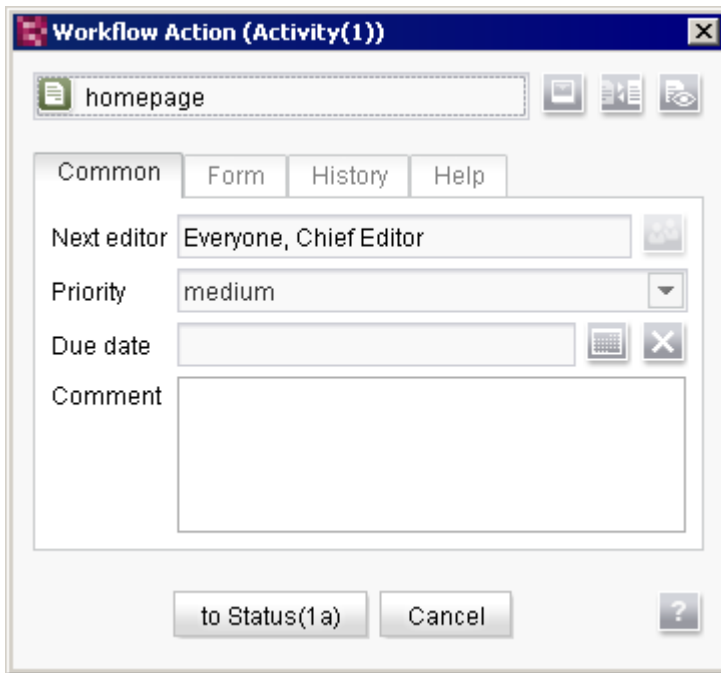


Figure 4-47: Activity dialog to switch the transition "to state (1a)"

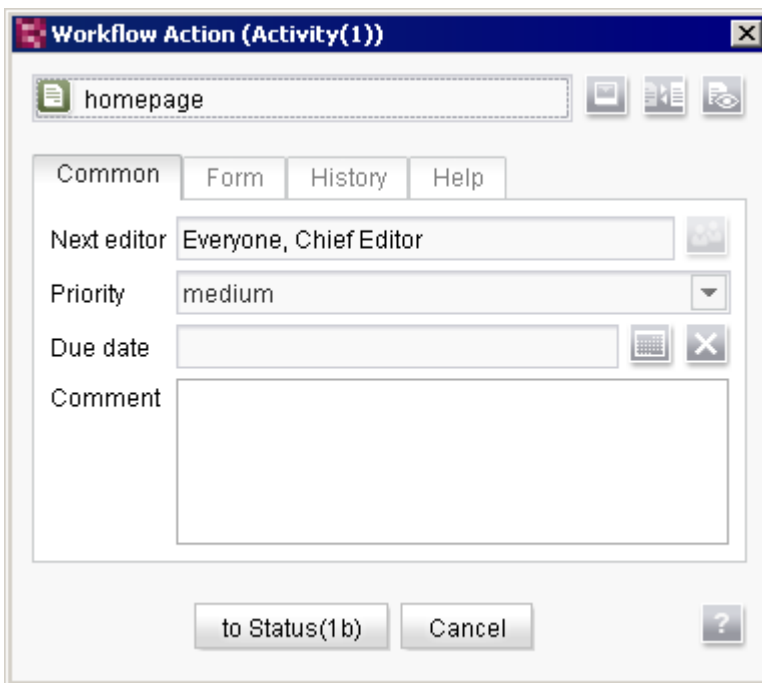


Figure 4-48: Activity dialog to switch the transition "to state (1b)"

3. If the instance of the workflow is in the state (1a), every user may query the following transition "to activity (2a)" via the context menu. The "Activity (2a)" dialog is shown.





Figure 4-49: Switching the workflow via the context menu

4. If the instance of the workflow is in State (1b), the "ChiefEditor" user can call up the following transition "to activity (2b)". The "Activity (2b)" dialog is shown.



Figure 4-50: Switching the workflow via the context menu

5. The "Activity(2a)" dialog provides the option to end the workflow in state "End(1)". The button to switch from "end(1)" appears for all users. (The field with the future "editors" is in this case empty, because it involves the last transition (see Chapter 4.6.2, page 166)).

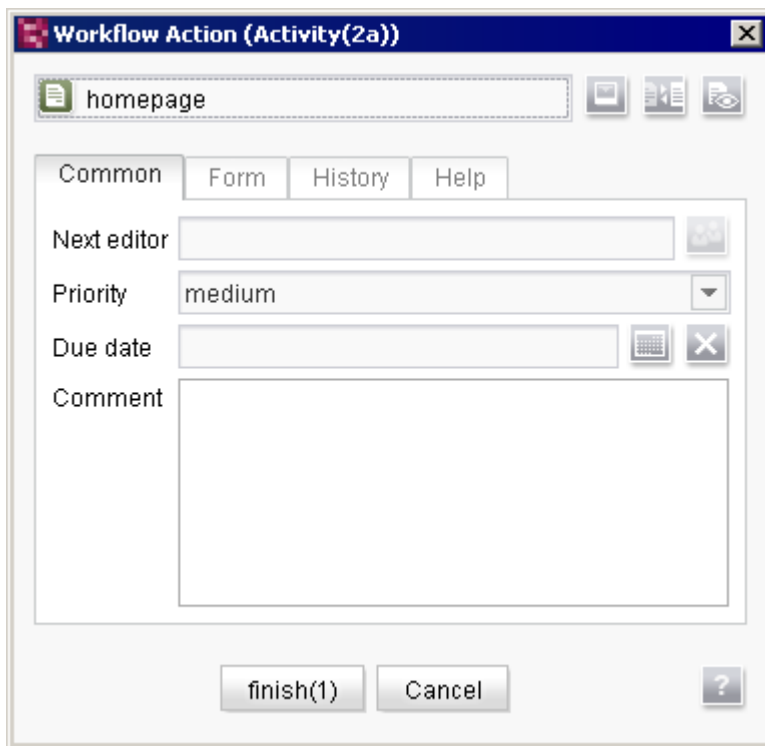


Figure 4-51: Activity dialog to switch the transition "End(1)"

6. The "Activity(2b)" dialog provides the option to end the workflow in state "End(2)". The button to switch to "End(2)" only appears for the "ChiefEditor" user. (The field with the future "editors" is in this case empty, because it involves the last transition (see Chapter 4.6.2, page 166)).



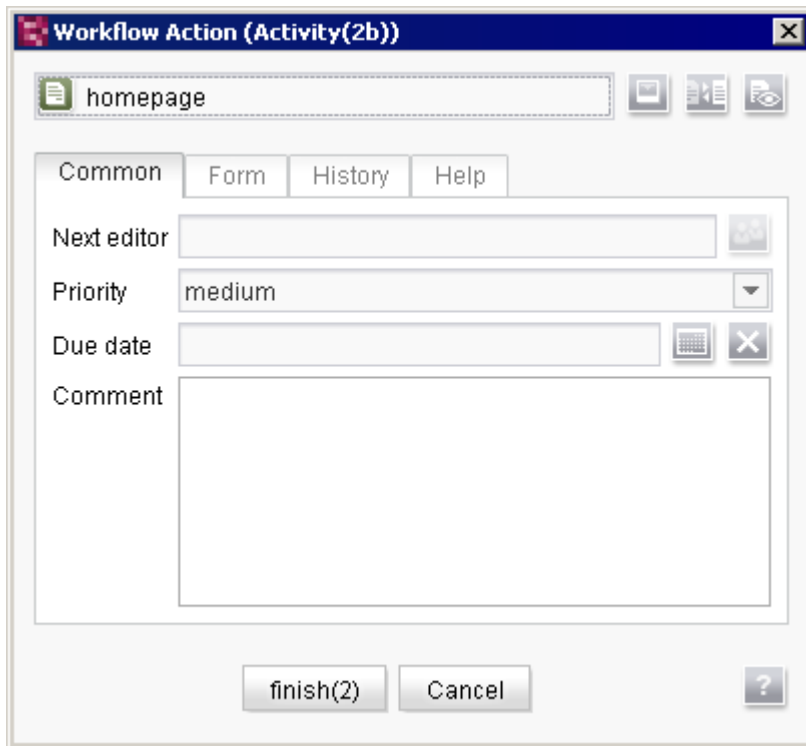


Figure 4-52: Activity dialog to switch the transition "End(2)"



If a user or a group has permission to advance a transition that leads to an activity dialog, this group and this user should also have permission to advance the transition into at least one following state. Otherwise, the activity dialog only includes a button to cancel the action. In this case, the permissions are to be checked and correspondingly redefined.

4.7 Write protection within workflows

4.7.1 General

When starting a (context-dependent) workflow, the element on which the workflow was started is equipped with write protection (see Chapter 4.5.3.1, page 153). This write protection should prevent an element from being changed by another editor while a workflow is running.

Write protection by means of running workflow instances:

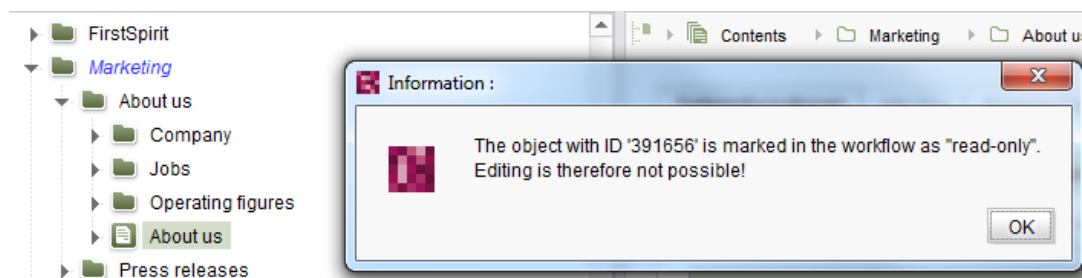


Figure 4-53: Write protection on subordinated objects

The write protection affects the current object and all subordinated objects of the running instance of the workflow. In the example from Figure 4-53, on the "Marketing" folder, a running workflow is set with write protection. If a user tries to block this folder from being edited, they will get the information that this element cannot currently be edited. The same message also appears if the user tries to block the "Company" folder or any object under the "Marketing" folder.

The write protection is set independently of whether a script is being used by the workflow and which actions will be run on the affected element.

4.7.2 Write protection when creating and moving

Within the FirstSpirit JavaClient, some actions can be run without the object being in edit mode. These changes to the edit concept should ensure that parallel work (with many users) functions as smoothly as possible, even in large projects. In this way, for example, while processing an element, the entire subtree for editing is no longer required, rather only the object that is currently to be changed. Creating or moving an element is therefore likewise possible without requiring previous write protection on the parent node (Editing mode).

Because during the workflows, however, potentially critical actions are involved (for example, release of an object), write protection of a workflow also prevents the creation or movement



within the currently running instance of the workflow.

If, for example, an editor tries to add a section on which a workflow is currently started, the following error message is shown:

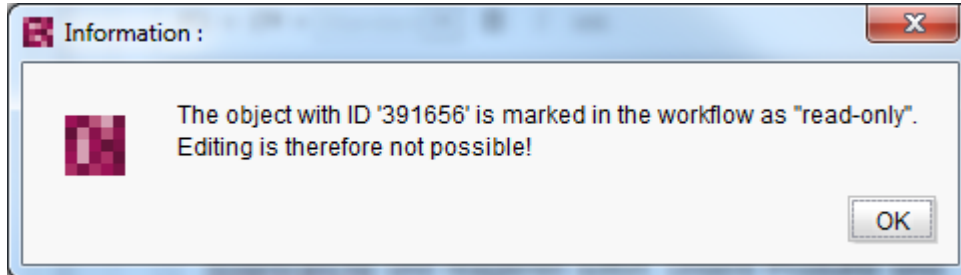


Figure 4-54: Write protection on a page (through a workflow)

4.7.3 Write protection within scripts

For some actions that are run via the FirstSpirit Access API, write-protection on the affected element is necessary, such as:

- Recursive deletion of elements in the project
- (Recursive) release of elements in the project

A problem which presents itself in real life is setting write protection in a script (on an element or subtree – API query `setLock(true, false)` or `setLock(true)`), if by starting the workflow. write protection is already on the element (through the workflow – see Chapter 4.5.3.1). The write protection of the workflow in this case prevents setting the "normal" write protection on the element.

For simple delete or release actions within the workflow, setting the write protection is however not necessary, because the affected element is already automatically blocked by the workflow upon switching the transition.

It is different if the deletion or release is recursive; in other words, it is to be run on a subtree of the project. In this case, a recursive write protection has to be set on the complete subtree and this is only possible if the write protection of the workflow is turned off. In addition, the (automatically set) write protection is temporarily removed via the state of the workflow and reset upon ending the deletion and release option (via the script). The exact procedure is described based on an example in Chapter 4.10.1.



4.8 Use of scripts in workflows

Scripts present a powerful aid in implementing customer-specific desires within the FirstSpirit workflows. As already noted in the description of the workflow editor elements, scripts can be bound exclusively within workflows to activities (see Chapter 4.5.4.1, page 156). An activity can be run manually by a user or automatically by a script (see Chapter 4.2.3.2, page 136).

The result of an activity is always related to the instance of a workflow. It either involves a status change into one of the following states accessible to the activity or the retention of the current state (corresponds to the "Cancel" semantics in the activity dialog). This also applies to scripts that are coupled to the activity. In other words, the script has to ensure on its own that a transition is carried out in the following state.

Usually, the activities connected to the script can be defined in the workflow model either as "manual" or as "automatic" – in both cases, it can make sense to use a script.



If scripts are used within workflows, NO automatic evaluation of the editing permissions (for example, during release) takes place. These permissions have to be suitably linked to the transition permissions within the workflow (see Chapter 4.5.5.2, page 162).

4.8.1 Automatic activities and scripts

Automatic: Automatic activities do not wait for user interaction and are run as soon as one of the states upstream in the model is reached (i.e. the action is triggered by the system and not by the user). Thus, an automatic action (and a connected script along with it) is run directly after reaching a state.



Through the use of automatic actions, potentially endless loops can be built. This situation is recognized by the FirstSpirit workflow interpreter; the execution of the corresponding workflow instance ends and an error message appears.

4.8.2 Manual activities and scripts

In this case, the action is run by a user. If no script is available, then the user is shown the standard form for workflows with all of the transitions allowed to them ("Activity dialog"). As soon



as the action is assigned a script, this dialog is no longer displayed automatically. If the activity dialog is to be shown to the user, then this has to be run via the script (see Chapter 4.8.3, page 181 and Chapter 4.8.4, page 183).

4.8.3 Workflow context

The script context for workflow make the following methods available:

```
Transition showAlertDialog();
```

Task: Display of the activity dialog (usually only relevant for "manual" actions). The transition selected by the user is returned as a "Transition" object. Attention: the actual transition is NOT carried out (for an example, see Chapter 4.8.4, page 183).

```
void doTransition(firstspirit.workflow.model.Transition transition)
```

Task: Execution of the specified transition. This can for example be selected by the user or another transition available (and allowed) in this action. If a transition is selected that is not allowed, then there will be an error message (for an example, see Chapter 4.8.4, page 183).

```
void doTransition(String transitionName)
```

Task: Execution of the transition indicated by name. If the transition in the model is not assigned a name, then a name is automatically generated in the form "->"+"Name of the objective state", which can be indicated here (for an example, see Chapter 4.8.5, page 185).

```
Transition[] getTransitions()
```

Task: Determines the quantity of all transitions that are available in their current state (for an example, see Chapter 4.8.4, page 183).

```
Data getData();
```

Task: A workflow model can be assigned a form. This form is shown to the editor in the activity dialog and they can enter or change data (see Chapter 4.4, page 147). Via this method, the script has access to the content of the form, and changes can also be made (see Chapter 4.4.1, page 149).

```
Map getSession()
```



Task: Each instance of a workflow is assigned (alongside the form) a special data structure (Java map) which allows a script to save its own instance state and change it if necessary. Because this state is part of a workflow instance, it is available to all scripts that are run during the life cycle of the instance. In this way, it is possible with this method to exchange (instance-related data) between scripts (for an example, see Chapter 4.8.5, page 185).

Examples:

Listing of all possible transitions with permissions starting from the current action:

```
#!/firstspirit.scripting.BeanShellWrapper

transitions = context.getTransitions();
print("Number of transitions:" + transitions.length);

for (i=0; i<transitions.length; i++) {
    print("Transition:" + transitions[i].getTarget());
    allowedUsers = transitions[i].getAllowedUsers();
    for (j=0; j<allowedUsers.size(); j++) {
        print("Allowed User:" + allowedUsers.get(j));
    }
}
```

State store in workflow instances (counter):

```
#!/firstspirit.scripting.BeanShellWrapper

state=context.getSession();
v=state.get("test");
if(v==null) v=0;
state.put("test",++v);
```

Generates an instance for every available workflow:

```
#!/firstspirit.scripting.BeanShellWrapper

import firstspirit.access.store.templatestore.*;
u=context.getUserService();
ts=u.getTemplateStore();
wfs=ts.getWorkflows().getAllChilds(Workflow.class);

for (i=0; i<wfs.length; i++) {
    print("Workflow:" + wfs[i].getName());
    try {
        u.createTask(null, wfs[i], wfs[i].getName());
    } catch (Exception e) { print("Error!");}
}
```

Other methods can be taken from the FirstSpirit Access API.



4.8.4 Example: Output of messages in workflows

Within a workflow, messages can be output to the user running it. The output of messages is realized via scripts within the workflow. A dialog via the script appears for the editor who is running the corresponding action within a workflow. There, certain information from the context of the workflow can be shown (see Chapter 4.8.3, page 181).

Example: Workflow "Message":

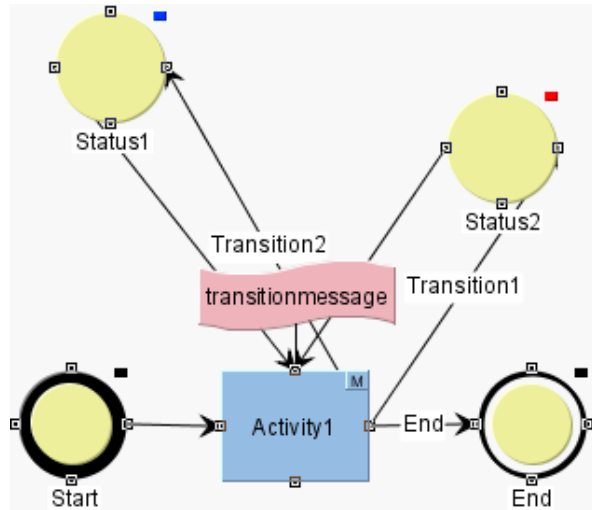


Figure 4-55: Example workflow "Message"

In this example workflow, before and after a transition is switched, an information dialog appears with the output "Hello \$USER":

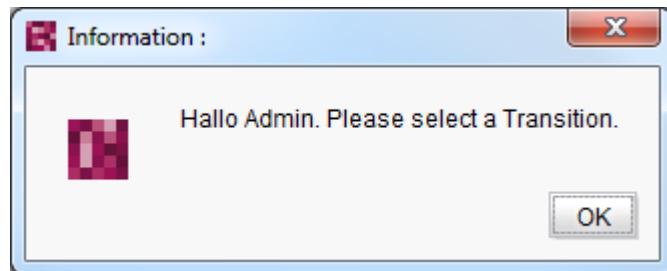


Figure 4-56: First information dialog

After advancing the transition dialog, an additional informational dialog is shown with the output "You have selected transition \$TRANSITION. Thank you for the comment \$KOMMENTAR"





Script "transitionMessage":

```
#!/Beanshell
import de.espirit.firstspirit.common.gui.*;
import de.espirit.firstspirit.ui.operations.RequestOperation;
import de.espirit.firstspirit.agency.OperationAgent;

userName = context.getGuiHost().getUserService().getUser().getLoginName();

text = "Hallo " + userName + ". Please select a Transition.";

requestOperation =
context.requireSpecialist(OperationAgent.TYPE).getOperation(RequestOperation.TYPE);
requestOperation.setKind(RequestOperation.Kind.INFO);
requestOperation.addOk();
requestOperation.perform(text);

context.showActionDialog();
transition = context.getTransitionParameters();

if (transition.getTransition() != null) {

text="You selected transition '" + transition.getTransition() + "'. A good
choice.\nThank you for your comment '" + transition.getComment() + "'";
requestOperation =
context.requireSpecialist(OperationAgent.TYPE).getOperation(RequestOperation.TYPE);
requestOperation.setKind(RequestOperation.Kind.INFO);
requestOperation.addOk();
requestOperation.perform(text);
context.doTransition(transition.getTransition());
} else {
requestOperation =
context.requireSpecialist(OperationAgent.TYPE).getOperation(RequestOperation.TYPE);
```



```
requestOperation.setKind(RequestOperation.Kind.INFO);
requestOperation.perform("You have not selected any transition.");
}
```

The information which is shown to the editor within the dialogs is retrieved in the script via the workflow context (`WorkflowScriptContext`), for example, the transition parameters (see example script):

```
context.getTransitionParameters();
```

4.8.5 Example: Persistent content within workflows

Within workflows, content can now be saved via the session and read out again after switching a transition.

Example: Workflow "Counter":

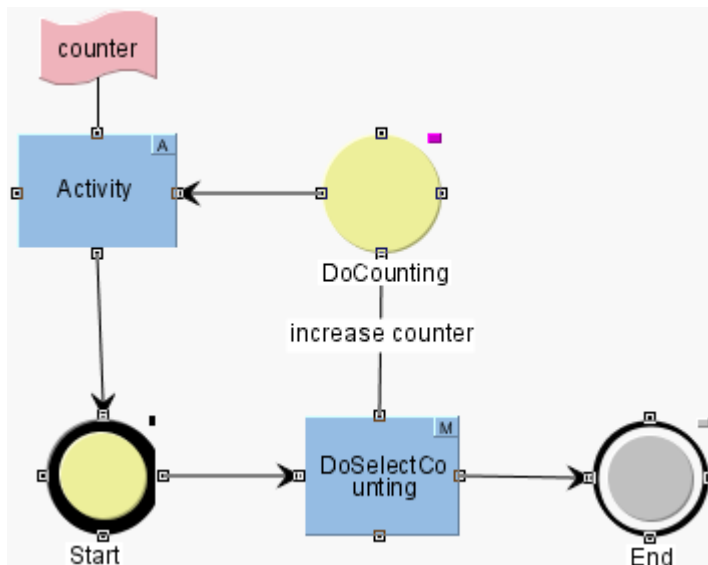


Figure 4-57: Example workflow "Counter"

Within the "DoSelectCounting" activity, a counter can be increased during every execution of the workflow by a value of 1. The value of the counter is saved, and upon a restart of the workflow, increased again by a value of 1. The value is shown to the user within an information dialog:



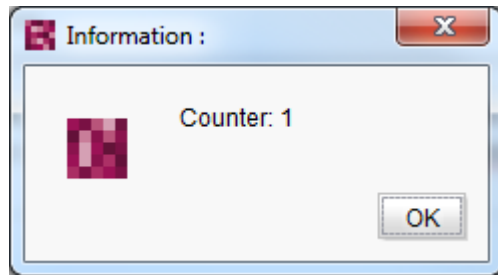


Figure 4-58: Value of the counter

"counter" script:

```
#!/Beanshell
import de.espirit.firstspirit.common.gui.*;
import de.espirit.firstspirit.ui.operations.RequestOperation;
import de.espirit.firstspirit.agency.OperationAgent;

session = context.getSession();
counter = session.get("counter");
if (counter == null) {
counter = new Integer(1);
}
text = "Counter: " + counter;
requestOperation =
context.requireSpecialist(OperationAgent.TYPE).getOperation(RequestOperation
.TYPE);
requestOperation.setKind(RequestOperation.Kind.INFO);
requestOperation.addOk();
requestOperation.perform(text);

session.put("counter", new Integer(counter + 1));
context.doTransition("->Start");
```



4.9 Deleting via a workflow

To delete elements in the FirstSpirit JavaClient and in the FirstSpirit WebClient, a project-specific workflow can be created and tied directly to the existing controls (buttons on the menu bar, context menu entry) of elements. Instead of simply deleting an object, such as a page, a more complex deletion function can be made available via the workflow, for example, the additional deletion of dependent objects on a page (demo workflow, see Chapter 4.9.2).



Deletion through a workflow is only available if the project was configured by the project administrator.

Within the clients, the new workflow is then started via the familiar control elements. The individual tasks of the workflow appear, as usual, in the task list (see Chapter 4.9.1, page 187).

If within a project, deletion via a workflow is configured, the permissions configuration for the workflow has to be adapted. The conventional editing permissions for deletion that are defined for a user or a group apply only if the permissions configuration is adapted correspondingly in the workflow (see Chapter 4.9.3, page 190).

4.9.1 Deleting via a workflow in the JavaClient

If deletion of elements in the project was bound to a workflow, then the workflow can be started and advanced in the JavaClient through the conventional control element. To do so, the following control elements are available:

- Mark element and click the button.
- Mark element and run the context menu entry "Delete".



- Mark element and click on the icon  in the icon bar

Similarly to multiple selection of workflows, deletion via a workflow can be run at the same time on a number of objects (see Figure 4-59 and Chapter 4.11, page 207).





The workflow can only be started if no workflow has been started on one of the marked objects and the user has the corresponding permissions to run the workflow. Otherwise, the corresponding control elements are deactivated.

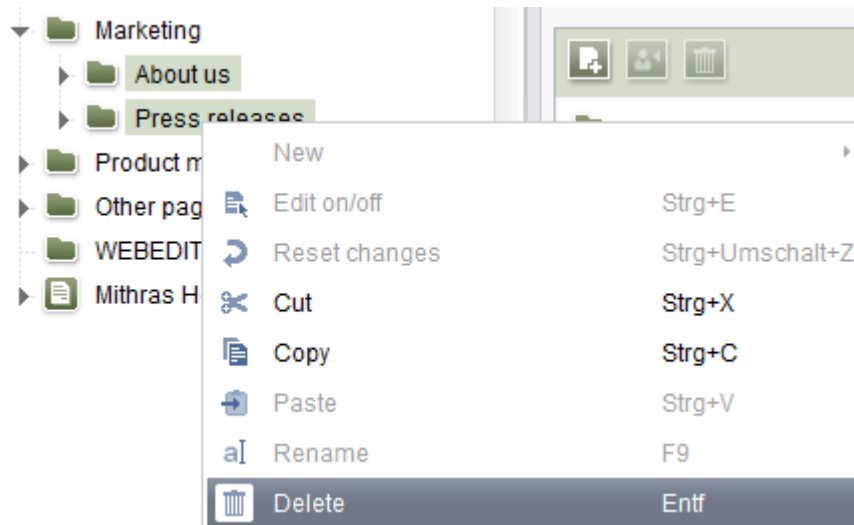


Figure 4-59: Multiple selection while deleting via a workflow



The "Delete" permission is also evaluated if elements are deleted via a workflow. If a user has permission to switch the workflow but NOT permission to delete elements, the workflow can be started (context menu entry "Delete" is activated) but deletion of the element is however not possible. The transition that deletes the element is not shown to these users.

4.9.2 Deleting via a workflow in the WebClient

If deleting elements in the project is bound to a workflow, then the workflow can be started in the WebClient via the "Contents/Delete" menu or via the state menu. Likewise, in the state menu, a workflow which was started for deletion of an element can be advanced.

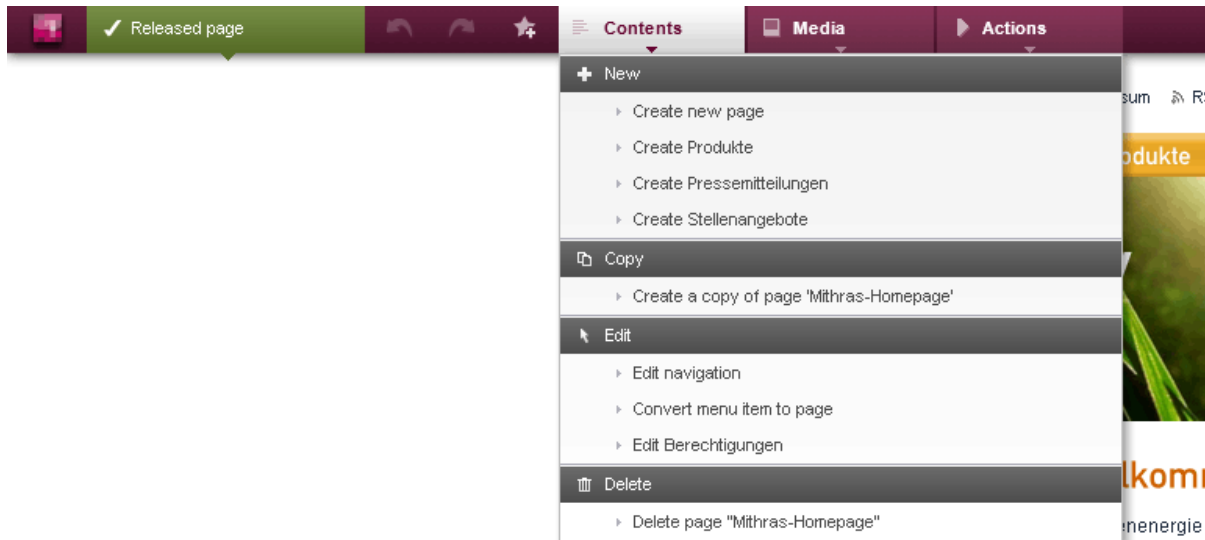


Figure 4-60: Workflow to delete a page in the WebClient – Content menu

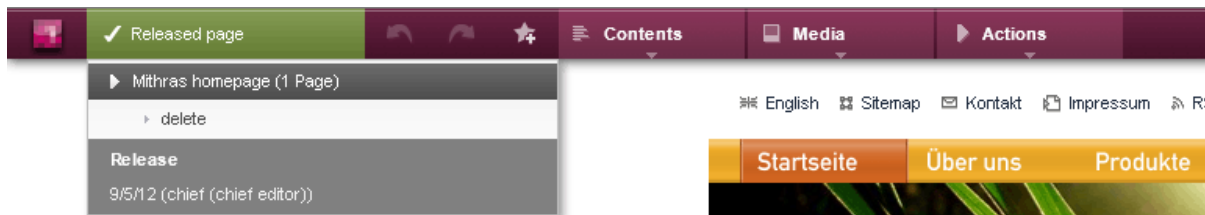


Figure 4-61: Workflow to delete a page in the WebClient – State menu

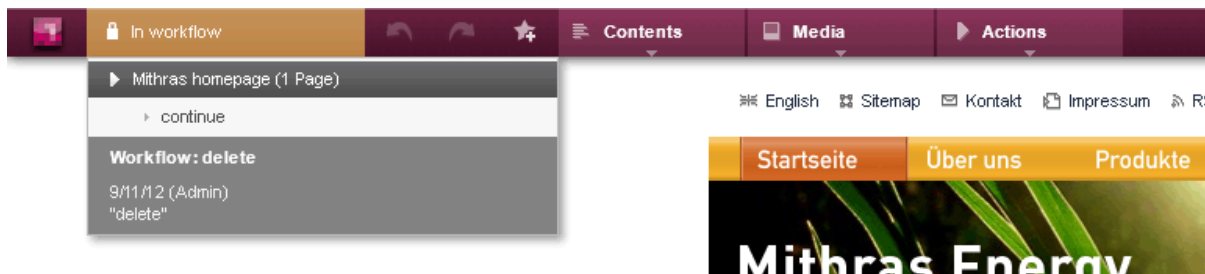



Figure 4-62: Advancing a workflow to delete an element

*It is to be noted that a workflow in the WebClient is **always** run on a page reference, and this represents the context for the workflow. If the corresponding page is also to be deleted, this has*



to be controlled via the script used by the workflow. Here, the order of elements to be deleted is to be noted (see also Chapter 2.2.8.1, page 32).



The workflow can only be started if no workflow has been started on one of the marked objects and the user has the corresponding permissions to run the workflow.

4.9.3 Permissions configuration

The permissions are assigned in FirstSpirit JavaClient. Here, all areas of the project can be assigned permissions for certain groups or users (see Chapter 4.6, page 165).


The permissions to delete elements (without workflow) are normally defined via the editing permissions. Editing permissions are defined for a user or a group on the respective element. In this way, permissions can be assigned for all editorial work. In addition, alongside "View" or "Change", there are, for example, the permission to "Delete object" or the permission to "Delete folder".

Permissions defined in this object

User Group	No Permissions	Visible	Read	Change	Create object	Create folder	Remove object	Remove folder	Release	Show metadata	Change metadata	Change permissions
Administrators	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Developer	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Everyone	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Figure 4-63: Editorial permissions "Delete object" and "Delete folder"

Additional information on editorial permissions is located in the FirstSpirit Manual for Editors, Chapter 13.1.



These editorial permissions are not automatically accessed if deletion is tied to a workflow. If these permissions are evaluated, the permissions configuration in the workflow has to be adapted first (see Figure 4-65).

If deleting in a project involves a workflow, the permissions configuration has to be relocated to the workflow. The permissions for running workflows are assigned, as with editing permissions for groups and users, in the "Permissions assignment" dialog within the administration area in the FirstSpirit JavaClient.



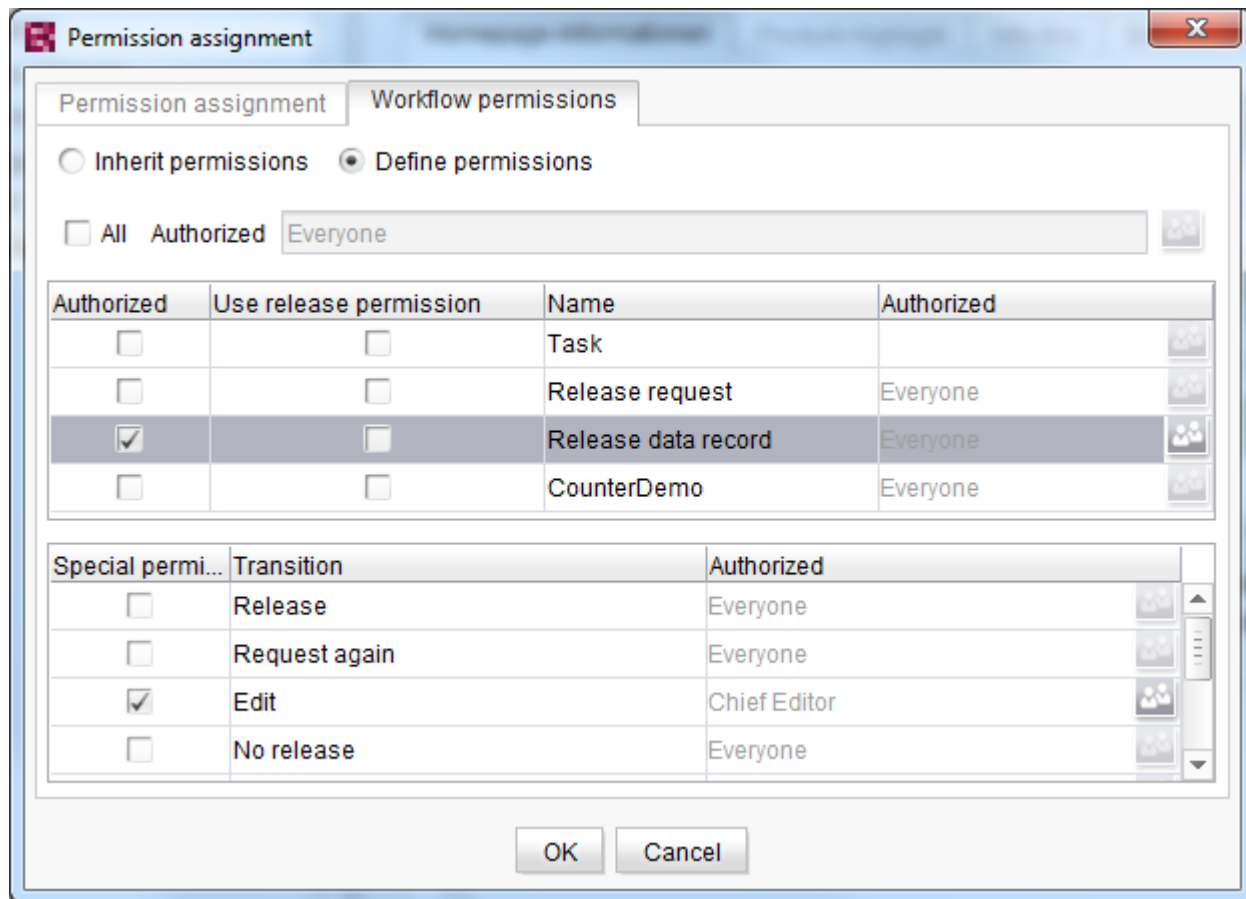


Figure 4-64: Permissions for running workflows

The permissions that are defined in the upper dialog area for running workflows ("Allowed") are related exclusively to starting the respective workflow (see Chapter 4.6.3, page 169).

The permissions for running a transition (from one step of the workflow to the next) are either:

- determined via the template developer in the workflow (see Chapter 4.6.1, page 165)
- or via the assignment of "special permissions" for the individual steps of a workflow (see Figure 4-64) (see Chapter 4.6.4, page 172)

Additional information on permissions for running workflows is located in the FirstSpirit Manual for Editors, Chapter 13.2

If the conventional editorial permissions ("Delete folder", "Delete object") and the permissions to run the workflow are to be suitably connected to one another, the permissions configuration has to be adapted within the workflow. Permissions assignment to the individual transitions is done within a workflow (see Chapter 4.5.5.2, page 162). This ensures that each individual activity can



only be carried out by authorized users. The permissions dialog opens when double-clicking the transition in a workflow's model. The permissions for switching a transition can be assigned on the "Permissions" tab:

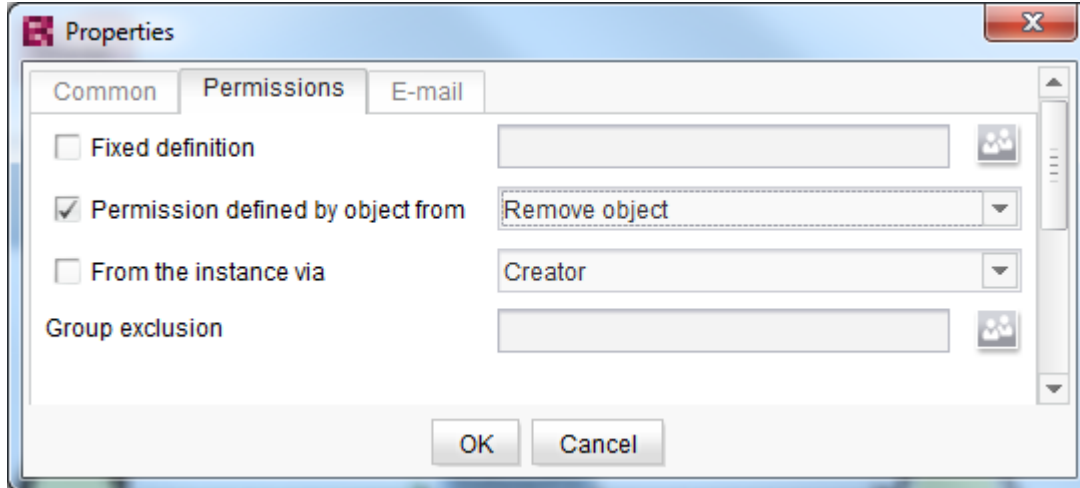


Figure 4-65: Linking editorial permissions and transition permissions

If the option "From the object" is activated, then the authorized users are determined from the editorial permissions, which were defined in the tree structure in JavaClient. In the field, you can select which permissions the user has to have over the object in consideration in order to be able to carry out this transition. If the permission "Delete object" or "Delete folder" is selected, then during workflow startup, whether the user has permission to "Delete" the element is examined. The permissions are then evaluated similarly to conventional deletion without a workflow.

Special case "Delete objects": When deleting an object through a workflow in combination with the permissions configuration via the "From the object" option, a special case applies. If the element is deleted, the permissions can no longer be determined from the object. In the example from 4.9.4, the object and with it the permissions defined for the object on the last transition, "End", is no longer available. In this case, the following applies: on a deleted object, "Everything allowed" is always true. If you do not want this, the permissions configuration for the corresponding transitions have to be changed (for example, on the "fixed, defined" groups or users).



If the "Special permissions" for the advancement of a workflow on an element are defined (see Figure 4-64), the permissions that were defined for the template developer will be overwritten.



4.9.4 Example: "Delete" workflow

The workflow to delete elements consists of the workflow and the corresponding scripts "clientdelete" (to delete individual objects) and "serverdelete" (to delete subtrees).

If the "clientdelete" action is run, the element is blocked by the corresponding script and then deleted. After deletion, the workflow is automatically advanced into the following state, "End".

If the action "server delete" is run, the element is blocked via the corresponding script recursively. Via the "serverdelete" action, not only individual elements, but also subtrees can be deleted. The deletion is done via a ServerHandle, which returns a results report and, in the case of error, throws an exception.

After successful deletion, the workflow is advanced to the following status, "End". For both actions, the following applies: In the event of an error, the workflow is advanced not into the end state, but rather into an error state that was modeled in the workflow.

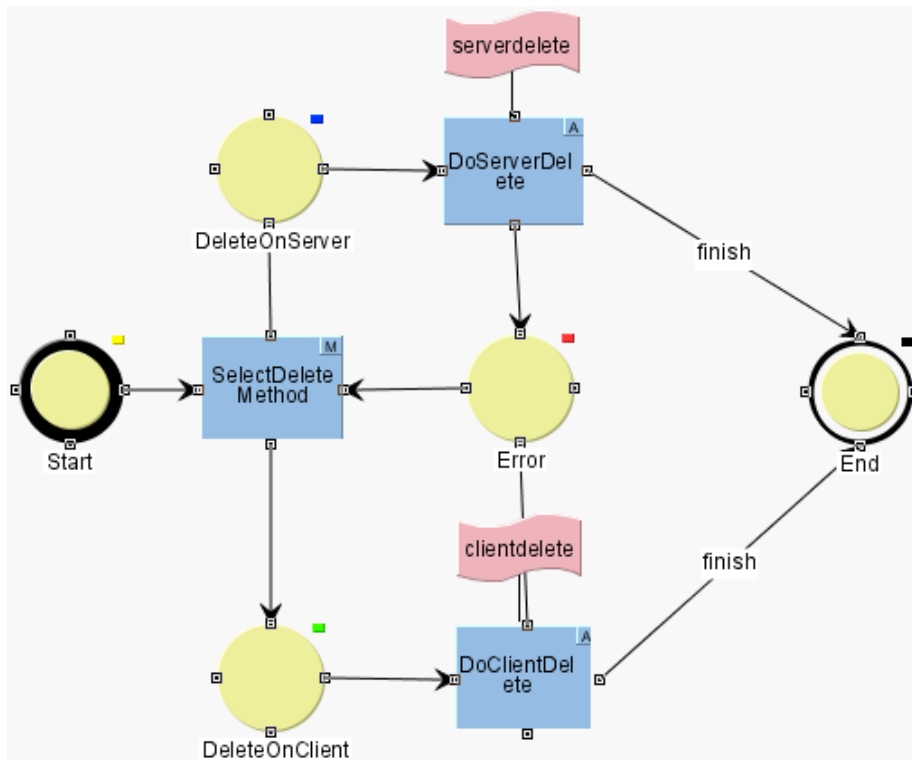


Figure 4-66: Example workflow "Delete"



Script "clientdelete":

```
#!/Beanshell
import de.espirit.firstspirit.common.gui.*;
import de.espirit.firstspirit.ui.operations.RequestOperation;
import de.espirit.firstspirit.agency.OperationAgent;

se = context.getStoreElement();
try {
se.setLock(true);
se.delete();
context.doTransition("->Ende");
} catch (Exception ex) {
text = "Error while deleting: " + ex;
requestOperation =
context.requireSpecialist(OperationAgent.TYPE).getOperation(RequestOperation
.TYPE);
requestOperation.setKind(RequestOperation.Kind.INFO);
requestOperation.addOk();
requestOperation.perform(text);

context.getSession().put("error", ex.toString());
context.doTransition("->Error");
}
```

Script "serverdelete":

```
#!/Beanshell
import de.espirit.firstspirit.common.gui.*;
import de.espirit.firstspirit.ui.operations.RequestOperation;
import de.espirit.firstspirit.agency.OperationAgent;

se = context.getStoreElement();
parent = se.getParent();
try {
se.setLock(false, false);
handle = de.espirit.firstspirit.access.AccessUtil.delete(se, true);
handle.getResult();
}
```



```
handle.checkAndThrow();

Set notDeleted = new HashSet();
progress = handle.getProgress(true);
notDeleted.addAll(progress.getDeleteFailedElements());
notDeleted.addAll(progress.getMissingPermissionElements());
notDeleted.addAll(progress.getLockFailedElements());
notDeleted.addAll(progress.getReferencedElements());
if (!notDeleted.isEmpty()) {
    text = "The following elements could not be deleted: " +
notDeleted;
    requestOperation =
context.requireSpecialist(OperationAgent.TYPE).getOperation(RequestOperation
.TYPE);
    requestOperation.setKind(RequestOperation.Kind.INFO);
    requestOperation.addOk();
    requestOperation.perform(text);
}

if (parent != null) {
    parent.refresh();
    context.getGuiHost().gotoTreeNode(parent);
}

if (!se.isDeleted()) {
    se.setLock(true, false);
}
context.doTransition("->End");

} catch (Exception ex) {
text = "Error while deleting: " + ex;
requestOperation =
context.requireSpecialist(OperationAgent.TYPE).getOperation(RequestOperation
.TYPE);
requestOperation.setKind(RequestOperation.Kind.INFO);
requestOperation.addOk();
requestOperation.perform(text);
```



```
context.getSession().put("error", ex.toString());
context.doTransition("->Error");
}
```

4.9.5 Example: "ContentDeleteDemo" workflow

Alongside deleting individual elements and subtrees (see Chapter 4.9.4, page 193), it is also possible to delete structured data using a workflow.

A distinction is made between normal elements ("StoreElements", such as pages, media, page references) and "Entities" in the workflow (in the script) for this purpose.

In the script, this information is retrieved using a workflow's context (`WorkflowScriptContext`) (see Chapter 4.8.3, page 181):

```
workflowable = context.getWorkflowable()
```

The `getWorkflowable()` method returns, in the form of a data record, whether the element where a workflow was started is a `StoreElement`, such as a media file, or an `Entity` (see example script). The output of a script can be adjusted accordingly, for instance:

```
if (workflowable instanceof ContentWorkflowable) {
...
} else {
...
}
```

In the example, the output is controlled depending on the context in which the workflow was started. If the delete functionality is started on a data record, the script delivers the output:

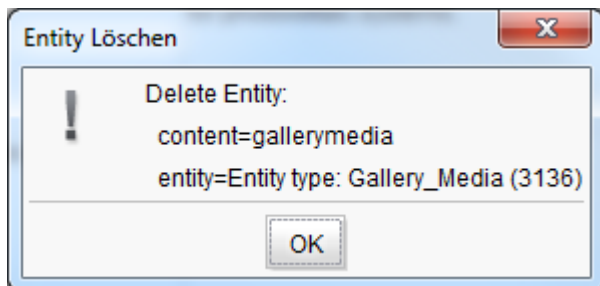


Figure4-67: Delete Entity



If it is started on a "StoreElement", the output is:

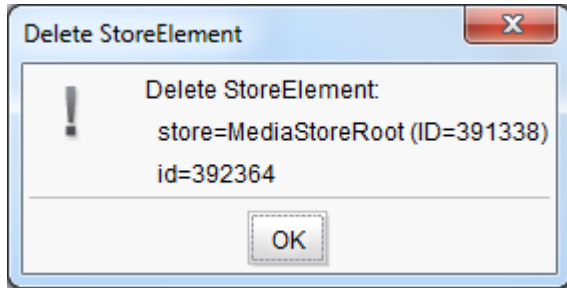


Figure4-68: Delete StoreElement

In this example, data is also deleted directly via the `WorkflowScriptContext` (see Chapter4.8.3, page 181):

```
workflowable.delete();
```

The delete method here is called at the `Workflowable` object in this instance and not, as in the example from Chapter 4.9.4, at the `StoreElement`. Using this Delete method, a `StoreElement` and a data record (`Entity`) can be deleted.

The workflow for deleting entities consists of the workflow and the "deletecontentdemo" script belonging to it (for deleting individual entities)

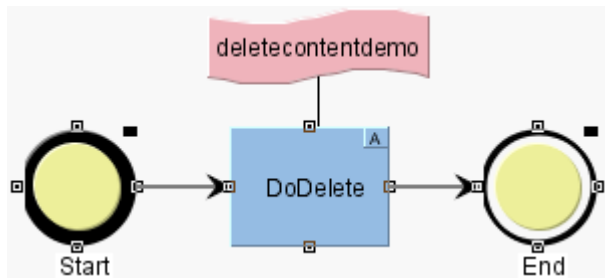


Figure 4-69: Example workflow "DeleteContentDemo"

After being deleted successfully, a workflow is advanced to the subsequent "End" state automatically.



Script ("deletecontentdemo"):

```
#!/Beanshell
import de.espirit.firstspirit.access.*;
import de.espirit.firstspirit.access.store.contentstore.*;
import de.espirit.firstspirit.ui.operations.RequestOperation;
import de.espirit.firstspirit.agency.OperationAgent;

workflowable = context.getWorkflowable();
if (workflowable instanceof ContentWorkflowable) {
message = "Delete Entity:\n  content=" + workflowable.getContent().getName()
+ "\n  entity=" + workflowable.getEntity().getKeyValue();

requestOperation =
context.requireSpecialist(OperationAgent.TYPE).getOperation(RequestOperation
.TYPE);

requestOperation.setKind(RequestOperation.Kind.INFO);
requestOperation.addOk();
requestOperation.perform(message);
}
else {
message = "Delete StoreElement:\n  store=" +
workflowable.getStore().getName() + "\n  id=" + workflowable.getId();

requestOperation =
context.requireSpecialist(OperationAgent.TYPE).getOperation(RequestOperation
.TYPE);

requestOperation.setKind(RequestOperation.Kind.INFO);
requestOperation.addOk();
requestOperation.perform(message);
}

workflowable.delete();
context.doTransition("->End");
```



4.10 Workflows with a complex function

Very complex functions can be implemented using workflow modeling and the BeanShell scripts within these models. A good example is the workflow described previously for deleting elements (see Chapter 4.9.4, page 193).

Within the scripts, actions are run on certain project content using FirstSpirit's Access API. In contrast to standard functions (such as for the "Delete" standard context menu entry), the developer of the workflow (or the associated scripts) has to ensure that all required boundary conditions, such as for deleting an element, are also covered by the script. For most actions, write protection ("Lock") on the affected element is necessary for this at the very least. However, what type of action is to be run on which elements is critical in this context. No write protection has to be set for simply deleting an element such as a media file; however, it does need to be set for recursive deletion such as a page with paragraphs (see Chapter 4.9.4, page 193).

The following chapters deal with write protection within the workflows with complex functionality.

4.10.1 Example: "RecursiveLock" workflow

This workflow for locking subtrees consists of the workflow and the corresponding "lockrecursive" script.

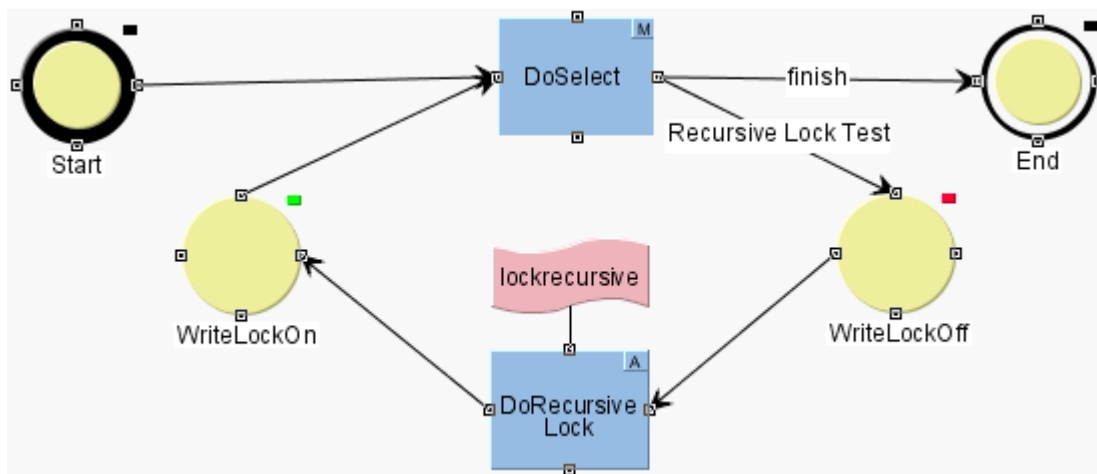


Figure 4-70: "RecursiveLock" example workflow

Within the script, recursive write protection is carried out on the element where the workflow was started. In order for this to work, the write protection for the workflow set automatically



beforehand has to be removed. To do so, write protection in the "WriteLockOff" state has to be removed first:

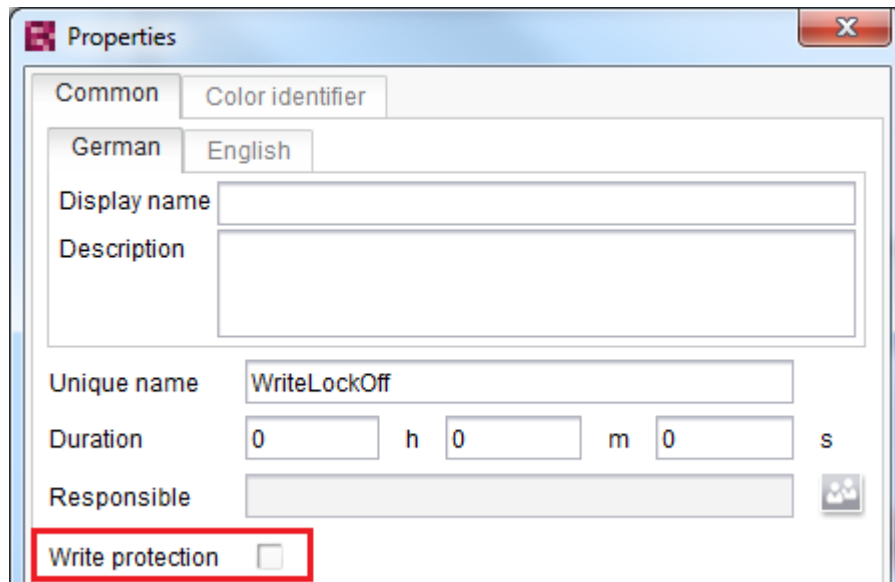


Figure 4-71: Removing write protection using a workflow's state

The "Write protection" checkbox is unchecked; write protection for the workflow is now removed when switching the "Recursive Lock Test" transition. The subsequent action, "DoRecursiveLock", is automatically run and is linked to the "lockrecursive" script.

Recursive write protection can now be set on the element by using the script:

```
// set recursive lock
se = context.getStoreElement();
se.setLock(true);
text = "Subtree locked";
    requestOperation =
context.requireSpecialist(OperationAgent.TYPE).getOperation(RequestOperation
.TYPE);
    requestOperation.setKind(RequestOperation.Kind.INFO);
    requestOperation.addOk();
    requestOperation.perform(text);
```

The elements are locked recursively; the editor is shown a dialog with the message "Subtree locked":



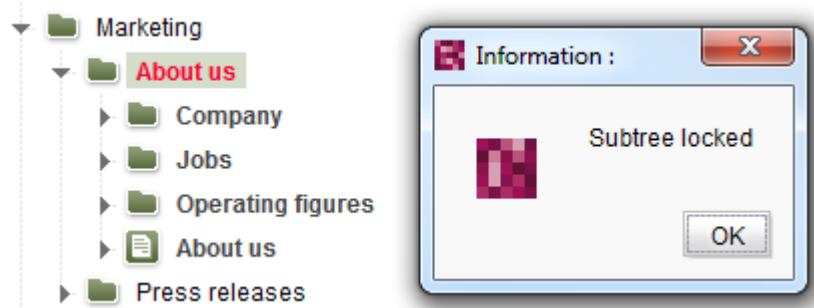


Figure 4-72: Write protection on the subtree

Confirming the message causes the script to continue running, the recursive write protection on the subtree is removed again:

```
// reset recursive lock
se.setLock(false);
```

In the next step, simple write protection has to be restored on the element:

```
// non recursive lock, normal state during workflow
se.setLock(true, false);
context.doTransition("->WriteLockOn");
```

This is necessary to be able to switch the subsequent transition within the workflow.

Then standard write protection for the workflow has to be restored. Additionally, the "Write protection" checkbox is reactivated in the "WriteLockOn" state.

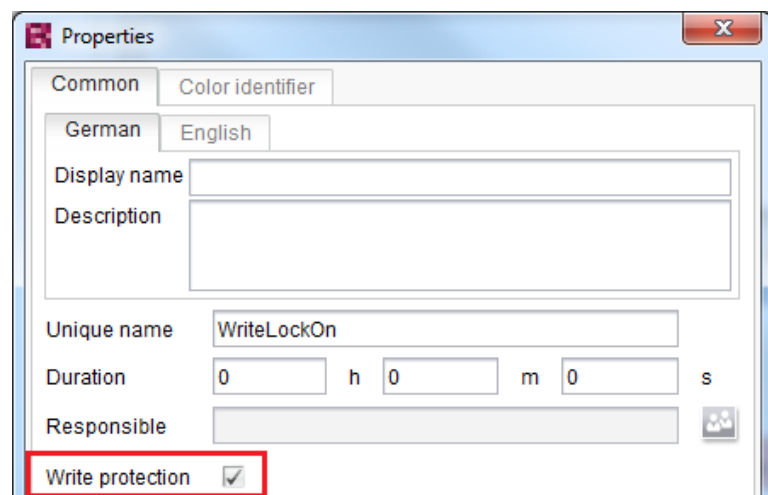


Figure 4-73: Setting write protection using a workflow's state



Script "lockrecursive":

```
#!/Beanshell
import de.espirit.firstspirit.common.gui.*;
import de.espirit.firstspirit.ui.operations.RequestOperation;
import de.espirit.firstspirit.agency.OperationAgent;

// set recursive lock
se = context.getStoreElement();
se.setLock(true);
text = "Subtree locked";
requestOperation =
context.requireSpecialist(OperationAgent.TYPE).getOperation(RequestOperation
.TYPE);
requestOperation.setKind(RequestOperation.Kind.INFO);
requestOperation.addOk();
requestOperation.perform(text);

// reset recursive lock
se.setLock(false);

// non recursive lock, normal state during workflow
se.setLock(true, false);
context.doTransition("->WriteLockOn");
```



4.10.2 Example: "RecursiveRelease" workflow

This workflow for recursive release consists of the workflow and the corresponding "serverrelease" script.

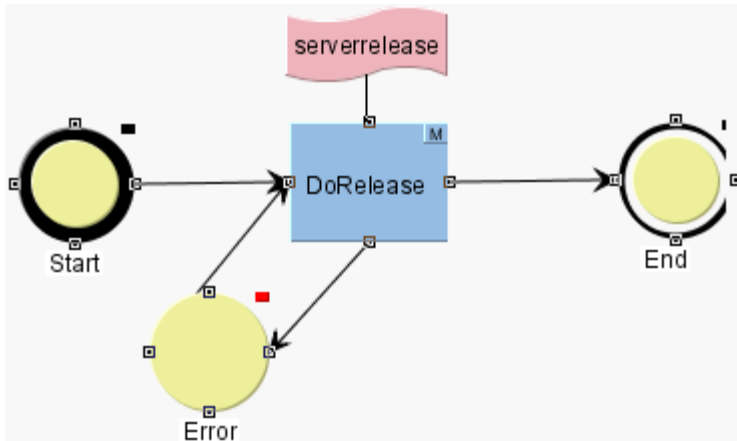


Figure 4-74: "RecursiveRelease" example workflow

The element, where the workflow was started and all dependent elements, is to be released recursively in the workflow.

Server-side release, which is used within the "serverrelease" script, controls the release and the internal setting of write protection for the affected element. If elements are found that already are provided with write protection, the server cannot carry out the server-side release. These affected elements can be retrieved via the return value of the server-side release (only in test mode):

```
handle.getProgress(true).getLockFailedElements()
```

Accordingly, no recursive write protection has to be set for server-side release using the script. So that the release can be made, however, no write protection may be set by the workflow. Therefore, write protection on the element is first removed using the script:

```
se.setLock(false, false);
```

Then server-side release is carried out by calling the method:

```
AccessUtil.release(IDProvider releaseStartNode, boolean checkOnly, boolean
releaseParentPath, boolean recursive, IDProvider.DependentReleaseType
dependentType)
```



In the example, the following transition parameters are configured for server-side release:

```
handle = AccessUtil.release(se, false, false, true,
de.espirit.firstspirit.access.store.IDProvider.DependentReleaseType.DEPENDEN
T_RELEASE_NEW_AND_CHANGED);
```

An explanation of the transfer parameters used can be found in Chapter 6, page 223.

Return parameters:

```
ServerActionhandle<? extends ReleaseProgress, Boolean >
```

The server-side release returns a `ServerActionHandle`, which contains all information about the release process.

Within the sample script, the result of the release process is first queried:

```
handle.getResult();
handle.checkAndThrow();
```

In connection with this, the errors during the release are examined. If elements cannot be released, for example because a write protection exists on the element or the processor does not have the corresponding permissions to release an element, these can be queried via the methods

`progress.getMissingPermissionElements()` OR
`progress.getLockFailedElements()`:

```
progress = handle.getProgress(true);
    notReleased.addAll(progress.getMissingPermissionElements());
    notReleased.addAll(progress.getLockFailedElements());
```

The script error handling shows the editor the elements which could not be released:

```
if (!notReleased.isEmpty()) {
text = "The following elements could not be released: " +
notReleased;

    requestOperation =
context.requireSpecialist(OperationAgent.TYPE).getOperation(RequestOperation
.TYPE);

    requestOperation.setKind(RequestOperation.Kind.ERROR);
    requestOperation.addOk();
    requestOperation.perform(text);
}
```



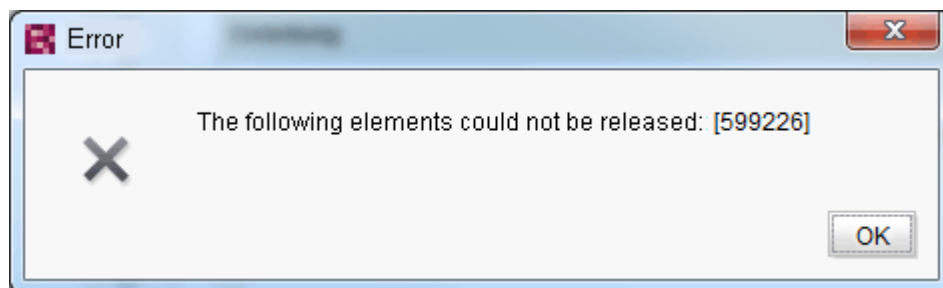


Figure 4-75: Error message – Unreleased elements



The error message is shown in test mode only ("checkOnly").

Script "serverdelete":

```
#!/Beanshell
import de.espirit.firstspirit.common.gui.*;
import de.espirit.firstspirit.access.*;
import de.espirit.firstspirit.access.store.*;
import de.espirit.firstspirit.ui.operations.RequestOperation;
import de.espirit.firstspirit.agency.OperationAgent;

se = context.getStoreElement();
try {
    se.setLock(false, false);
    handle = AccessUtil.release(se, false, false, true,
de.espirit.firstspirit.access.store.IDProvider.DependentReleaseType.DEPENDEN
T_RELEASE_NEW_AND_CHANGED);
    handle.getResult();
    handle.checkAndThrow();
    Set notReleased = new HashSet();
    progress = handle.getProgress(true);

    notReleased.addAll(progress.getMissingPermissionElements());
    notReleased.addAll(progress.getLockFailedElements());
    if (!notReleased.isEmpty()) {
        text = "The following elements could not be released: " +
notReleased;
    }
}
```



```
        requestOperation =
context.requireSpecialist(OperationAgent.TYPE).getOperation(RequestOperation
.TYPE);

        requestOperation.setKind(RequestOperation.Kind.ERROR);
        requestOperation.addOk();
        requestOperation.perform(text);
    }
se.refresh();
context.getGuiHost().gotoTreeNode(se);
se.setLock(true, false);
context.doTransition("->End");
} catch (Exception ex) {
text = "Error while releasing: " + ex;
requestOperation =
context.requireSpecialist(OperationAgent.TYPE).getOperation(RequestOperation
.TYPE);
requestOperation.setKind(RequestOperation.Kind.ERROR);
requestOperation.addOk();
requestOperation.perform(text);

context.getSession().put("error", ex.toString());
context.doTransition("->Error");
}
```



4.11 Multiple workflow selection

In FirstSpirit, many dialogs have the option to select and edit elements multiple times. A multiple selection is, for example, possible within the tree view in the FirstSpirit JavaClient (see Figure 4-76). With it, multiple elements can be marked on which a certain action (such as moving, copying, deleting) can also be run.

It is possible to select multiple items at once by pressing the SHIFT or CTRL key at the same time. Moreover, the key combination STRG + A can be used to select all visible elements of a store area (within the tree view) or all elements within a table (for example, within the task list).

The multiple selection of elements within the tree view is limited to the current store area. In other words, for example, if an element is marked in the page store, then afterward, no other element from a different store area can be selected.



If the key combination CTRL + A is used within the tree view of the FirstSpirit JavaClient, only the currently visible (expanded) elements of the tree view are marked. If, for example, a folder in the page store is not expanded, the pages under it are not a part of the selection.

4.11.1 Multiple workflow selection

The selection of multiple workflows enables starting and switching a workflow for a quantity of objects.

To do so, the desired objects can be marked within the tree view. In connection with this, the context menu is opened as usual and the desired workflow is selected.



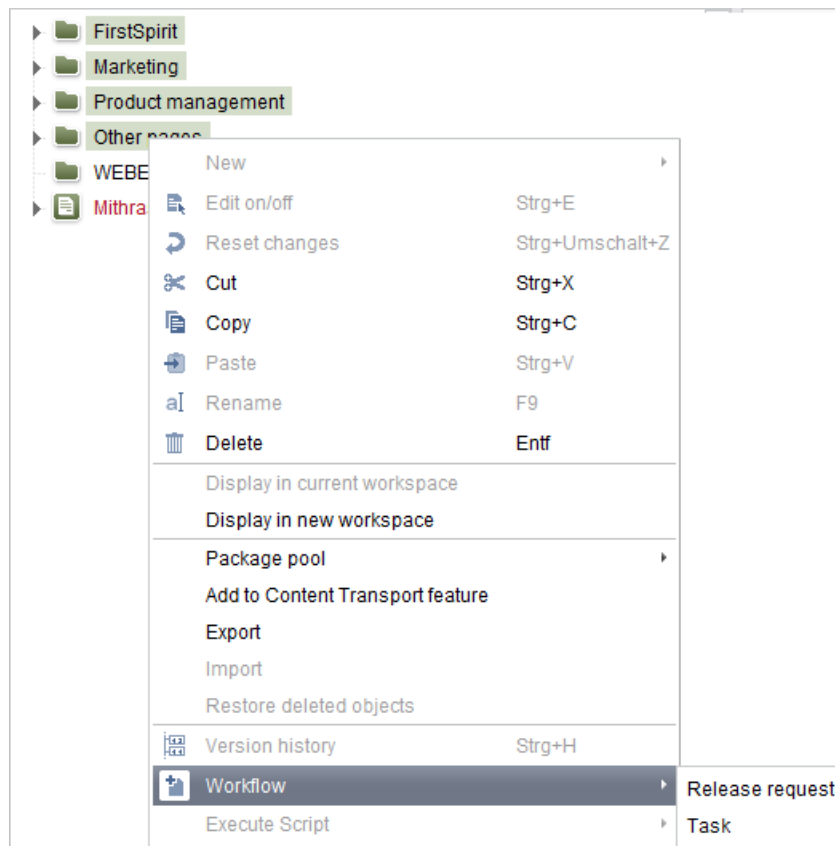


Figure 4-76: Starting a workflow on multiple elements

4.11.2 Requirements for starting and advancement

Starting or advancing a workflow can only happen if all elements have the same status of the workflow or, until now, no workflow has been started on the selected elements (see Figure 4-76).

During multiple element selection, for every element, it is determined which workflows and which transitions of a workflow can be shown via the context menu. In this, for example, it is taken into account whether:

- a workflow was already started on the element,
- the user has the required permissions to start the workflow on this element,
- a workflow may be started on this element,
- a workflow was already started on the elements, but the elements did not reach the same state as the workflow.

If these requirements are not fulfilled for even one element of the multiple selection, the context



menu delivers the evaluation "not available" for all selected elements.



Figure 4-77: Context menu – not available

In this case, the multiple selection should be canceled, the individual elements checked again and reselected where necessary.

4.11.3 Multiple selection via the task list

Alongside the multiple selection via the tree view, workflows started once can also be moved forward via the task list (see Figure 4-78) or via the "Workflows" overview in the template store (see Chapter 4.1, page 127).

The task list provides all tasks not yet completed ("open tasks") and all started tasks ("initiated tasks") within a table view. Alongside the name of the workflow, the current state of the respective instance of the workflow is shown here.

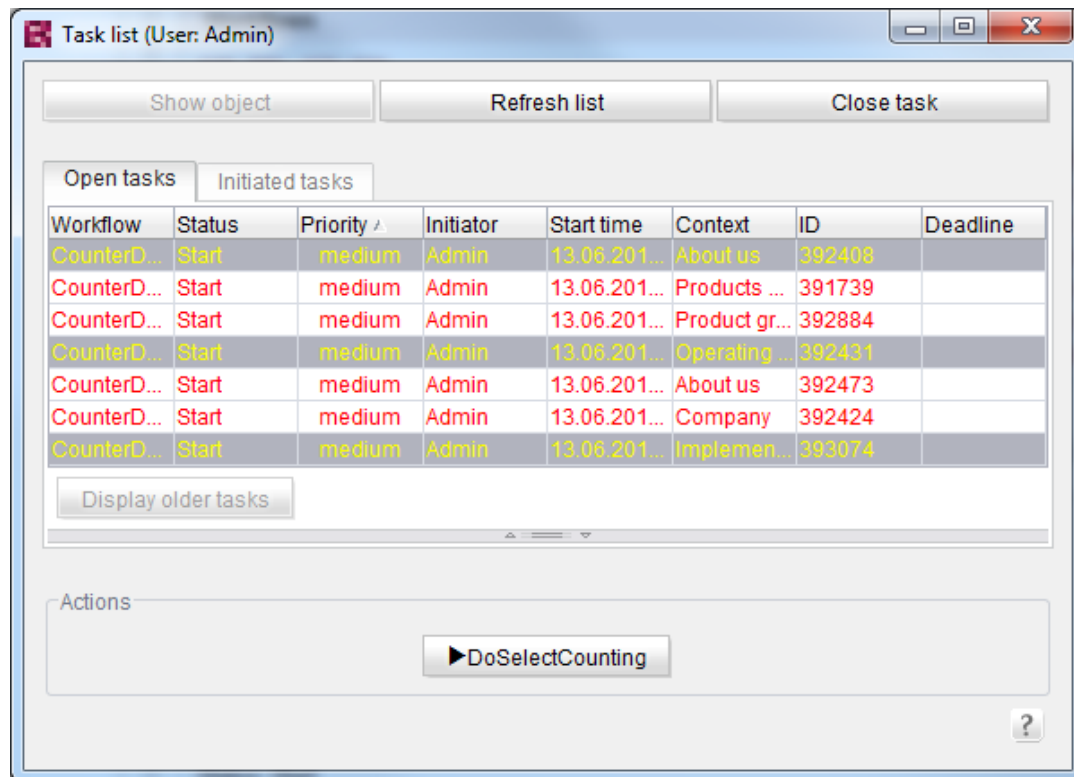


Figure 4-78: Multiple selection via the task list

Within the tabular task list, multiple tasks can be selected. As long as these tasks have the same



workflows and the same status and the user has the required permissions to run the workflow for the selected elements, the possible actions are shown in the lower area of the task list.

The tasks can thus be advanced simultaneously. Unlike when starting via the tree view, multiple elements from different administrative areas can be marked and switched simultaneously in the task list.

4.11.4 Multiple selection via the "Workflows" overview

Alongside the task list, within the template store there is an additional overview on the "Workflows" node over all previously started tasks (see Chapter 4.1, page 127). Differently than in the task list, the tasks can be filtered here according to specific search criteria and also shown according to already concluded tasks (see Chapter 4.1.1, page 129).

Within the overview, multiple tasks can be selected. Directly advancing the workflow is not possible here; however, clicking the "Edit" button opens the task list (see Chapter 4.1.2, page 131). The elements previously marked in the task list are selected directly and can be advanced there (see Chapter 4.11.3, page 209).

Alongside editing, the overview can close multiple selected tasks (see Chapter 4.1.3, page 132).



5 Tracking changes via revision-metadata

FirstSpirit provides an option for tracking changes via the FirstSpirit Access API. Access to the revision metadata is possible via certain API functions (see Chapter 5.2, page 213). Revision metadata contains information on the type (which changes took place?) and the scope (which elements were changed?) of a change to the project. The information which is made available via the revision metadata is highly granular. With changes to content, it can be determined, for example, which properties of an element were changed, for example whether the content was changed in a certain editing language, whether a child element was added or removed or whether certain attributes, such as permissions to the corresponding element, were changed.

Via the extended revision information, all changes that were made from the time of a certain revision to the time of a certain revision within a project can be determined.

Access to this information is possible via the FirstSpirit Access API, for example by BeanShell Script.

The following chapters present methods in order to obtain one or more revisions of a project which are to be examined based on changes (see Chapter 5.1, page 212) and methods to determine the corresponding information on the changes (see Chapter 5.2, page 213). Independent of respective change type, different metadata information is available (see Chapter 5.2.1, page 213).

Additionally, examples of using track changes are described in the project.

The first example determines all the database changes which took place since the last released revision of a project (see Chapter 5.3, page 215).

The second example determines changes to the content that took place between a to-be-defined start revision and a to-be-defined end revision in the project (see Chapter 5.4, page 219).

All code excerpts in this chapter involve fragments, which are insufficient to put together the entire script!



5.1 Get revision

FirstSpirit works with a revision-based repository. With it, special administration of chronological development of data is used. This is revision management.

A revision can be presented as a type of "Snapshot" across the entire repository at a certain point in time. In contrast to a version, which is usually only related to a single object, during a revision, the total state of all objects in the repository are listed.

Revisions are listed with sequential numbering (revision ID), where there is always exactly one current revision for the whole repository. If a repository is edited, all changes carried out are linked to a new revision number. The revision number is the last current revision number of the entire repository, increased by one. All unchanged objects retain their old revision numbers. If an object is changed, it is not overwritten in the repository, but rather inserted as a new object (with a higher revision number).

In order to determine in what time frame certain changes took place within the program, first the corresponding revision for the repository has to be retrieved.

The revision can be retrieved directly via the project. In doing so, either the desired unique revision ID, such as:

```
project.getRevision(revisionId);
```

or the date of the desired revision were transferred, such as:

```
project.getRevision(context.getStartTime());
```

The transitioned date does not have to be assigned to a unique revision. Any desired data value can be transferred. If at this date a revision exists, it is returned; otherwise the methods return the next lowest revision.

A selection of revisions within a certain time frame can be made available via the method:

```
project.getRevisions(Revision from, Revision to, int maxCount,  
Filter<Revision> filter);
```

made available. Two revisions are transferred. The first revision ("from") defines the lower revision limit and the second revision ("to") defines the upper revision limit. Along with both of these revisions, all revisions which have a higher revision ID than the lower revision limit and a lower revision ID than the upper revision limit are returned.



The respective most current version can be retrieved via:

```
getRevision(new Date());
```

via:

```
start = project.getRevision(context.getStartTime());
end = project.getRevision(new Date());
revisions = project.getRevisions(start, end, 0, null);
```

Optionally, the parameter "maxCount" can be passed, which limits the number of the returned revisions to a highest value, and – likewise optionally – provides a filter for further limitation.

5.2 Determining changes to a revision

Via the revisions (see Chapter 5.1, page 212), in connection with the metadata, additional information on the changes can be retrieved:

```
revision.getMetaData();
```

The metadata administers different information which is dependent on the type of the changes respectively (see Chapter 5.2.1, page 213). In the process, language-dependent content changes to an element are taken into account as well as structural changes (such as a move) or a change to the element attributes (such as name, permission definition, etc.) (see Chapter 5.2.2, page 214).

5.2.1 Determining the type of change

The changes that have taken place in a revision can be retrieved via:

```
metaData.getOperation();
```

via:

The provided revision operation (`RevisionOperation`) provides information on the type of change (`RevisionOperation.OperationType`), for example:

```
operation.getType();
```

In this context, different types of changes are available for different project contents.

For contents of type `IDProvider`, the following types of changes are possible:

- **CREATE** an object has been newly created in the project
- **MODIFY** An object has been modified in the project



- **MOVE** An object has been moved in the project
- **DELETE** An object has been deleted in the project
- **RELEASE** An object has been released in the project
- **SERVER_RELEASE** An object has been released on the server

The corresponding revision operation (for example, `ModifyOperation`) provides an object of type `BasicElementInfo` with additional information on the respective object (for example, the `UniqueIdentifier`).

For contents of type `Entity`, the following types of changes are possible:

- **CONTENT_COMMIT** Database contents have been modified

The corresponding revision operation (for example, `ContentOperation`) returns an object of type `EntityInfo` with additional information on the respective data records (for example, the ID of the data record or the ID of the associated database schema).

5.2.2 Determining changed elements

Depending on the respective change operation, additional information on changes can be called up, such as which data records have been released in the project (for operation type: **CONTENT_COMMIT**):

```
operation.getReleasedEntities();
```

or, for example, which contents have been newly created in the project (for operation type: **CREATE**):

```
operation.getCreatedElement();
```

Additional methods are in the examples in the two following chapters (see Chapter 5.3 and Chapter 5.4).

For an overview of all available methods, see documentation on the FirstSpirit Access API ⁵.

⁵ Via the FirstSpirit Online Documentation in the Template development – FirstSpirit API area



5.3 Changes since the last deployment

The changes in the project carried out since the time of the deployment are to be shown in the first example. The following example involves a project in which the reports on the FirstSpirit content store can be managed. An overview of the changes to the data records is to be made available each time a project is deployed. A post-deployment script is first created in the deployment schedules to determine the changes.

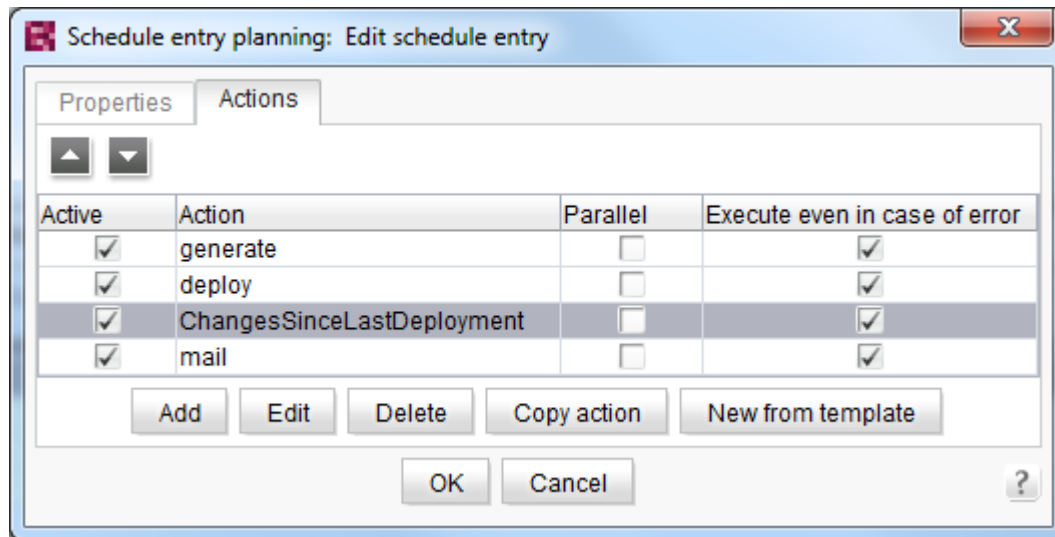


Figure 5-1: Configuration of the post-deployment script

The script first determines the ID of the revision which was current at the time of the last deployment of the project:

```
task = context.getTask();
lastExecutionRevisionId = (Long) context.getVariable(task.getName() + ".revision");
if (lastExecutionRevisionId != null) {
    context.logInfo("revision of last execution=" + lastExecutionRevisionId);
    revId = lastExecutionRevisionId.longValue();}
```



Subsequently, all revisions of the project since the last deployment are retrieved. The revision with the just-determined revision ID is retrieved as the lower revision limit ("startRev"). The current revision at the time of deployment is determined as the upper revision limit:

```
startRev = project.getRevision(revId);
endRev = project.getRevision(context.getStartTime());
context.logInfo("startRev=" + startRev.id + ", endRev=" + endRev.id);
if (startRev.id == endRev.id) {
    context.logInfo("no changes detected");
}
revisions = project.getRevisions(startRev, endRev, 0, null);
```

All of the determined revisions are then examined for changes in a loop:

```
checkChanges(revisions) {
    for (revision : revisions) {
        metaData = revision.getMetaData();
        operation = metaData.getOperation();
        if (operation != null) {
            type = operation.getType();
            switch (type) {
                case RevisionOperation.OperationType.CONTENT_COMMIT:
                    ..
                    ..
                    break;
            }
        }
    }
}
```

In the process, only changes to database content – in other words, of operation type CONTENT_COMMIT – are to be taken into account, and only the newly created and changed data records of a specific database table.

All newly generated and released data records are first determined via:

```
createdEntities = operation.getCreatedEntities();
releasedEntities = operation.getReleasedEntities();
```

This selection is then restricted to a specific database table (here: MyEntityType):

```
ENTITY_TYPE = "MyEntityType";
```




```

if (ENTITY_TYPE.equals(created.getEntityTypeName())) {
    ..
}
if (ENTITY_TYPE.equals(released.getEntityTypeName())) {
    ..
}

```

Related:

```

case RevisionOperation.OperationType.CONTENT_COMMIT:
    createdEntities = operation.getCreatedEntities();
    for (created : createdEntities) {
        if (ENTITY_TYPE.equals(created.getEntityTypeName())) {
            createdCertificates.put(created.getEntityId(), revision);
            context.logInfo("\t created entity " + created.getEntityId() + " in
revision " + getRevisionString(revision));
        }
    }
    releasedEntities = operation.getReleasedEntities();
    for (released : releasedEntities) {
        if (ENTITY_TYPE.equals(released.getEntityTypeName())) {
            releasedCertificates.put(released.getEntityId(), revision);
            context.logInfo("\t released entity " + released.getEntityId() + " in
revision " + getRevisionString(revision));
        }
    }
    break;

```

The required information (e.g. review numbers) is retrieved using the IDs of the data records determined to be changed and released ("created" and "released") and then saved for further use.

```

context.setProperty("created", createdList);
context.setProperty("updated", updatedList);

```

In the process, the values saved via `context.setProperty(..)` are only persistent within the current schedule; this means they can continue to be used in a subsequent action in the schedule with `context.getProperty(..)`. In this example, the contents continue to be used in the following "Mail" action (see Figure 5-1) in the e-mail template:

```

Hello,

$CMS_SET(created, #context.getProperty("created"))$CMS_SET(updated,
#context.getProperty("updated"))$

```



```

new(created != null)$($CMS_VALUE(created.size)$)$CMS_END_IF$ and
modified$CMS_IF(updated != null)$($CMS_VALUE(updated.size)$)$CMS_END_IF$
certificates have been published on
http://www.gutachten-online.de

$CMS_IF(created.size > 0)$New certificates:
=====

$CMS_FOR(entity, created)$ * $CMS_VALUE(entity.Gutachtennr)$
($CMS_VALUE(entity.Datum.format("dd.MM.yy"))$) - $CMS_VALUE(entity.Kennzeichen)$
$CMS_END_FOR$$CMS_END_IF$

$CMS_IF(updated.size > 0)$Updated certificates:
=====

$CMS_FOR(entity, updated)$ * $CMS_VALUE(entity.Gutachtennr)$
($CMS_VALUE(entity.Datum.format("dd.MM.yy"))$) - $CMS_VALUE(entity.Kennzeichen)$
$CMS_END_FOR$$CMS_END_IF$
--

This is an automatically generated e-mail which is sent when new certificates are
published.

If you have any questions, please contact info@gutachten-online.de

```

The template now generates an e-mail when carrying out a deployment schedule with the generated and modified contents:

Example (e-mail):

```

Hello,

new(4) and modified(2) certificates have been published on
http://www.gutachten-online.de

New certificates:
=====

* AZ33048/D (10.08.10) - DO-WZ 1234
* AZ45134/D (10.08.10) - DO-XY 4321
* AZ46200/D (11.08.10) - EN-AA 1111
* AZ50261/D (13.08.10) - BO-YZ 5566

Updated certificates:
=====

```



```
* AZ44356/D (10.08.10) - DO-ZZ 3388
* AZ47709/D (05.05.08) - D-YY 9999
--
This is an automatically generated e-mail which is sent when new certificates are
published.
If you have any questions, please contact info@gutachten-online.de
```

Content saved via `context.setVariable(..)` also continues to be persistent while running the current schedule (in contrast to saving content via `context.setProperty(..)`). This option is used in the example in order to save the revision at the time of the current schedule:

```
context.setVariable(task.getName() + ".revision",
new Long(endRev.getId()));
```

When starting the next schedule, this information can then be used to retrieve the revision ID which was current at the time of the last deployment of the project:

```
lastExecutionRevisionId = (Long) context.getVariable(task.getName() + ".revision");
```

The complete script and the templates described here can be requested as needed via the FirstSpirit help desk.

5.4 Changes between two revisions

In the second example, the revisions can be selected conveniently via a GUI. To do so, a new script has to be created in the project's template store first.

Input components for selecting a start and an end date for the desired revision limits can be configured in the form area of the script:



Choose Time Range

Start Date

9/3/12 12:10 PM

End Date

The editor must not be empty!

Please correct your input!

Date

September 2012

CW	Su	Mo	Tu	We	Th	Fr	Sa
35							1
36	2	3	4	5	6	7	8
37	9	10	11	12	13	14	15
38	16	17	18	19	20	21	22
39	23	24	25	26	27	28	29
40	30						

Today is Tuesday, September 11, 2012

Time 12 : 10 : 41

OK Cancel

Figure 5-2: GUI for selecting the revision limits

The XML file for configuring the form area can be requested via the FirstSpirit help desk as needed.

Similarly to the first example, the corresponding revisions can then be retrieved using the selected data:

```
data = context.showForm();
if (data != null) {
    context.logInfo("data=" + data);
    from = data.get(context.getProject().getMasterLanguage(),
        "from").get();
    to = data.get(context.getProject().getMasterLanguage(),
        "enddate").get();

    if (from != null) {
        context.logInfo(from + " -- " + to);
        start = project.getRevision(from);
        end = project.getRevision(to);
        context.logInfo("startRev=" + start.id + ", endRev=" + end.id);
    }
}
```



```
        if (start.id <= end.id) {
            revisions = project.getRevisions(start, end, 0, null);
        } else {
            revisions = project.getRevisions(end, start, 0, null);
        }

        context.logInfo("found '" + revisions.size() + "' revisions -> first=" +
revisions.get(0) + ", last=" + revisions.get(revisions.size() - 1));

        checkChanges(revisions);
    }}
```

In `checkChanges` (see Chapter 5.3, page 215), the changes that took place in this revision are found. In this example, changes to project content of operation types DELETE and CREATE (within the page store) are taken into account. In addition, changes to the project's database content are determined.

Changes due to removing elements in the page store:

```
case RevisionOperation.OperationType.DELETE:
    deleteRoot = operation.getDeleteRootElement();
    if (deleteRoot.getStoreType() == Store.Type.PAGESTORE) {
        // include only pagestore
        context.logInfo("found delete in pagestore (deleted node=" +
deleteRoot.getUid() + ") in revision=" +
getRevisionString(revision));
    }
    break;
```

Changes due to adding elements in the project:

```
case RevisionOperation.OperationType.CREATE:
    created = operation.getCreatedElement();
    parent = operation.getParent();
    context.logInfo("found created element in store '" +
created.getStoreType() + "' (created node=" +
created.getUid() + ", parent node=" + parent.getUid() + ") in
revision=" + getRevisionString(revision));
    break;
```

Changes due to creating, deleting, changing or releasing database content:

```
case RevisionOperation.OperationType.CONTENT_COMMIT:
    created = operation.getCreatedEntities();
```



```
changed = operation.getChangedEntities();
deleted = operation.getDeletedEntities();
released = operation.getReleasedEntities();
context.logInfo("found content changes in revision=" +
    getRevisionString(revision));
context.logInfo("\t created entities(" + created.size() + ") "
    + created);
context.logInfo("\t changed entities(" + changed.size() + ") "
    + changed);
context.logInfo("\t deleted entities(" + deleted.size() + ") "
    + deleted);
context.logInfo("\t released entities(" + released.size() + ") "
    + released);
break;
```

The script is output via the Java console:

```
..
INFO 20.05.2008 15:44:50.317
startRev=6804, endRev=7677
found created element in store 'PAGESTORE' (created node=Testpage_131,
parent node=Test_6C22873) in revision=7335
- Thu May 15 16:02:51 CEST 2008 (importStoreElement) - Admin - CREATE
..
```



6 Server-side release

In addition to release via a workflow, all objects in FirstSpirit can be released server-side via the Access API. To do so, there are methods of defining the different release settings for an object. In this way, the specific release can be used to release additional objects dependent on the current object, such as the complete parent chain and child elements of the object to be released.

In general, a distinction is made among the following release options:

- Standard release (see Chapter 6.1, page 224):
Release for the object to be released, including additional, defined release options for the default case. These predefined release options are different depending on the object. In this way, a page in the page store is released via the default release options, including the subordinated sections and the parent elements that have never been released. In contrast, the default release of a page reference in the site store only takes the page reference itself into account. The default release options cannot be changed.
- Specific release (see Chapter 6.2, page 224):
Release for the object to be released, including optional release options established by the user. The different release options can be combined with each other in any way to realize a comprehensive release within a short time. However, the release of all objects involved in the release process is may be undesirable in certain circumstances and therefore should be carried out with caution.



6.1 Default release

The release of the current object (for example, page or folder of the page store), including defined, object-dependent release options for the default case, is carried out using this option.

Direct release of an object is carried out directly using the following API method:

```
AccessUtil.release(IDProvider toRelease, boolean checkOnly)
```

Transition parameters:

`toRelease`: Element to be released

`checkOnly`: If the value `true` is passed, the default release is just tested. The objects to be released are not transferred to the release state. Instead, for example, the standard release is run in order to discover errors prior to the actual release of an object.

Return parameters:

```
ServerActionhandle<? extends ReleaseProgress, Boolean >
```

The server-side release returns a `ServerActionHandle`, which contains all information on the release process and, for example, contains the status of the release or the log info.

6.2 Specific release

The specific release takes even more (dependent) objects into account in the release process, depending on the release parameters defined.

- **Ensure accessibility (parent chain):** Starting at the selected object, all new (never released), higher-level nodes are also released (see Chapter 6.2.4, page 231). This option makes sense, for example, if a new page within a new folder was created in the page store and both are to be released together. In contrast to recursive release, other new pages below the folder would not be released. While the combination of this option with the "recursive release" option remains limited to the current store (see Chapter 6.2.5, page 233), it also has an effect on the parent chain of the dependent objects and, thus, on other stores in combination with the "Dependent release" option (see Chapter 6.2.6, page 234).
- **Recursive release:** Depending on the object selected, all subordinate nodes are also released. This selection makes sense, for example, if within a folder in the page store many



pages were changed and now all changes are to be released together. This option remains limited to the current store area (see Chapter 6.2.1, page 227).

- **Only release new dependent objects:** Starting from the selected object, all objects that are dependent on the selected object (for example, a media file used in an image input component) and that have not yet been released (newly created objects) are also released. If this release option is combined with other options (for example, the release of the parent chain), the dependent release also has an effect on other objects and stores included in the release process.
- **Release new and changed dependent objects:** Starting from the selected object, all objects that are dependent on the selected object (for example, a media file used in an image input component) are also released. Objects that have never been released (newly created objects) and objects that have been reedited after being successfully released (changed objects) are taken into account in the process. If this release option is combined with other options (for example, the release of the parent chain), the dependent release also has an effect on other objects and stores included in the release process.

Specific release of an object is carried out using the following API method:

```
AccessUtil.release(IDProvider releaseStartNode, boolean checkOnly,  
boolean releaseParentPath, boolean recursive, IDProvider.DependentReleaseType  
dependentType)
```

Transition parameters:

`releaseStartNode`: Start nodes for the release

`checkOnly`: If the value `true` is passed, the specific release is just tested. The objects to be released are not transferred to the release state. Instead, the defined release options are run in order to discover errors before the real release, for instance.

`releaseParentPath`: If the value `true` is passed, the complete parent chain of the object to be released is determined and all objects that have never been released before are also released. If the option `releaseParentPath=false` is set, the parent chain is not released; the elements to be released are, however, added to the release child list of the parent node. The following applies here:

- With changed parent nodes: The object to be released can be accessed in release state. The parent element, however, is not released.
- With new parent nodes: Because the parent node was never released, the object to be released in the release state cannot be reached. That can lead to invalid references in the release state (see Chapter 6.2.4 and Chapter 6.2.5).



`recursive`: If the value `true` is passed, all child elements of the object to be released are recursively determined and likewise released. If the value `false` is passed, the child elements are not taken into account during release (see Chapter 6.2.1, Chapter 6.2.3 and Chapter 6.2.5).

`dependentType`: Objects dependent on the object to be released are determined and also released using this parameter. If, for example, a media file is referenced on a page, this media file can also be released directly during specific release of the page. The following dependencies can be taken into account (see Chapter 6.2.2, Chapter 6.2.3 and Chapter 6.2.6):

- `DEPENDENT_RELEASE_NEW_AND_CHANGED`: New and modified dependent objects are taken into account.
- `DEPENDENT_RELEASE_NEW_ONLY`: Only newly created (not yet released objects) are taken into account
- `NO_DEPENDENT_RELEASE`: Dependent objects are not taken into account and have to be released separately if necessary (default setting).

The different release options can be combined with each other in any way to realize a comprehensive release within a short time. However, the release of all objects involved in the release process is may be undesirable in certain circumstances and therefore should be carried out with caution.

The server-side release will therefore be explained in the following chapters based on some examples.

Return parameters:

```
ServerActionhandle<? extends ReleaseProgress, Boolean >
```

The server-side release returns a `ServerActionHandle`, which contains all information about the release process.



6.2.1 Recursive release

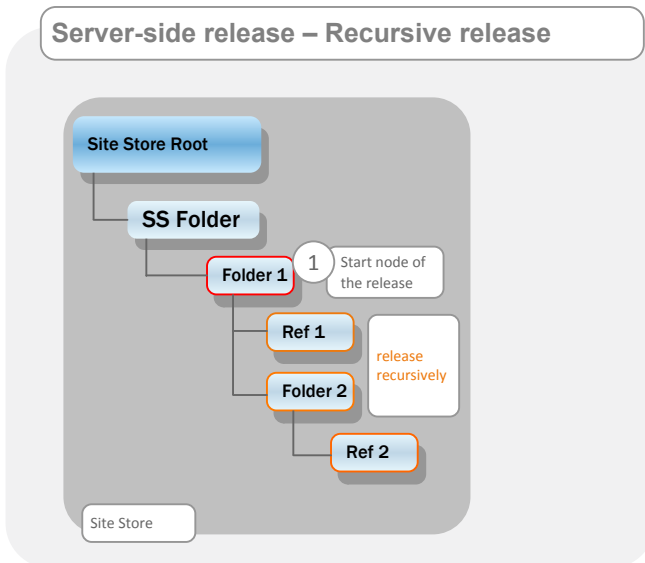


Figure 6-1: Server-based release – recursive

For calling `AccessUtil.release(...)` the following parameters were set:

```
releaseStartNode: Folder 1
releaseParentPath: false
boolean recursive: true
DependentReleaseType: NO_DEPENDENT_RELEASE
```

The selected start node for the release is the "Folder 1" menu level.

Recursive release: At the start point of the "Folder 1" release, the option `recursive` is evaluated. The recursive release has an effect *solely* on the child elements of the release start point. Thus the "Ref 1", "Folder 2" and "Ref 2" child elements are released by the option in the example from Figure 6-1.

Recursive releases of additional dependent elements are not run, even in combination with other release options. The recursive release thus does *not* have an effect on the release of child elements of dependent objects in other stores.



6.2.2 Dependent release

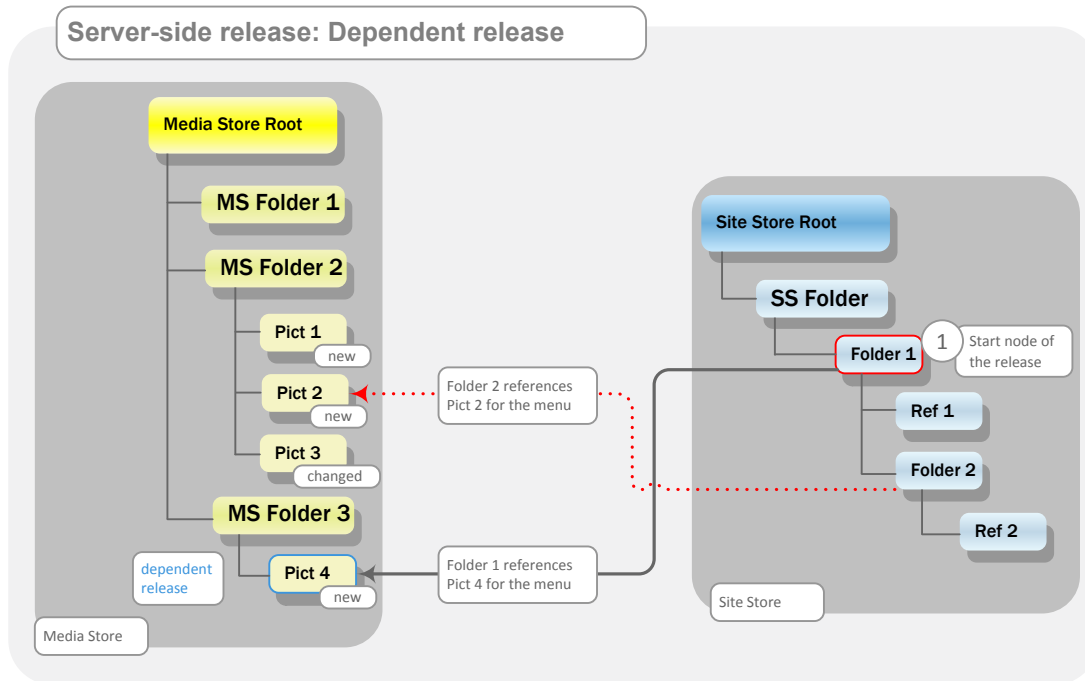


Figure 6-2: Server-side release – only new or new and modified releases

For calling `AccessUtil.release(...)` the following parameters were set:

```
releaseStartNode: Folder 1
releaseParentPath: false
boolean recursive: false
DependentReleaseType:
DEPENDENT_RELEASE_NEW_AND_CHANGED||DEPENDENT_RELEASE_NEW_ONLY
```

The selected start node for the release is the "Folder 1" menu level.

Dependent release: The options `DEPENDENT_RELEASE_NEW_ONLY` and `DEPENDENT_RELEASE_NEW_AND_CHANGED` have an effect on basically *all* dependent objects in the page store, the site store and the media store. This release option thus does not just affect start nodes, but rather all objects that are taken into account during the release process. All outgoing references in the "Folder 1" menu level are examined and released by the option in the example from Figure 6-2. If only the dependent release is activated (without recursive release), only "Pict 4" would be conditionally released (see Figure 6-2); if additional release options are activated, the release can be substantially more extensive, however (see Chapter 6.2.3, page 229, Chapter



6.2.6, page 234 and Chapter 6.2.7, page 236).

! All outgoing references for the dependent release are only completely taken into account in one direction. If all dependent objects are to be included in the release process, then the release has to be carried out in a certain order (see Chapter 6.2.8, page 238).

6.2.3 Dependent release with recursive release

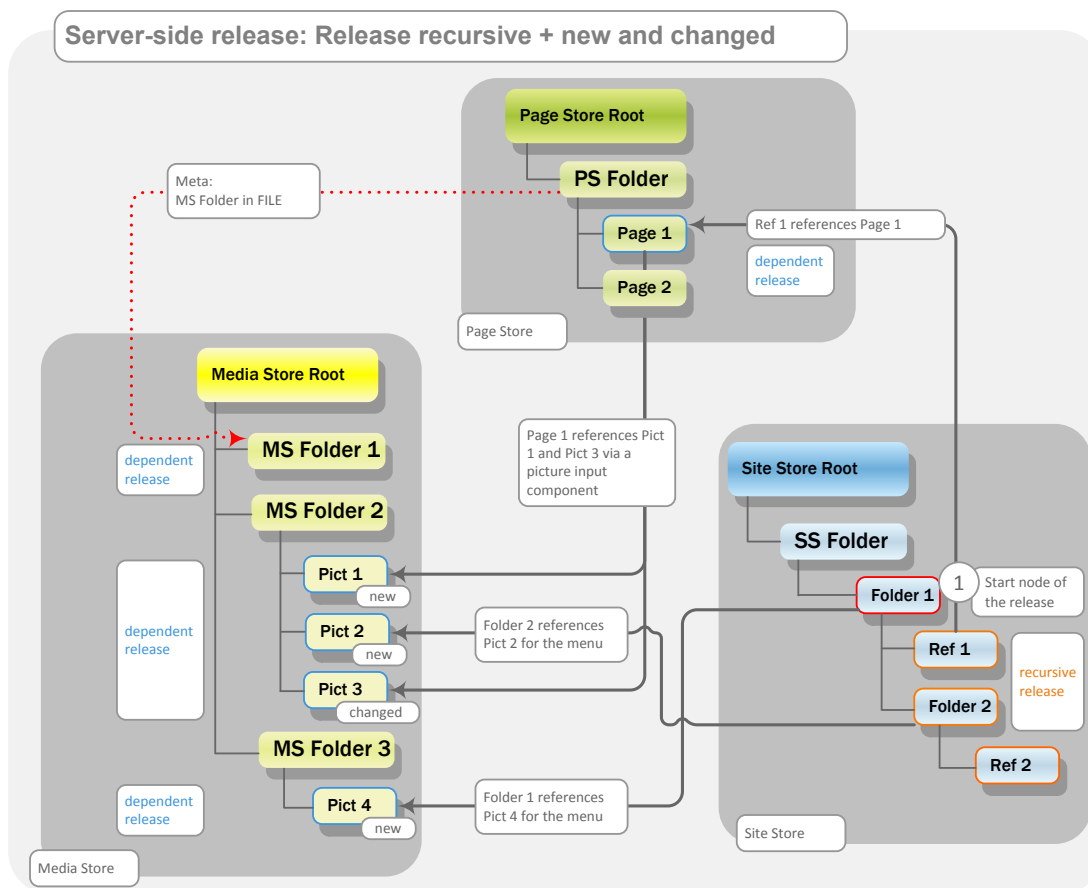


Figure 6-3: Server-side release – releasing recursively and dependently

For calling `AccessUtil.release(...)` the following parameters were set:

```
releaseStartNode: Folder 1
releaseParentPath: false
boolean recursive: true
DependentReleaseType:
```



DEPENDENT_RELEASE_NEW_AND_CHANGED | DEPENDENT_RELEASE_NEW_ONLY

The selected start node for the release is the "Folder 1" menu level.

Dependent release and recursive release: If the options `DEPENDENT_RELEASE_NEW_ONLY` or `DEPENDENT_RELEASE_NEW_AND_CHANGED` are combined with the `recursive` option, the dependent release has an effect on all objects below the start node. Thus, in the example from Figure 6-3, both the outgoing references of the "Folder 1" menu level (see Chapter 6.2.2, page 228) and the outgoing references from the subordinate child objects are examined:

- Relating to the example, "Ref 1" underneath "Folder 1" is also examined, which has a reference in the page store. The page reference "Ref 1" is released due to recursive release; page "Page 1" is also released due to dependent release.
- The "Folder_2" menu level, which has become a part of the release process via the recursive release option, has a reference to a media file in media store. The folder "Folder 2" and the subordinate page reference "Ref 2" are released due to recursive release. The referenced media file "Pict 2" is released due to dependent release.
- The page "Page 1", which was released dependently, also has outgoing references in media store. The media referenced, "Pict 1" and "Pict 3", are likewise released dependently.

Additional dependent or recursive objects are no longer taken into account, as they are not covered by any of the release options.



All outgoing references for the dependent release are only completely taken into account in one direction. If all dependent objects are to be included in the release process, then the release has to be carried out in a certain order (see Chapter 6.2.8, page 238).



6.2.4 Ensuring accessibility (parent chain)

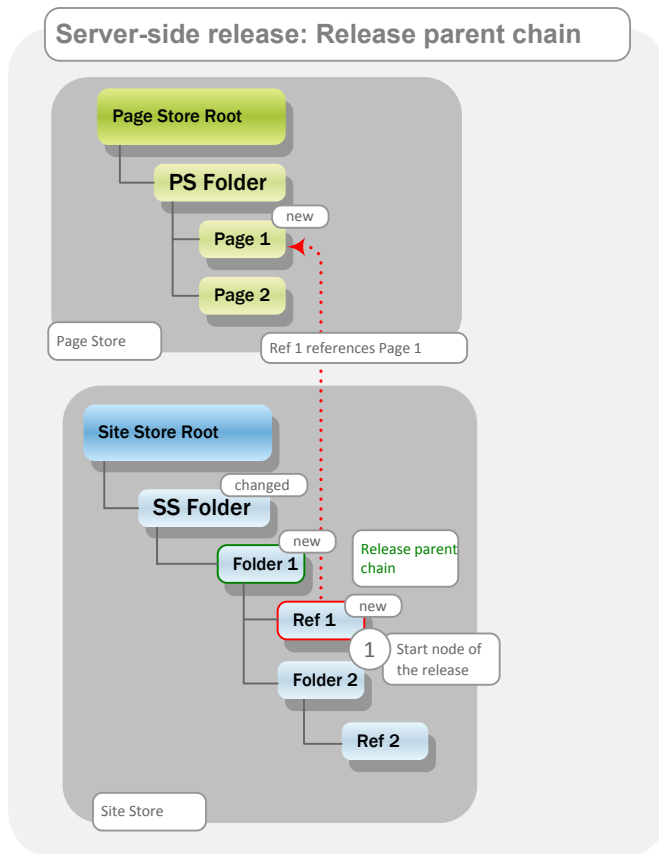


Figure 6-4: Server-side release – release parent chain

For calling `AccessUtil.release(...)` the following parameters were set:

```
releaseStartNode: Ref 1
releaseParentPath: true
boolean recursive: false
DependentReleaseType: NO_DEPENDENT_RELEASE
```

The selected start node for the release is page reference "Ref 1".

Ensuring accessibility (parent chain): Starting from the start node of the release "Ref 1", the complete parent chain of the object up to the root node of the store is considered. Through the option `releaseParentPath`, all nodes of the parent chain are released which were *not yet ever released*. Concretely, this means that objects which were already released once (changed objects) are not released through the option `releaseParentPath`, not even if they have been changed by an addition, for example of a page reference (see Figure 6-4).



If the option `releaseParentPath=false` is set, the parent chain is not released; the elements to be released are, however, added to the release child list of the parent node. The following applies here:

- With changed parent nodes: The object to be released can be accessed in release state. The parent element, however, is not released.
- With new parent nodes: Because the parent node was never released, the object to be released in the release state cannot be reached. This can lead to invalid references in the release state.

Background: If a page reference is to be released even though the editor has no permission to release within the higher menu level, the page reference should still be taken over in the release state. Possible content changes within the menu level (for example, additional references), should, however, not be released with the `releaseParentPath` option (see Figure 6-5).

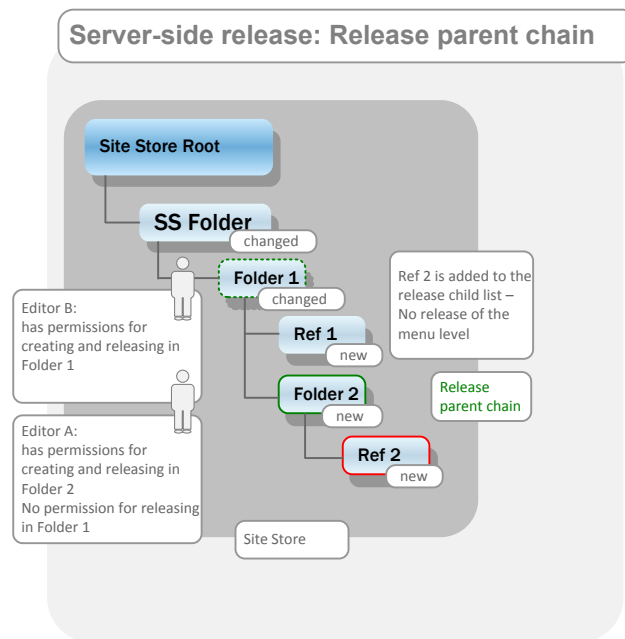


Figure 6-5: Server-side release – release parent chain (child release list)

In the example from Figure 6-5, through a release of page reference "Ref 2" from "Editor A", the newly created page reference and the newly created menu level "Folder 2" would be released. Menu level "Folder 1" is *not* released. So that the new menu level "Folder 2" (and with it the page reference "Ref 2") is reachable in a released state, "Folder 2" is added with the option `releaseParentPath` to the release child list of menu level "Folder 1". With that, page reference "Ref 2" can be reached within the release state, but not the newly created page reference "Ref 1". If "Editor B" now releases page reference "Ref 1", this is also added to the release child list of the menu levels "Folder 1". Because "Folder 1" is already reachable as a changed object via the



release child list of the likewise changed folder "SS Folder", the release is concluded with this. Both menu levels ("Folder 1" and "SS Folder") are not released via the option, because it does not involve newly created objects.

6.2.5 Ensure accessibility (parent chain) and recursive release

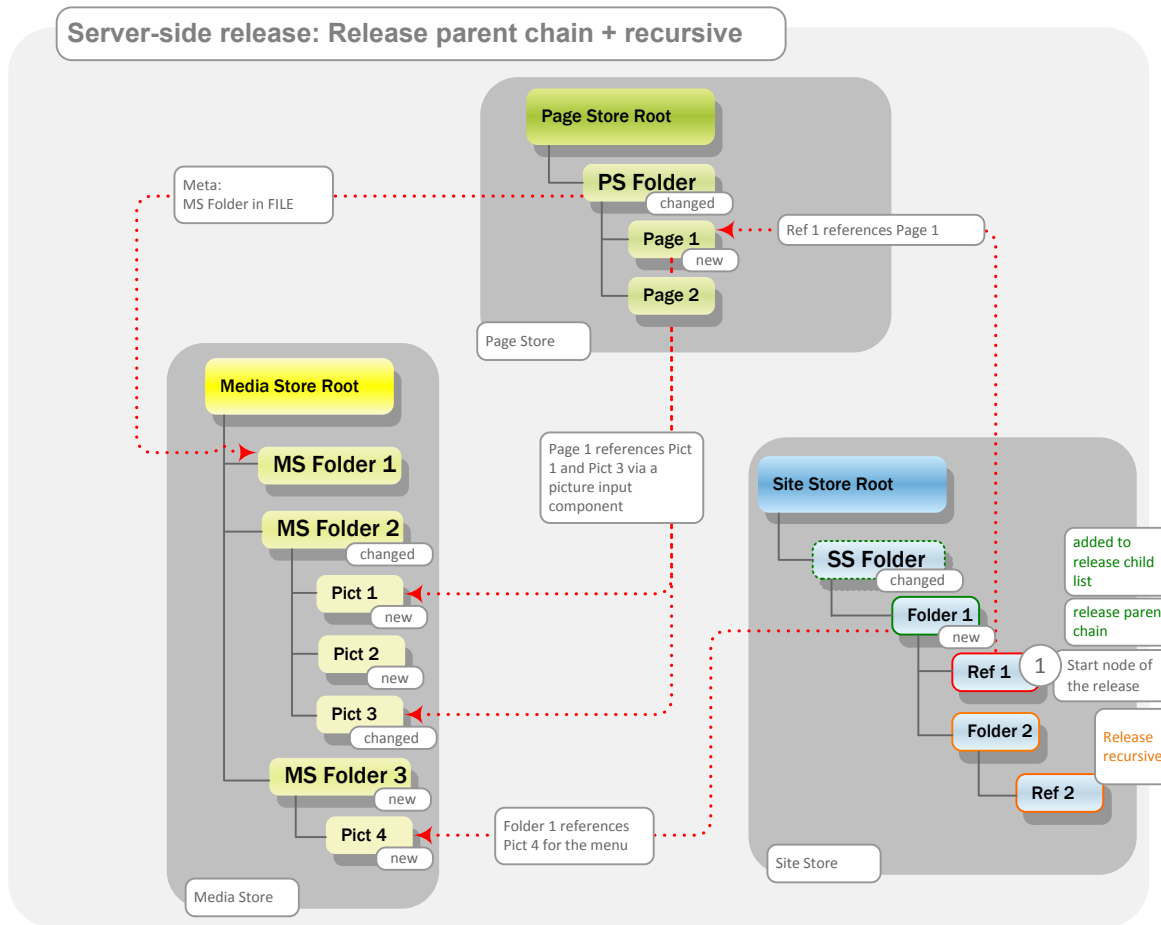


Figure 6-6: Server-side release – parent chain and recursive release

For calling `AccessUtil.release(...)` the following parameters were set:

```
releaseStartNode: Ref 1
releaseParentPath: true
boolean recursive: true
DependentReleaseType: NO_DEPENDENT_RELEASE
```

The selected start node for the release is page reference "Ref 1".

Ensuring accessibility (parent chain) and recursive release: Starting from the start node of the



release "Ref 1", the complete parent chain of the object up to the root node of the store is considered. Through the option `releaseParentPath`, all nodes of the parent chain are released which were *not yet ever released* (compare to Chapter 6.2.4, page 231). Additionally, all child elements of the start point are recursively released (compare to Chapter 6.2.1, page 227). Based on the example in Figure 6-6, it is easy to recognize that this release is limited to the site store, because no dependencies are taken into account here (in contrast to Figure 6-7). During this release it is to be noted that defective references may arise if one of the objects referenced in the site store was newly created, this in the example, "Page 1" and the media "Pict 1" and "Pict 4". The current configuration in the example (compare to Figure 6-6) will thus lead to an error within the release, because the page referenced, "Page 1", was never released. If the references indicate objects that were already released once ("changed"), the respective last released versions of the objects are referenced. In this case, the release from the example could be successfully run.

6.2.6 Ensure accessibility (parent chain) and dependent release

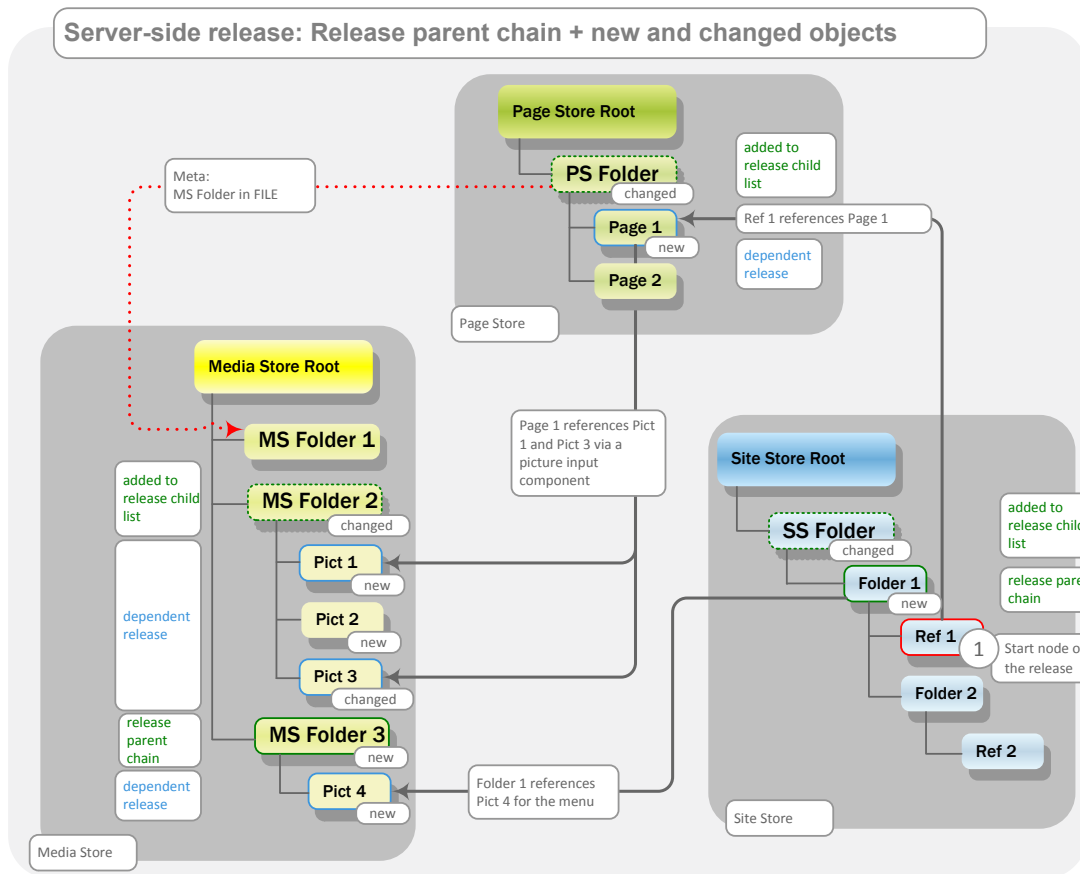


Figure 6-7: Server-side release – release parent chain and dependent objects



For calling `AccessUtil.release(...)` the following parameters were set:

```
releaseStartNode: Ref 1
releaseParentPath: true
boolean recursive: false
DependentReleaseType:
DEPENDENT_RELEASE_NEW_AND_CHANGED | DEPENDENT_RELEASE_NEW_ONLY
```

The selected start node for the release is page reference "Ref 1".

Ensure accessibility (parent chain) and dependent release: If the options `DEPENDENT_RELEASE_NEW_ONLY` or `DEPENDENT_RELEASE_NEW_AND_CHANGED` are combined with the option `releaseParentPath`, the dependent release affects the current start node and the release of never before released elements of the parent chain. For the release of a page reference, for example, this means that the page referenced there is released. The whole parent chain is now run through for the referenced page as well, and a search is run for elements that were never released. These elements are likewise released. The same applies to dependent objects in the media store.

- For the page reference "Ref 1", the entire parent chain is run through. There, all never released objects are released, in other words, in the example, the new menu level "Folder 1" is released, but not the changed menu level "SS Folder".
- The menu level "Folder 1" has an outgoing reference in the media store. Through the dependent release, the medium "Pict 4" is also released.
- For the "Pict 4" medium, now, in turn, the entire parent chain is run through and all never released objects are released. In the example, only the new media folder "MS Folder 3" is released.
- During release of page reference "Ref 1", the page referenced, "Page 1", is released.
- For page "Page 1", now, in turn, the entire parent chain is run through and all never released objects are released. In the example, no object is affected, because the parent node "PS Folder" was already released once. Dependent objects of the "PS Folder" folder are therefore not taken into account during the dependent release.
- However, "Page 1" that was released dependently still has outgoing references in the media store. The media referenced, "Pict 1" and "Pict 3", are likewise released dependently.
- For both media, the parent chain is now likewise examined. Because the common parent node "MS Folder 2" had only changed, no release is run here.



! All outgoing references for the dependent release are only completely taken into account in one direction. If all dependent objects are to be contained in the release process, the release thus has to be carried out in a certain order (see Chapter 6.2.8, page 238).

6.2.7 Ensure accessibility (parent chain), recursive and dependent release

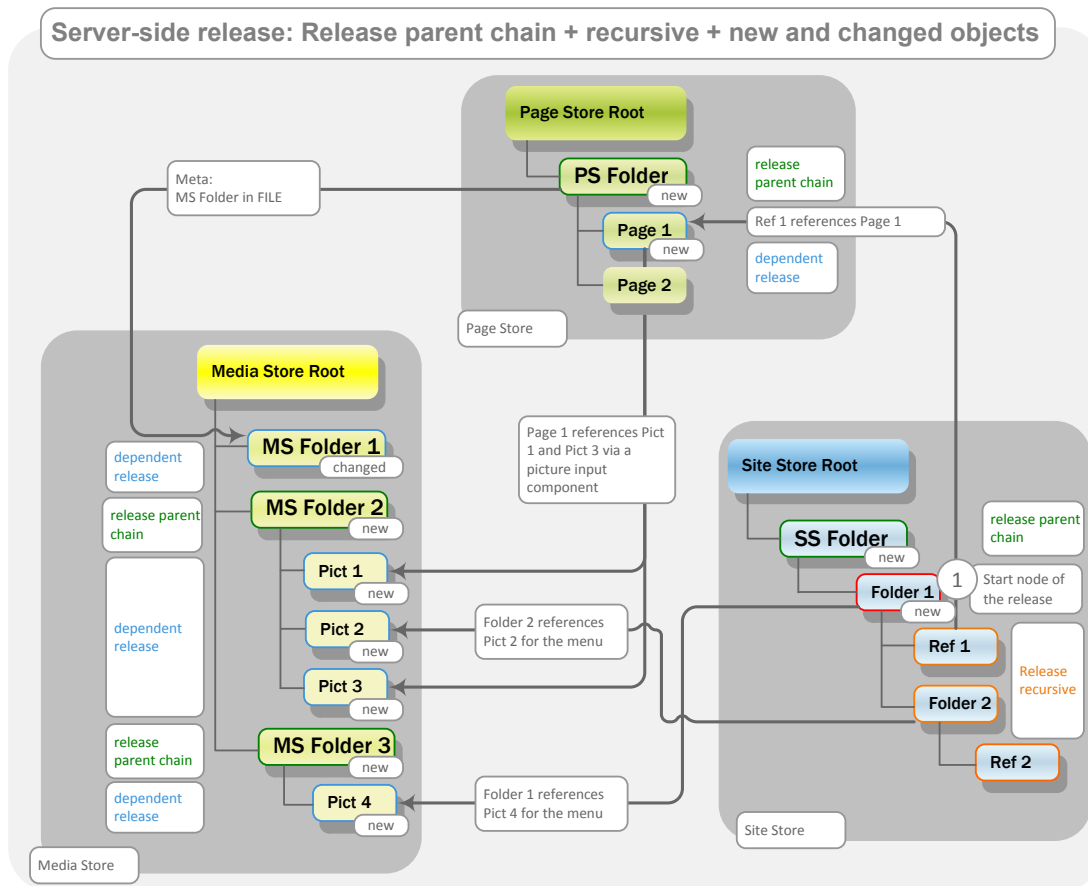


Figure6-8: Server-side release including all options

For calling `AccessUtil.release(...)` the following parameters were set:

```

releaseStartNode: Folder 1
releaseParentPath: true
boolean recursive: true
DependentReleaseType:
DEPENDENT_RELEASE_NEW_AND_CHANGED || DEPENDENT_RELEASE_NEW_ONLY
    
```



The selected start node for the release is the "Folder 1" menu level.

Ensure accessibility (parent chain), release recursively and dependently: The most comprehensive release is executed if all release options are combined with one another. In this case, all never-before released elements of the parent chain as well as elements beneath the start node are released. In addition, the dependent objects of all nodes affected by the release process are released; the entire parent chain is examined and released there as well. The recursive release has no effect on the dependent objects, unlike the release of the parent chain. Based on the example in Figure6-8, it is clear that the release has an effect on nearly all objects shown – only "Page 2" is not affected:

- At the release start point "Folder 1", the option `recursive` is evaluated. The recursive release has an effect *solely* on the child elements of the release start point. In the example from Figure6-8, through the option, the child elements Ref 1, Folder 2 and Ref 2 are released.
- All outgoing references of the released objects are released. In the example, this is the objects "Pict 4" (via the reference within the menu level "Folder 1"), "Page 1" (via the page reference "Ref 1"), "Pict 2" (via the reference within the menu level "Folder 2")
- From these released objects, the outgoing edges are again examined and released. In the example, these are the media "Pict 1" and "Pict 3" (via the reference within page "Page 1").
- The complete parent chains are examined for all released elements, and all never-released parent nodes are released. In the example, this is the "SS Folder" (parent element start node), "PS Folder" (parent element "Page 1"), "MS Folder 2" (parent element "Pict 1" and "Pict 2"), "MS Folder 3" (parent element "Pict 4").
- Now, the dependent objects of the released parent nodes are released. In the example, this is "MS Folder 1" (via the reference in "PS Folder"). Differently than with the release option `releaseParentPath`, "MS Folder 1" is then also released if it was only "changed", in other words, was already released once.



All outgoing references for the dependent release are only completely taken into account in one direction. If all dependent objects are to be contained in the release process, the release thus has to be carried out in a certain order (see Chapter 6.2.8, page 238).



6.2.8 Order for the release

All outgoing references for the dependent release are only completely taken into account in one direction, in order to eliminate cyclical dependencies during the release.



Objects from page and media stores, which are referenced in the content, structure or content store, are taken into account during the dependent release.

The reverse direction (site / media store → page / site / content store) does not function.



Objects from remote projects are not taken into account during dependent release.

If all dependent objects are contained in the release process, the following order has to be maintained:

- Release in the site store contains outgoing references to the content and the media store
- Release in the page store contains outgoing references to the media store

The following are not taken into account:

- Release in the page store contains *no* outgoing references to the site store.
- Release in the media store contains *no* outgoing references to the site store or the page store.

Other cases in which dependent objects are shown in the reference graph but are not released during the dependent release.

- Page→Page reference: Page with an FS_REFERENCE component, in which a page reference is referenced.
 - ⇒ Only the page is released, not the dependent page reference.
- Page→Medium: Page with the page template in which a medium reference is hard-coded. For example: `$CMS_REF(media:"XXX")$` in the HTML channel.
 - ⇒ Only the page is released, not the dependent medium.
- Medium→Media file: In a CSS file (file parsing: yes), an additional hard-coded reference to a media file (for example, a picture) is made. Both media are not yet released.
 - ⇒ If the CSS file is released ("Specific release -> Release dependent objects"), the medium referenced is not released with it.



- Page with LINK/DOM editor→Page reference: Both referenced objects are not yet released.
 - ⇒ The medium referenced (image) was also released, but the page reference referenced was not.
- Page→Data records: Page with the CONTENTLIST/FS_LIST/... component in which data records are referenced.
 - ⇒ CS object is not released with it.



Under certain circumstances, there can be cyclical dependencies that cannot be released automatically and therefore have to be triggered manually.

Example: *There are 2 pages in the page store ("Page 1" and "Page 2"), each with one section and a section reference to the other page's section:*

- Page 1
 - Section A
 - Section reference on section B of page 2
- Page 2
 - Section B
 - Section reference to section A of page 1

If the section references are not yet released, neither page 1 nor page 2 can be released automatically in this configuration. In order to release the pages, first one of the section references has to be deleted in order to display the cyclical dependency, for example, "Section reference on section B of page 2". Then page 2 can be released. Then the section reference has to be restored, after which page 1 can also be released.

There are some cases in which the dependent objects are shown in reference graphs but are not released during the dependent release.



7 Code completion for forms

In order to support template developers better during form creation, with FirstSpirit Version 5.0, a code completion program has been introduced on the form tab. Using this code completion, all available FirstSpirit input components and all corresponding parameters along with available values are shown at the press of a button and inserted at the insert mark on the form tab, for example:

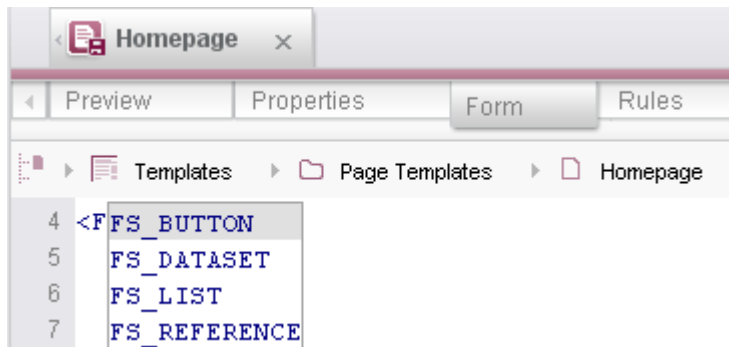


Figure 7-1: Auto completion on the form tab

To do so, the insert mark has to be positioned within a `<CMS_MODULE>` tag.



You can look up tags and parameters for the input components, data and design elements with the respective values and their syntax and meaning in the FirstSpirit Online Documentation, Chapter "Template development" / "Forms".

7.1.1 Inserting the input component tags

In order to determine input component tags (`FS_...` and `CMS_...`), a pointy bracket has to be opened (`<`) and the cursor positioned after it. The tags are then shown in a list, if `<Ctrl>` and the space bar are held down at the same time. The desired tag can then be applied using the keyboard (cursor button up and down and `<Enter>`) or the mouse (double click or click and `<Enter>`) on the form tab. The opening and closing tag and mandatory parameters (usually `name`) are inserted in the process, for example, when selecting `FS_BUTTON`:

```
<FS_BUTTON name=""></FS_BUTTON>
```

The cursor is then located between the quotation marks of the `name` parameter.



The number of tags shown can be limited by entering the initial letter(s) of the desired input components after the pointy bracket, for example <C for the input components beginning with "CMS_" or <F for the input components beginning with "FS_".



Crossed-out entries on the list are old and should not be used.

7.1.2 Inputting tags, parameters and key terms

In order to be able to show and select an input component's available tags, parameters and key terms, the cursor has to be positioned as follows depending on form syntax:

- **In opening tags:** In order to show parameters within an open tag, there has to be a blank space behind the cursor.
- **Between opening and closing tags:** In order to show tags between opening and closing tags, an opening pointed bracket has to be behind the cursor (<).
- **In quotation marks:** In order to show FirstSpirit preset values ("key terms") for a parameter, the cursor has to be positioned inside the quotation marks.

Only tags, parameters and key terms that are available for the selected tag or parameter are ever shown. Tags or parameters already used for the form that can only be used once are no longer shown in the list.

If the desired tags, parameters and key terms are already known, the first letter(s) can be entered. With <Ctrl> + space bar, the number of entries to be selected is reduced and the entry is directly inserted. Mandatory parameters are also inserted directly to the extent possible.



Crossed-out entries on the list are old and should not be used.

