FirstSpirit™
Your Content Integration Platform

FirstSpirit Manual for Developers (Part 1: Basics)

FirstSpirit Version 4.0, 4.1 and 4.2

Version	**1.31**
Status	**RELEASED**
Date	**2011-09-02**
Department	Techn. Documentation
Author/ Authors	B.Ehle
Copyright	2011 e-Spirit AG

File DEVB40EN_FIRSTspirit_DeveloperDocumentationBasics

e-Spirit AG

Barcelonaweg 14
44269 Dortmund | Germany

T +49 231 . 286 61-30
F +49 231 . 286 61-59

info@e-spirit.de
www.e-spirit.de

e-Spirit^AG

Table of contents

1 Introduction

The aim of this manual is to describe the implementation of FirstSpirit projects from the developer's perspective. The structure of the documentation has been chosen to give the most comprehensive overview possible of the FirstSpirit mechanisms relevant for the developer and the respective purposes (see Chapter 1.1 page 10).

Several areas required, especially for template development, are already explained in detail in the FirstSpirit Online Documentation. Chapter 1.3 gives an explanation of general terms used (from page 13) as an introduction to the FirstSpirit concept from a template development view.

! *New functions not released until FirstSpirit Version 4.1 are shown in this document in the new Look & Feel, existing functions (apart from a few exceptions) have, for the time being at least, been shown with the old Look & Feel.*

1.1 Topics covered in this documentation

This document describes the relevant functions and aspects for template development in FirstSpirit. The layout is roughly based on the user interface of the FirstSpirit JavaClient.

Chapter 2 describes the Template Store of the FirstSpirit JavaClient with all the available functions (see Chapter 2 from page 22).

FirstSpirit has efficient mechanisms for linking databases (content). **Chapter 3** deals with the layer types available in FirstSpirit for the database link and gives several general recommendations on how to handle data sources (content) in FirstSpirit (see Chapter 3 page 140).

Workflows are sequences of tasks which are worked through in a fixed, predefined structure. They can be used, for example, to model release processes. **Chapter 4** explains the workflow editor used in FirstSpirit, including all configuration options (see Chapter 4 from page 152).

In certain application cases it may be desirable to collate several FirstSpirit pages to form a single document, for example, if a PDF is to be created from several pages of the Page Store. The document groups concept used in FirstSpirit for this requirement and all configuration options as well as specific application examples are explained in **Chapter 5** (see Chapter 5 from page 241).

FirstSpirit provides a change tracking option via the FirstSpirit Access-API. **Chapter 6** describes access to the metadata of a revision via specific API functions. The revision metadata contains information about the type (what changes have taken place?) and the scope (which elements were changed?) of a change in the project (see Chapter 6 page 259).

Apart from release via a workflow, all objects in FirstSpirit can be released on the server side via the Access API. **Chapter 7** shows the methods for defining the different release settings for an object (see Chapter 7 page 272).

WebEdit was developed in addition to JavaClient and enables the editor to quickly and directly edit website content. The default specified functions and the appearance of the preview pages which the editor edits in WebEdit can be modified by the template developer. **Chapter 8** shows the requirements for the use of WebEdit, the functional scope and the options for template development as well as the Easy-Edit functionality (see Chapter 8 page 292).

1.2 Position within the overall documentation

Several areas required, especially for template development, are already documented in detail in the FirstSpirit Online Documentation. How the developer documentation fits into the overall documentation is illustrated in Figure 1-1.

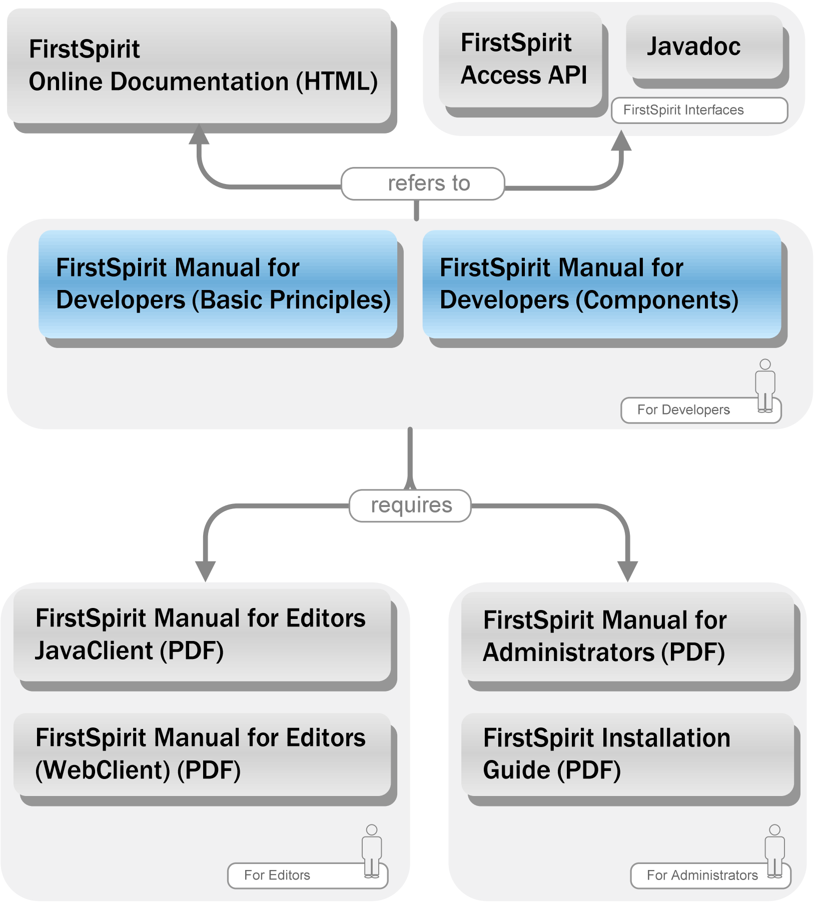

FirstSpirit Online Documentation (HTML)
FirstSpirit Access API
Javadoc
FirstSpirit Interfaces
refers to
FirstSpirit Manual for Developers (Basic Principles)
FirstSpirit Manual for Developers (Components)
For Developers
requires
FirstSpirit Manual for Editors JavaClient (PDF)
FirstSpirit Manual for Editors (WebClient) (PDF)
For Editors
FirstSpirit Manual for Administrators (PDF)
FirstSpirit Installation Guide (PDF)
For Administrators

Figure 1-1: Position of the developer documentation within the overall documentation

Basic knowledge of the Manual for Editors and the Manual for Administrators is assumed and is necessary as a minimum to understand the following chapters. A detailed description of individual template components and the interfaces is given in the FirstSpirit Online Documentation.

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

The Manual for Developers (basics) covers, among other things, the following aspects:

- The context menus and editing options with the FirstSpirit Template Store (see Chapter 2 from page 22)
- Terms and concepts for working with data sources (content) in FirstSpirit (see Chapter 3 page 140).
- The creation of workflows (see Chapter 4 page 152).
- The configuration and use of document groups (see Chapter 5 page 241).
- Access to revision metadata, e.g. for tracking changes (see Chapter 6 page 259).
- Information on server-side release (see Chapter 7 page 272).
- Information about template development in WebEdit (see Chapter 8 page 292).

The Developer Manual for Components deals with, among other things, the following aspects:

- Installation and configuration of modules
- Structure of modules and components
- The FirstSpirit GUI object model GOM
- The FirstSpirit Security Architecture
- Example listings

Knowledge of the following is also helpful to understand the chapters which describe extensions and adjustments of FirstSpirit:

- Programming in Java / BeanShell
- Technology of relational databases

1.3 General terms

1.3.1 Templates

Templates form the basis of each internet presence. The complete layout of the website is taken into account in them (among other things the Corporate Design and Corporate Identity). Templates are required to join the content entered in the Page Store and the media integrated in the Media Store with the structure deposited in the Site Store to form a complete presentation when the website is generated.

The basic principles of template development are taught in detailed step-by-step instructions in the **FirstSpirit Online Documentation**. The creation of the first templates is explained by way of a simple example. The example refers to the startpage of the demo project **FIRSTools**, the output language is HTML (see FirstSpirit Online Documentation / Chapter Templates (basics) / Step-by-step).

Different types of templates are available to the developer in FirstSpirit:

- **Page templates** create the basic framework of a page. Page templates are used to specify where, e.g. logos and navigations are positioned, whether a page is to consist of frames or not and similar general settings. In addition, the page templates define in which places an editor can insert content.
- **Section Templates** are used to insert content in this basic framework. Section templates are divided into individually specified input fields with which the editor maintains the editorial content of the section (in the Page Store).
- **Mapping** is used to define rules with which links can be made between page and section templates, e.g. to obtain a print view for a page.
- **Format templates** are used to define formatting which can subsequently be used in the DOM-Editor input element in the Page Store.
- **Link templates** are used to specify in detail the appearance of links within a FirstSpirit project. The template developers define all input fields with which the editors can enter all the necessary content and the display of the link on the HTML page.

All types of templates are maintained and managed in the so-called "Template Store" of FirstSpirit.

1.3.2 New input components (Status: Under development) (from V4.2)

Change to the input component model: FirstSpirit Version 4.2 marks the start of fundamental revision and consolidation of FirstSpirit's input component model (cf. "FirstSpirit Roadmap 2009-2012"). Within the scope of these activities, a whole range of input components previously implemented separately will be brought together. It is planned to bring together the following input component groups:

- single value input components: links to other FirstSpirit objects, e.g. CMS_INPUT_FILE, CMS_INPUT_PICTURE, CMS_INPUT_PAGEREF etc.
- set-valued input components: CMS_INPUT_CONTENTLIST, CMS_ INPUT_TABLIST, CMS_INPUT_CONTENTAREALIST, CMS_INPUT_LINKLIST

This is an extensive consolidation project in which compatibility and migration aspects also play an important role. This is why the implementation is in two phases:

- With **FirstSpirit Version 4.2** a new generation of input components will be gradually introduced with the name prefix "FS_" instead of "CMS_INPUT_".

These new input components will be gradually added within the scope of continued development of FirstSpirit and will be adjusted to clients' needs.

- The new input components cannot be officially released until **FirstSpirit Version 5.0**. This process is necessary, as release of the components in Version 4.2 would already bindingly specify the parameter assignment and API. This would make flexible further development of the components, taking into account customers' interests, no longer possible.

Status – Under development: As the official release will not take place until FirstSpirit Version 5.0, the input components ("FS_") can already be used in FirstSpirit Version 4.2, but in Version 4.2 their status is "**under development**". In specific terms, this status means:

- The input component is supported within the scope of the usual quality assurance and debugging.
- The persistence format will be kept compatible during the continued development, i.e. once data has been entered, it can continue to be imported. However, there is no downwards compatibility with FirstSpirit Version 4.1.
- The aim is not to make too many changes to the user prompting. However, as especially optimisation of the user prompting, is the objective of the iterative procedure, editorially relevant changes may well occur
- The aim is to keep the parameter assignment of the input component compatible. Should this not be possible, a corresponding announcement will be made within the scope of the "FirstSpirit Release Notes".
- The API of the input components will change during the course of the development. If possible (and meaningful) the API changes will be compatible. However, incompatible changes are possible and will also be announced within the scope of the "FirstSpirit Release Notes".
- As release of the "FS_" components is not planned until FirstSpirit Version 5.0 with WebEdit 5.0, support for these component in WebEdit 4.2 will be very rudimentary. An exception here is all new input components which are necessary for the migration to generic link editors.

The implications for use in productive projects are: Use of the components is possible, if the project developers dispense with use of the API and they are willing to potentially subsequently adjust the parameter assignment of the input components. The editorial users should be prepared for changes in the user prompting. Should this be unacceptable, use of the new input components should be dispensed with in Version 4.2.

For detailed information about the new input components please see also FirstSpirit Online Documentation[1].

1.3.3 Content Store

The Content Store is used for entering and managing highly structured databases, which are to be used or maintained in FirstSpirit. Such databases are, e.g. product catalogues and address lists. These databases are not only highly structured but are also subject to frequent changes. Such data is usually kept in databases. The Content Store data is saved in a relational database system supported by FirstSpirit.

Layout, content and structure should continue to be separated for entering data in the Content Store. To ensure this, both the structure of the data and the layout for the corresponding data entry screen are defined in the Schemata area (in the Template Store). This layout is then used to manage the content in database tables in the Content Store. These databases can then be inserted into the structure of the website in the Site Store.

In an initial step, a graphic editor is used to create a database schema (content schema) in the Template Store. This schema can either be created on the basis of an existing database structure and then, if necessary, adjusted in the schema editor, or it can be generated as an empty schema, and then organised with the help of the schema editor. The tables and relations of a data model must then be depicted in this schema. The input elements for the table columns are then defined in the table templates and queries for the databases are formulated.

The responsible editors maintain the databases in the Content Store. To this end, database tables are created based on the settings in the Template Store and are filled with content via the configured input elements.

To display the structured content on a website, the database table is inserted as content on a page of the Page Store. This page is then referenced in the Site Store. Settings for the display of Content Store data can be made on this page reference. For example, if a specific section of the database table only is to be displayed, the queries defined in the Template Store can be opened at this point.

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

[1] ../Template development/Forms/Input components (new)

For details of the concept of working with "Schemata, table templates, views of a database" see Chapter 3 page 140.

1.3.4 Workflows

A workflow is a sequence of tasks which are worked through in a fixed, specified structure. The tasks serve to transfer an object, for example a page from the Page Store, from its initial state (e.g. "Page changed") into an end state (e.g. "Changed Page checked and released"). Both due dates and authorised groups of people can be specified for the tasks to be executed between these two states.

The workflows can be displayed in the Template Store with the help of a graphic editor. The workflow editor's task is to describe the workflow as abstractly and completely as possible. The graphically created model can then be used as the basis of support for the user when performing the work process.

The structure (sequence of the tasks) and properties (e.g. context-free) of a workflow and the definition of the authorised persons of groups who are allowed to switch or forward a workflow from one task to the following task are defined within the Template Store (see Chapter 4 page 152).

One example of a FirstSpirit workflow is the frequently used release process. The task of the release process is to ensure that a new article or contribution created by the editor or a change to the content is subjected to a check before the "Live Transmission". The release process can vary depending on which workflows are already established or are to be established within the company.

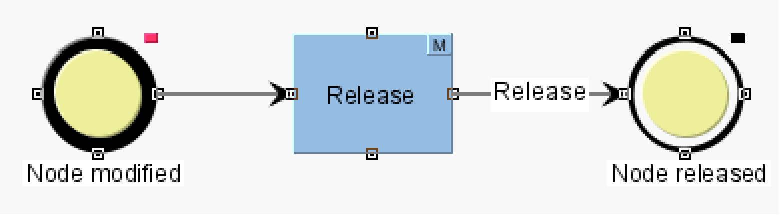


Figure 1-2: Example of a "simple" release

In this example, the chief editor is responsible for checking the contributions of the editors. Deployment (publication) is not possible until the chief editor has checked the changes (see Figure 1-2).

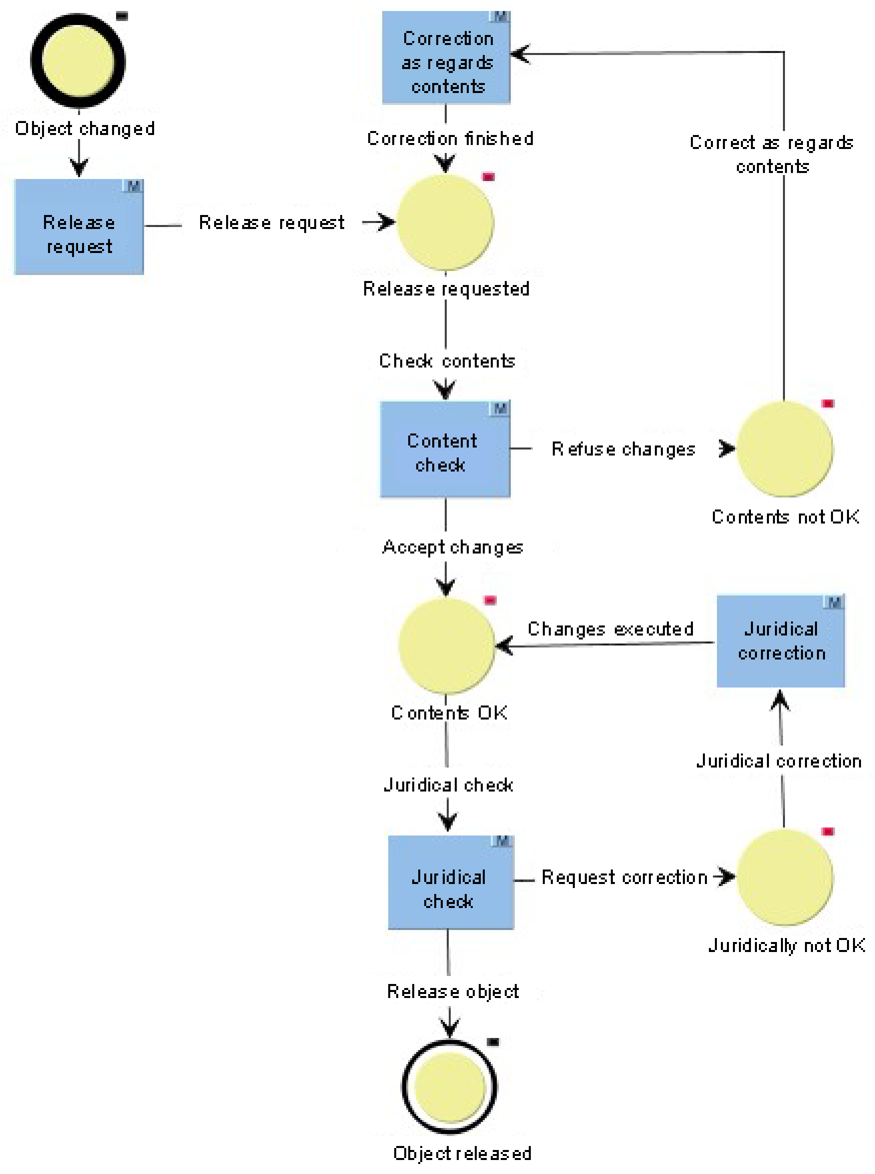


Figure 1-3: Example: Release with "factual and legal" check

In this example the check for deployment of certain contributions is divided into a "factual" (i.e. as regards content) and a "legal" (i.e. juridicial) substep (see Figure 1-3). These substeps are usually carried out by different people. In this case, the workflow also ensures that the legal check is not carried out until after the content check so that any necessary content corrections also run through this checking step. Should a correction be necessary under legal aspects, in this case the model assumes that renewed content checking is not necessary (although this could definitely be necessary in other types of applications).

Important aspects when using workflows are, among other things:

- Assignment of workflows to "logical" sub-areas:
  Example: In the Media Store, workflow "B" and "C" only is possible in area "A", while in the Site Store the workflow "E" must be used.

- Assignment of users/groups and workflows:
  Example: Workflow "B" may only be carried out by user group "G".

- Defining rights (permissions) in workflows:
  Example: Transition to "Juridical check" status can only be carried out by the group "lawyers".

- Defining data fields which can (or must) be filled by the user on running through the workflow:
  Example: Insertion of a "legal check note“ in the corresponding form field by the person checking.

For further information on creating workflows see Chapter 4, page 152 ff.

1.3.5 Integrated preview (from V4.2)

From FirstSpirit Version 4.2 the JavaClient offers a direct WYSIWYG preview by means of the function “Integrated preview“ (menu “View“ / “Display integrated preview“).A template developer can use the integrated preview to check changes in the output channel of the templates (e.g. HTML or XSL-FO) directly in the preview window, as each time the template is saved, the (configured) preview page is automatically updated.

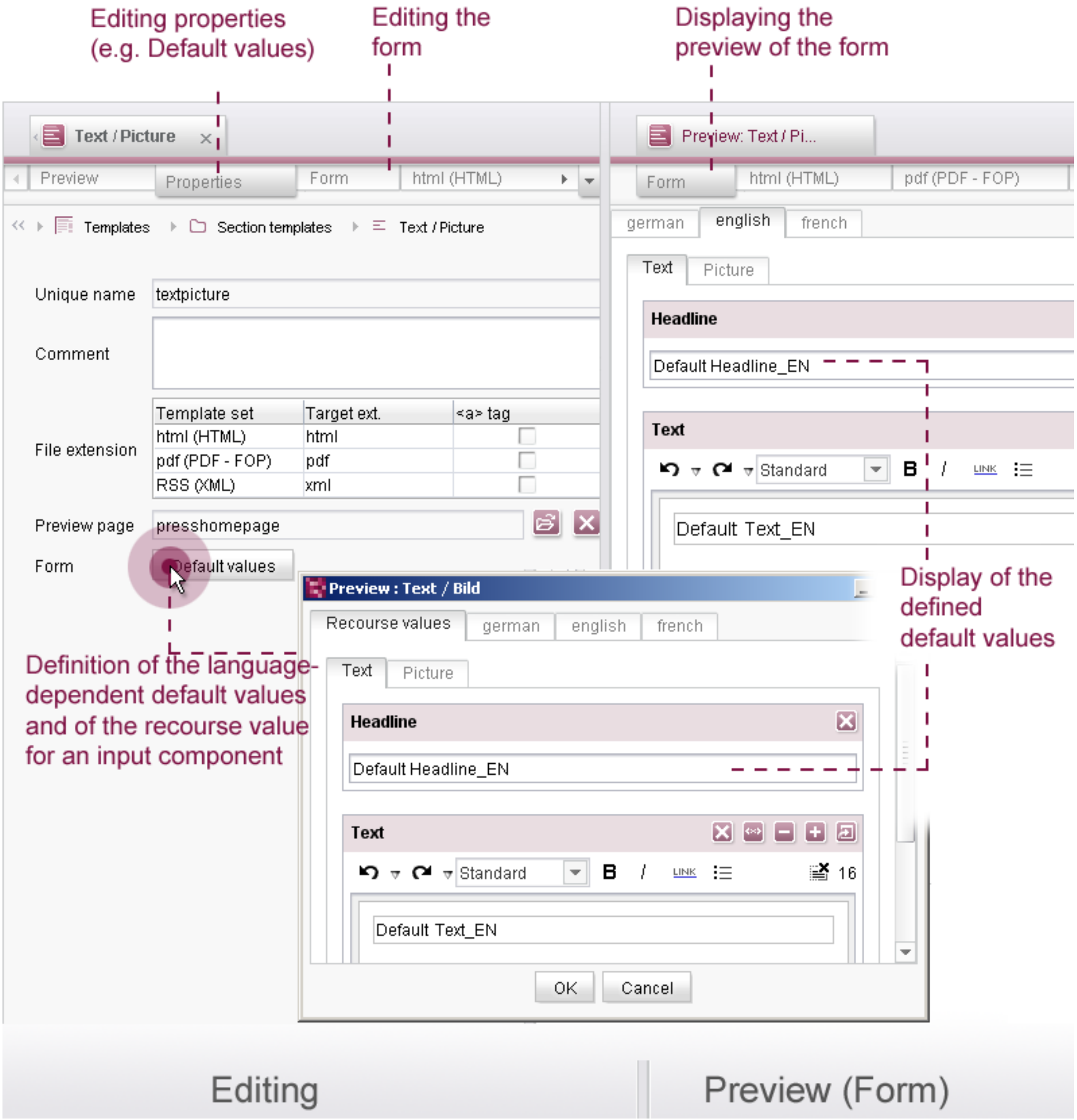
In addition, an integrated form preview is available within the Template Store. If the form area of a template is selected, a live view of the edited form appears in the preview area with the defined retrieval values of the input components (see Figure 1-4).

The integrated preview can be optionally displayed on the right next to the workspace or, if smaller monitors are used, in an external window.

All input fields can be directly edited within the integrated form preview. To do this, the template must not be locked to prevent editing.

The editing option is merely a tool for the template developer. In addition, for example, it is also possible to directly check whether a defined remote configuration provides the required results for an input component. However, the content entered is not saved in the form view. Default values cannot be defined in the form preview.

The **default values for the input components** can be defined using the "Default Values" button in the "Properties" tab of a template (see Chapter 2.5.2 page 83 and Chapter 2.6.2 page 90). The language-dependent default values are displayed in the preview window immediately after saving the properties.



Unique name: textpicture

Comment:

File extension:

Template set	Target ext.	<a> tag
html (HTML)	html	□
pdf (PDF - FOP)	pdf	□
RSS (XML)	xml	□

Preview page: presshomepage

Form: Default values

Figure 1-4: Preview of the form area in the Template Store

1.3.6 Content Highlighting (from V4.2)

The "Content Highlighting" function was introduced to make it easier for editors to search for content and to navigate in JavaClient.

Existing FirstSpirit projects do not have to be migrated. Content Highlighting is an additional function, i.e. existing projects can initially continue to be used as usual without adjustments (and therefore without Content Highlighting).

Analogous to use of Easy-Edit (see Chapter 8.3 page , the templates of a project must be adjusted first for use of the “Content Highlighting" function. Special format templates and a CSS stylesheet are used for this; these are already included in the FirstSpirit scope of supply. The format templates must be added to the page, section and/or table templates in which the Content Highlighting function is to be used.

A precise description of the format templates and of the stylesheet is given in the FirstSpirit Online Documentation[2].

The technologies used for the "Content Highlighting" function intervene in the HTML source code of a FirstSpirit project far more than the functions used to date. As the JavaScript share here is smaller than, for example, when using "Easy-Edit", conflicts should occur far less often. Nevertheless, it is not possible to guarantee that "Content Highlighting" can be used without changes to the project HTML. Particularly "pixel-precise" layouts in conjunction with CHTML can cause problems here, as several additional pixels are required in the HTML environment due to the framing of the highlighted content.

[2] FirstSpirit Online Documentation – Chapter: ../Advanced topics/Content Highlighting

2 Template Store of the FirstSpirit JavaClient

2.1 General information

Template-Store

Templates form the basis of each internet presence. The complete layout of the website is taken into account in them (among other things the Corporate Design and Corporate Identity). Templates are required to join the content entered in the Page Store and the media integrated in the Media Store with the structure deposited in the Site Store to form a complete presentation when the website is generated.

Different types of templates can be defined and edited in the Template Store.

- Page templates (see Chapter 2.5 page 81).
- Section templates (see Chapter 2.6 page 89).
- Format templates (see Chapter 2.7 page 91).
- Style templates (see Chapter 2.8 page 95).
- Table format templates (see Chapter 2.9 page 106).
- Link templates (see Chapter 2.10 page 114).

In addition, other edit options are available for:

- Scripts (see Chapter 2.11 page 120).
- Database Schemata (see Chapter 2.12 page 125).
- Workflows (see Chapter 2.13 page 139).

Move using "Drag & Drop": Folders and templates can be moved within the Template Store with the help of the mouse using "Drag & Drop" (identified by a small rectangle on the mouse pointer).

Copy using "Drag & Drop": Further, it is also possible to copy folders and Templates in the Template Store using the mouse and by pressing the Ctrl key at the same time (indicated by a small plus symbol on the mouse pointer).

The user must have the necessary permissions to "Drag & Drop" (move, copy) from nodes into the Template Store. Otherwise objects cannot be moved or copied by “Drag & Drop”.

In FirstSpirit Version 4.2 some new “Drag & Drop“ options have been implemented. They can also be used in the Template Store. For detailed information please see FirstSpirit Manual for Editors (JavaClient) and FirstSpirit Release Notes Version 4.2.

2.2 General Template Store context menus

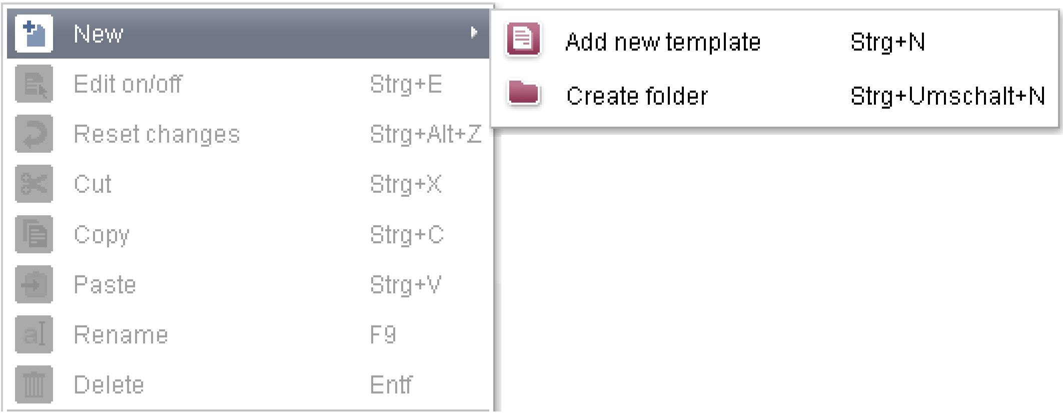


Figure 2-1: General Template Store context menu

The Template Store context menus are described in the following chapters:

Structuring the context menus (general, specific, administrative):
All context menus are structured in the same way:

- the top part contains general functions (see this chapter)
  it contains the following menu items:

- New	(see Chapter 2.2.1 page 24)
- Edit on/off	(see Chapter 2.2.2 page 40)
- Reset Changes	(see Chapter 2.2.3 page 41)
- Cut	(see Chapter 2.2.4 page 41)
- Copy	(see Chapter 2.2.5 page 42)
- Paste	(see Chapter 2.2.6 page 42)
- Rename	(see Chapter 2.2.7 page 43)
- Delete	(see Chapter 2.2.8 page 44)

- the middle part contains specific functions for the selected nodes (see Chapter 2.3 page 47).
- the bottom area contains functions which are usually only required by project administrators. These cannot normally be carried out by the normal user and are therefore in most cases disabled (see point 3) (see Chapter 2.4 page 71).

Opening a context menu: To open a context menu, an object, for example a folder or a template, is selected in the tree view in the left-hand half of the screen and is then right-clicked (i.e. right-hand mouse key is pressed) to open the context menu for this node. The required menu item can be selected by clicking the left-hand mouse key (left-click).

Disabled menu items are "grey". In this case the function is not available to the user. Possible reasons for this are:

- the object is currently being edited by another template developer
- the status of the current object
- the user does not have the necessary permissions to execute a specific action.

2.2.1 New

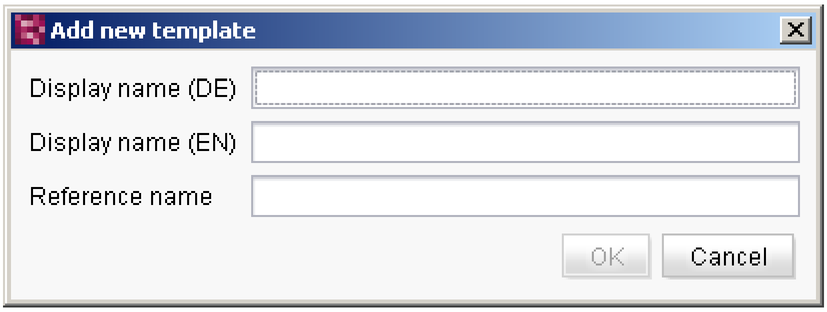
New objects can be inserted in the project using the "New" context menu entry or the "New" icon in the toolbar. The selection available depends on the object type on which the context menu or function was opened:

▪ Create template	(see Chapter 2.2.1.1 page 25)
▪ Create folder	(see Chapter 2.2.1.2 page 26).
▪ Create format template	(see Chapter 2.2.1.3 page 27).
▪ Create table format template	(see Chapter 2.2.1.4 page 27).
▪ Create style template	(see Chapter 2.2.1.5 page 28).
▪ Create link configuration	(see Chapter 2.2.1.6 page 28).
▪ Create link template	(see Chapter 2.2.1.7 page 30).
▪ Create new script	(see Chapter 2.2.1.8 page 30).
▪ Create schema	(see Chapter 2.2.1.9 page 31).
▪ Create schema from database	(see Chapter 2.2.1.10 page 35).
▪ Create table templates	(see Chapter 2.2.1.11 page 37).
▪ Create query	(see Chapter 2.2.1.12 page 38).
▪ Create new workflow	(see Chapter 2.2.1.13 page 40).

2.2.1.1 New: Create template

This function can be used on the following elements of the Template Store:

- Root nodes and folders of the page templates, section templates and format templates
- on link configurations



Add new template

Display name (DE)

Display name (EN)

Reference name

Figure 2-2: Add new template

The following entries are necessary to create a new page, section, format or link template:

Display name: The language-dependent display name can be individually defined for all project languages. If the language-dependent display of the tree view is enabled in the JavaClient (Menus Extras / Tree View), the display names entered here are displayed depending on the selected project language. Unlike unique reference names, the language-dependent names can be changed at any time.

Reference name: A unique name for the new template is given in the "Reference name" field. The field for entering the reference name is automatically filled when the template is created. To this end, the display name entered (for the master language), without spaces and special characters, is copied into the field. The user can change the reference name on creating the template; after the template has been created however, it is no longer possible to change the reference name as otherwise all relations within the project would be lost. The reference name can be used to create a unique relation to the template within a project. For example, the unique reference name is used within the input components (in the form area) to establish relations to templates (cf. FirstSpirit Online Documentation, e.g. to input components CMS_INPUT_DOM or CMS_INPUT_OBJECTCHOOSER). If a reference name already assigned within a name space is entered here, FirstSpirit automatically replaces it with a unique name, mostly by appending a number. (Invalid special characters are also automatically replaced.)

Example: A page template called *template* already exists. If a new section template called *template* is created, it would then automatically be saved under the reference name *template_1*.

The reference name of a template can be determined in the "Properties" tab (see Chapter 2.5.2 page 83).

> *A reference name of a template should not be changed after a template has been created, otherwise all relations within the project would be lost!*

[OK] Click the button to insert the new template in the directory tree, from where it can be further edited.

[Cancel] Click the button to cancel the action. A new template is not created.

2.2.1.2 New: Create folder

For improved clarity, it is helpful to create all elements of the Template store in meaningful folder structures. This function enables new folders to be created.

This function can be used on the following elements of the Template Store:

- Root nodes and folders of the page templates, section templates, format templates, scripts, database schemata (content schemata) and workflows, from FirstSpirit Version 4.2 also link templates

A new folder is created in a similar way to a template (see Chapter 2.2.1.1 page 25).

Reference name: A name for the new template must be entered in the "Reference Name" field. Unlike the reference name of templates (cf. Chapter 2.2.1.1), the folder name given here does not have to be unique.

2.2.1.3 New: Create format template

This function can be used on the following elements of the Template Store:

- Root node, folders and other format templates of the format template node

A new format template is created in a similar way to a template (see Chapter 2.2.1.1 page 25).

Reference name: A unique name for the new template must be entered in the "Reference Name" field (cf Chapter 2.2.1.1). The name of the format template can be viewed in the "Abbreviation" field of the "Properties" tab (see Chapter 2.7.1 page 92). The internally used abbreviation of the format template (e.g.: "b" for "Bold") must be unique within the whole project and may not contain any special characters. Therefore, the entry of invalid special characters is suppressed when a new format template is created. The abbreviation is needed in the form area of the page or section template to specify the valid format templates for the input component. The corresponding XML tag name is formed from the abbreviation, e.g. for the "Bold" format template (see also FirstSpirit Online Documentation – Area Template development / Format templates).

If a name is entered which has already been assigned within a name space, FirstSpirit automatically replaces the name with a unique name, mostly by appending a number.

> ! *A unique name or the abbreviation of a format template cannot be changed after the template has been created, otherwise all relations within the project would be lost!*

A detailed description of reference and display names is given in Chapter 2.2.1.1 page 25.

2.2.1.4 New: Create table format template (from V4.1)

> ! *This function is released for FirstSpirit Version 4.1 and higher only. Screenshots are therefore displayed in the new "LightGray" Look & Feel. The display in the "Classic" Look & Feel can differ slightly.*

This function can be used on the following elements of the Template Store:

- Root node and folders of the formal templates node

A new table format template is created in a similar way to a template (see Chapter 2.2.1.1 page 25).

Reference name: A unique name for the new template must be entered in the "Reference Name" field (cf Chapter 2.2.1.1).

A detailed description of reference and display names is given in Chapter 2.2.1.1 page 25.

2.2.1.5 New: Create style template (from V4.1)

> ! *This function is released for FirstSpirit Version 4.1 and higher only. Screenshots are therefore displayed in the new "LightGray" Look & Feel. The display in the "Classic" Look & Feel can differ slightly.*

This function can be used on the following elements of the Template Store:

- Root node and folders of the formal templates node

A new style template is created in a similar way to a template (see Chapter 2.2.1.1 page 25).

Reference name: A unique name for the new template must be entered in the "Reference name" field (cf Chapter 2.2.1.1).

A detailed description of reference and display names is given in Chapter 2.2.1.1 page 25.

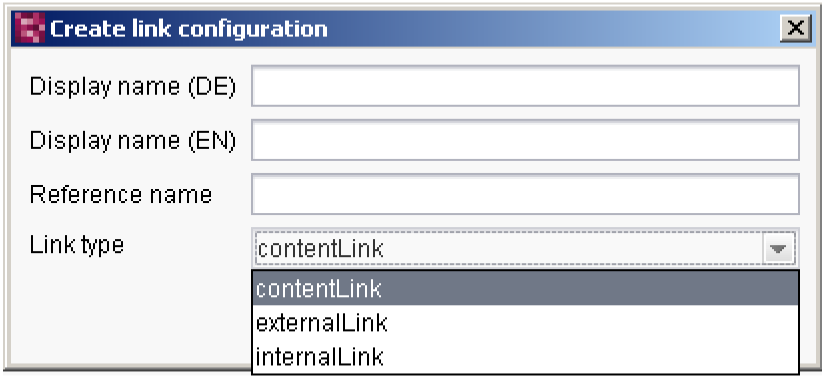
2.2.1.6 New: Create link configuration

This function can be used on the following elements of the Template Store:

- Root node of the link templates

Any number of instances of standard link types (internal link, external link, link to a Content Store element) can be created below the root node of the link templates.

These instances are called link configurations. Link configurations affect all link templates located below them (cf. Chapter 2.2.1.7). The following information is required to create a link configuration:



Create link configuration

Display name (DE)

Display name (EN)

Reference name

Link type: contentLink

- contentLink
- externalLink
- internalLink

Figure 2-3: Create link configuration

Display name: The language-dependent display name can be individually defined for all project languages (cf. Chapter 2.2.1.1).

Reference name: A unique name for the new link configuration must be entered in the "Reference name" field (cf Chapter 2.2.1.1).

A detailed description of reference and display names is given in Chapter 2.2.1.1 page 25.

Link type: In FirstSpirit, a differentiation is made between three standard types of link (see Chapter 2.10.1 page 115).

An instance (or link configuration) of the respective standard link type can be created by selecting a standard link type from the list. The possibility of defining several instances (link configurations) of a link type enables different link configurations to be defined for different input components in the form area of a page or section template. In this way, internal links entered, for example, by the editor within the DOM editor input component, can be configured and displayed differently to links entered, for example, within the link list input component.

From FirstSpirit Version 4.2 also “Generic link editors“ can be created (see Chapter 2.10.6 page 118). For this purpose, the link type “genericLink“ must be selected.

OK Click the button to insert the new link configuration in the tree view, from where it can be further edited.

Click the button to cancel the action. A new link configuration is not created.

2.2.1.7 New: Create link template

This function can be used on the following elements of the Template Store:

- On link configurations in the link templates node

A new link template is created in a similar way to a template (see Chapter 2.2.1.1 page 25).

The link template is only used to define the output of the editorial content. The configuration of the link template, i.e., for example the input fields made availalbe to an editor for creating a new link, is defined in the respective higher level link configuration (instance of a link type) (see Chapter 2.10 page 114).

A detailed description of reference and display names is given in Chapter 2.2.1.1 page 25.

> *From FirstSpirit Version 4.2 also "Generic link editors" can be created by means of this context menu entry (see Chapter 2.10.6 page 118).*

2.2.1.8 New: Create new script

This function can be used on the following elements of the Template Store:

- Root node, folders and other scripts in the "Scripts" node

A new script is created in a similar way to a template (see Chapter 2.2.1.1 page 25).

A detailed description of reference and display names is given in Chapter 2.2.1.1 page 25.

2.2.1.9 New: Create new schema

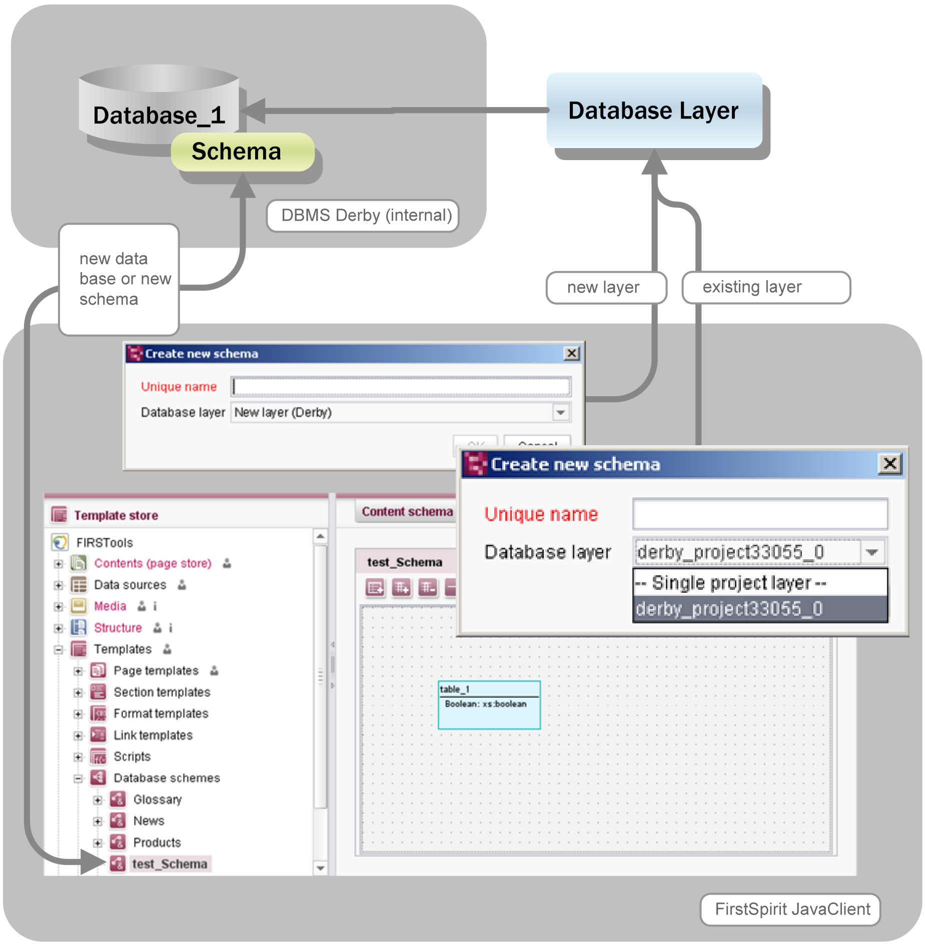
A database schema (content schema) is used to define which data is saved in a database and in what form and how this data is related to each other. The graphic editor in the FirstSpirit JavaClient can be used to model database tables with the corresponding columns and the relations between the individual tables (see Chapter 2.12.1 page 126).

The creation of a new schema in the JavaClient – apart from adding a schema node below the "Database schemes" root node – creates a new database or a new database schema, configured for the project concerned, in the database. For example, if the project administrator has configured the "Standard database" for the project, the "New – Create new schema" context menu is used to create a new database within the standard database (Derby). (The performance depends on the configuration of the database layer – see "FirstSpirit Manual for Administrators" regarding the "No Schema Sync" setting). The editor obtains access to the database via the corresponding tables in the Content Store of FirstSpirit and can enter content in the relevant tables, which are written in the database (provided the project administrator has not defined the database as being "write protected") (see Chapter 3.5 page 149).

If a new schema for a data source (content) is created in FirstSpirit, it should be decided in advance in which database this should be stored in productive use and which permissions the DBMS account used by FirstSpirit should have in productive use. Subsequent conversion of the layer type is not easily possible (cf. "FirstSpirit Manual for Administrators"). In case of doubt, a separate MultiProjectLayer should be created in the project for each FirstSpirit schema (see Chapter 3.2.2 page 145).

! *In general, we recommend the development always be carried out in an environment corresponding to productive use. In particular, the Derby DBMS contained in FirstSpirit is not suitable for productive use and should therefore be used for tests only.*

! *Especially when using an Oracle database, saving database schemes and changes to a schema can take some time in version 4.2R4 and higher.*



Create new schema

Unique name

Database layer New layer (Derby)

Create new schema

Unique name

Database layer derby_project33055_0

-- Single project layer --

derby_project33055_0

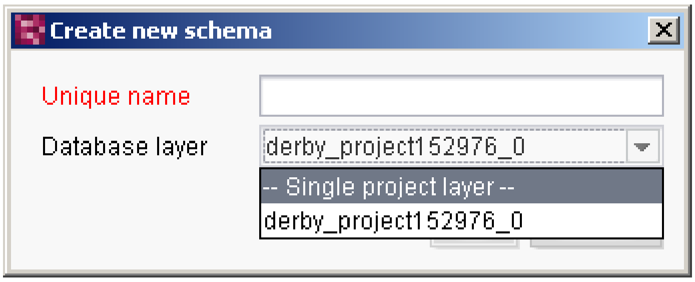
Figure 2-4: Creating a new schema

A new, empty database (see Figure 2-4 "Database_1") and a schema (see Figure 2-4 "Schema") is created using the "New – Create new schema" context menu. To this end, the name of the database layer must be given within the JavaClient (see Figure 2-5 "Database layer") or (if there is no layer) a new (standard) layer must be created (see Figure 2-6). The new schema created can be given any name (in conformity with database) – but it must be unique within the whole project.

This function can be used on the following elements of the Template Store:

- Root node and folders in the "Database schemes" node

The following input is required to create a new schema:



Create new schema

Unique name

Database layer: derby_project152976_0

-- Single project layer --
derby_project152976_0

Figure 2-5: Create new schema

Unique name: A name for the new schema must be entered in the "Unique name" field. This reference name can be used to create a unique relation to the schema within a project. If a reference name already assigned within a name space is entered here, FirstSpirit automatically replaces it with a unique name, mostly by appending a number. (Invalid special characters are also replaced by FirstSpirit.)

> *The reference name of a schema cannot be changed after it has been created, otherwise all relations within the project would be lost!*
>
> *The "unique name" defined for a schema within a FirstSpirit project is not the same as the physical name of the schema in the database. The physical name is automatically issued depending on the database – for example, in accordance with the following pattern for the standard database (Derby): derby_projectID_schemaID*

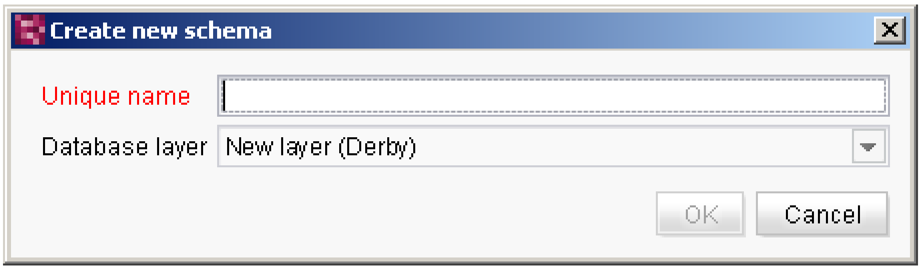
Database layer: An existing database layer of a database in which the individual database tables for this schema are to be saved must be selected in the "Database layer" field. So-called "Single project layers" and (optional) "Multiproject layers" are available to choose from:

- Multiproject layer: A multiproject layer must be selected if the intention is to work directly on an existing database schema (cf. Figure 2-8). This layer can be used in several FirstSpirit projects, which all write in the same database or read content from it. Only one of the projects involved should have permission to write to the database to ensure overlapping does not occur during use of a multiproject layer (see Chapter 3.2 page 143).
- Single project layer: If a single project layer is selected, a separate schema or database is created for the respective project (during the first Sync) when the schema is created. In this case, overlapping on writing the content is not possible (see Chapter 3.3 page 147).

In FirstSpirit Version 4.2 the naming of the layer types has been changed. The functionality is retained – existing projects need not to be changed. For further information see Chapter 3.4 page 148.

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

If *no database layer* is available yet for the project, a new layer can be created directly using the "Create new schema" dialog. Then, instead of a database layer (cf. Figure 2-5), the "New layer (Derby)" entry is displayed in the "Database layer" drop-down list. When the dialog is confirmed, the new layer (see Figure 2-4) is created in addition to the schema or database.



Database layer: New layer (Derby)

Figure 2-6: Create schema – if no layer is available

OK Click the button to insert the new, empty schema in the tree view and create a schema or database in the configured DBMS. The schema can be further edited with the help of the graphic editor.

Cancel Click the button to cancel the action. A new schema is not created.

A new schema can also be created by importing an export file from another FirstSpirit project (see Chapter 2.3.5.1 page 64).

2.2.1.10 New: Create schema from db

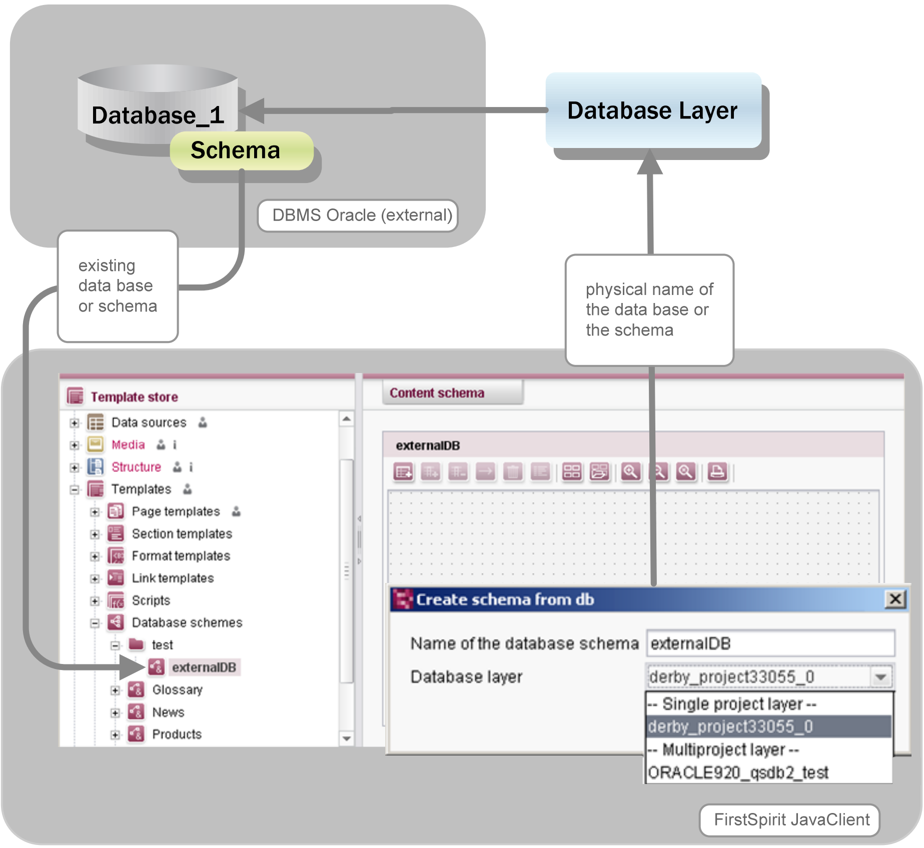
This function can be used to copy an already existing schema from an (external) database into the new schema node.

Instead of creating an empty schema node (cf. Chapter 2.2.1.9), a new schema node is therefore created in the FirstSpirit project on the basis of a database's existing tables and relations. The new schema is created from a database using the "Create schema from database" context menu entry (cf. Chapter 2.2.1.9).

> ! *The structure and content of an external database may not be changed. Unlike internal databases, read access only is possible for external databases, not write access. To this end, the project administrator must enable the "No schema sync" and "Read only" restrictions. In this case the content can be read out of an external schema and created in the FirstSpirit JavaClient as a new schema node. Table templates can then be used to display (but not to change) the content in the Content Store.*

For further information, see "FirstSpirit Manual for Administrators".

The creation of a new schema from an existing database in the JavaClient inserts a new schema node below the "Database schemes" root node. At the same time, the system tries to automatically copy available tables and content from an existing database or from an existing database schema into the graphic schema editor *(for details of "Restrictions", see "FirstSpirit Manual for Administrators")*. For example, if the project administrator has configured an external Oracle database for the project, a schema based on the external database is created via the "New – Create schema from db" context menu. The corresponding tables are also copied. Depending on the database configuration, the editor is given read access to the database via the Content Store and can read out and generate content from the relevant tables (for details of concept, see Chapter 3.6 page 151):



Create schema from db

Name of the database schema: externalDB

Database layer: derby_project33055_0

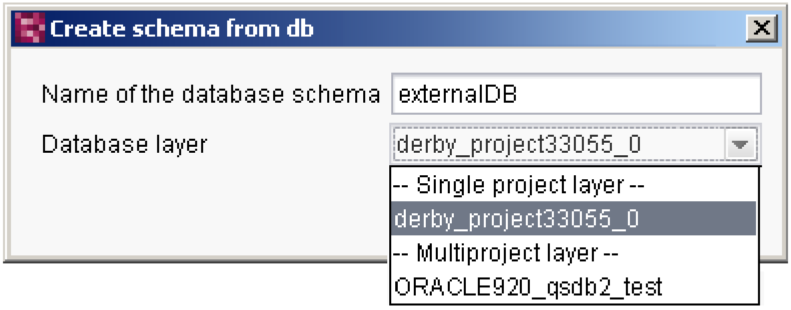
- -- Single project layer --
- derby_project33055_0
- -- Multiproject layer --
- ORACLE920_qsdb2_test

Figure 2-7: Create a schema from an external database

This function can be used on the following elements of the Template Store:

- Root node and folders in the "Database schemes" node

The following input is required to create a new schema:



Create schema from db

Name of the database schema: externalDB

Database layer: derby_project33055_0

- -- Single project layer --
- derby_project33055_0
- -- Multiproject layer --
- ORACLE920_qsdb2_test

Figure 2-8: Create schema from database

Name of the database schema: The name for the database schema must be entered in this field. Unlike the creation of a new, empty schema (cf. Chapter 2.2.1.9), the name *cannot* be freely selected; instead it must be exactly the same as the physical name of the database schema (or the database).

If a name is chosen which does not exist in the database concerned, an empty schema node is created. In this case it is not possible to copy any content (e.g. tables) from the selected database.

Database layer: An existing database layer must be selected in this field (see Chapter 2.2.1.9 page 31).

In FirstSpirit Version 4.2 the naming of the layer types has been changed. The functionality is retained – existing projects need not to be changed. For further information see Chapter 3.4 page 148.

Selection of an external database is only available if the project administrator has configured access to an external database for the project in the project properties.
Please contact the project or system administrator if you have any questions about the database.

Click the button to insert the new schema and the corresponding tables in the tree view and further edit it with the help of the graphic editor.

Click the button to cancel the action. A schema is not created.

A schema can also be created by importing an export file from another FirstSpirit project (see Chapter 2.3.5.1 page 64).

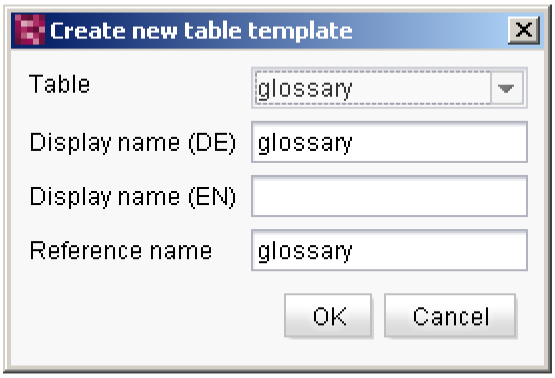
2.2.1.11 New: Create new table template

A table template must be created below the schema for each table entered in the database model. These table templates are used to define the input components via which the editor can subsequently enter data in the corresponding tables and where in the database this data is to be saved (see Chapter 3.6 page 151)).

This function can be used on the following elements of the Template Store:

- Schema node and other table templates in the "Database schemes" node

The following input is required to create a new table template:



Create new table template

Table	glossary
Display name (DE)	glossary
Display name (EN)	
Reference name	glossary

Figure 2-9: Create new table template

Table: The combobox can be used to select the table of the database schema (see Chapter 2.12.1 page 126) for which the table template is to be created.

If the schema does not contain any tables the combobox is empty. It is then not possible to create a table template.

Display name: The language-dependent display name can be individually defined for all project languages (cf. Chapter 2.2.1.1).

Reference name: A unique name for the new table template must be entered in the "Reference name" field. This reference name can be used to establish a unique relation to the table template within a project, for example, a new table can be created within the Content Store on the basis of this table template.

A detailed description of reference and display names is given in Chapter 2.2.1.1 page 25.

OK Click the button to insert the new table template in the tree view and display it in the graphic editor, where it can be further edited.

Cancel Click the button to cancel the action. A new table template is not created.

2.2.1.12 New: Create new query

Several queries can be created for each content schema to limit the number of data records for subsequent output. The conditions a data record must fulfil in order to be included in the results list are specified in these queries.

This function can be used on the following elements of the Template Store:

- Schema node and other queries in the "Database schemes" node

The following input is required to create a new query:

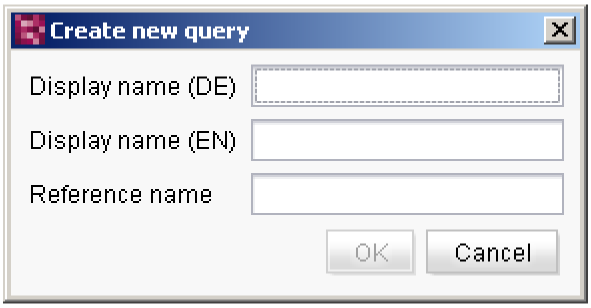


Figure 2-10: Create new query

For a description of the input fields, see Chapter 2.2.1.1 page 25.

OK Click the button to insert the new query in the tree view, where it can be further edited.

Cancel Click the button to cancel the action. A new query is not created.

2.2.1.13 New: Create new workflow

This function can be used on the following elements of the Template Store:

- Root node, folders and other workflows in the "Workflows" node

The following input is required to create a new workflow:

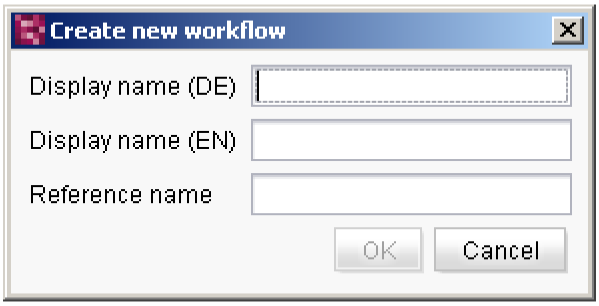


Figure 2-11: Create new workflow

For a description of the input fields, see Chapter 2.2.1.1 page 25.

OK Click the button to insert the new workflow in the tree view; it can then be further edited within the graphic editor.

Cancel Click the button to cancel the action. A new workflow is not created.

2.2.2 Lock/Unlock (Edit Mode On/Off)

To make changes to an object, it is first necessary to switch on edit mode (i.e. lock the object). This prevents simultaneous editing by another user and prevents conflicts which could occur if an element is simultaneously changed.

New objects can be blocked to prevent editing by using the "Edit on/off" context menu entry or the "Edit on/off" icon in the toolbar.

> *After the required changes have been made, edit mode must be switched back off again (i.e. the object unlocked) to release the relevant object for editing by other users. When edit mode is quit all the changes made are automatically saved.*

This function can be used on the following elements of the Template Store:

- Page and section templates
- Format, table format and style templates
- Link configurations and link templates
- Scripts
- Database Schemata
- Table templates
- Queries
- Workflows

2.2.3 Reset Changes

This function can be used to undo changes made during the current editing and which have *not yet been saved*. A confirmation prompt appears before the reset so that content cannot be inadvertently deleted.

The function is only active if edit mode is enabled on an object (cf. Chapter 2.2.2).

This function can be used on the following elements of the Template Store:

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

2.2.4 Cut

This function can be used to cut out an object in the current tree position and place it in the temporary memory (clipboard).

> ! *The "Cut" function is not carried out until the cut object is inserted again (paste). If a cut object is not inserted again (pasted), it is retained in the original tree position, i.e. it is not deleted.*

The Paste function (see Chapter 2.2.6) can then be used to insert these objects in another place.

This function can be used on the following elements of the Template Store:

- All folders of the Template Store
- Page and section templates
- Format (exception: System format templates and folders, which contain system format templates), table format and style templates
- Scripts
- Database Schemata
- Table templates
- Queries
- Workflows

2.2.5 Copy

This function is used to create a copy of the current object and store it in the temporary memory (clipboard). The Paste function (see Chapter 2.2.6) can then be used to insert these objects in another place.

This function can be used on the following elements of the Template Store:

- All folders of the Template Store
- Page and section templates
- Format, table format and style templates
- Scripts
- Database Schemata
- Table templates
- Queries
- Workflows

2.2.6 Paste

This function is used to paste (insert) the content of the temporary memory (clipboard) in the current position of the tree structure. This function is only active if there is data in the temporary memory (on the clipboard) which may be inserted in the current position.

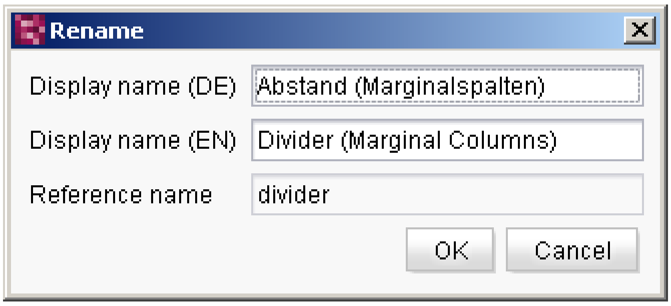
Objects may only be pasted in the areas provided. For example, it is not possible to insert a section template under the page templates node. In this case, the "Paste" entry is disabled.

This function can be used on the following elements of the Template Store:

- All root nodes of the Template Store
- All folders of the Template Store
- Database schemata (for table templates only)

2.2.7 Rename

This function can be used to change the language-dependent display name of the current object in the tree structure of the FirstSpirit JavaClient. After the function has been called a window opens with the display name to date; this can now be changed.



Rename

Display name (DE): Abstand (Marginalspalten)

Display name (EN): Divider (Marginal Columns)

Reference name: divider

Figure 2-12: Rename

Unique reference names may not be changed, as otherwise the relations to this object in the project would be lost (e.g. reference names of section or page templates). In this case the "Reference name" field is disabled and cannot be edited. Folders do not have unique reference names. This name can therefore be renamed at any time).

This function can be used on the following elements of the Template Store:

- All folders of the Template Store
- Page and section templates
- Format, table format and style templates
- Link templates and link configurations

- Scripts
- Database Schemata
- Table templates
- Queries
- Workflows

2.2.8 Delete

This function can be used to delete the currently marked (selected) object or the currently selected sub-tree. Inadvertent deletion is prevented by a confirmation prompt.

The "Delete Objects" function is available on the following elements:

- Page templates (for details of deleting section restrictions, see Chapter 2.2.8.1)
- Section templates (see Chapter 2.2.8.2)
- Format (system format templates cannot be deleted), table format and style templates
- Link configurations and link templates
- Scripts
- Workflows
- Database schemata, queries and table templates

The "Delete sub-trees" function is available on the following elements:

- All folders of the Template Store

For further information on deleting objects and sub-trees, see FirstSpirit Manual for Editors, Chapter 3.2.8 "Delete".

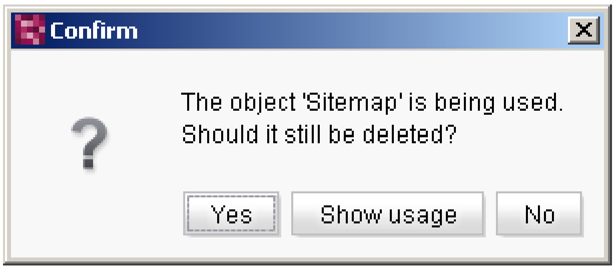
2.2.8.1 Special case: Deletion of section restrictions

! *If all section restrictions below a page template are deleted, there are no longer any restrictions for the creation of sections within a page. Under certain circumstances the layout of the page can be damaged at the same time, for example, because sections are created which are not suitable for the page's layout.*

Restrictions for the use of section templates can be defined within a page template. These restrictions cannot be deleted using the context menu. They are deleted by selecting the restriction below the page template and then clicking <Del>.

2.2.8.2 Deleting section templates (use as section restriction)

Section templates which are still being used as a section restriction within a page template can also be deleted. In this case, the existing uses of the template are displayed to the user before it is deleted:

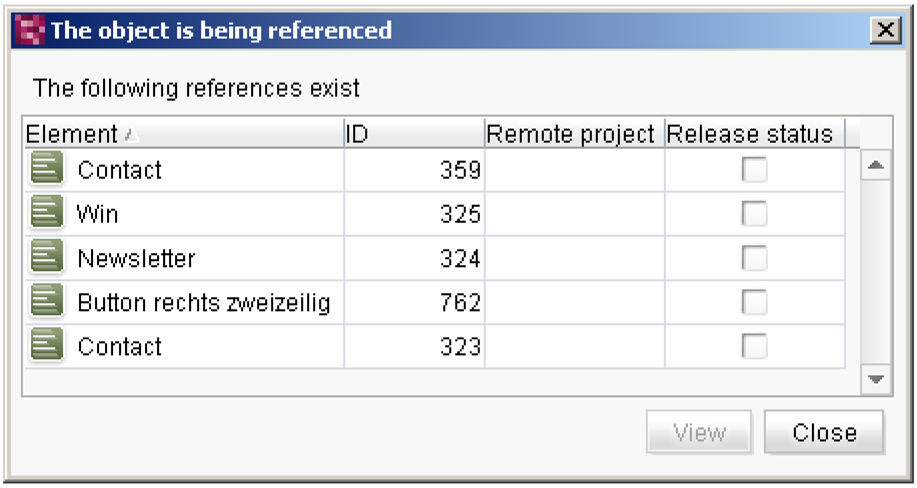


Confirm

The object 'Sitemap' is being used.
Should it still be deleted?

Figure 2-13: Delete object

Show usage Click the button to open the "The object is being referenced" dialog in which the objects are displayed which are still using the object to be deleted (see Figure 2-14). If the section template is still being used as a section restriction within a page template, the corresponding page template is displayed here.



The object is being referenced

The following references exist

Element	ID	Remote project	Release status
Contact	359		□
Win	325		□
Newsletter	324		□
Button rechts zweizeilig	762		□
Contact	323		□

Figure 2-14: The object is being referenced

Yes Click the button to delete the section template despite the existing uses within the project. Below the page template, the section restriction is marked with <DELETED> as labelling within the tree view.

Unlike page templates which do not have any section restriction (all section templates can be used here), a page template with a "DELETED" restriction continues to be usable to a limited extent only. This means:

- After deleting the section template (the only template restriction), NO other section templates can be selected for the corresponding page in the Page Store.
- Only when the "DELETED" restriction below the page template is removed also (see Chapter 2.2.8.1), is there no longer any restriction. Then ALL section templates can be selected for the corresponding page in the Page Store.

! *This function is available for project and server administrators only.*

! *From FirstSpirit Version 4.2 section restrictions are defined using the "Properties" tab of a page template (see Chapter 2.5.2 page 83).*

2.3 Special context menus of the Template Store

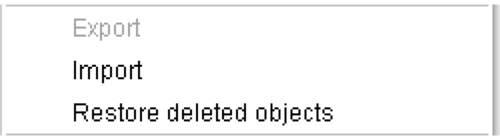


Figure 2-15: Special context menus – Page templates (root)

Special functions for the respective object are available in the middle part of the context menu. The functions are therefore highly dependent on the type of object selected.

Following menu items are available – depending on the selected object:

- Refresh this store (only root node Template Store, see Chapter 2.3.1 page 47)
- Create update (only root node Template Store, see Chapter 2.3.2 page 48)
- Install update (only root node Template Store, see Chapter 2.3.3 page 53)
- Export (see Chapter 2.3.4 page 59)
- Import (see Chapter 2.3.5 page 62)
- Restore deleted objects (see Chapter 2.3.6 page 67)
- Edit extern (see Chapter 2.3.7 apge 69)

2.3.1 Refresh this store

This function can be used on the following elements of the Template Store:

- Root node of the Template Store

This menu entry can be used to update (refresh) the view of the Template Store. This is necessary if several people simultaneously work on and make changes to a project.

> ! *This function may not be used if an object is currently being edited and the changes have not yet been saved! Otherwise the changes not yet saved would be overwritten by the version of the object on the server and would therefore be lost.*

2.3.2 Create update

The update functions "Create update" and "Install update" can be used to swap FirstSpirit templates between a source and a target project. Here a project takes on the role of the development project. This project is the so-called source project in which all templates are created, edited and tested. After testing the "Create update" context menu function is used to create an update file with the required templates. This can then be installed in the target projects (see Chapter 2.3.3).

The update function is basically comparable to the licence-dependent "PackagePool" function, but provides a far smaller scope of functions (see FirstSpirit Module Documentation "PackagePool").

This function can be used on the following elements of the Template Store:

- Root node of the Template Store

> ! *This function is available for project administrators only.*

The "Create update" function is used to create a new update version in the source project. A window opens in which a selection list can be created for the template update:

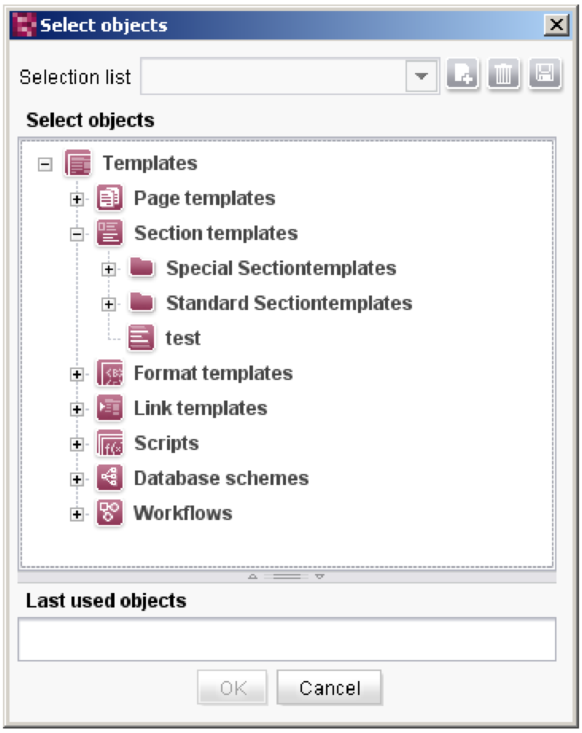
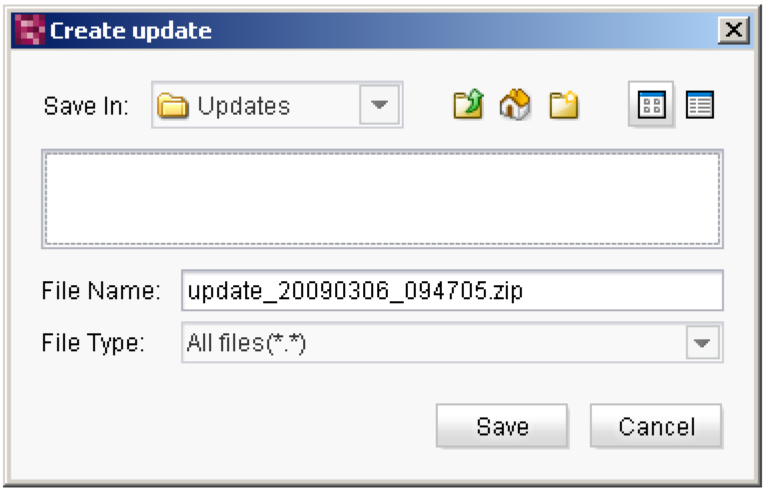


Figure 2-16: Select objects for the template update

- Create selection list (see Chapter 2.3.2.1 page 50).
- Select selection list (see Chapter 2.3.2.2 page 51).
- Delete selection list (see Chapter 2.3.2.3 page 52).
- Select objects (see Chapter 2.3.2.4 page 52).
- Last used objects (see Chapter 2.3.2.5 page 53).

OK Click the button to confirm the template selection. A window opens with the option of filing the packed templates as a Zip file in the local file system. The Zip file contains all the templates in the selection list.



Create update

Save In: Updates

File Name: update_20090306_094705.zip

File Type: All files(*.*)

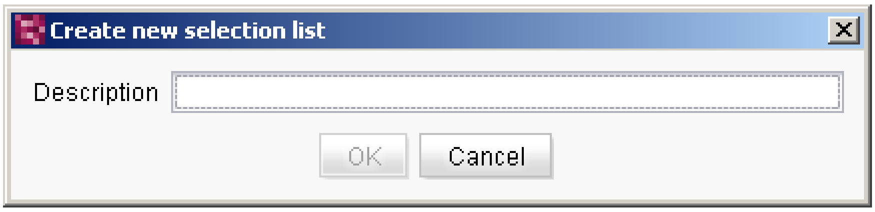
Figure 2-17: Create update – Save Zip file

Cancel Click the button to cancel the action. An update file is not created.

2.3.2.1 Create selection list

A new selection list for the template update can be created in the top part of the "Select objects" dialog box (see Figure 2-16).

If the icon is clicked the "Create new selection list" dialog opens:



Create new selection list

Description

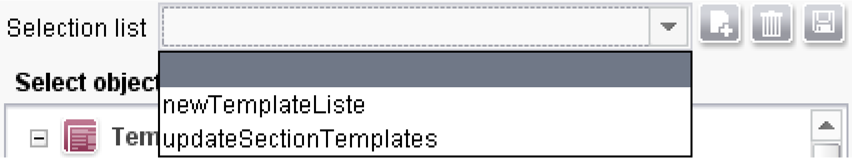
Figure 2-18: Create new selection list for the template update

A description can be added here, under which the selection list is displayed in future in the "Select objects" dialog (see Figure 2-16).

OK Click the button to display the description entered in the selection list in the "Select objects" dialog (see Figure 2-16). Objects for the update can now be added to the new selection list (see Chapter 2.3.2.4 page 52).

Cancel Click the button to cancel the action. A new selection list is not created.

2.3.2.2 Select selection list



Selection list

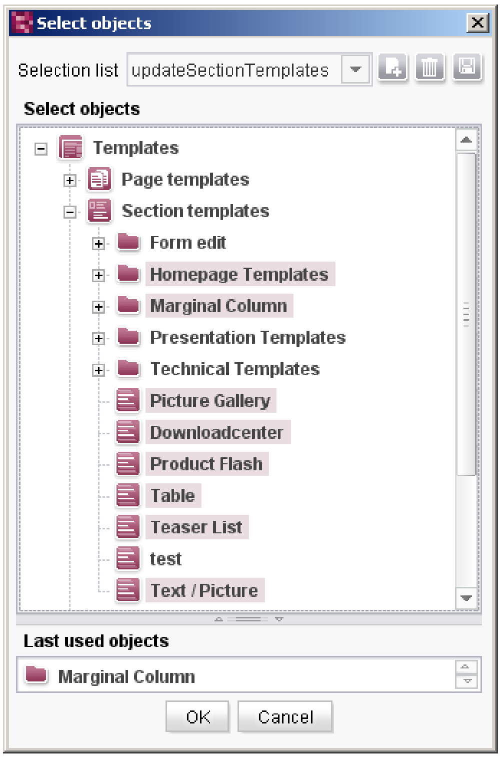
Select object

newTemplateListe
updateSectionTemplates

Figure 2-19: Select selection list

If one or several selection lists are already available for updating the templates, they can be selected using the drop-down list in the top part of the window. The description entered on creating a new selection list (see Chapter 2.3.2.1 page 50) should therefore describe, as precisely as possible, which templates are involved or the intended purpose of the update. The description is also displayed when the update is installed in the target project (see Figure 2-23).

If an existing selection list was selected, all objects of the selection list are displayed marked (selected) in the "Select objects" window.



Select objects

Selection list: updateSectionTemplates

Select objects

- Templates
  - Page templates
  - Section templates
    - Form edit
    - Homepage Templates
    - Marginal Column
    - Presentation Templates
    - Technical Templates
    - Picture Gallery
    - Downloadcenter
    - Product Flash
    - Table
    - Teaser List
    - test
    - Text / Picture

Last used objects

Marginal Column

OK Cancel

Figure 2-20: Selected objects of a selection list

The selected selection list can be edited. Existing objects can be deleted and new ones can be added.

If the changes in the selection list are to be retained, they must be saved by clicking the 💾 button.

2.3.2.3 Delete selection list

🗑 If a selection list is no longer required it can be deleted again by clicking the button. The description of the selection list is no longer displayed in the "Select objects" dialog (see Figure 2-16).

A confirmation prompt appears before the list is deleted to ensure content is not inadvertently deleted.

2.3.2.4 Select objects for the update

The required objects within the tree view can be selected in the "Select objects" area (see Figure 2-20). (For a multiple selection, the "CTRL" key must be pressed at the same time.)

The following rules apply:

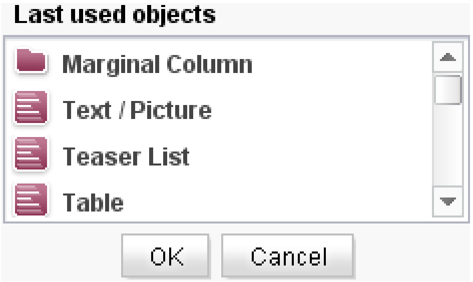
- If an object is selected in the tree view, the whole parent chain of the object is also selected. The higher level elements of the parent chain are therefore a part of the update file.
- If a higher level element is selected, for example a Template Store folder, all lower level elements are also part of the update file

If the selection in the selection list is to be retained, the list must be saved by clicking the 💾 button.

> ! *It is also possible to select an object for the update without creating a new selection list (cf. Chapter 2.3.2.1) or selecting an existing selection list (cf. Chapter 2.3.2.2). In this case, the "empty selection" must be selected in the "Selection list" drop-down list, the required objects are then selected and the update is finished by clicking the* OK *button.*

2.3.2.5 Last used objects

The "Last used objects" area can be maximised or minimised by clicking the ▲▼ icon in the bottom part of the window. The most recently used objects are displayed in this area.



Last used objects

- Marginal Column
- Text / Picture
- Teaser List
- Table

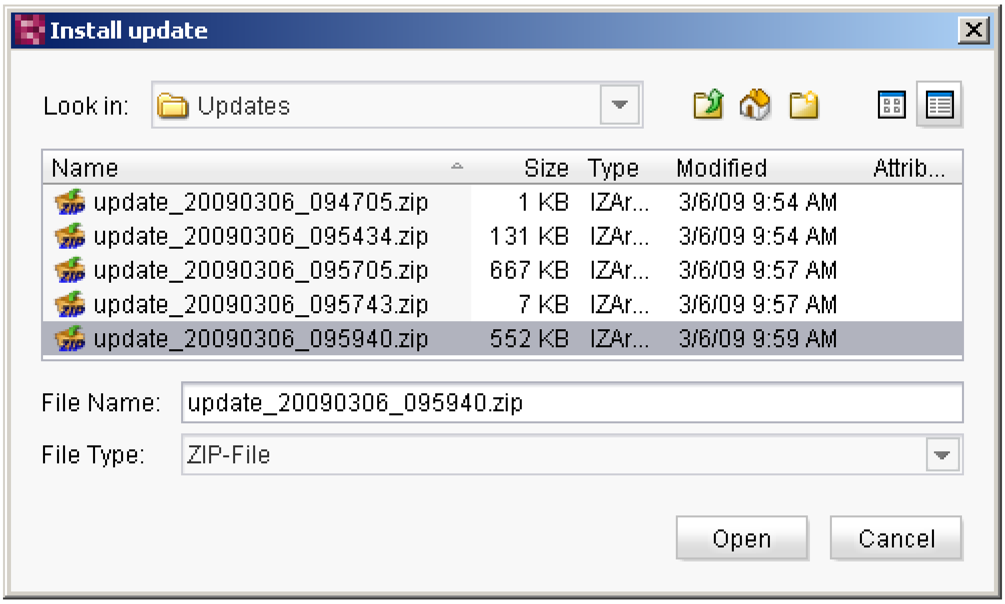
Figure 2-21: Last used objects

2.3.3 Install update

The update functions "Create update" (see Chapter 2.3.2) and "Install updates" can be used to swap templates between a source and a target project. In this case, the update files are created in one project (source project), and can then be installed in one or several other FirstSpirit projects (target projects). The update files can be copied from the source project using the "Install update" context menu function.

> ! *This function is available for project administrators only.*

The "Install update" function is used to load the new update file into the target project. A window opens in which the computer's file system is displayed. The required update version can then be selected from the file system.



Install update

Look in: Updates

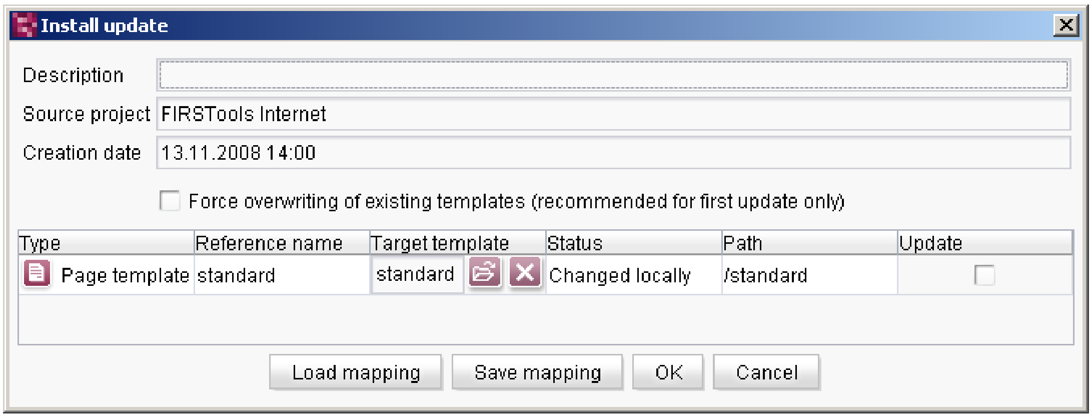
Name	Size	Type	Modified	Attrib...
update_20090306_094705.zip	1 KB	IZAr...	3/6/09 9:54 AM	
update_20090306_095434.zip	131 KB	IZAr...	3/6/09 9:54 AM	
update_20090306_095705.zip	667 KB	IZAr...	3/6/09 9:57 AM	
update_20090306_095743.zip	7 KB	IZAr...	3/6/09 9:57 AM	
update_20090306_095940.zip	552 KB	IZAr...	3/6/09 9:59 AM	

File Name: update_20090306_095940.zip

File Type: ZIP-File

Figure 2-22: Install update – Select update file

A window then opens in which detailed information on the selected update version is displayed:



Install update

Description

Source project FIRSTools Internet

Creation date 13.11.2008 14:00

□ Force overwriting of existing templates (recommended for first update only)

Type	Reference name	Target template	Status	Path	Update
Page template	standard	standard	Changed locally	/standard	□

Figure 2-23: Install update

Description: The name of the selection list in which the update was created in the target project is displayed here (cf. Chapter 2.3.2). The field remains empty if a selection list was not saved for the selected objects.

Source project: Here, the project is displayed, in which the update file was created (cf. Chapter 2.3.2).

Creation date: Date and time at which the update file was created in the source project.

Force overwriting of existing templates: If this option is *selected* (i.e. checkbox is ticked), an update is performed for all the displayed templates of the target project. The update is also carried out for templates with a conflict status in the target project (cf. Chapter 2.3.3.1). On installing the templates, the templates in the target project with the same name are overwritten and local changes are lost. If the checkbox is *deselected*, only updates for which there is no conflict status in the target project are carried out.

All elements of the selected update file are displayed within the table:

Type: Type of element, for example the template type.

Reference name: Reference name of the element in the source project.

Target template: Shows the name of the template in the target project which is to be replaced by the content of this template update. The icons can be used to change the given assignment (mapping) (see Chapter 2.3.3.2).

Status: Status of the template in the target project (for further information, see Chapter 2.3.3.1).

Path: Relative path of the template in the source project.

Update: If the checkbox is selected, the corresponding template is updated in the target project when the dialog is confirmed.

Click this button to load a previously saved template assignment (mapping) from the source project into the templates in the target project. To this end, a dialog box for selecting a mapping file (".map") from the user's local file system opens.

! *The loaded mapping must match the current, selected update. If the mapped (assigned) templates do not exist in the update file, they cannot be mapped.*

Click this button to save the assignment of the templates (mapping) from the source project with the templates in the target project. A dialog opens for selecting a storage location within the user's local file system. The mapping of the source template to a target template can be saved as a file with a ".map" extension.

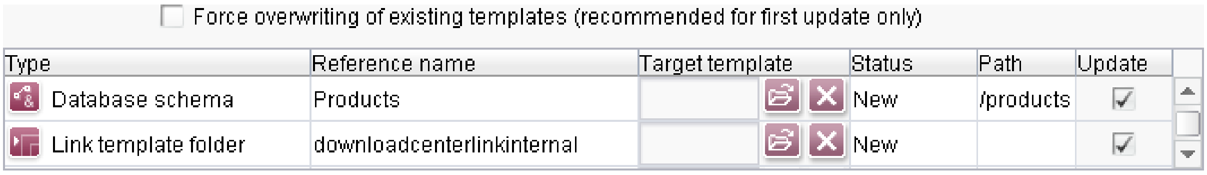
Click this button to confirm the settings from the "Create updates" dialog and perform the update of the templates in the target project selected using the "Perform update" checkbox.

Cancel Click this button to cancel the update of the templates; the settings and assignments made in the dialog are lost – unless they were previously saved – and an update is not performed in the target project.

2.3.3.1 Status of the template on updating

When template updates are installed, different statuses are displayed:

New: This status is displayed when a template update is installed for the first time in the target project. In this case a template with this name does not yet exist in the target project. There is therefore no entry in the "Target template" column. The "Update" checkbox is selected for all templates with this status (see Figure 2-24).

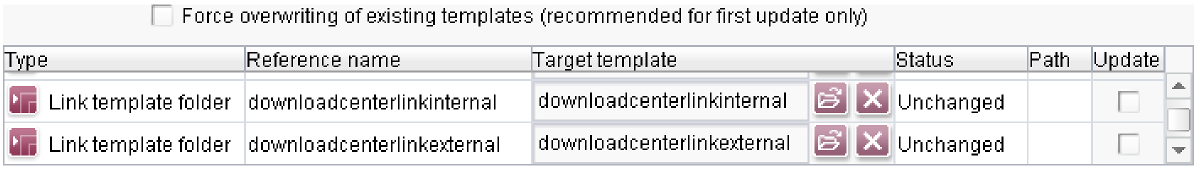


□ Force overwriting of existing templates (recommended for first update only)

Type	Reference name	Target template	Status	Path	Update
Database schema	Products		New	/products	☑
Link template folder	downloadcenterlinkinternal		New		☑

Figure 2-24: New status – for the first installation in the target project

Unchanged: "Unchanged" status is displayed when an attempt is made to perform an update already installed in the target project. This means, this version of the template already exists in the target project. As an update is not necessary, the "Update" checkbox is automatically deselected (see Figure 2-25).



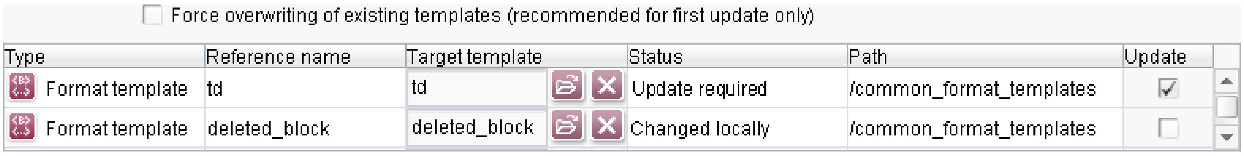
□ Force overwriting of existing templates (recommended for first update only)

Type	Reference name	Target template	Status	Path	Update
Link template folder	downloadcenterlinkinternal	downloadcenterlinkinternal	Unchanged		□
Link template folder	downloadcenterlinkexternal	downloadcenterlinkexternal	Unchanged		□

Figure 2-25: Unchanged status –template version is already installed

Changed locally: "Changed locally" status is displayed if a template has been changed in the target project. In this case the local change in the target project would be overwritten by the update from the source project. The "Update" checkbox is automatically deselected so that the content in the target project is not inadvertently deleted. If the update is required, the checkbox must be selected manually (see Figure 2-26). (The status is displayed even if no status information about the installed updates is available.)

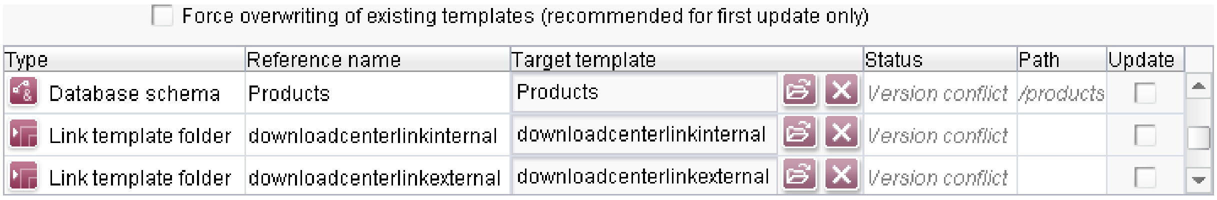
Update required: "Update required" status is always displayed if a new version of the template is created in the source project and the content of the template in the target project has not been changed locally. This status should the the standard case for template updates. As an update is necessary and there is no conflict in the target project, the "Update" checkbox is automatically selected (see Figure 2-26).



□ Force overwriting of existing templates (recommended for first update only)

Type	Reference name	Target template	Status	Path	Update
Format template	td	td	Update required	/common_format_templates	☑
Format template	deleted_block	deleted_block	Changed locally	/common_format_templates	□

Figure 2-26: Changed locally and Update required status



□ Force overwriting of existing templates (recommended for first update only)

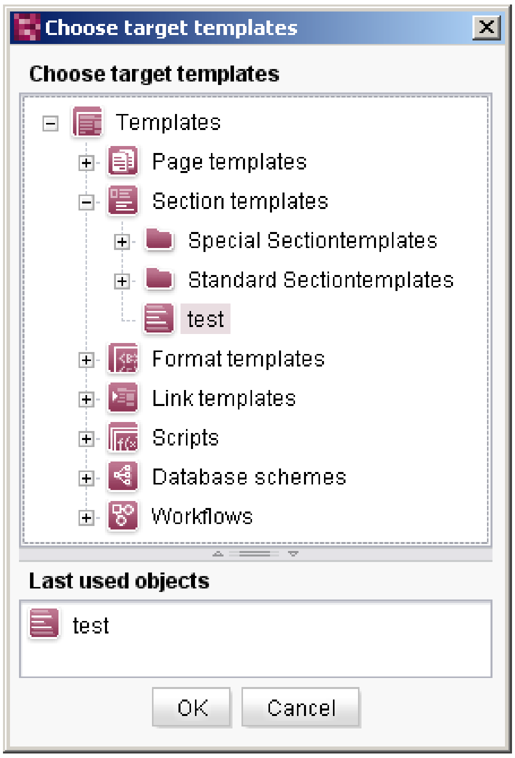
Type	Reference name	Target template	Status	Path	Update
Database schema	Products	Products	*Version conflict*	*/products*	□
Link template folder	downloadcenterlinkinternal	downloadcenterlinkinternal	*Version conflict*		□
Link template folder	downloadcenterlinkexternal	downloadcenterlinkexternal	*Version conflict*		□

Figure 2-27: Conflict status – on attempting to install an older version

Version conflict: This status arises if an attempt is made to install a template in the target project which is older than a previously installed version. As an update is probably not wanted, the "Update" checkbox is automatically deselected. If resetting to an older version is required, the checkbox must be selected manually (see Figure 2-27).

2.3.3.2 Change mapping to template in the target project

Click this button to open the "Choose target templates" dialog with a tree view of the Template Store in the current project (target project). Exactly one target template in the view can be selected for the template update. The new mapping is then copied into the "Install updates" table (cf. Figure 2-23).



Choose target templates

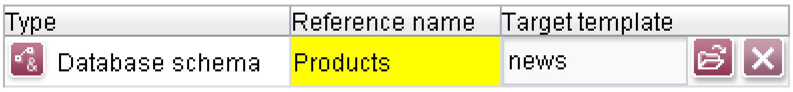
Choose target templates

- Templates
  - Page templates
  - Section templates
    - Special Sectiontemplates
    - Standard Sectiontemplates
    - test
  - Format templates
  - Link templates
  - Scripts
  - Database schemes
  - Workflows

Last used objects

test

Figure 2-28: Choose target template



Type	Reference name	Target template
Database schema	Products	news

Figure 2-29: Mapping if names in source and target project differ

If a target template has been selected whose name is not the same as the name of the template from the source project the name in the table has a yellow background.

☒ Changes to the template mapping can be undone by clicking this icon.

2.3.4 Export

The "Export" context menu entry can be used to group together objects from a project in a compressed Zip file and save them in the local file system. The export files can then be used to import the exported content of a project (source project) into other FirstSpirit projects (target project) (see Chapter 2.3.5 page 62). The selection available depends on the object type on which the context menu or function was opened:

> ! *The "Export" context menu available in the FirstSpirit JavaClient is a client-side function and therefore sets substantial requirements for the main memory of the client system when large quantities of data are involved. This function should therefore only be used to export small quantities of data.*

This function can be used on the following elements of the Template Store:

- All folders of the Template Store (see Chapter 2.3.4.1 page 59).
- Page and section templates (see Chapter 2.3.4.2 page 59).
- Format templates (not system format templates)(see Chapter 2.3.4.2 page 59)
- Style and table format templates (see Chapter 2.3.4.3 page 60).
- Link templates and link configurations (see Chapter 2.3.4.4 page 60).
- Scripts (see Chapter 2.3.4.5 page 61).
- Database Schemata (see Chapter 2.3.4.6 page 61).
- Table templates (see Chapter 2.3.4.7 page 61).
- Queries (see Chapter 2.3.4.7 page 61).
- Workflows (see Chapter 2.3.4.8 page 62).

When the context menu is opened, an export window opens first for selection of the required storage location for the export file in the local file system of the workstation.

2.3.4.1 Export folders

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

2.3.4.2 Export templates

Templates can be exported and imported into other FirstSpirit projects for use there. Use of the update function for templates is recommended if templates are to be frequently exported from a source project and imported into a target project, because extended information for the mapping and on the update status in the target project is available (see Chapter 2.3.2 page 48).

2.3.4.3 Export style and table format templates (from V4.1)

! *This function is released for FirstSpirit Version 4.1 and higher only.*

Link configurations and link templates are exported in a similar way to templates (see Chapter 2.3.4.2 page 59).

Style and table format templates are closely linked (see also Chapter 2.8 and 2.9 from page 95) and should therefore be exported together wherever possible. This is best done by grouping them together in one folder, which can be exported as described in Chapter 2.3.4.1. But style templates can also be easily exported individually and re-imported at a later date; the style template used as the standard and the style templates used in the display rules must always be exported together with table format templates.

2.3.4.4 Export link configurations and link templates

Link configurations and link templates are exported in a similar way to templates (see Chapter 2.3.4.2 page 59).

Wherever possible, link configurations should always be exported together with the corresponding link templates. If this is not possible, for example, because new link templates have been created in the source project, they can also be exported individually. In this case the template must be imported into the target project on the node of the matching link configuration. Otherwise an error message is displayed on importing ("Export data not compatible").

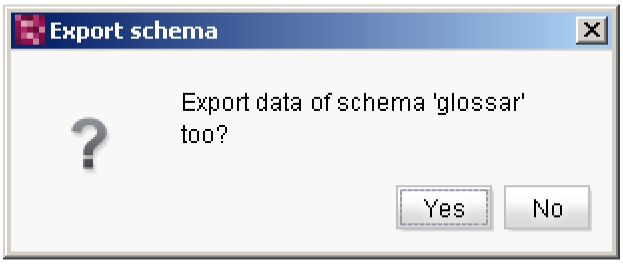
2.3.4.5 Export scripts

Scripts are exported in a similar way to templates (see Chapter 2.3.4.2 page 59).

2.3.4.6 Export schema

Schemata can be exported and imported into other FirstSpirit projects for use there (see Chapter 2.3.5.1 page 64).

The following dialog is displayed after the export window in which the required storage location for the export file can be selected from the local directories of the workstation has been displayed:



Export data of schema 'glossar' too?

Figure 2-30: Export data on exporting a schema

Yes Click this button to add the data of the schema of the Content Store of the current project to the export file. This data is then available to the users of the second project when the export file is imported into another FirstSpirit project.

No Click this button to add the schema only, but not the data of the schema, from the Content Store of the current project to the export file. When the export file is imported into another FirstSpirit project the schema is available to the users of the second project, but not the data from the Content Store of the first project.

The schema's corresponding table templates are automatically added to the export file.

2.3.4.7 Export table templates and queries

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

In this case the template (and/or query) must be imported on the matching schema node in the target project. Otherwise errors can occur in the project, because the mapping of the table template no longer matches the tables of the schema ("The referenced table "xy" does not exist").

2.3.4.8 Export workflows

This function can be called using the context menus at folder level and on the workflow level.

This function can be used to export both individual workflows and folders with all the subfolders and workflows it contains into the computer's file system. In this way, workflows can be used at a later date, e.g. in other projects.

To do this, an export window opens in which you can select the required storage location for the export file in the local directories of the workstation.

If scripts are used within a workflow, they can also be added to the export file. To do this, the required workflow is selected first in the tree view and then the script with Ctrl + click the required script. If the "Export" function is now selected in the context menu, both objects are contained in the Zip file. However, scripts can also be exported separately at a later date (see Chapter 2.3.4.5 page 61).

2.3.5 Import

The "Import" context menu entry can be used to insert previously exported objects from a source project into another FirstSpirit project (target project). To do this, the required Zip file must first be selected from the local file system of the workstation and imported in a suitable position in the target project.

If the imported content does not match the context of the target project they are – as far as possible – automatically imported in the correct context of the target project. In this case the import is performed independently of the object on which the "Import" context menu was selected. For example, if an attempt is made to import the export file of a script into the "Workflows" area, the selected script is nevertheless imported into the target project, however, not into the "Workflows" area but into the correct "Scripts" area of the target project.

This automatic correction does not happen in all cases. If the system cannot map the object in the target project to which the import file matches, an error message is displayed instead.

The selection available depends on the object type on which the context menu or function was opened:

This function can be used on the following elements of the Template Store:

- All folders of the Template Store
- Style and table format templates (see Chapter 2.3.5.1 page 64).
- Link configurations
- Database Schemata (see Chapter 2.3.5.2 page 65).
- Workflows (see Chapter 2.3.5.3 page 67)

This function can be used to import exported page/section templates into the file system as well as exported folders with all the templates and sub-folders contained in the selected folder.

To do this, an import window opens in which you can search for the required export file in the local directories of the workstation.

Mapping of template sets in the target project: When importing a template from a source project into a target project, FirstSpirit tries to import the content of the template sets too:

- First it is tried to map the template sets on the basis of the name (*name of the template set in the source project* to *name of the template set in the target project*). I.e., if the names of the template sets in the source and target project are identical, the content will be imported into the target project.
- If a mapping by name does not succeed, because the names of the template sets of the target project differ from those in the source project, it is tried to map the template sets an the basis of the presentation channel in the next step (*name in the source project* to *presentation channel in the target project*). For example, a template set with the name „html" from the source project will be mapped to the first located presentation channel „HTML" in the target project (independent from the name of the channel in the target project).
- If neither mapping by name nor mapping by presentation channel is successful, the content of the template sets can not be imported from the source project into the target project and must be entered or copied manually.

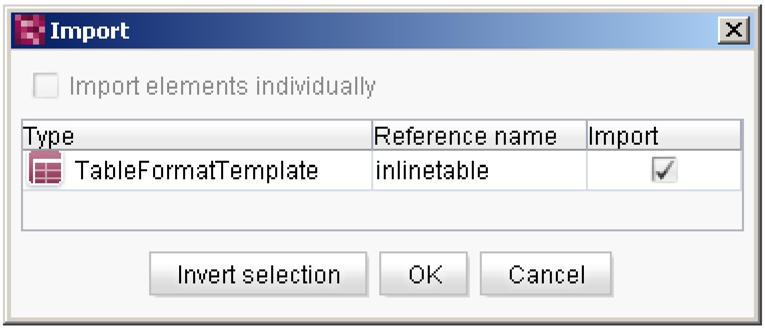
2.3.5.1 Import style and table format templates (from V 4.1)

This function is released for FirstSpirit Version 4.1 and higher only. Screenshots are therefore displayed in the new "LightGray" Look & Feel. The display in the "Classic" Look & Feel can differ slightly.

Style and table format templates can be imported from other FirstSpirit projects. To do this, an export file of the required templates must be exported first from another FirstSpirit project (see Chapter 2.3.4.3 page 60).

Style templates can also easily be imported individually. Table format templates should be imported together with the style templates used (see Chapter 2.8 from page 95). If these style templates are not exported at the same time, table format templates can nevertheless be imported, but any references to the style templates are lost.

To import style and table format templates, the context menu is opened on the "Format templates" root node or on a folder below this root node and the "Import" function is selected. After selecting the required export file, the Import dialog appears:



☐ Import elements individually

Type	Reference name	Import
TableFormatTemplate	inlinetable	☑

Figure 2-31: Import table format template

Import elements individually: This function 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 selected the element concerned is imported into the target project, if the checkbox is deselected the element is not imported.

Invert selection Click this button to selection made within the "Import" column is inverted.

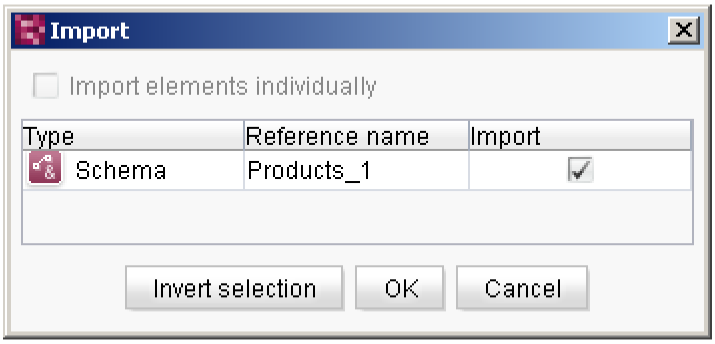
OK Click this button to confirm the selection in the dialog and the "Select database layer" dialog opens (see Figure 2-33).

Cancel Click the button to cancel the import.

2.3.5.2 Import schema

Schemata can be imported from other FirstSpirit projects. To do this, an export file of the required schema must be exported first from another FirstSpirit project (see Chapter 2.3.4.6 page 61).

The export file of the first FirstSpirit project can now be imported into the second project. To do this, the context menu is opened on the "Database Schemata" root node or a folder below this root node and the "Import" function is selected. After selecting the required export file, the Import dialog appears:



☐ Import elements individually

Type	Reference name	Import
Schema	Products_1	☑

Figure 2-32: Import schema with table templates

Import elements individually: This function is not available in the Template Store.

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

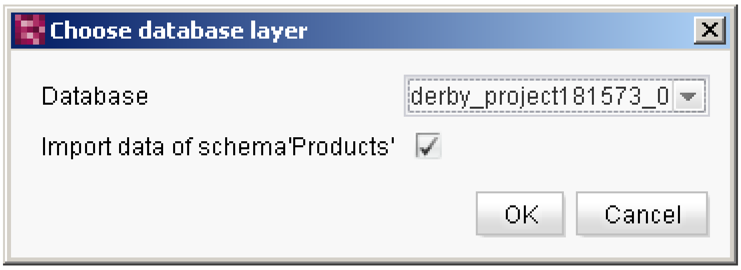
Reference name: Name of the element contained in the export file.

Import: If this checkbox is selected, the element concerned (and all subordinate elements, e.g. table templates) is imported into the target project, if the checkbox is deselected the element is not imported.

Invert selection Click this button to selection made within the "Import" column is inverted.

OK Click this button to confirm the selection in the dialog and the "Select database layer" dialog opens (see Figure 2-33).

Cancel Click the button to cancel the import.



Choose database layer

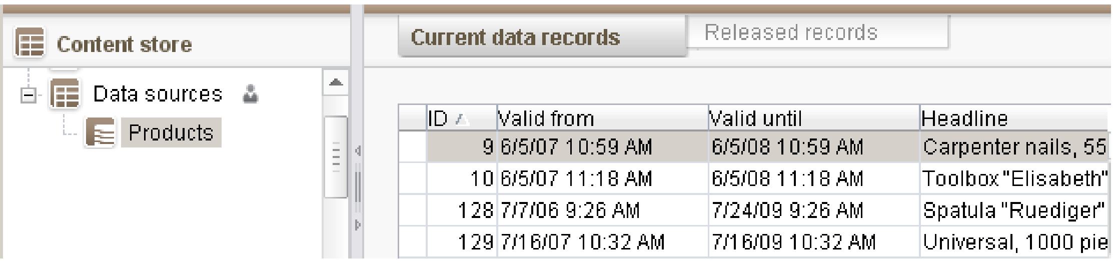
Database: derby_project181573_0

Import data of schema'Products' ☑

Figure 2-33: Import schema

Database: Select the required database layer. All layers enabled for the project by the project administrator are displayed in the drop-down list.

Import data from schema 'xy': If this checkbox is selected, the data entered for this schema to date in the Content Store of the source project is copied into the target project. On creating a table in the Content Store of the target project, based on the imported schema information, the structured data entered to date in the target project is displayed:



ID	Valid from	Valid until	Headline
9	6/5/07 10:59 AM	6/5/08 10:59 AM	Carpenter nails, 55
10	6/5/07 11:18 AM	6/5/08 11:18 AM	Toolbox "Elisabeth"
128	7/7/06 9:26 AM	7/24/09 9:26 AM	Spatula "Ruediger"
129	7/16/07 10:32 AM	7/16/09 10:32 AM	Universal, 1000 pie

Figure 2-34: Table view of the structured data in the target project

The data can also be changed within the target project, i.e. it is not write-protected.

If the schema only is to be copied, but not the data entered to date for this schema in the Content Store of the source project, the schema must either be exported in the source project without data (see Figure 2-30) or the "Import data from schema "xy" checkbox must be deselected in the target project. In both cases, the data to date (based on the schema concerned) is ignored during the import, i.e. is protected against access from the target project.

If the data to date from the source project is to be displayed in the target project, but changes to the structured data is to be prevented, write protection must be enabled on the selected database layer after importing the data.

2.3.5.3 Import workflows

This function can be called using the context menus in the workflows area and at folder level.

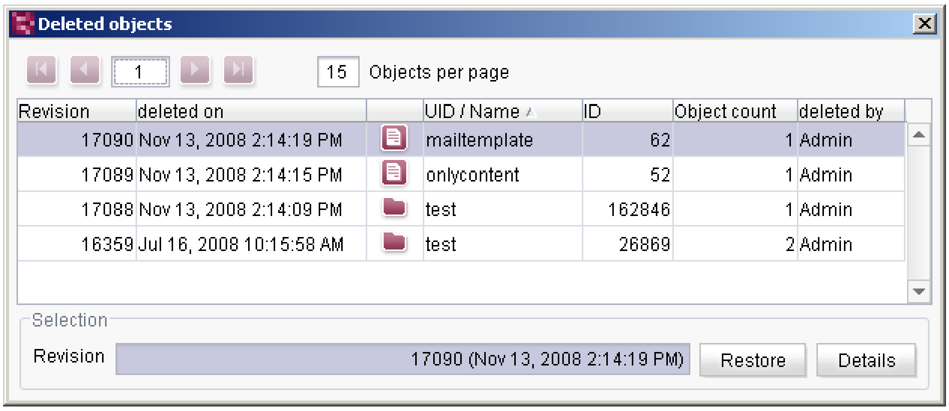
This function can be used to import exported workflows into the file system as well as exported folders with all the sub-folders and workflows (and any scripts) contained in the selected folder.

To do this, an import window opens in which you can search for the required export file in the local directories of the workstation.

> ! *The rights (permissions) configuration may have to be project-specifically adjusted on importing (see Chapter 4.6 page 194).*

2.3.6 Restore deleted objects

The “Restore deleted objects” function can be called for page, section and format templates and scripts, both at root and at folder level; in addition, it can be called for database schemata and link templates at schema and link configuration level too; in the case of workflow it can be called at root level only. If an object has been mistakenly deleted from the tree structure it can be restored with the help of this function. After clicking a window opens with the deleted objects:



15 Objects per page

Revision	deleted on		UID / Name	ID	Object count	deleted by
17090	Nov 13, 2008 2:14:19 PM		mailtemplate	62	1	Admin
17089	Nov 13, 2008 2:14:15 PM		onlycontent	52	1	Admin
17088	Nov 13, 2008 2:14:09 PM		test	162846	1	Admin
16359	Jul 16, 2008 10:15:58 AM		test	26869	2	Admin

Selection

Revision 17090 (Nov 13, 2008 2:14:19 PM)

Figure 2-35: Deleted objects

All objects for which a backup exists are displayed at root level, while at folder level only the objects located underneath this folder are displayed. The following information is given for each object:

Revision: Version number of the deleted object.

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

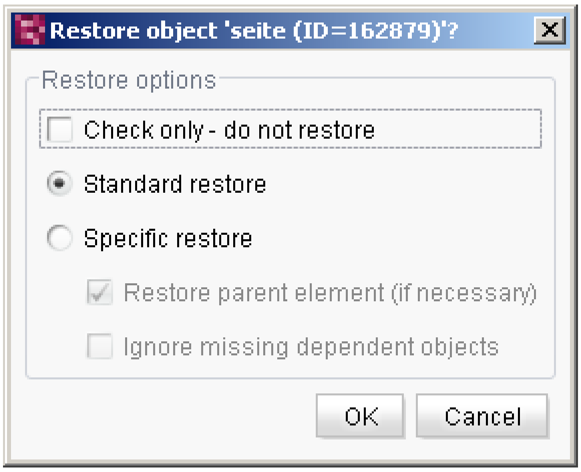
UID / Name: The reference name of the deleted object.

ID: The unique ID number of the deleted object.

Number of objects: The number of objects located in the tree structure below the deleted object. The Details button can be used to display these in a popup window. These hierarchically lower level objects are also inserted again by the restore function.

deleted by: Name of the user who deleted the object.

To restore, it is only necessary to select the required object and press the Restore button.



Restore object 'seite (ID=162879)'?

Restore options

□ Check only - do not restore

● Standard restore

○ Specific restore

☑ Restore parent element (if necessary)

□ Ignore missing dependent objects

Figure 2-36: Restore deleted objects

Check only – do not restore: If this option is selected, the system checks whether the object can be restored without errors. To this end, the restore action is simulated, but the deleted object is not restored. A popup window then appears showing whether the object can be restored or not.

Standard restore: This option is set as a default. If the object is restored with this option, the restore action is performed directly depending on the object. Therefore, different options can be selected in the "Specific restore" area, depending on the object.

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

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

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

! *This option is available to project administrators only.*

In the next dialog, the position at which the deleted object is to be added can be selected.

2.3.7 Edit extern

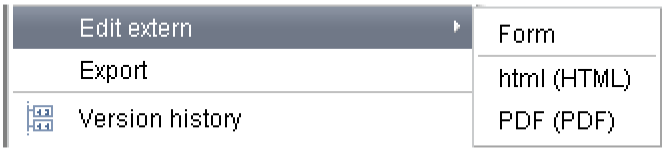
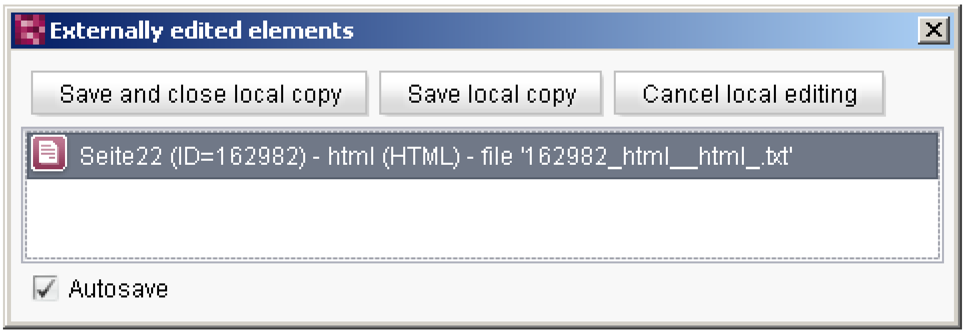


Figure 2-37: Edit extern function

This function can be called using the context menu on page templates and section templates and is further divided into several areas: All **template sets** set in the server and project configuration for this project are listed, in addition there is a **Form** and a **DTD** area.

If one of the existing editing areas is enabled, the corresponding source file opens in an external editor. To edit a source file in an external editor, an editor should be entered in the user settings of the Global Store. In addition, another window appears in which all the open templates are displayed.



Seite22 (ID=162982) - html (HTML) - file '162982_html__html_.txt'

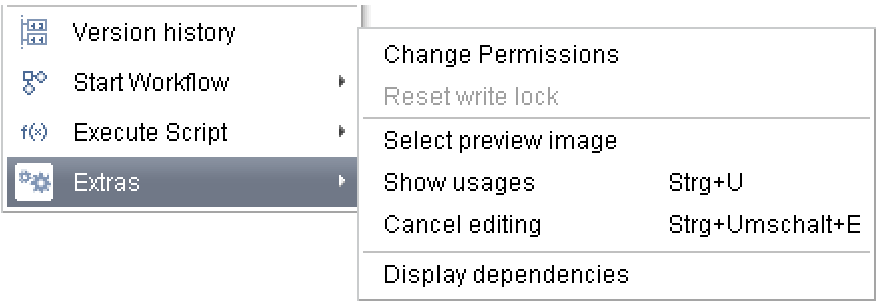
☑ Autosave

Figure 2-38: Edit externally

After selecting the templates, changes to the source text are saved by pressing the "Save and close local copy" button or the "Save local copy" button. In the first case the editor is then closed. In the same way, changes not yet saved can be undone using "Cancel local editing".

Autosave: If this tick is set, all changes saved in the external editor are automatically saved in the FirstSpirit client too.

2.4 Administrative context menus of the Template Store

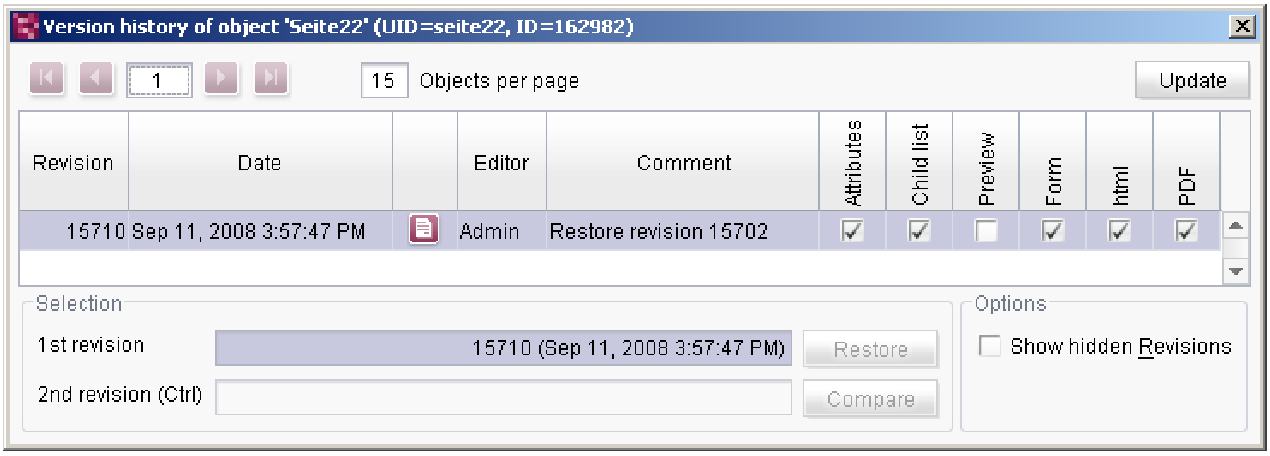


Following menu items are available – depending on the selected object:

▪ Version history	(see Chapter 2.3.1 page 47)
▪ Start Workflow	(see Chapter 2.4.2 page 72)
▪ Execute Script	(see Chapter 2.4.3 page 73)
▪ Search in templates	(see Chapter 2.4.4 page 73)
▪ Extras – Change Permissions	(see Chapter 2.4.5 page 73)
▪ Extras – Reset write lock	(see Chapter 2.4.6 page 73)
▪ Extras – Select preview image	(see Chapter 2.4.7 page 73)
▪ Extras – Show properties (from V4.2)	(see Chapter 2.4.8 page 74)
▪ Extras – Show usages	(see Chapter 2.4.9 page 76)
▪ Extras – Accept template changes	(see Chapter 2.4.10 page 76)
▪ Extras – Cancel editing	(see Chapter 2.4.11 page 77)
▪ Extras – Convert link template (from V4.2)	(see Chapter 2.4.12 page 77)
▪ Extras – Change reference name	(see Chapter 2.4.13 page 78)
▪ Extras – Display dependencies (from V4.1)	(see Chapter 2.4.14 page 78)
▪ Extras – Create a copy of this workflow	(see Chapter 2.4.15 page 80)

2.4.1 Version history

This function can be used to call the version history of each object in the Template Store.



Version history of object 'Seite22' (UID=seite22, ID=162982)

Revision	Date		Editor	Comment	Attributes	Child list	Preview	Form	html	PDF
15710	Sep 11, 2008 3:57:47 PM		Admin	Restore revision 15702	☑	☑	☐	☑	☑	☑

Selection

1st revision: 15710 (Sep 11, 2008 3:57:47 PM)

2nd revision (Ctrl):

Options

☐ Show hidden Revisions

Figure 2-39: Version history on a page template

General information about FirstSpirit version history and the functions of the dialog in Figure 2-39 is given in the *FirstSpirit Manual for Editors*, Chapter 11.10.

In addition to the general revision information available (revision, date, editor, comment) the right-hand part of the list shows the element of the object at which a change was made resulting in the issue of a new revision number (e.g. attributes, child list, preview, output channels). This depends on the objects on which the version history was opened.

2.4.2 Start Workflow

If workflow is not yet active for the selected object, all workflows defined in the permissions system for these nodes in the tree structure 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 it can be switched to another workflow action/state under this menu item.

Detailed documentation of workflows is given in Chapter 4 from page 152 and in the *FirstSpirit Manual for Editors*, Chapter 12.

2.4.3 Execute Script

All scripts which can be opened in this position in the JavaClient are listed under this menu item. Scripts enable pre-programmed actions or calculations to be executed. Information on script development in FirstSpirit is given 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 of the FirstSpirit menu bar. Further information on this search is given in the *FirstSpirit Manual for Editors*, Chapter 3.

2.4.5 Extras – Change Permissions

This function can be used to define the permissions for the current nodes in the tree structure. It can be opened on all nodes using the context menu.

Detailed documentation on the definition of permissions is given in the *FirstSpirit Manual for Editors*, Chapter 13.

✓ 4.1 For improved clarity, the entries of the lists in the "Inherited permissions" and "Permissions defined in this object" areas are automatically sorted alphabetically in FirstSpirit Version 4.1. Groups are displayed first, then the users.

2.4.6 Extras – Reset write lock

If write protection (write lock) exists for the selected node due to an active workflow the write lock can be cancelled using this function. (The write lock is indicated by italic lettering in the tree.) Detailed information on the write lock within workflows is given in Chapter 4.7 from page 208.

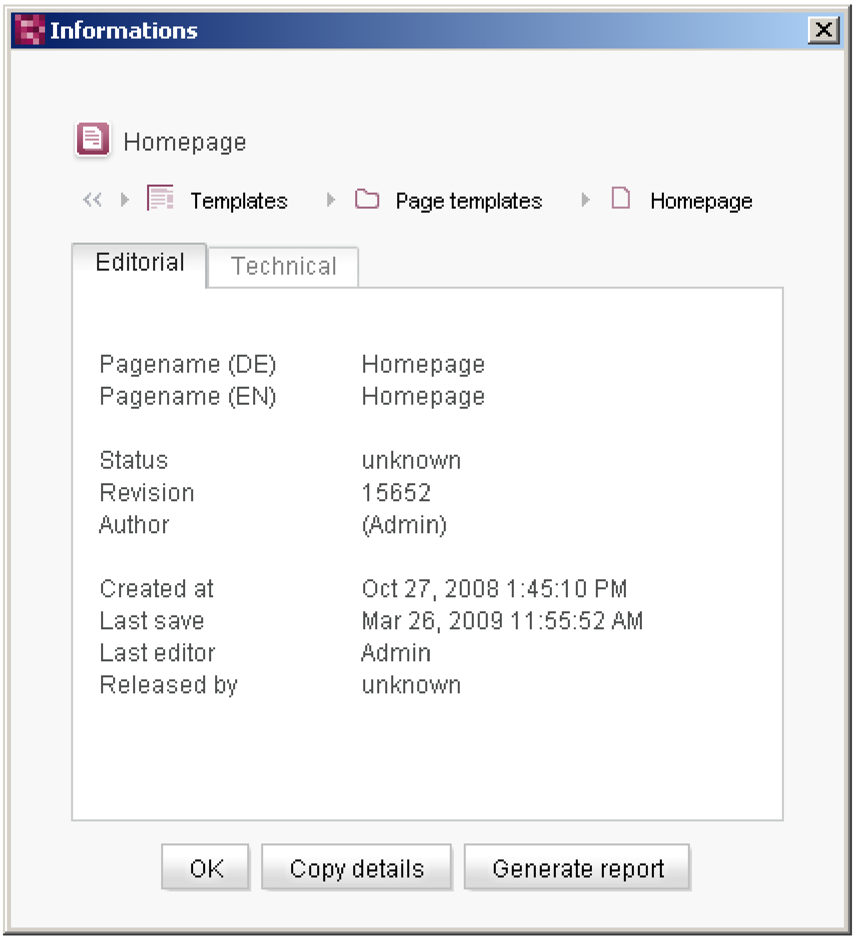
2.4.7 Extras – Select preview image

This function can be opened using the context menu at page/section level. The respective object must be in Edit mode.

This function can be used to select a preview graphic for the Preview tab of the respective object. To do this, a file window opens in which you can search for the required preview graphic in the local directories of the workstation. The graphic file must have the extension "gif", "jpg" or "png".

2.4.8 Extras – Show properties (from V4.2)

Using this function the properties of an object including the following information can be shown. The information can vary according to the object type.



Pagename (DE)	Homepage
Pagename (EN)	Homepage
Status	unknown
Revision	15652
Author	(Admin)
Created at	Oct 27, 2008 1:45:10 PM
Last save	Mar 26, 2009 11:55:52 AM
Last editor	Admin
Released by	unknown

Figure 2-40: Properties of a page template – Editorial

Use the path to show the properties of other objects.

Tab Editorial

Pagename: Display names of the object (language-dependent)

Status: shows the status (e.g. “Not released“, “Released“, “Changed (not released)“)

Revision: shows the revision

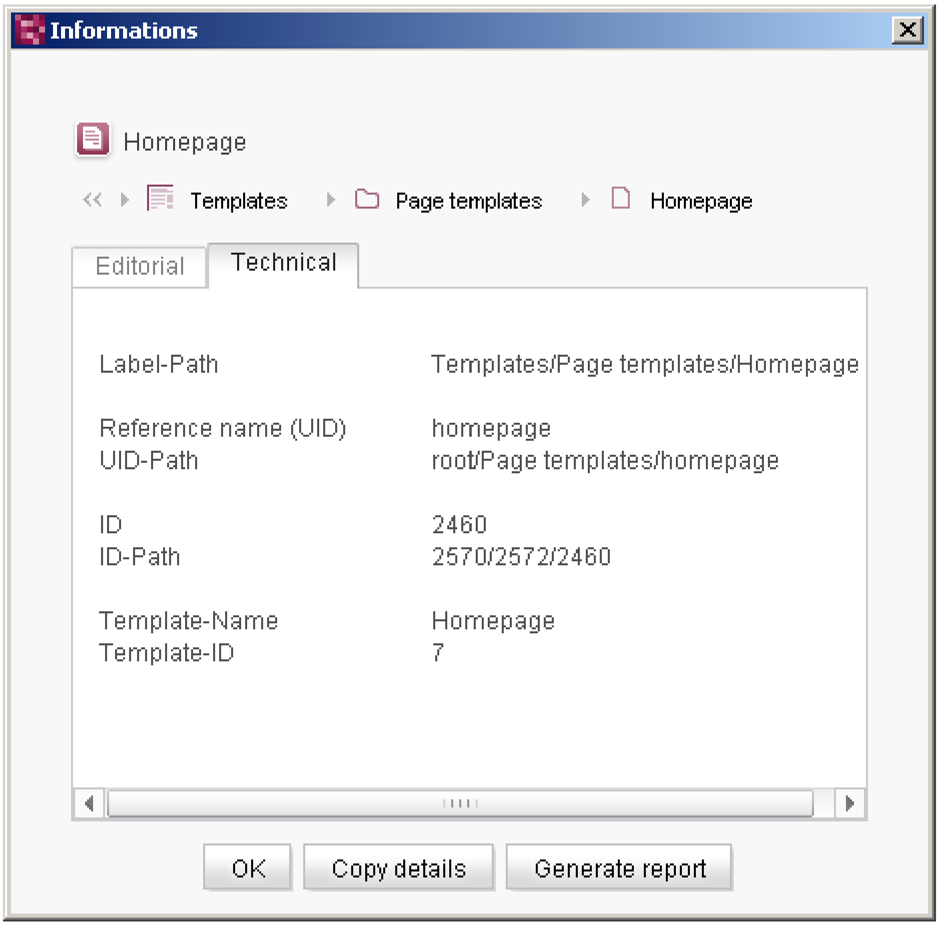
Author: name of the user who has created the object

Created at: Date and time of the object's creation in the JavaClient

Last save: Date and time of the object's last saving

Last editor: name of the user who last changed the object

Released by: name of the user who has released the object



Label-Path	Templates/Page templates/Homepage
Reference name (UID)	homepage
UID-Path	root/Page templates/homepage
ID	2460
ID-Path	2570/2572/2460
Template-Name	Homepage
Template-ID	7

Figure 2-41: Properties of a page template – Technical

Tab Technical

Label-Path: Path to the selected object (display names)

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

UID-Path: Path to the selected object (reference names)

ID: ID of the object

ID-Path: Path to the selected object (IDs)

Template-Name: Display name of the template

Template-ID: ID of the template

Depending on the object type the link “View template“ is displayed. Use this linkg to switch directly to the template on which the object bases.

Using the button “OK“ the properties dialogue will be closed. Using the button “Copy details“ all information of this dialogue can be copied to the clipboard. Use the button „Generate report“ to generate a HTML page from this information. You can also add a comment.

2.4.9 Extras – Show usages

This function can be opened using the context menu on page, section and format templates, scripts and table templates.

It can be used to automatically switch to nodes in other stores based on the object on which this function was performed. If the multiple uses of the object exist, a new window opens in which all nodes (e.g. sections from the Page Store) which are based on the current object are displayed. Double-click one of these entries to display the corresponding nodes in the directory tree.

2.4.10 Extras – Accept template changes

This function can be used to adopt changes to the definition of the content areas in a page template for existing pages.

The function is available on page templates in the Template Store only.

For example, if a content area is added within a page template, this change does not automatically affect an existing page. The "Accept template changes" function can be used to update the existing pages when a page template is changed. The function checks the definition of the content areas in the page template against the template areas of the pages which use this template:

- If content areas are found which are defined in the template but which do not exist on the existing page, these content areas are added in the page.
- If the reverse is true, i.e. content areas are found which are missing in the template but exist in the page,
  - they are removed from the page if they do not contain any sections.

- they are kept on the page if they contain sections.

2.4.11 Extras – Cancel editing

This function can be used to cancel edit mode on nodes without saving any changes made. However, changes which have already been saved using CTRL + S or the Save function of the toolbar cannot be undone.

2.4.12 Extras – Convert link template (from V4.2)

In FirstSpirit Version 4.2, the configuration options for links are considerably enhanced by the introduction of generic link editors (see Chapter 2.10.6 page 118).

Whereas static and generic link editors can still be used in parallel in FirstSpirit Version 4.2, **static link editors will no longer be supported from FirstSpirit Version 5.0**. The introduction of "generic link editors" in FirstSpirit Version 4.2 therefore also helps to migrate existing projects to FirstSpirit Version 5.0.

Conversion of the existing link templates to the new, generic link editors is supported by FirstSpirit. The automatic conversion can be started on the existing link templates using the context menu (“Extras” – “Convert link template”). With this, the static input fields used to date are replaced by the new input components in the form area. Project-specific manual adjustment of the layout may be necessary following the automatic conversion.

> ! *The migration of the link editors must be fully completed in FirstSpirit Version 4.2. Otherwise the projects cannot be migrated to FirstSpirit Version 5.0.*

> ! *These changes are not downwards compatible. These changes must be manually reset on downgrading from FirstSpirit Version 4.2 back to Version 4.1.*

The changeover to generic link editors changes the internal data structure of DOM elements. This can potentially lead to problems with existing scripts. The scripts must be manually adjusted if necessary.

2.4.13 Extras – Change reference name

Using this function the reference name of objects can be changed afterwards.

The object may have still references in the project. These will become invalid if the reference name of the object is changed. For this reason, a warning is shown. If you select “Change nevertheless“ you can change the reference name of the object in the following dialogue.

2.4.14 Extras – Display dependencies (from V4.1)

Essential functions of FirstSpirit are based on the so-called reference graph of a project. This had to be calculated for a project for the first time ever in Version 3.1 and since then has been constantly extended and further developed. The reference graph of a project is used to recognise dependencies within the project and is therefore an essential component of complex functions, for example, the server-side release (see Chapter 7 page 272).

From FirstSpirit Version 4.1, project administrators can request visualisation of the reference graph for an object via the "Extras – Display dependencies" context menu. This means it is possible to identify the dependencies of an object, even in complex projects.

Reference graphs of single data sets of the Content Store can be displayed using the context menu of the respective data set.

The tabs show the dependencies of the object, in the form of incoming and outgoing edges, both for the current status and for the most recently released status (see Figure 2-42).

The display can be changed over to a hierarchical view, which is advisable especially for complex dependencies (see Figure 2-42). Direct updating on making changes and zooming within the view is possible using the buttons in the top part of the window. The view can also be stored as an image for subsequent use.

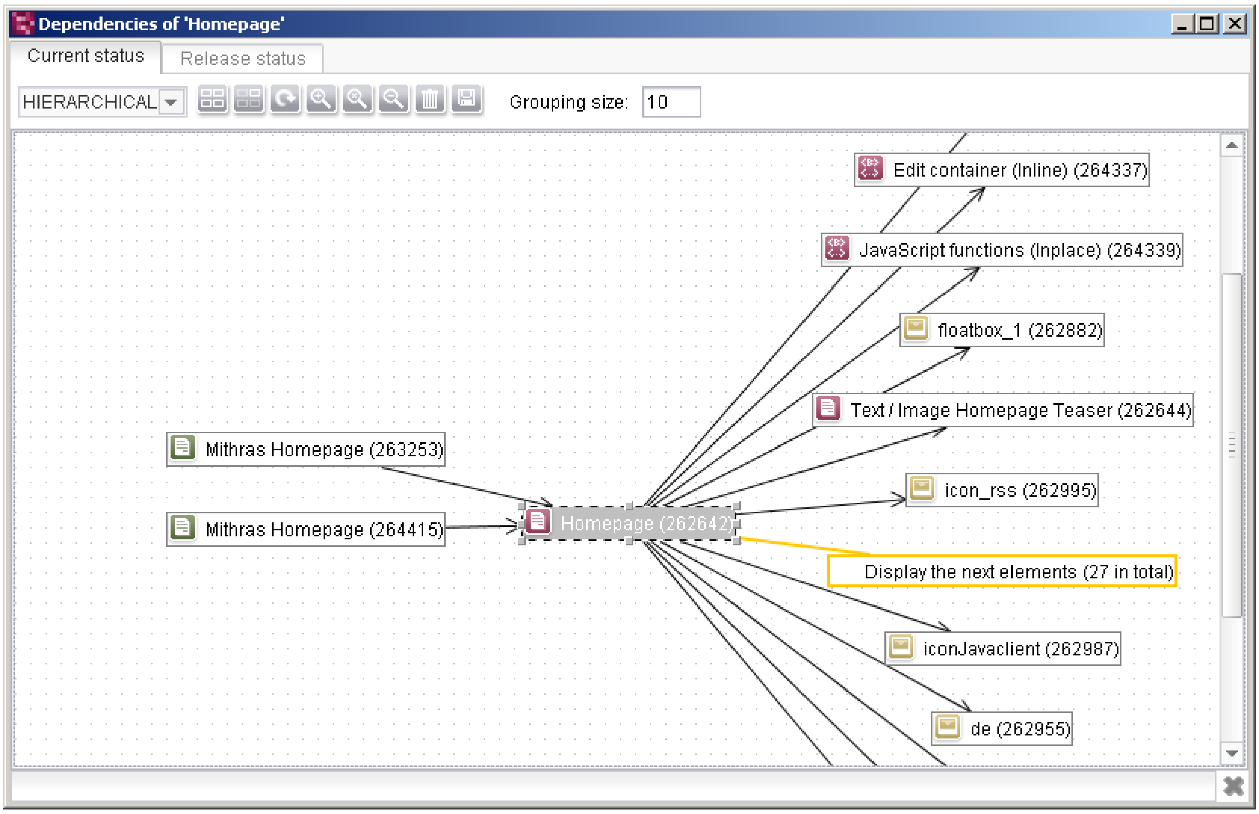

Dependencies of 'Homepage'
Current status
Release status
HIERARCHICAL
Grouping size:
10
Edit container (Inline) (264337)
JavaScript functions (Inplace) (264339)
floatbox_1 (262882)
Text / Image Homepage Teaser (262644)
Mithras Homepage (263253)
icon_rss (262995)
Mithras Homepage (264415)
Homepage (262642)
Display the next elements (27 in total)
iconJavaclient (262987)
de (262955)

Figure 2-42: Displaying dependencies via the reference graph

From FirstSpirit Version 4.2 format templates used within the input components CMS_INPUT_DOM and CMS_INPUT_ DOMTABLE can now also be displayed using the reference graph. This enhancement requires a change to the FirstSpirit Access API:

For the return type `DomEditorValue`, a new API interface `DomElement` has been introduced. Scripts which use the previously unstable "`DomElement`" "class" must be adjusted to the new interface:

Example: Generating content in the DOM Editor (to date – unstable):

```
final DataValue dataValue = data.get("cs_payment_text");
final DomEditorValue editor = (DomEditorValue) dataValue.getEditor();
xmlBuf = new StringBuilder();
xmlBuf.append("<p>myDom</p>");
editor.set(language, DomElement.fromXml(xmlBuf.toString()));
```

Example: Generating content in the DOM Editor (from FirstSpirit Version 4.2):

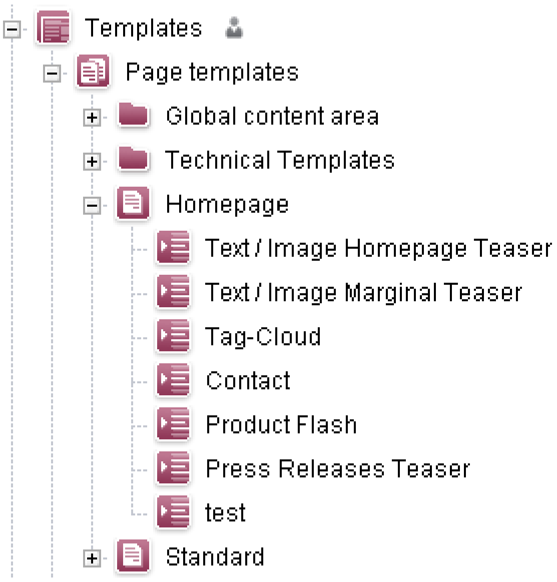
```
final DataValue dataValue = data.get("cs_payment_text");
final DomEditorValue editor = (DomEditorValue) dataValue.getEditor();
xmlBuf = new StringBuilder();
xmlBuf.append("<p>myDom</p>");
final DomElement domElement = editor.get(language);
domElement.set(xmlBuf.toString());
editor.set(language, domElement);
```

! *The changes are downwards compatible. After adjusting in Version 4.2, the scripts also function in FirstSpirit Version 4.0 and 4.1.*

2.4.15 Extras – Create a copy of this workflow

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

2.5 Page templates



- Templates
  - Page templates
    - Global content area
    - Technical Templates
    - Homepage
      - Text / Image Homepage Teaser
      - Text / Image Marginal Teaser
      - Tag-Cloud
      - Contact
      - Product Flash
      - Press Releases Teaser
      - test
    - Standard

Figure 2-43: Tree view Template Store – Page templates

Page templates create the basic framework of a page. Page templates are used to specify where, e.g. logos and navigations are positioned, whether a page is to consist of frames or not and similar general settings. In addition, the page templates define the places at which an editor can insert content.

Tree elements within the page templates:

Root element of the page templates

Folders within the page templates node

Page templates

Mapping of a section template to a page template

2.5.1 Preview tab

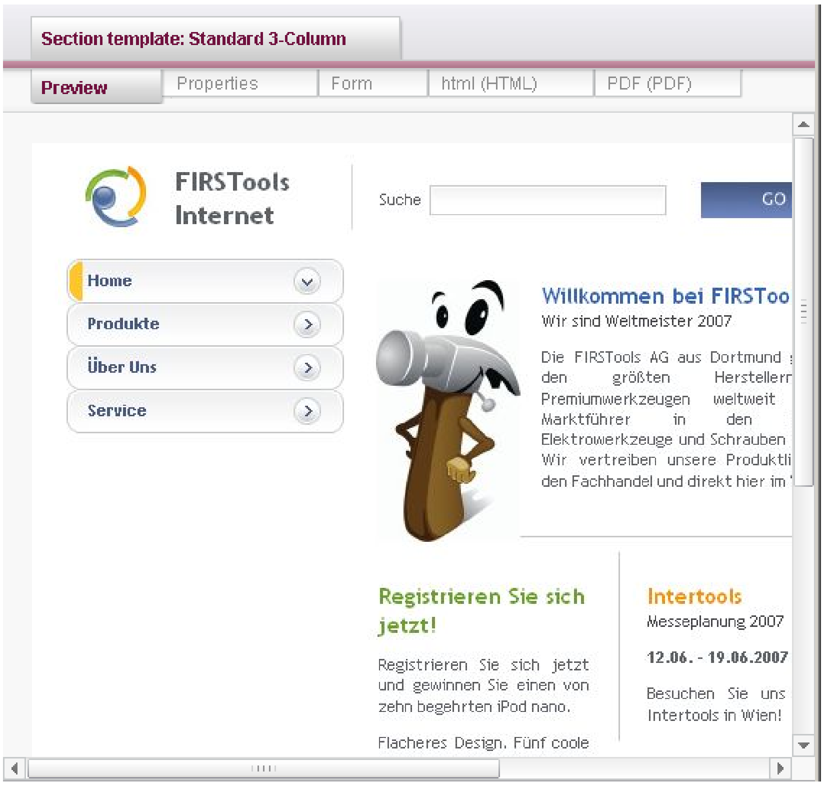
A previously made preview graphic (e.g. a screenshot) can be displayed in the "Preview" tab to obtain an idea of how a template will be subsequently displayed in the browser. In this way, each user can immediately recognise which template they have just selected.

A graphic can be inserted in this tab in three different ways:

1. Lock the template, then click "Select preview image” from the Template context menu and select a file of the type: "gif", "jpg" or "png".

2. Lock the template, then select a file of the type "gif", "jpg" or "png" from the file explorer, drag it to the preview position with the mouse and drop it there.

3. Lock the template, select a link-free graphic (Ctrl key pressed) from a website (MS IE only), drag to the Preview tab with the mouse and drop it there.



Willkommen bei FIRSToo
Wir sind Weltmeister 2007

Die FIRSTools AG aus Dortmund
den größten Herstellern
Premiumwerkzeugen weltweit
Marktführer in den
Elektrowerkzeuge und Schrauben
Wir vertreiben unsere Produktli
den Fachhandel und direkt hier im

Registrieren Sie sich jetzt!

Registrieren Sie sich jetzt und gewinnen Sie einen von zehn begehrten iPod nano.

Flacheres Design. Fünf coole

Intertools
Messeplanung 2007

12.06. - 19.06.2007

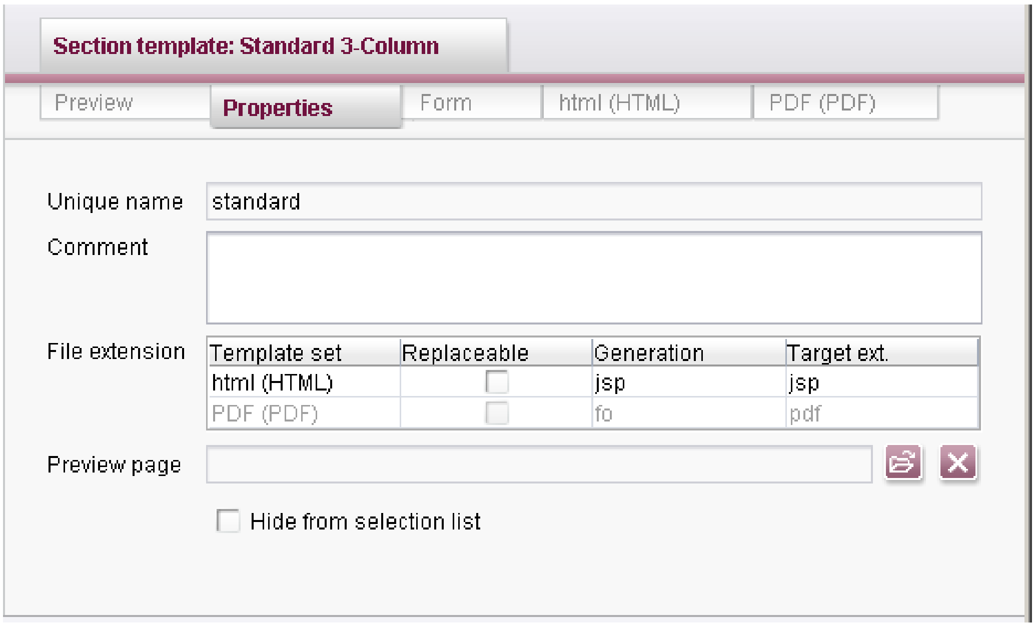
Besuchen Sie uns Intertools in Wien!

Figure 2-44: Page template – "Preview" tab

The inserted preview image of a page template is also displayed in the Page Store, if the content area of the corresponding page is enabled.

2.5.2 Properties tab

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



Unique name: standard

Comment:

File extension:

Template set	Replaceable	Generation	Target ext.
html (HTML)	□	jsp	jsp
PDF (PDF)	□	fo	pdf

Preview page:

□ Hide from selection list

Figure 2-45: Page template – "Properties" tab

Unique name: A unique name is given in this field, under which the template is stored in the file directory (see "Reference name" in Chapter 2.2.1.1 page 25).

Comment: A comment describing the page or section template in greater detail can be entered here.

File extension – Template set: Name and type of template sets which the project administrator defined for the current project in the server and project configuration. Disabled template sets are shown "greyed out" and cannot be edited.

File extension – Replaceable: If the tick is set, this means that the extensions of a page template given in one of the next two input fields can be overwritten by a section template.

File extension – Generation: The extension of the template to be used when generating the page. Double-click the field to edit the extension.

This option is no more available from FirstSpirit Version 4.1.

File extension – Target ext.: The extension of the template to be linked to. Double-click the field to edit the extension.

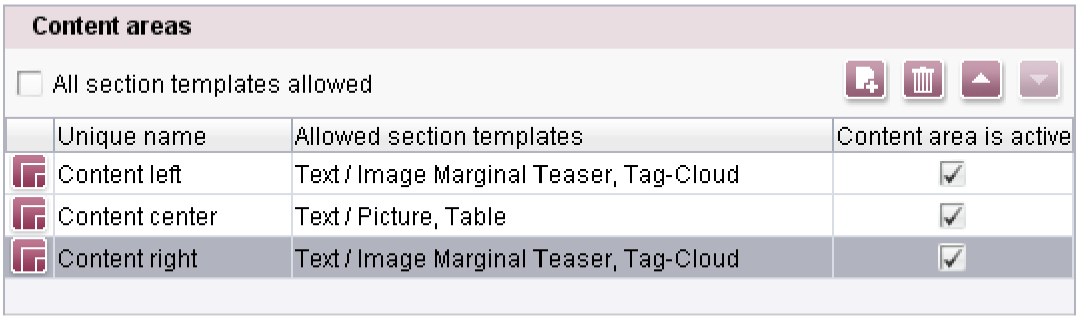
Preview page: A page from the Site Store in which the template is used can be selected here. In this way, any changes made to the template can be directly checked in the Template Store using the preview function.

Hide from selection list: Activating this option prevents an editor from using this template when creating a new page.

New functions from FirstSpirit Version 4.2 are listed below:

Formular: From FirstSpirit Version 4.2 the button „Default values“ is also available on this tab. It can be used to define default values for this template. A dialogue for pre-define the default values. The language-dependent default values are displayed directly in the preview area after saving the properties.

Content areas / section restrictions: Content areas are defined on the „Properties“ tab from FirstSpirit Version 4.2:



Content areas

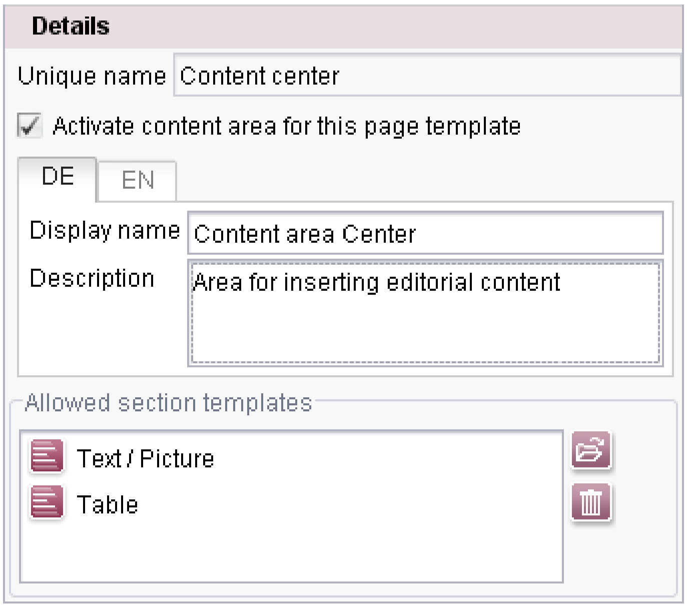
□ All section templates allowed

Unique name	Allowed section templates	Content area is active
Content left	Text / Image Marginal Teaser, Tag-Cloud	☑
Content center	Text / Picture, Table	☑
Content right	Text / Image Marginal Teaser, Tag-Cloud	☑

Figure 2-46: Defining content areas for a page template

Use the icons to add a new content area to the page template, to remove an existing content area or to resort the list of content areas.

Clicking on a content area you can define **section restrictions** for this page template:



Details

Unique name: Content center

☑ Activate content area for this page template

DE | EN

Display name: Content area Center

Description: Area for inserting editorial content

Allowed section templates

- Text / Picture
- Table

Figure 2-47: Defining section restrictions

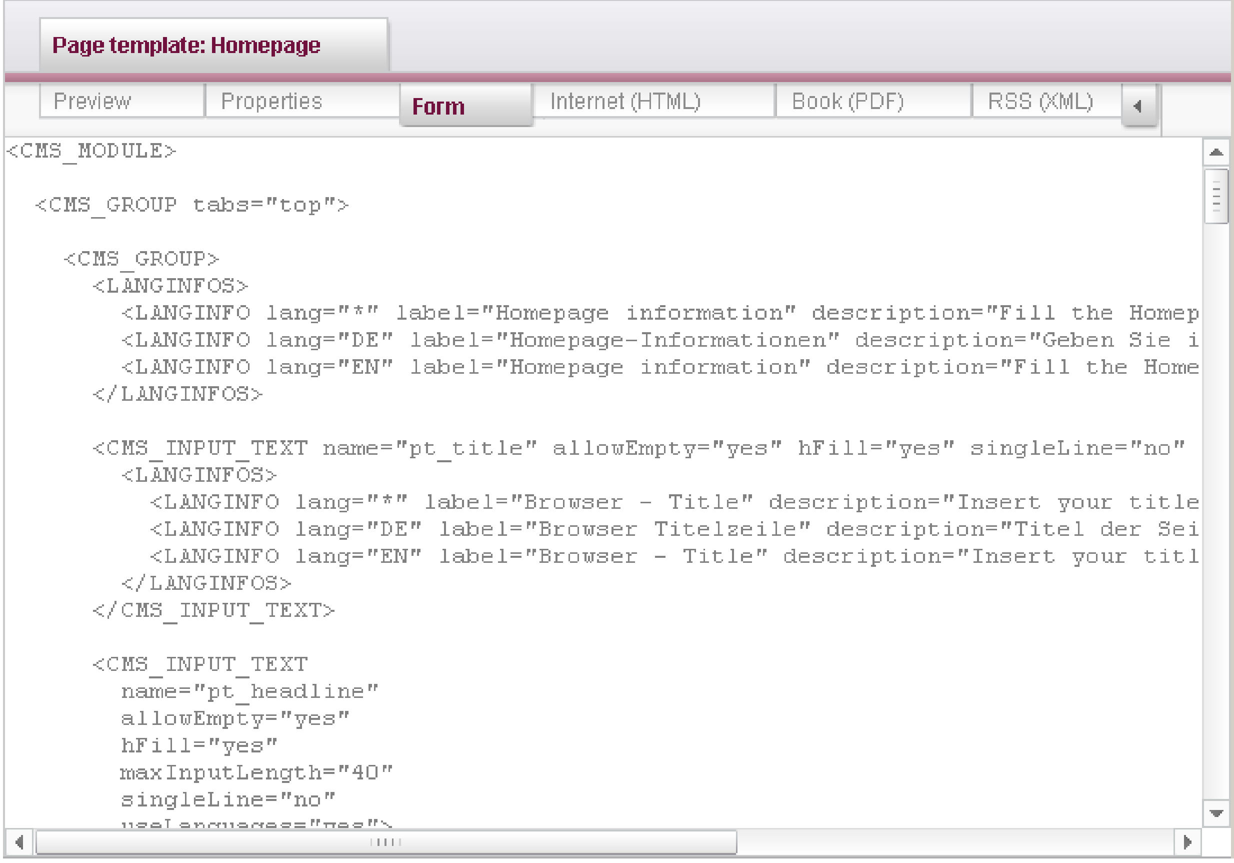
To this end, the required section templates can either be allowed or prohibited by adding them to or removing them from a list (for a content area). For the corresponding content areas, this means that only the respective selected section templates are allowed. The definition of section restrictions used to date by using Drag & Drop to drop section templates onto a content area (see Chapter 2.5.5 page 87), is therefore no longer supported.

Optionally, all section templates can also be allowed for all content areas of a page template. In this way, each section restriction for the page template is cancelled.

The addition of language-dependent display names for content areas is also new. Content areas can now be assigned one (or several) language-dependent display names and a unique reference name..

> **!** *Note about automatic adjustment of the project’s templates: FirstSpirit automatically adjusts existing content areas in the header area of a page template and the defined section restrictions of a page template to the new definition in the "Properties" tab. For further notes see FirstSpirit Release Notes Version 4.2, Chapter 8.1.5.*

2.5.3 Form tab



```
<CMS_MODULE>

  <CMS_GROUP tabs="top">

    <CMS_GROUP>
      <LANGINFOS>
        <LANGINFO lang="*" label="Homepage information" description="Fill the Homep
        <LANGINFO lang="DE" label="Homepage-Informationen" description="Geben Sie i
        <LANGINFO lang="EN" label="Homepage information" description="Fill the Home
      </LANGINFOS>

      <CMS_INPUT_TEXT name="pt_title" allowEmpty="yes" hFill="yes" singleLine="no"
        <LANGINFOS>
          <LANGINFO lang="*" label="Browser - Title" description="Insert your title
          <LANGINFO lang="DE" label="Browser Titelzeile" description="Titel der Sei
          <LANGINFO lang="EN" label="Browser - Title" description="Insert your titl
        </LANGINFOS>
      </CMS_INPUT_TEXT>

      <CMS_INPUT_TEXT
        name="pt_headline"
        allowEmpty="yes"
        hFill="yes"
        maxInputLength="40"
        singleLine="no"
        useLanguages="yes">
```

Figure 2-48: Page template – "Form" tab

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

When the GUI.XML is saved a DTD validation is performed. Well-formedness violations are fatal, in this case the GUI.XML cannot be saved. Other errors are only displayed.

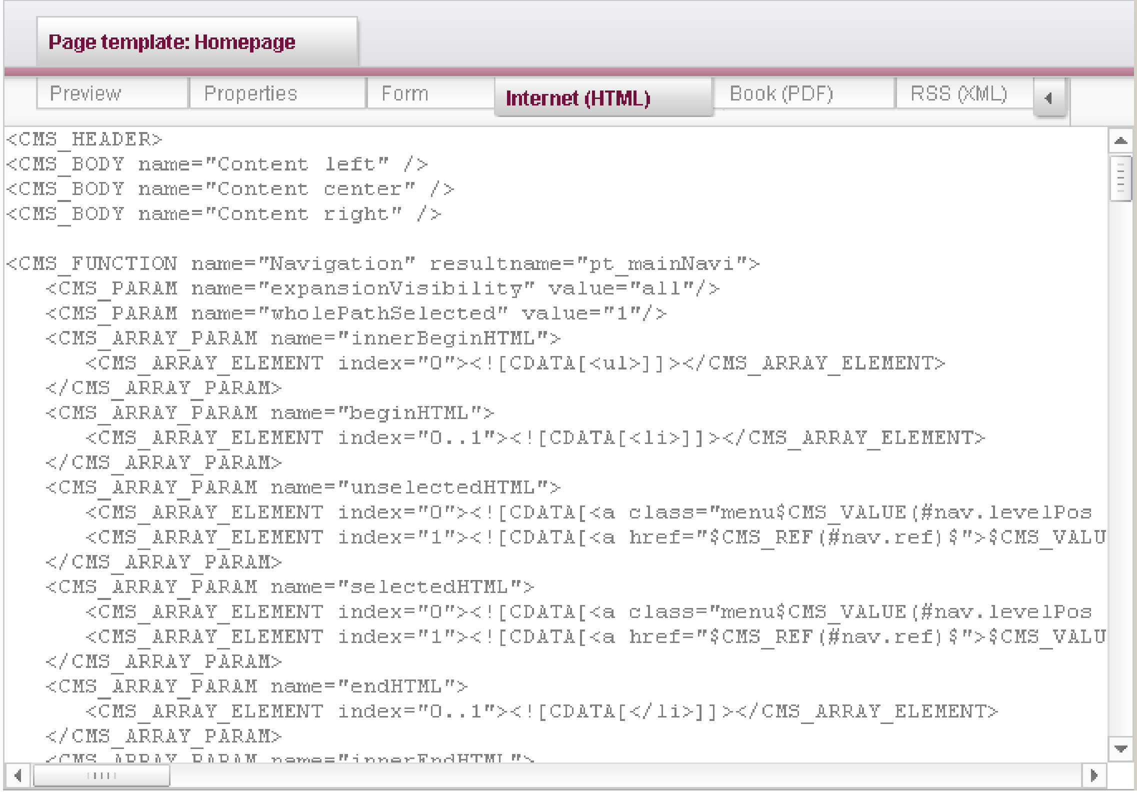
If a preview of the GUI.XML is requested in locked state, then the changes are automatically saved beforehand. (A new version is not created – that only takes place when unlocked!)

For details of updating content areas in existing pages (on changing the definition within the page template) see Chapter 2.4.10 page 76.

The "FirstSpirit Online Documentation" contains a list of all available input elements.

The input elements are explained with all their attributes and a schematic example under the menu item ***Template development – Forms****.*

2.5.4 Template sets tabs



```
<CMS_HEADER>
<CMS_BODY name="Content left" />
<CMS_BODY name="Content center" />
<CMS_BODY name="Content right" />

<CMS_FUNCTION name="Navigation" resultname="pt_mainNavi">
   <CMS_PARAM name="expansionVisibility" value="all"/>
   <CMS_PARAM name="wholePathSelected" value="1"/>
   <CMS_ARRAY_PARAM name="innerBeginHTML">
      <CMS_ARRAY_ELEMENT index="0"><![CDATA[<ul>]]></CMS_ARRAY_ELEMENT>
   </CMS_ARRAY_PARAM>
   <CMS_ARRAY_PARAM name="beginHTML">
      <CMS_ARRAY_ELEMENT index="0..1"><![CDATA[<li>]]></CMS_ARRAY_ELEMENT>
   </CMS_ARRAY_PARAM>
   <CMS_ARRAY_PARAM name="unselectedHTML">
      <CMS_ARRAY_ELEMENT index="0"><![CDATA[<a class="menu$CMS_VALUE(#nav.levelPos
      <CMS_ARRAY_ELEMENT index="1"><![CDATA[<a href="$CMS_REF(#nav.ref)$">$CMS_VALU
   </CMS_ARRAY_PARAM>
   <CMS_ARRAY_PARAM name="selectedHTML">
      <CMS_ARRAY_ELEMENT index="0"><![CDATA[<a class="menu$CMS_VALUE(#nav.levelPos
      <CMS_ARRAY_ELEMENT index="1"><![CDATA[<a href="$CMS_REF(#nav.ref)$">$CMS_VALU
   </CMS_ARRAY_PARAM>
   <CMS_ARRAY_PARAM name="endHTML">
      <CMS_ARRAY_ELEMENT index="0..1"><![CDATA[</li>]]></CMS_ARRAY_ELEMENT>
   </CMS_ARRAY_PARAM>
   <CMS_ARRAY_PARAM name="innerEndHTML">
```

Figure 2-49: Page template – Template sets tabs

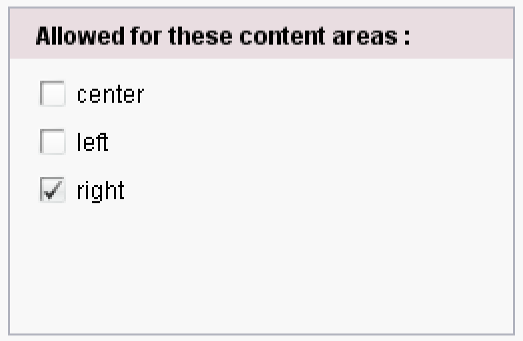
The "Internet", "Book" and "RSS" tabs are template sets which the project administrator has created in the server and project configuration for this project. The tabs show the source text of the different template sets for the current template. If the template is locked, direct changes can be made here.

If a change has been made in the source text, the change is checked for well-formedness of the CMS_HEADER on saving. If an error occurs, this is immediately displayed in a new window.

2.5.5 Restricting the content areas for page templates (up to and including V4.1)

By dragging section templates () onto page templates () it is possible that the respective selected section templates only are then allowed for the content areas of the corresponding page template.

If such a section restriction () is selected, checkboxes appear in the right-hand window in which the content areas of the page template are listed.



Allowed for these content areas :

□ center

□ left

☑ right

Figure 2-50: Allowed areas

If the corresponding page template is locked, the required content areas can be enabled or disabled. One or several content areas can be selected for each section restriction. An editor who inserts a new section in a content area is then only shown and can only choose from the section templates allowed for this section area. All other section templates are hidden.

If there are no restrictions for a content area, all section templates are automatically allowed and are available for an editor to select.

For details of how to delete section restrictions, see Chapter 2.2.8.1.

From FirstSpirit Version 4.2 content areas are defined via the “Properties“ tab.

2.6 Section templates

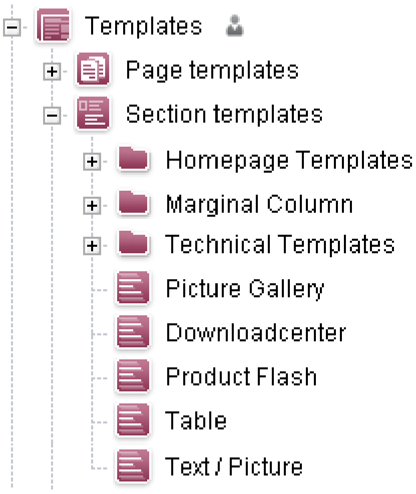


Figure 2-51: Template Store tree view – Section templates

Section templates are used to insert content, defined in the page templates, in the basic framework of a page. All input elements which are to hold the dynamic content of the page (text, tables, pictures, data records,...) are defined within a section template. Any number of sections can be inserted in each section area of a page. In most cases, several different section templates are also available for the different possible content of a page.

- **Tree elements within the section templates:**

Root element of the section templates

Folders within the section templates node

Section templates

2.6.1 Preview tab

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

For information on the "Preview" tab, see Chapter 2.5.1 page 81.

2.6.2 Properties tab

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

For information on the "Properties" tab, see Chapter 2.5.2 page 83.

The "Hide from selection list" option (cf. Chapter 2.5.2) is not available for section templates.

2.6.3 Form tab

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

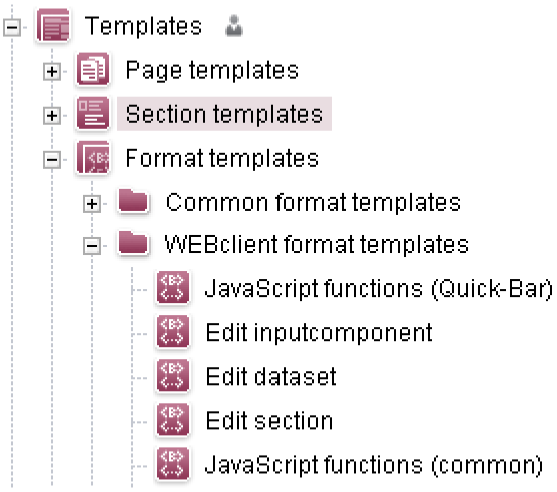
For information on the "Form" tab, see Chapter 2.5.3 page 86.

2.6.4 Template sets tabs

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

For information on the "Template sets" tab, see Chapter 2.5.4 page 87.

2.7 Format templates



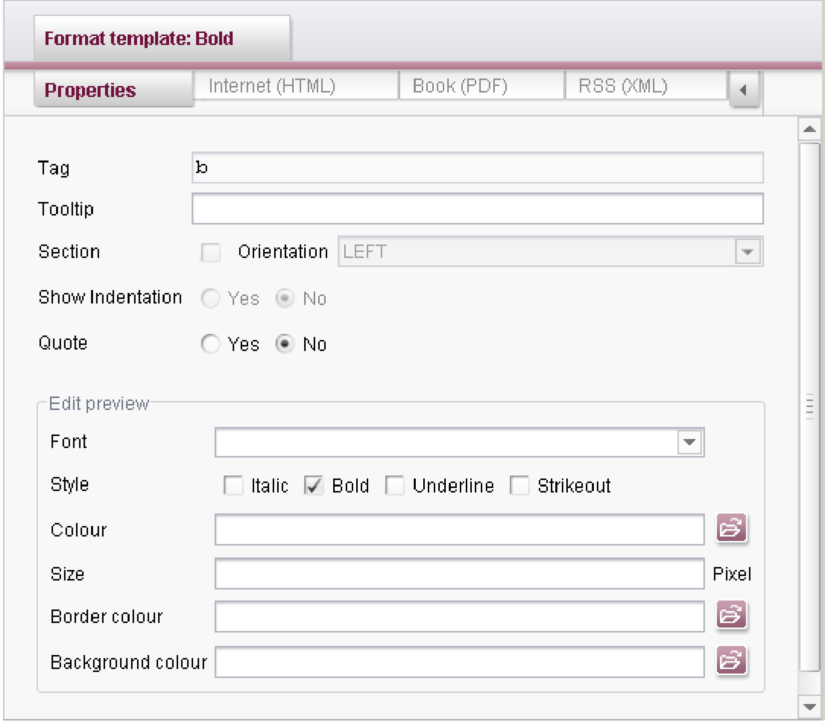
Templates
Page templates
Section templates
Format templates
Common format templates
WEBclient format templates
JavaScript functions (Quick-Bar)
Edit inputcomponent
Edit dataset
Edit section
JavaScript functions (common)

Figure 2-52: Template Store tree view – Format templates

Format templates are used to define text formatting, which can subsequently be used in the DOM-Editor and DOM Table input elements in the Page Store. The standard (default) format templates are: Bold, Italic, LineBreak, Link, Standard and Underline. These "Default format templates" may not be deleted (see Chapter 2.2.8 page 44).

In addition to these default format templates, template developers can create other project-specific format templates.

2.7.1 Properties tab



Format template: Bold

Properties | Internet (HTML) | Book (PDF) | RSS (XML)

Tag: b

Tooltip:

Section □ Orientation LEFT

Show Indentation ○ Yes ● No

Quote ○ Yes ● No

Edit preview

Font

Style □ Italic ☑ Bold □ Underline □ Strikeout

Colour

Size Pixel

Border colour

Background colour

Figure 2-53: Format template – Properties tab (new Look&Feel)

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

Tag: The tag (abbreviation for the format template's name) is needed in the form area of the page or section template to specify the valid format templates for the input component. The corresponding XML tag name is formed from this abbreviation, e.g. for the "Bold" format template (see also FirstSpirit Online Documentation – Template development / Format templates area). The name must be unique and may not contain any special characters; it is automatically issued according to the unique reference name of the format template. The abbreviation should not be changed manually to ensure its uniqueness is not put at risk (see Chapter 2.2.1.3 page 27).

! *A unique name or the abbreviation of a format template cannot be changed after the template has been created, otherwise all relations within the project would be lost!*

Tooltip The text entered in this field is displayed as Help text when the mouse is moved over the formatting, for example in the DOM Editor.

Section: If the "Section" checkbox is selected, the whole section is always formatted. If the checkbox is not selected, the formatting is applied to individual, selected characters only.

Orientation: If the "Section" checkbox has been selected, the orientation (alignment) of the text, for example in the DOM Editor, can be specified here.

Show Indentation, defines how the corresponding formatted text is to be displayed. If this option is selected, all spaces are displayed and the text is no longer automatically wrapped. If this option is deselected, spaces are displayed in "HTML Notation".

Quote: Select **Yes** to apply the complete conversion rules to the individual template sets (convert part and quote part). If **No** is selected, the convert part only of the conversion rules is applied to the individual template sets.

The **"Edit preview"** field can be used to define other formats only displayed in the editor.

Font: A font in which the text is to be shown can be selected here. (This font must be installed on each client computer, otherwise a similar font is used.)

Style: Here you can select whether the text is to be displayed in the DOM Editor as bold, italic or underlined text.

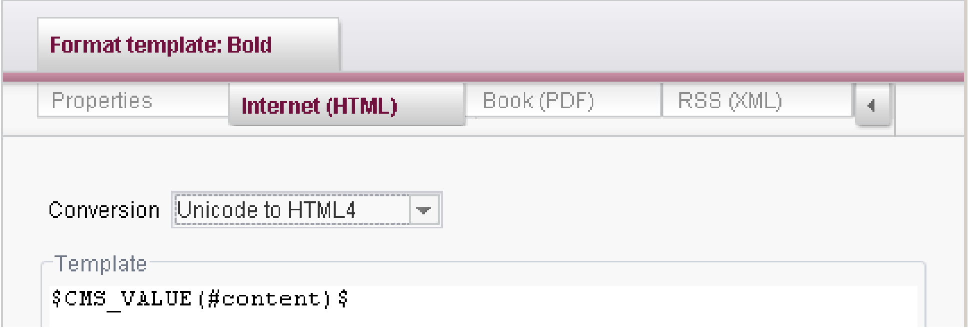
Colour: A colour for the displayed text can be selected here.

Size: The size in which a text is to be displayed in the DOM Editor is determined here. Relative information (+2, -1, etc.) is also possible.

Border colour: A colour selection for a border within the input component can be made here.

Background colour: A colour selection for the background within the input component can be made here.

2.7.2 Template Sets tabs



Conversion: Unicode to HTML4

Template

```
$CMS_VALUE(#content)$
```

Figure 2-54: Format template – Template Sets tabs

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

Template: The HTML code which the required formatting generates for the text on the website is entered here. The #content expression stands for the text entered in the DOM Editor (for details of outputting formatted texts via the #content system object, see "FirstSpirit Online Documentation").

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

2.8 Style templates (from V4.1)

This function is released for FirstSpirit Version 4.1 and higher only. Screenshots are therefore displayed in the new "LightGray" Look & Feel. The display in the "Classic" Look & Feel can differ slightly.

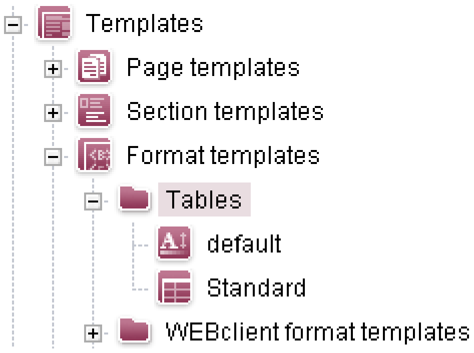


Figure 2-55: Template Store tree view – Style templates

2.8.1 Introduction: Inline tables (from V4.1)

So-called "inline tables" can be integrated in the text flow by extending the DOM Editor input component (CMS_INPUT_DOM input component). Any number of layout design options can be made available to the editor, down to cell level.

The table layout is determined on the one hand by table format templates (see Chapter 2.9 page 106), and on the other hand by style templates (from Chapter 2.8.2 page 96). Style templates are used to define table layout features, i.e. background colour, text orientation (alignment), font, line break control, borders and border margins.

Each table format template can have exactly one standard style template mapped to it (for the whole table) and several other style templates for separate display of individual cells of the table (see Chapter 2.9.1 page 108). The style templates define the layout of the individual table cells, e.g. the background colour ("bgcolor"), the alignment of text in the cell ("align") or the colour of the text within the cell ("color").

Therefore, a style template must be created first before the inline tables can be used in the DOM Editor (see Chapter 2.8.2 page 96).

For an improved clarity, style and table format templates should be grouped together in one folder (e.g. "Tables").

Inline table icons:

Folder

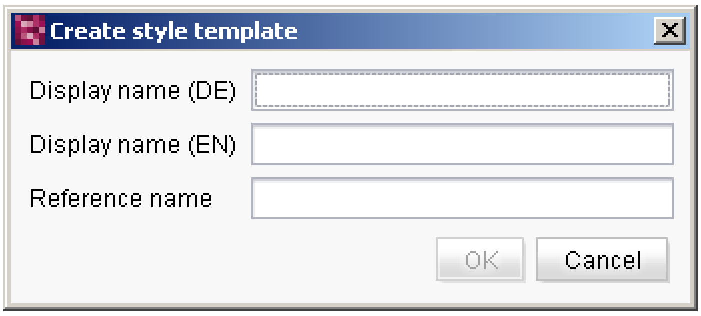
Table format template

Style template

> *From FirstSpirit Version 4.2 the inline tables functionality is available in WebEdit too. However, the cell propoerties can not be edited. This means, the related button is always disabled in WebEdit.*

2.8.2 Create style template (from V4.1)

Style templates are created in the "Format templates" area. To do this, the **New – Create style template** function is selected in the context menu. A reference name for the style template must be entered in the window that opens. Entry of a display name is optional.



Create style template

Display name (DE)

Display name (EN)

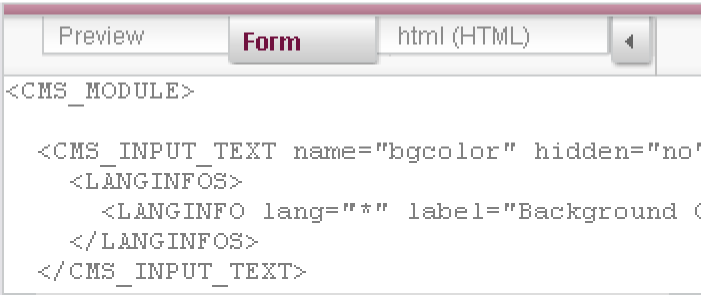
Reference name

Figure 2-56: New – Create style template

Click OK to create the new style template. The form area of a style template can be used to create input components which affect the properties of the layout, e.g. background colour, text alignment, font, line break control, border and border margins (see Chapter 2.8.3 page 96).

2.8.3 Form area of a style template (from V4.1)

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



```
<CMS_MODULE>

  <CMS_INPUT_TEXT name="bgcolor" hidden="no'
    <LANGINFOS>
      <LANGINFO lang="*" label="Background (
    </LANGINFOS>
  </CMS_INPUT_TEXT>
```

Figure 2-57: Form area of a style template

Several predefined layout attributes (with reserved identifiers) have a direct effect on the display of the table within the DOM Editor:

- `bgcolor`: defines the background colour of a table cell
  (for an example, see Chapter 2.8.7.1 page 102)
- `color`: defines the font colour of a text within the table cell
  (for an example, see Chapter 2.8.7.2 page 103)
- `align`: defines the alignment of a text within the table cell
  (for an example, see Chapter 2.8.7.3 page 104)

! *The predefined identifiers may not be changed. The attributes in the input component must always be given with name="Identifier", e.g. <CMS_INPUT_TEXT name="bgcolor" .../>*

Of course, apart from these standard attributes, other freely defined attributes can be entered using the input components of the form area, e.g. CSS attributes (for an example, see Chapter 2.8.7.4 page 105).

Supported input components for maintaining the layout attributes:

- `CMS_INPUT_TEXT` / `CMS_INPUT_TEXTAREA`: Text field for entering a value, e.g. for the background colour.
  (For an example, see Chapter 2.8.7.1 page 102)
- `CMS_INPUT_COMBOBOX:` Selection from a pre-defined set of values, e.g. for entering a background colour or an alignment
  (For an example, see Chapter 2.8.7.2 page 103)
- `CMS_INPUT_RADIOBUTTON:` Selection from a pre-defined set of values, e.g. for entering a background colour or an alignment
  (For an example, see Chapter 2.8.7.3 page 104)

- `CMS_INPUT_TOGGLE:` Selection between two possible values (e.g. On/Off, right/left)
  (For an example, see Chapter 2.8.7.4 page 105)
- `CMS_INPUT_NUMBER:` Indication of a numerical value (e.g. a value for the background colour of a cell)

> ! *The following applies to all input components used within the form area of a style template: the components should be defined independent of language (`useLanguages="no"`). In this case, the language-dependency of the component is covered by the language selection within the DOM Editor instance, which is edited by the editor.*

> ! *Other input components for maintaining layout attributes (in style templates) are not currently supported.*

2.8.3.1 Prevent layout editing for editors (from V4.1)

Maintenance of layout attributes can be prevented for editors. To do this, the following attribute must be defined within the input component: `hidden="yes"`. The effect of the `hidden="yes"` attribute is that the input component is only visible within the Template Store, but not during maintenance of the table in the Page Store. The template developer can therefore use the attribute to prevent editing of the layout by the editor and can instead specify defined values for the layout, for example, for the background colour of the cells (see Chapter 2.8.4 page 99).

If the `hidden="yes"` attribute is defined (for all input components of the style template), it is not possible for the editor to change the layout properties of a table cell in the Page Store. In this case the corresponding "Cell properties" button is inactive.

If, on the other hand, individual components are "visible" (`hidden="no"`) and others are "hidden” (`hidden="yes"`), the "Cell properties" button is active within the DOM Editor (in the Page Store), but in the following dialog only the "visible" components are displayed to the editor.

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

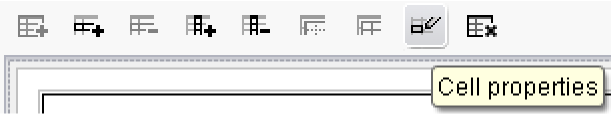


Figure 2-58: "Cell properties" button within the DOM Editor

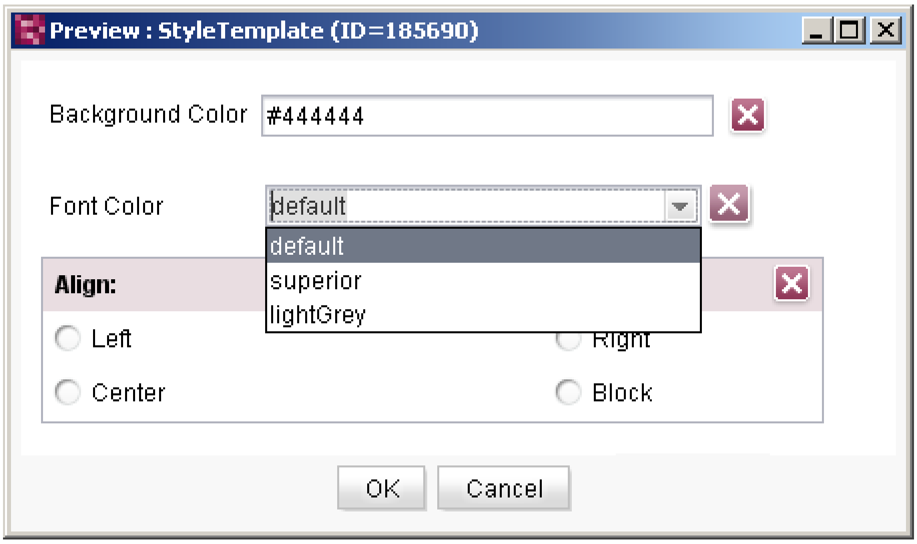
By clicking the button the editor opens a dialog for editing the project-specific layout attributes (see *FirstSpirit Manual for Editors*).

2.8.4 Pre-configuration of the layout attributes (from V4.1)

Retrieval values for the layout attributes can be defined within the Template Store.

Pre-configuration of an input component (e.g. with a colour value) no longer (since FirstSpirit Version 4.0) takes place by defining the input component but instead is made within a separate dialog within the Template Store.

After definition of the input components in the form area of the style template has been completed and saved, the button with the magnifying symbol (preview) can be used in the "Form" tab to open the maintenance dialog for specifying the pre-configuration:



Preview : StyleTemplate (ID=185690)

Background Color: #444444

Font Color: default

Figure 2-59: Style template preview

The template developer can pre-configure the input component within the "Retrieval Values" dialog. For example, a colour value can be entered in the text field for "Background Color" (cf. Figure 2-59). This colour value is copied for all table cells based on the corresponding style template when an inline table is created in the DOM Editor.

Depending on the defined value for the `hidden` attribute within the definition of the input component, this default selection can be changed by the editor (see Chapter 2.8.3.1 page 98).

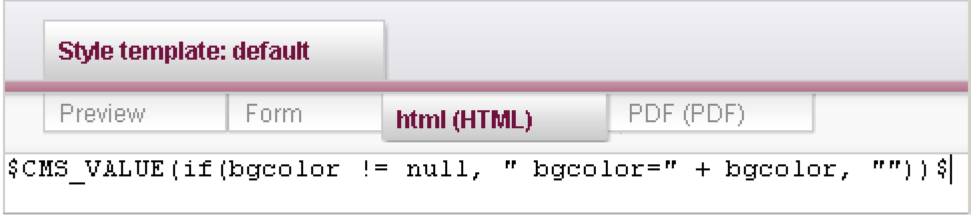
If editing is possible (`hidden="no"`), the editor can overwrite this default selection on editing a table cell within the DOM Editor (see *FirstSpirit Manual for Editors*).

> ! *The values in the dialog can only be edited and/or saved if the template is locked ("Switch to Edit Mode" button)!*

> ! *These default values **cannot** be changed in WebEdit.*

2.8.5 Output channel of a style template (from V4.1)

The values entered within the input components can be read out again within an output channel (e.g. "HTML") of a style template (see Chapter 2.8.4 page 99).



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

Figure 2-60: "HTML" output channel of a style template

To do this, the name of the input component must be output by means of the `$CMS_VALUE(...)$` instruction:

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

or

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

For further information on outputting variables, see FirstSpirit Online Documentation[3].

2.8.6 Linking with standard table format templates (from V4.1)

The `#style` system object can be used to link the values of the style template with the default format templates for the generation (and preview) of tables in the project. The default format templates for tables provided 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 the `#style` system object is used within the default format template `td`, the value defined within the "Cell properties" dialog by the editor (see *FirstSpirit Manual for Editors*) or the values predefined for the style template by the template developer (see Chapter 2.8.4 page 99), is taken into account in the generation of the table.

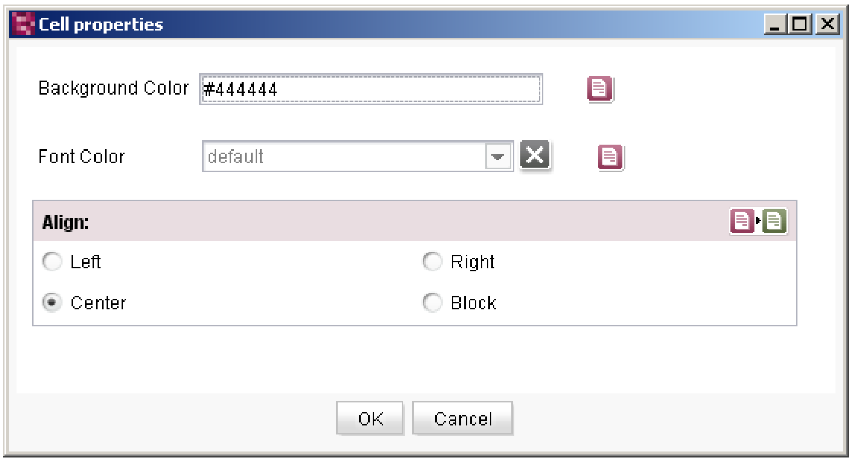
Example of the output in the HTML channel of the default format template `td`:

```
<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 further information about how to access properties and information of tables and their content see FirstSpirit Online Documentation, System objects #cell, #content, #table and #tr in the area Template development / Template syntax / System objects.*

The values of the layout attributes, which were defined by the editor (or the template developer) in the Cell properties dialog are now taken into account on generating the table (see Figure 2-61):

[3] FirstSpirit Online Documentation in the area: Template Development / Variables



Background Color #444444

Font Color default

Align:

○ Left
○ Right
● Center
○ Block

Figure 2-61: Properties of the table cell

The source text of the table cell is now generated as follows:

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

2.8.7 Examples (from V4.1)

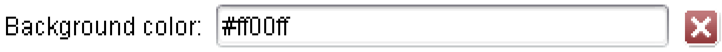
2.8.7.1 Example: Text input component for entering a background colour

Definition of the 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 colour. This name may not be changed because it is a fixed key value.

Entering a colour value via the input component:



Background color: #ff00ff

Figure 2-62: Input component for entering a background colour

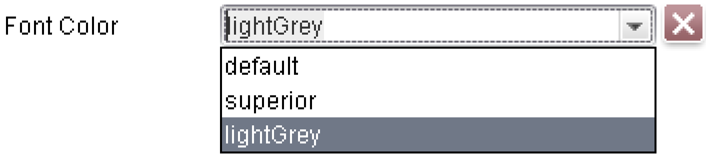
For details of how to output the value within the output channel of the style template, see Chapter 2.8.5 page 100.

2.8.7.2 Example: Input component for entering a font colour

Definition of the 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 colour value via the input component:



Font Color lightGrey

Figure 2-63: Input component for selecting a colour value for the font colour

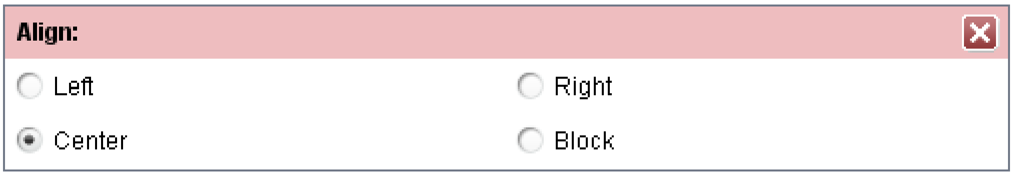
For details of how to output the value within the output channel of the style template, see Chapter 2.8.5 page 100.

2.8.7.3 Example: Input component for selecting a text alignment

Definition of the 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 via the input component:



Align:

○ Left ○ Right

● Center ○ Block

Figure 2-64: Input component for selecting a text alignment

For details of how to output the value within the output channel of the style template, see Chapter 2.8.5 page 100.

2.8.7.4 Example: Input component for entering a CSS attribute

Definition of the component in the form area:

```
<CMS_MODULE>
  <CMS_INPUT_TOGGLE name="ft_css" useLanguages="no">
    <LANGINFOS>
      <LANGINFO lang="*" label="Important:"/>
    </LANGINFOS>
    <On>
      <LANGINFO lang="*" label="Important"/>
    </On>
    <Off>
      <LANGINFO lang="*" label="Not important"/>
    </Off>
  </CMS_INPUT_TOGGLE>
</CMS_MODULE>
```

Selecting a CSS attribute via the input component:

Important: ☑

Figure 2-65: Input component for selecting a CSS attribute

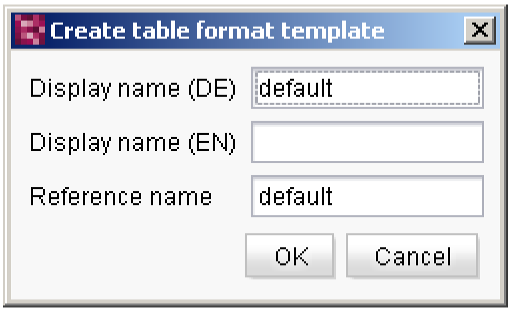
For details of how to output the value within the output channel of the style template, see Chapter 2.8.5 page 100.

2.9 Table format templates (from V4.1)

Table format templates are required to create so-called inline tables (see Chapter 2.8.1 page 95). A table format template must be created in the "Format templates" area for each required table layout.

To do this, the **New – Create table format template** function is selected in the context menu. A reference name for the table format template must be entered in the window that opens. Specification of a display name is optional.

A detailed description of reference and display names is given in Chapter 2.2.1.1 page 25.



Create table format template

Display name (DE) default

Display name (EN)

Reference name default

Figure 2-66: New – Create table format template

After clicking OK , the "Properties" tab opens. The size of the table can be defined here and the style templates created in Chapter 2.8.2 can be assigned.

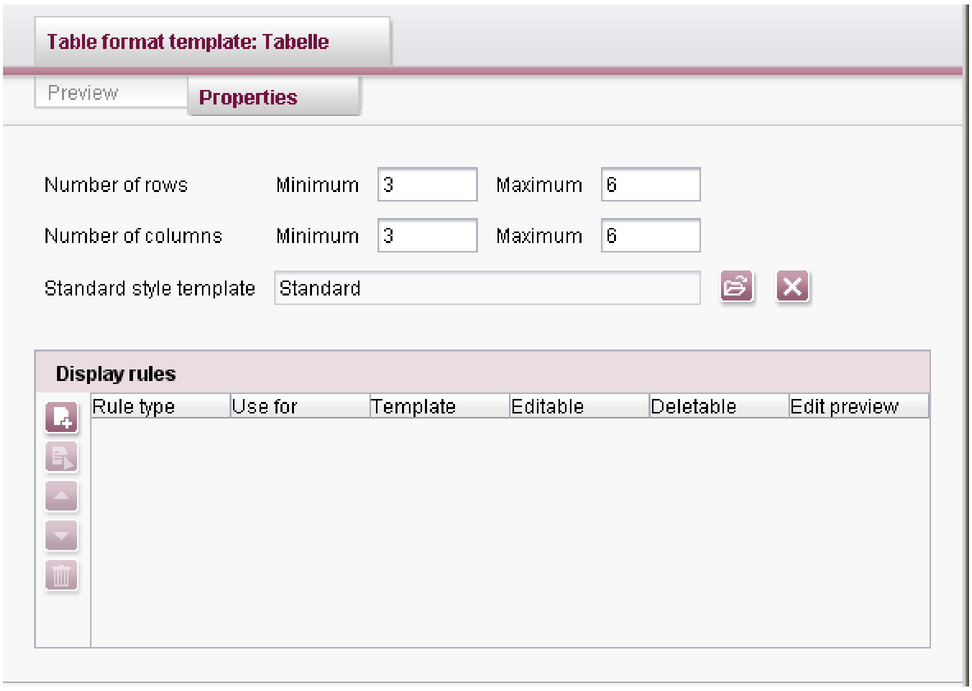


Table format template: Tabelle

Number of rows	Minimum	3	Maximum	6
Number of columns	Minimum	3	Maximum	6
Standard style template	Standard			

Display rules

Rule type	Use for	Template	Editable	Deletable	Edit preview

Figure 2-67: Table format template

Number of rows: The **Minimum** and **Maximum** fields are used to define the minimum number of rows the table must have or the maximum number it may have. If the editor subsequently inserts an inline table with this table format template into the DOM Editor, it is automatically created there with the minimum number of rows given here. The editor cannot exceed or undershoot these default values when editing the table. In this case the corresponding buttons are disabled. If, for example, the minimum number of rows is defined as two rows, the "Remove row"" button is disabled in the DOM Editor as soon as the table only contains two rows.

Number of columns: The **Minimum** and **Maximum** fields are used to define the minimum number of columns the table must have and the maximum number it may have. If the editor subsequently inserts an inline table with this table format template into the DOM Editor, it is automatically created there with the minimum number of columns given here. The editor cannot exceed or undershoot these default values when editing the table. In this case the corresponding buttons are disabled. For example, if the maximum number of columns is defined as being 6 columns, the "Add Column" button is disabled in the DOM Editor as soon as the table contains six columns.

Standard style template: In this field the required style template on which the whole table is to be based can be selected by clicking the 📂 icon. All the available style templates are displayed in the window that opens.

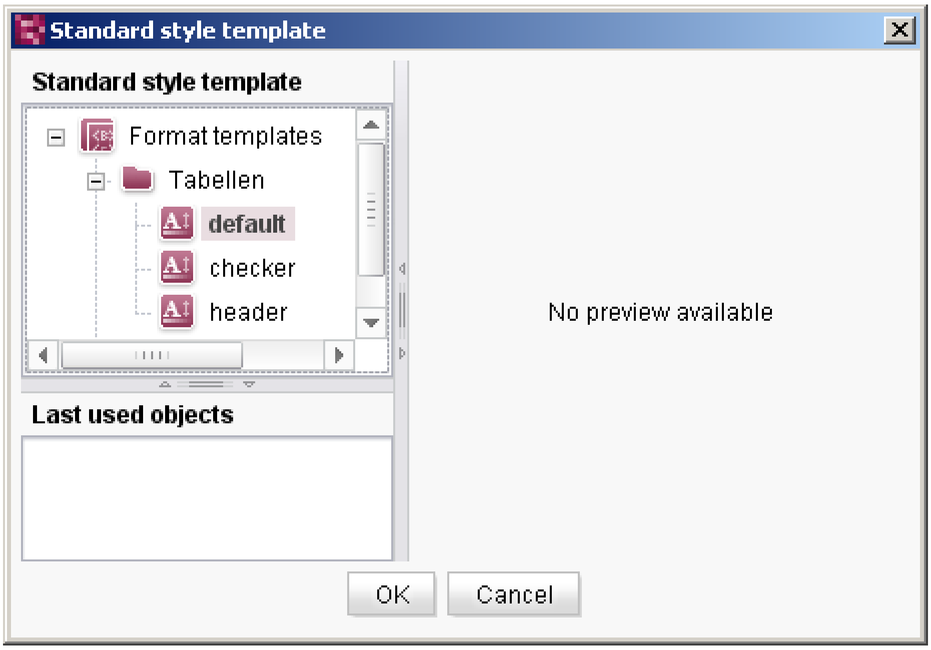


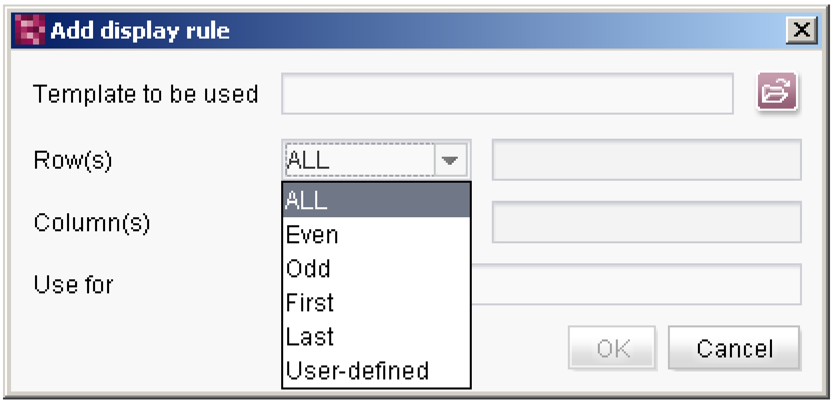
Figure 2-68: Table format template – Standard style template

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

2.9.1 Creating and editing display rules (from V4.1)

The standard style template defined in the table format template is the basis for the layout of a table (cf. Figure 2-68). In addition, other layout options for formatting rows, columns and individual cells can be made available to the editor in the **Display rules** area; these overwrite the layout specifications of the standard style template. These layout options are based on the previously created style templates (see Chapter 2.8.2 page 96).

The ➕ icon is used to create new display rules. After it is clicked the following window opens:



Add display rule

Template to be used

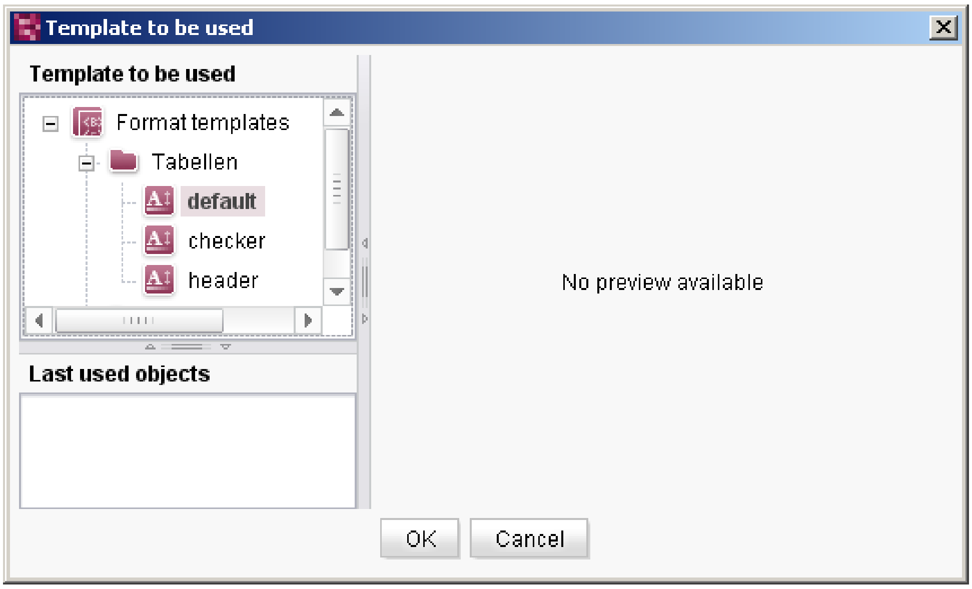
Row(s): ALL

Column(s)

Use for

Figure 2-69: Table format template – Display rule

Click the icon to select the required style template to be used for the display rule. All the available style templates are displayed in the window that opens.



Template to be used

Figure 2-70: Table format template – Style template

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

The **Row(s)** and **Columns(s)** comboboxes are used to define conditions for application of the rule and therefore for use of the selected style template. Both conditions must be fulfilled for the rule to be applied.

ALL: The display rule applies to all rows and columns without restrictions. For example, if the "ALL Columns" option is selected, the display rule only takes into account the restriction defined under the "Rows" option and vice versa.

It is not possible to define a display rule which concerns ALL columns and ALL rows. Such a rule would correspond to the standard 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).

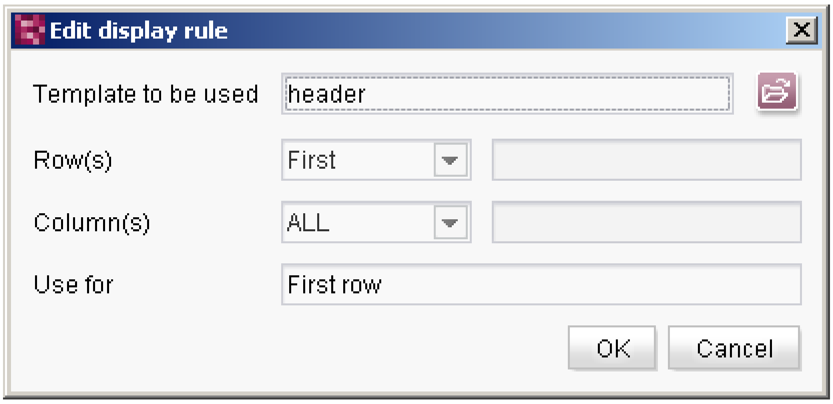
First: The display rule applies to the first row or column.

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

User-defined: The display rule applies to a specific row or column. If this option is selected, the number of the required row/column must be entered in the field to the right of the combobox.

The "Use for" field shows the row(s) and columns(s) to which the selected style template applies.

Examples:



Template to be used: header

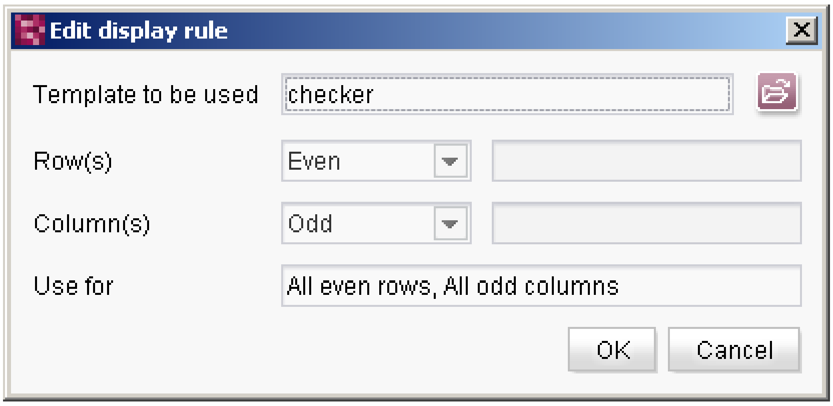
Row(s): First

Column(s): ALL

Use for: First row

Figure 2-71: Display rules – Example 1

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

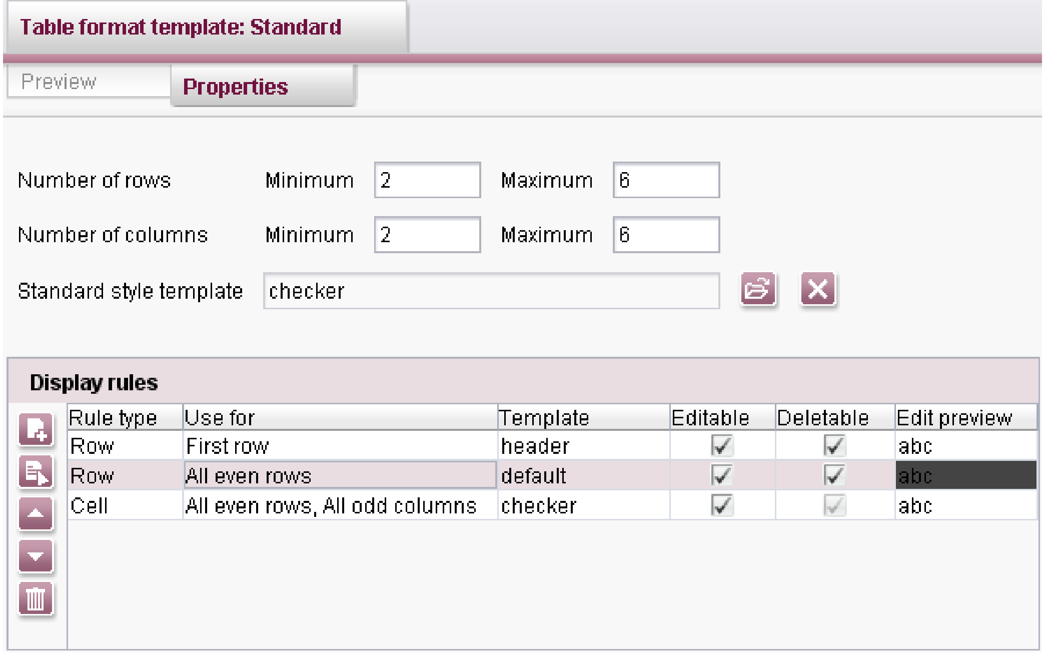


Template to be used	checker	
Row(s)	Even	
Column(s)	Odd	
Use for	All even rows, All odd columns	

Figure 2-72: Display rules – Example 2

In this example, the "checker" style template is used for **Even rows** in **Odd columns**, i.e. in all cells for which both conditions are true.

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



Number of rows	Minimum	2	Maximum	6
Number of columns	Minimum	2	Maximum	6
Standard style template	checker			

Display rules

Rule type	Use for	Template	Editable	Deletable	Edit preview
Row	First row	header	✓	✓	abc
Row	All even rows	default	✓	✓	abc
Cell	All even rows, All odd columns	checker	✓	✓	abc

Figure 2-73: List of the display rules

Rule type: Indicates whether the rule is valid for rows, columns or cells.

Use for: Indicates to which row(s) and/or column(s) the rule is applied.

Template: Indicates which style template is used.

Editable: This checkbox is selected as a default so that the editor can change the properties of the cell(s) to which this display rule applies. If the checkbox is deselected the editor cannot change the properties of the cell(s) concerned.

Deletable: This checkbox is selected as a default, if it is a rule for rows or columns. The editor can delete the row(s) or columns to which this display rule applies. If the checkbox is deselected the editor cannot change the properties of the row(s) or columns concerned. If the rule applies to a cell the checkbox is selected. It cannot be deselected, as individual cells cannot be deleted from tables.

Edit preview: This column shows the background colour and text alignment of the row/column/cell, if corresponding values are defined.

Edit: Click this icon (or double-click the display rule) to open an existing display rule for editing.

One position up: If several display rules exist, this icon can be used to move them up in the list by one position.

One position down: If several display rules exist, this icon can be used to move them down in the list by one position.

Delete: Click this icon to delete the selected display rule.

The width of the columns of this list can be changed as and when necessary by clicking the column lines and dragging the line with the mouse pressed.

If there are several rules in the list they are evaluated from the top down.

2.9.2 Evaluation order (from V4.1)

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

1. First, the **display rules** in the list (Figure 2-73) are evaluated from the top down.
2. The **Standard style template** is used on cells to which none of the display rules apply.

2.9.3 Insert inline tables in the DOM Editor (from V4.1)

To make inline tables available to the editor in the DOM Editor this type of input component must be inserted into the required section template. To do this, the `table="yes"` parameter must be added to the CMS_INPUT_DOM input component.

Example:

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

The input component can look like the following:

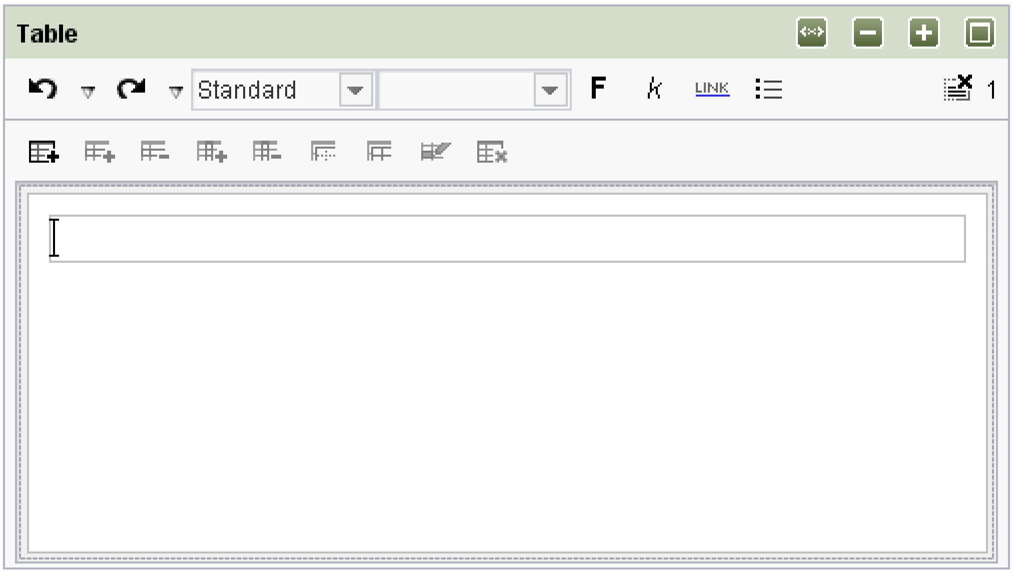


Figure 2-74: DOM Editor with inline table

2.10 Link templates

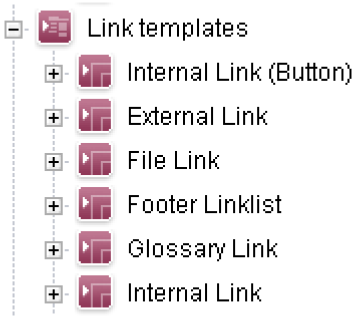

Link templates
Internal Link (Button)
External Link
File Link
Footer Linklist
Glossary Link
Internal Link

Figure 2-75: Template Store tree view – Link templates

Template developers can use link templates to specify the layout of links in detail within a FirstSpirit project. The editors enter all the necessary content via a screen mask. Which fields can be filled in the screen mask depends on the configuration of the link templates (Chapter 2.10.3 page 116). A differentiation is made between three standard types of link (see Chapter 2.10.1 page 115). Any number of instances of these three types of link can be created below the "Link Templates" node (**Link configuration**). Each instance must have a unique name and correspond to one of the standard link types.

New **Link templates** can be created below the link configurations. The configuration of all link templates below the instance are defined via the instance. The link template is only used to define output of the editorial content (for details of link configurations and link templates, see also Chapter 2.2.1.6 page 28 and 2.2.1.7 page 30).

! *From FirstSpirit Version 4.2 so-called "Generic link editors" can be used in addition. For further information please see Chapter 2.10.6 page 118.*

For further information on link templates, see "FirstSpirit Online Documentation".

Tree elements within the link templates:

Root element of the link templates

Link configuration

Link template

2.10.1 Standard link types

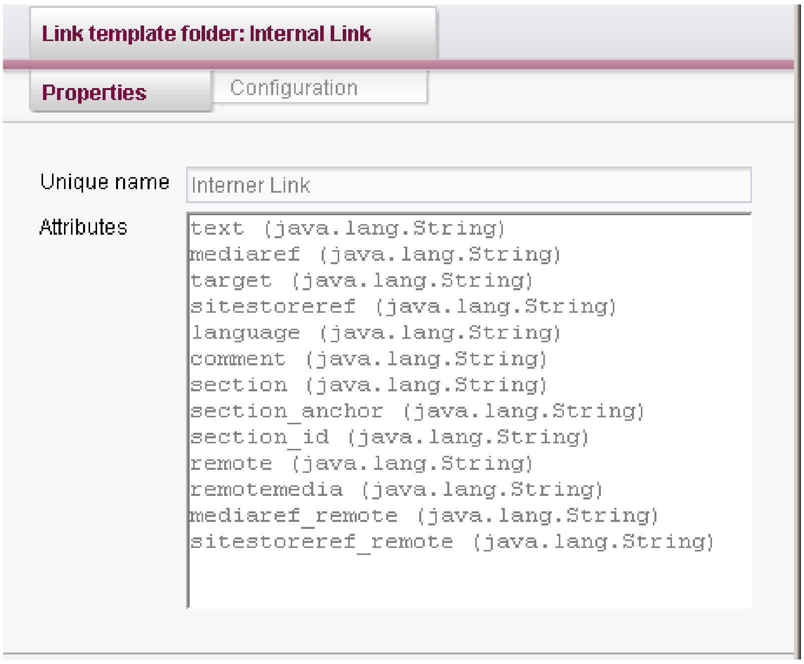
externalLink: for external links, i.e. links to elements outside the project, for example to an external website.

internalLink: for internal links to an element within the project. This link type can also be used to implement links to another FirstSpirit project.

contentLink: for links to an element from the project's Content Store.

genericLink (from FirstSpirit Version 4.2): for generic link editors (see Chapter 2.10.6 page 118)

2.10.2 Link configuration – Properties tab



Unique name: Interner Link

Attributes:

```
text (java.lang.String)
mediaref (java.lang.String)
target (java.lang.String)
sitestoreref (java.lang.String)
language (java.lang.String)
comment (java.lang.String)
section (java.lang.String)
section_anchor (java.lang.String)
section_id (java.lang.String)
remote (java.lang.String)
remotemedia (java.lang.String)
mediaref_remote (java.lang.String)
sitestoreref_remote (java.lang.String)
```

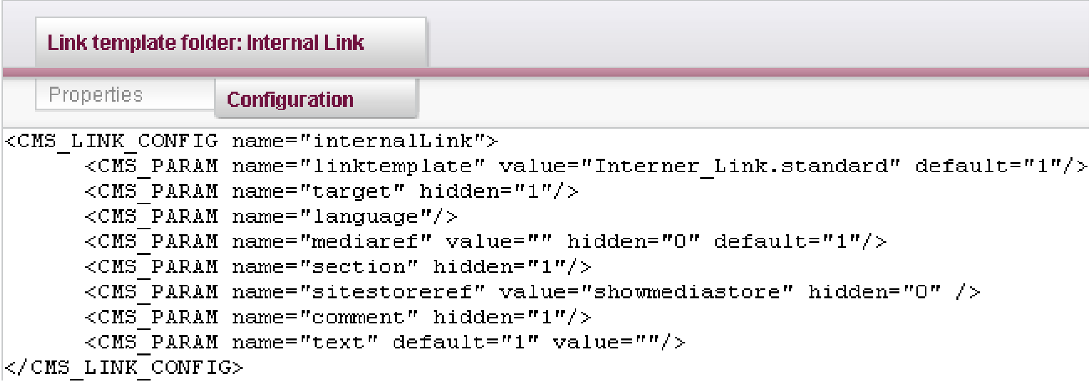
Figure 2-76: Link configuration – "Properties" tab (new Look & Feel)

Unique name: Reference name of the link configuration.

Attributes: Special attributes are available to the template developer for configuring link templates which they can view for each link type in the "Properties" tab.

For further information on the meaning of the individual attributes, see "FirstSpirit Online Documentation" – "Configuring a link template" chapter.

2.10.3 Link configuration – Configuration tab



```
<CMS_LINK_CONFIG name="internalLink">
      <CMS_PARAM name="linktemplate" value="Interner_Link.standard" default="1"/>
      <CMS_PARAM name="target" hidden="1"/>
      <CMS_PARAM name="language"/>
      <CMS_PARAM name="mediaref" value="" hidden="0" default="1"/>
      <CMS_PARAM name="section" hidden="1"/>
      <CMS_PARAM name="sitestoreref" value="showmediastore" hidden="0" />
      <CMS_PARAM name="comment" hidden="1"/>
      <CMS_PARAM name="text" default="1" value=""/>
</CMS_LINK_CONFIG>
```

Figure 2-77: Link configuration – "Configuration" tab (new Look & Feel)

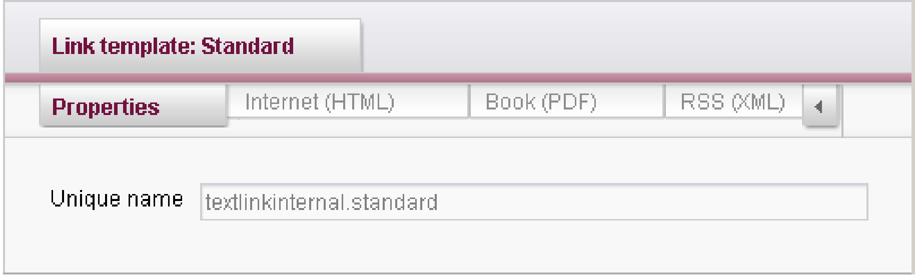
The "Configuration" tab can be used to configure the fields of the input mask for each instance of a link template type. The configuration is defined in the Editing window within the opening and closing <CMS_LINK_CONFIG> tags and is valid for all link templates of the link configuration currently being edited.

The individual fields or selections lists are configured using the definition of parameters. The parameters are defined within the <CMS_LINK_CONFIG> tags via <CMS_PARAM> tags.

If a change has been made to the link template configuration, on being saved the source text is checked for well-formedness of the XML source text. If an error is discovered during this check, this is indicated to the user in a new window.

For further information on the meaning of the individual attributes, see "FirstSpirit Online Documentation" – "Configuring a link template" chapter.

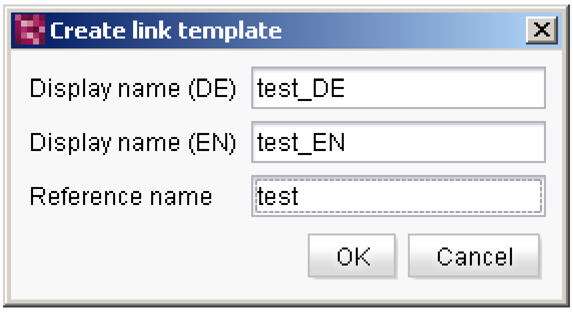
2.10.4 Link templates – Properties tab



Unique name: textlinkinternal.standard

Figure 2-78: Link template – "Properties" tab (new Look&Feel)

The unique reference name of the link template is displayed in the "Properties" tab. The name is automatically formed from the reference name of the link configuration (cf. Figure 2-77) and the reference name issued by the template developer on creating the link template, e.g.:



Display name (DE) test_DE

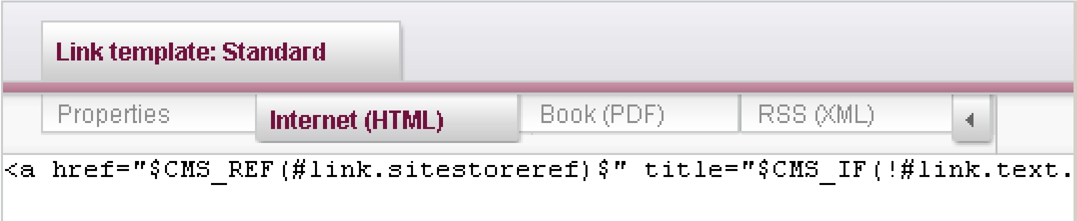
Display name (EN) test_EN

Reference name test

Figure 2-79: Create a link template (new Look&Feel)

This ensures that the link template is unique within the name space. The reference name may not be changed.

2.10.5 Link templates – Template sets tabs



```
<a href="$CMS_REF(#link.sitestoreref)$" title="$CMS_IF(!#link.text.
```

Figure 2-80: Link template – Template Sets tab

In order for the editorial contents of input components to be visible in the respective output channel the output must be defined within the templates. The same applies to the input components for entering and maintaining links. The contents entered by the editor in the relevant fields of the screen form are output via the link templates.

If a link template is selected in the tree view an Editing window appears in the right-hand part of the window with one or several tabs for each available output channel. The template developer must define the corresponding instructions for each output channel of the link template.

The field content is either output via the instruction $CMS_VALUE(...)$ or the instruction $CMS_REF(...)$, depending on which input component is involved

For further information on the output of links, see the "FirstSpirit Online

Documentation" – "Output of links" chapter.

2.10.6 Generic link editors (from V4.2)

From FirstSpirit Version 4.2, the configuration options are considerably enhanced by the introduction of generic link editors. The configuration of link templates can then, analogous to page and section templates, be created by adding input components into the form area. All input options for the maintenance of links can be reproduced using the normal form syntax of FirstSpirit.

As part of the introduction of generic link editors, the link templates can now also be structured in folders.

The conventional input options for links (of the static link editors, see Chapter 2.10.1 to 2.10.5 from page 115) can of course also be generated using the new, generic editors. Several new input components were introduced with FirstSpirit Version 4.2, to enable all the functions of the static link editors used to date to be reproduced on the new generic editors. For example, selection via the "mediaref" field of the static link editors could not be reproduced on the existing input components of FirstSpirit Version 4.1. The input components CMS_INPUT_PICTURE and CMS_INPUT_FILE each only support selection of one reference type, i.e. either pictures or files, but not both. Therefore, in FirstSpirit Version 4.2, the input component FS_REFERENCE was introduced, which supports any reference types.

Analogous to this, enhancements to the input component CMS_INPUT_OBJECTCHOOSER for reproduction of a link on database content, for selection of datasets from *one* specific database table and the new input component FS_DATASET, for selection of datasets from *any* database tables, were introduced.

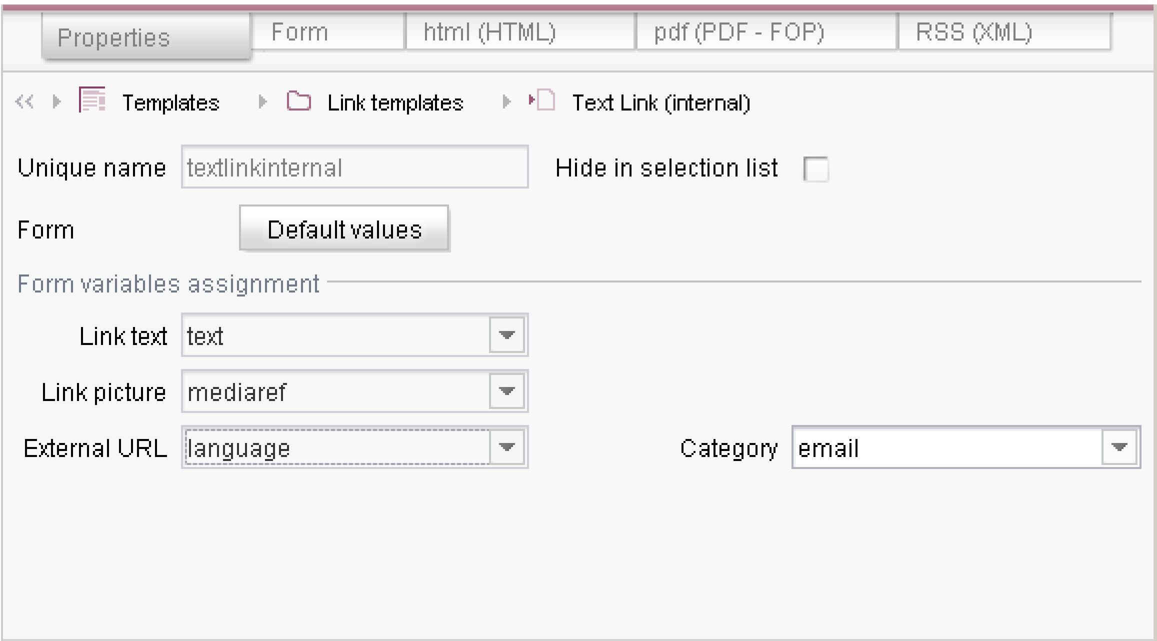
Use of the new components is subject to certain limitations, as the components will not be officially released until FirstSpirit Version 5.0 (see Chapter 1.3.2 page 14).

In FirstSpirit Version 4.2, the new components are mainly used for migration of the existing static editors. Further use of the components is possible, if the project developers dispense with use of the API and they are willing to potentially subsequently adjust the parameter assignment of the input components (see Chapter 1.3.2 page 14).

The new functions and enhancements in FirstSpirit Version 4.2 are intended to prepare existing projects for the new major release of FirstSpirit Version 5.0. The new concepts introduced can still be used in FirstSpirit Version 4.2 in parallel with the functions used to date (in Version 4.1). With the release of FirstSpirit Version 5.0, individual functions will then no longer be supported, but will instead be replaced by new, enhanced concepts. An example is the introduction of generic link editors in FirstSpirit Version 4.2. These provide greater functional scope and more flexible design options than the static editors used to date. ***The static link editors will no longer be supported from FirstSpirit Version 5.0.*** *The introduction of "generic link editors" in FirstSpirit Version 4.2 therefore also helps to migrate existing projects to FirstSpirit Version 5.0.*

The distinction between definition (form) and output is supended for generic link editors. To create a new link template, only a link configuration need to be inserted (beneath the root node "Link templates" or beneath a folder). Select "genericLink" as link type.

The “Properties“ tab can appear as follows:



Unique name: textlinkinternal
Hide in selection list □
Form
Form variables assignment
Link text: text
Link picture: mediaref
External URL: language
Category: email

Figure 2-81: Generic link – “Properties“ tab (new Look&Feel)

Every input component can be used on the “Form“ tab.

For further information about the generic link editors see “FirstSpirit Online Documentation“ – Chapter “Link templates“ / “Generic link editors“.

The generic link editors are supported – with restrictions – also in WebEdit.

2.11 Scripts

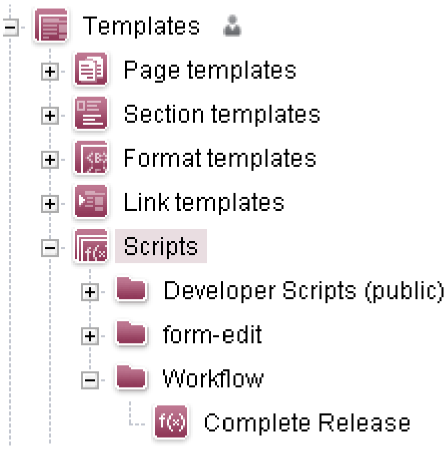


Figure 2-82: Template Store tree view – Scripts

Scripts can be used to automate different types of operating sequences or user input sequences in FirstSpirit. A script is used to describe the sequence to be executed and if necessary can make changes to FirstSpirit data structures. Scripts enable fast implementation of functions which are not yet available in FirstSpirit. Further areas of use are, for example, complex migration scenarios and linking external systems.

The script language supported in FirstSpirit is BeanShell[4]. The BeanShell syntax is to a large degree based on JAVA, but offers numerous simplifications, for example, dynamic instead of static typification of global variables and functions, as well as (limited) reflexive access to the program itself and many other functions.

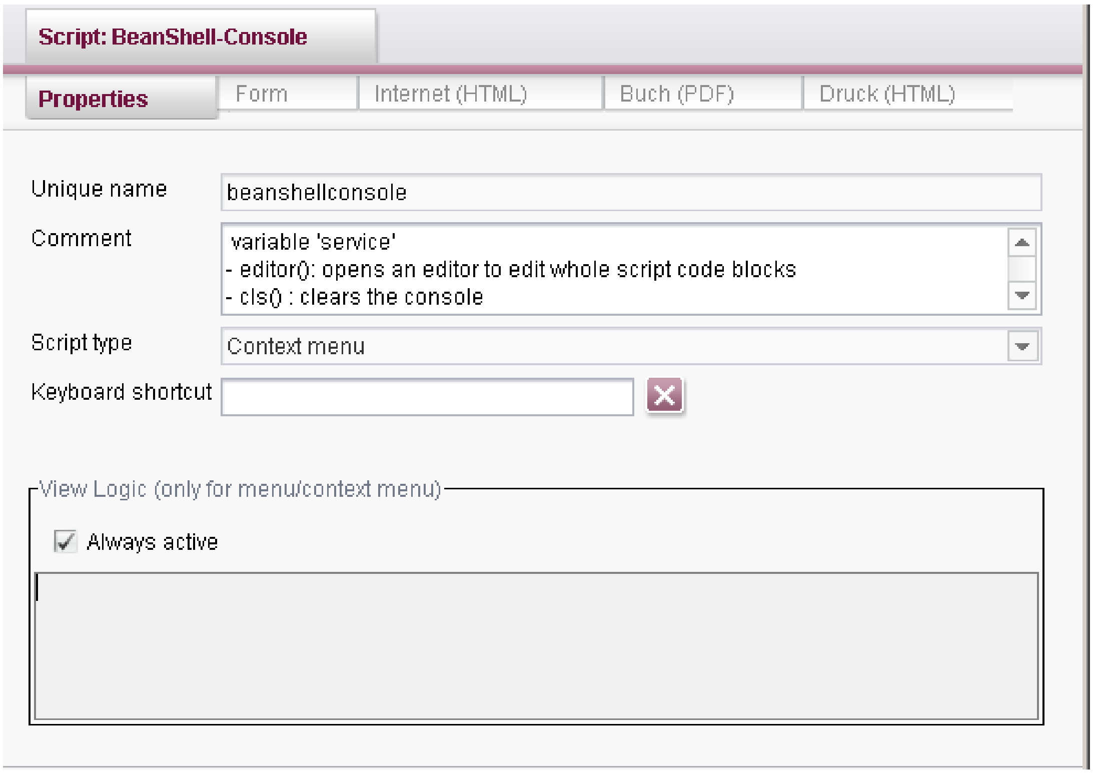
Scripting with BeanShell provides a high degree of flexibility for the template developer. However, working with scripts is not trivial, therefore, before using a script it is necessary to precisely check whether a corresponding function is already available in FirstSpirit or not!

[4] Further information about this script language is given on the website www.beanshell.org, which also provides a detailed manual (EN).

For further information on script development in in BeanShell see "FirstSpirit Online Documentation"– "Scripting" chapter.

Examples of the use of scripts in workflows (see Chapter 4.8 page 210).

2.11.1 Properties tab



Unique name: beanshellconsole

Comment:
variable 'service'
- editor(): opens an editor to edit whole script code blocks
- cls() : clears the console

Script type: Context menu

Keyboard shortcut:

View Logic (only for menu/context menu)

☑ Always active

Figure 2-83: Scripts – "Properties" tab (new Look&Feel)

Unique name: Reference name of the script.

Comment: An optional comment describing the scripts in greater detail can be entered here.

Script type: The context in which a script is to be executed can be set here:

- Template: The script can be called and executed within a template via `$CMS_RENDER(script:..)$` , e.g for rendering specific content for the PDF output channel:

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

- Menu: The script can be executed via the "Extras" menu – "Execute Script".
- Context menu: The script can be called and executed via the context menu on a specific element in the tree view of the FirstSpirit JavaClient.
- Uninterpreted: the script is not checked for BeanShell syntax on saving. As a result, for example, HTML syntax can also be saved (for example, to display list elements). These scripts should no longer be used in FirstSpirit Version 4.x. The corresponding templates should be changed over to format template in projects which use uninterpreted scripts.

Use on homepage: From FirstSpirit version 4.1 this option can be activated for scrips of the type “Menu“. This script will then be, depending on the settings made in the area “View Logic“ (see below), shown on the project home page in the area “My Actions“ and can be executed directly by one click on the link.

Keyboard shortcut: A unique keyboard shortcut can be defined for scripts in this field. In this case the script no longer has to be executed via the context menu or the "Extras" menu but instead can be opened directly via the defined keyboard shortcut. The cursor must be located within the field to define a new keyboard shortcut. It is then sufficient to enter the required key combination via the keyboard. The input is then copied into the input field. Text input is not possible. To change the keyboard shortcut, position the cursor in the field again and then call the new key combination.

Press the icon ☒ to delete a defined keyboard shortcut for the workflow.

The keyboard shortcuts can only be used for scripts of the type: "Context menu" or "Menu".

View Logic (only for menu/context menu): The view logic can be used to show or hide scripts depending on specific properties (analogous to the view logic of workflows, cf. Chapter 4.5.2 page 179). For example, a script of the type "Context menu“ can only be displayed if the context menu is opened on a page reference in the Site Store:

```
//!Beanshell
import de.espirit.firstspirit.access.store.sitestore.PageRef;
e = context.getStoreElement();
if (e instanceof PageRef) {
    context.setProperty("visible", true);
}
```

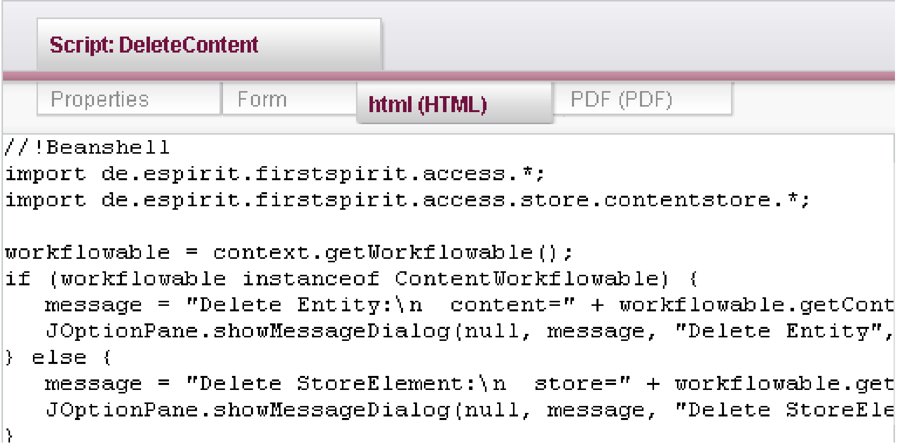
Always active: The "Always active" checkbox can be selected if the view logic is to be disabled. In this case the script is always shown, regardless of the view logic. The deposited view logic is no longer evaluated, but is retained and can be reactivated by deselecting the checkbox.

2.11.2 Form tab

Analogous to page and section templates, individual input components can be defined in the "form" tab which can be invoked at the script run time. The input component values can be returned to the script for processing (analogous to form support within workflows, cf. Chapter 4.4 page 174).

*The "FirstSpirit Online Documentation" contains a list of all available input elements. The input elements are explained with all their attributes and a schematic example under the menu item **Template Development – Forms**.*

2.11.3 Template sets tabs



```
//!Beanshell
import de.espirit.firstspirit.access.*;
import de.espirit.firstspirit.access.store.contentstore.*;

workflowable = context.getWorkflowable();
if (workflowable instanceof ContentWorkflowable) {
   message = "Delete Entity:\n  content=" + workflowable.getCont
   JOptionPane.showMessageDialog(null, message, "Delete Entity",
} else {
   message = "Delete StoreElement:\n  store=" + workflowable.get
   JOptionPane.showMessageDialog(null, message, "Delete StoreEle
}
```

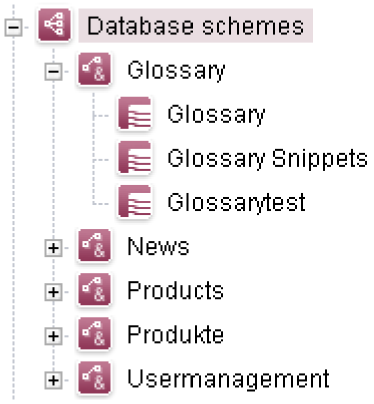
Figure 2-84: "Scripts" output channels (new Look&Feel)

The BeanShell source code is defined in the output channels in which the script is to be executed. By entering the character string `//! Beanshell` in the first line of the script, the subsequent source text is interpreted as BeanShell script.

For examples of script development within workflows, see (see Chapter 4.8 page 210).

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

2.12 Database schemes



Database schemes
Glossary
Glossary
Glossary Snippets
Glossarytest
News
Products
Produkte
Usermanagement

Figure 2-85: Template Store tree view – Database schemes

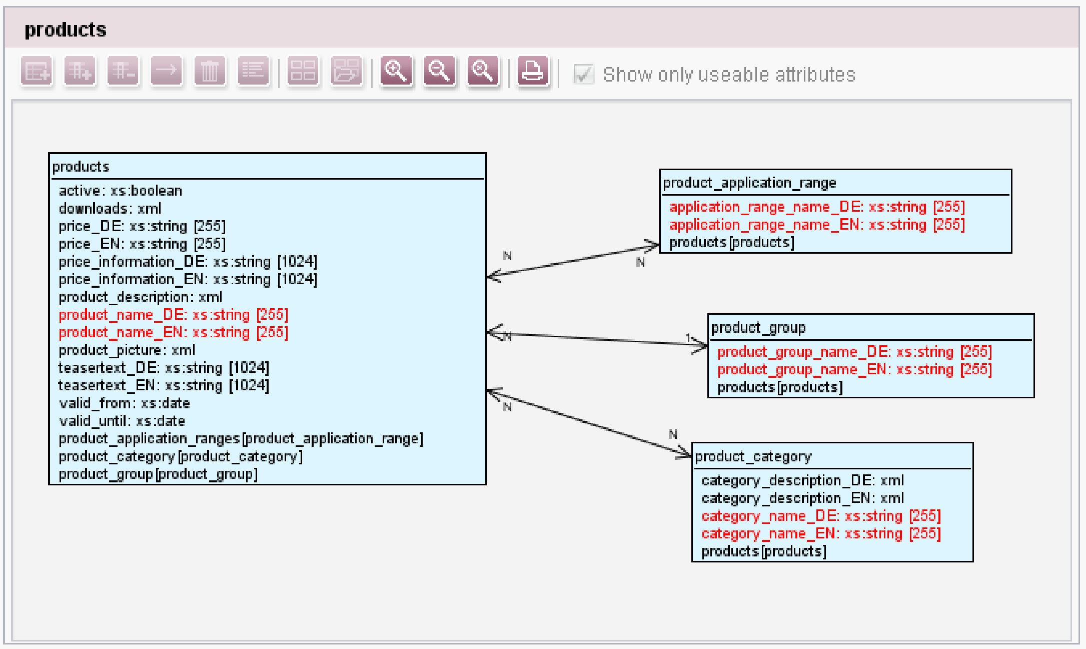
FirstSpirit has efficient mechanisms for linking databases (see Chapter 3 page 140).

In the Template Store, a graphic schema editor can be used to create and modify database tables (see Chapter 2.12.1 page 126), define templates for maintaining and displaying the data records (see Chapter 2.12.2 page 132) and formulating queries for filtering the data records (see Chapter 2.12.8 page 136). To this end, FirstSpirit implements a database abstraction layer, which maps the universal FirstSpirit content type system on the specific database system to be used (see Chapter 3 page140).

2.12.1 The FirstSpirit schema editor

A graphic editor for editing a schema in the FirstSpirit JavaClient is available on the right-hand side of the window, which can be used to create the required database schema. Depending on the configuration, a schema can use existing database structures or can create new table structures in an existing database.

The editor is used either via the editor's toolbar or via a context menu which can be opened in any position of the editor by right-clicking.



products
active: xs:boolean
downloads: xml
price_DE: xs:string [255]
price_EN: xs:string [255]
price_information_DE: xs:string [1024]
price_information_EN: xs:string [1024]
product_description: xml
product_name_DE: xs:string [255]
product_name_EN: xs:string [255]
product_picture: xml
teasertext_DE: xs:string [1024]
teasertext_EN: xs:string [1024]
valid_from: xs:date
valid_until: xs:date
product_application_ranges [product_application_range]
product_category [product_category]
product_group [product_group]

product_application_range
application_range_name_DE: xs:string [255]
application_range_name_EN: xs:string [255]
products [products]

product_group
product_group_name_DE: xs:string [255]
product_group_name_EN: xs:string [255]
products [products]

product_category
category_description_DE: xml
category_description_EN: xml
category_name_DE: xs:string [255]
category_name_EN: xs:string [255]
products [products]

Figure 2-86: Database schema editor

Figure 2-86 shows the example of the database schema of the "FIRSTools" test project's product database. It shows the products (*products*), product category (*product_category*), product group (*product_group*) and product application ranges (*product_application_range*) tables. It also shows, for example, that products and product groups are linked with each other by a 1:N relationship and products and product groups are linked by a M:N relationship.

> ! *Especially when using an Oracle database, saving database schemes and changes to a schema can take some time in version 4.2R4 and higher.*

Create table: this button can be used to insert a new table in the database schema. The following window opens:

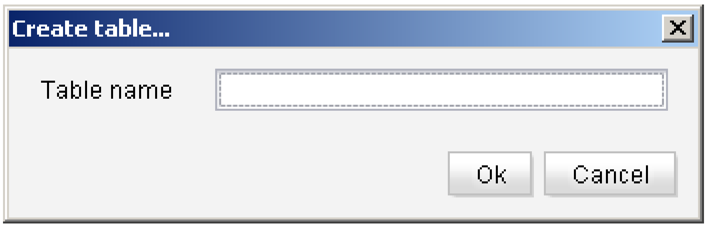


Figure 2-87: Create table

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

Create column; this button can be used to insert a new column in the activated table. The following window opens:

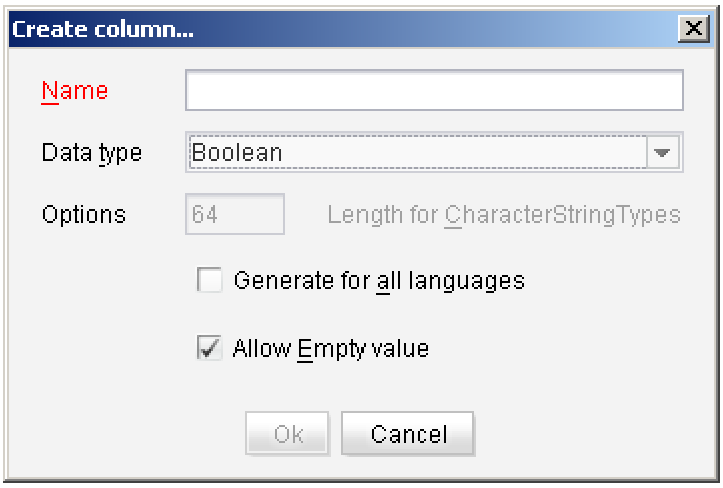


Figure 2-88: Create column

Name: The column name must be entered in this field. As long as the field is empty, *Name* is displayed in red lettering and the new column cannot be saved.

Data type: This combobox can be used to select the required data type of the new table column.

Boolean: This data type enables two values: `true` or `false`. In the schema editor this data type is given an `xs: boolean`.

Date: This data type is used for date values. In the schema editor this data type is given an `xs: date`.

Double: This data type enables the entry of floating point numbers. In the schema editor this data type is given an `xs: decimal`.

FirstSpirit-Editor: This data type enables DOM editors to be used. The maximum string length is 65535 characters. In the schema editor this data type is given an `xml`.

FirstSpirit-Link: This data type enables links to be used. The maximum string length is 65535 characters. In the schema editor this data type is given an `xml`.

Integer: This data type is used for whole numbers. In the schema editor this data type is given an `xs: integer`.

Long: This data type is also used for whole numbers, but the value range is larger than that of the data type integer. In the schema editor this data type is given an `xs:long`.

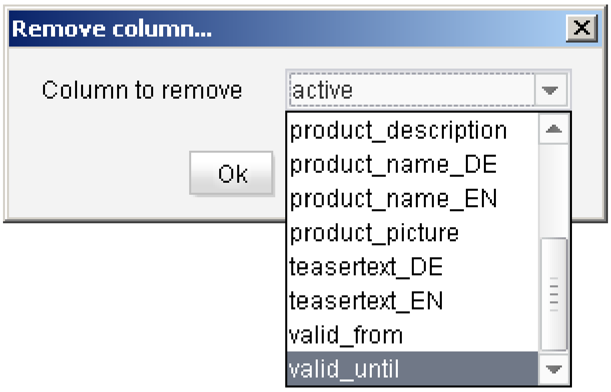
String: This data type is used for character strings. In the schema editor this data type is given an `xs: string`. The maximum number of characters allowed can also be specified for this data type.

Options: The maximum string length for the column type String must be given in this field. The respective value is displayed in square brackets behind the `xs: string`.

Generate for all languages: This option is used to enable language-dependent input of the values by the editor. If the checkbox is selected, an individual column is generated for each attribute in each language. It is useful if the language of the attribute terms differs. Product pictures, for example, do not usually have to be entered on a language-dependent basis. The columns are assigned an appropriate language abbreviation for each language.

Allow empty value: If this option is selected the editor is allowed to create a new data record without entering a value in this column. If an empty value is not allowed (i.e. it is a mandatory input), the column name is displayed in red lettering in the database schema model.

▥ Remove column; this function can be used to remove the individual columns of a table in the database schema. The required column can be selected from the combobox in the following dialog:



Remove column...

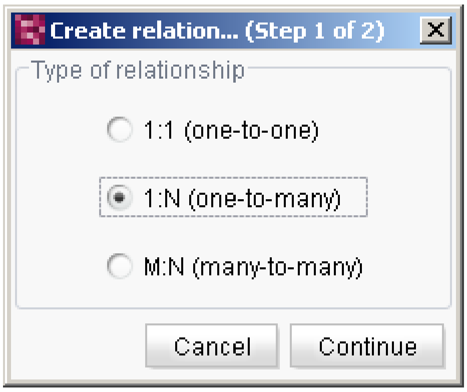
Column to remove: active

Figure 2-89: Remove column

Create foreign key relation; this button can be used to establish a relation between the activated table and another table.

The creation of a relationship is explained using the example of the product database shown in Figure 2-86. We want to establish the relationship between the products and product groups tables: a product group can contain several products, they therefore have a 1:N relationship.

Therefore, first of all the product group table must be activated and in addition (with pressed Shift key) the products table is activated (activated tables change their border colour). If the *Create foreign key relation* button is now selected the first step of the process appears in which the type of relationship must be defined. The two tables are to have a 1:N relationship.



Create relation... (Step 1 of 2)

Type of relationship

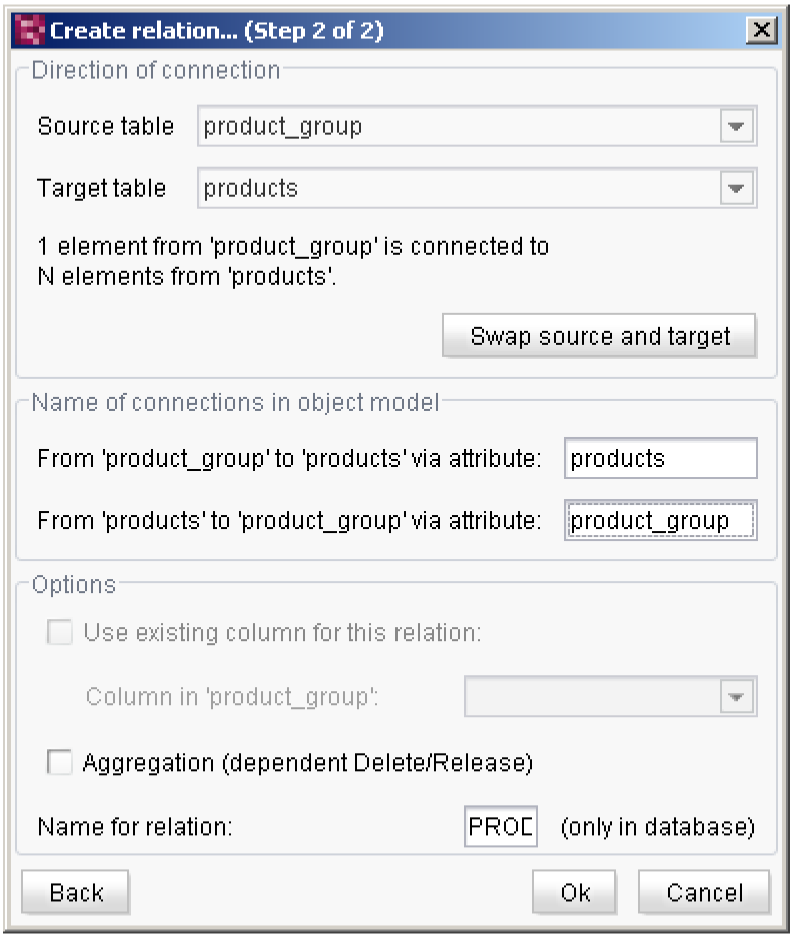
○ 1:1 (one-to-one)

● 1:N (one-to-many)

○ M:N (many-to-many)

Figure 2-90: Create relationship – Step 1

The second window of the relationship looks like this (however, the appearance can differ depending on the selection made in the first window):



Create relation... (Step 2 of 2)

Direction of connection

Source table: product_group

Target table: products

1 element from 'product_group' is connected to
N elements from 'products'.

Name of connections in object model

From 'product_group' to 'products' via attribute: products

From 'products' to 'product_group' via attribute: product_group

Options

□ Use existing column for this relation:

Column in 'product_group':

□ Aggregation (dependent Delete/Release)

Name for relation: PROD (only in database)

Figure 2-91: Create relationship – Step 2

The direction of the connection is already predefined by the order in which the tables are activated so that 1 element from the product table is connected to N elements from the products table. If the tables were inadvertently activated in the wrong order, the *Swap source and target* button can be used to change over the order. The other information in this window can usually be accepted as suggested by the system. The names given in the "Names of connections in object model" area are used subsequently to follow the databases along their relations.

Delete elements: this button can be used to delete the activated table from the database schema. It is deleted immediately, without a confirmation prompt; any data not saved is lost and cannot be restored.

Properties: this button can be used to display the name of the activated table.

Automatic layout; click this button to automatically arrange the displayed tables in the editor.

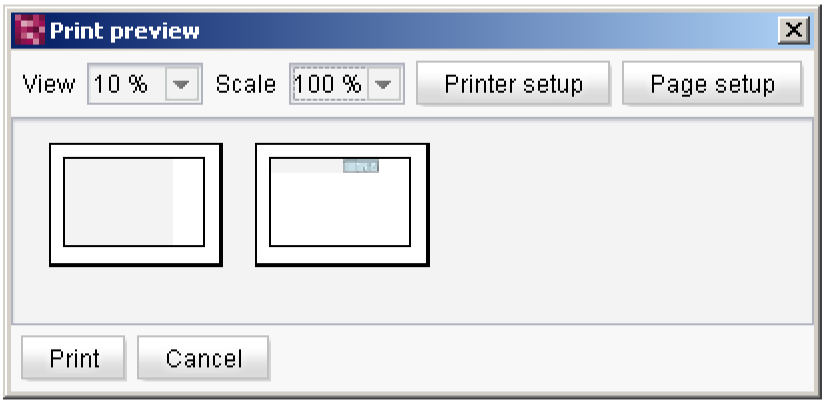
Load saved layout; this button can be used to undo changes to the arrangement of the schema's tables. The last saved arrangement is displayed again.

Zoom in; this button can be used to enlarge the display (zoom in) of the database schema's elements.

Zoom out; this button can be used to reduce the display size (zoom out) of the database schema's elements.

Normal view; this button can be used to display the elements of the database schema with their original size again.

Print: this button can be used to printout the database schema. The following print preview window opens first:



Print preview

View 10 %

Scale 100 %

Figure 2-92: Print preview

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

Scale: If necessary, the database schema is printed out with a reduced size. The possible scaling levels are 10%, 25%, 50% and 100%.

Printer setup: Opens the dialog for the printer settings.

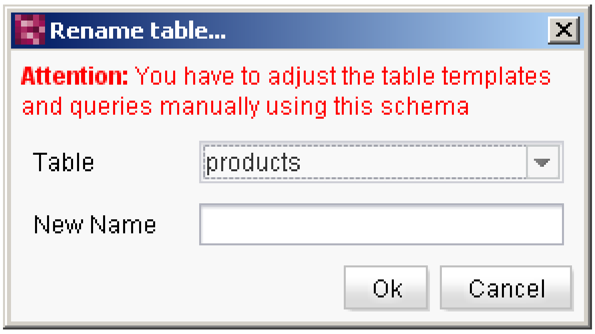
Page setup: Opens the dialog for the page settings.

The Print button is used to start the print job with the current settings.

☑ Show only usable attributes: if this tick is set, all the attributes of a table which cannot be filled with content by the editor are hidden.

The context menu can be used to open two other functions in addition to the icon functions:

Rename table: This function can be used to issue a new name for an existing table. A window appears in which an existing table of the database schema can be selected and a new name can be entered for the table. When renaming it is necessary to note that the table templates and queries based on this table must be adjusted.



Rename table...

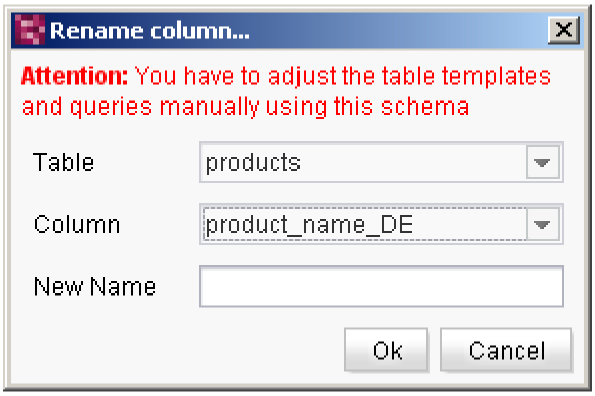
Attention: You have to adjust the table templates and queries manually using this schema

Table: products

New Name:

Figure 2-93: Rename table

Rename column: This function can be used to issue a new name for an existing column of a table. A window appears in which an existing table and column of the database schema can be selected and a new name can be entered for the column. When renaming it is necessary to note that the table templates and queries based on this column must be adjusted.



Rename column...

Attention: You have to adjust the table templates and queries manually using this schema

Table: products

Column: product_name_DE

New Name:

Figure 2-94: Rename column

2.12.2 Table templates – Preview tab

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

For information on the "Preview" tab, see Chapter 2.5.1 page 81.

2.12.3 Table templates – Properties tab

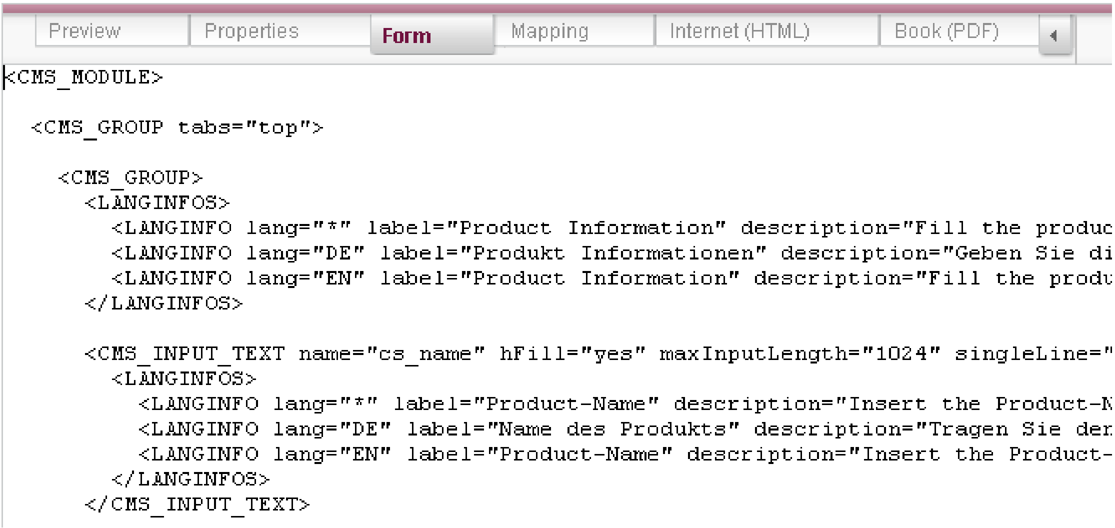
The tab for table templates is identical to the tab for page templates with the same name and can be edited in the same way.

For information on the "Properties" tab, see Chapter 2.5.2 page 83.

The "Hide from selection list" option (cf. Chapter 2.5.2) is not available for table templates.

2.12.4 Table templates – Form tab

Analogous to the page and section templates (cf. Chapter 2.5.3 page 86), the input elements for the editor are defined in the form area.



```
<CMS_MODULE>

  <CMS_GROUP tabs="top">

    <CMS_GROUP>
      <LANGINFOS>
        <LANGINFO lang="*" label="Product Information" description="Fill the produc
        <LANGINFO lang="DE" label="Produkt Informationen" description="Geben Sie di
        <LANGINFO lang="EN" label="Product Information" description="Fill the produ
      </LANGINFOS>

      <CMS_INPUT_TEXT name="cs_name" hFill="yes" maxInputLength="1024" singleLine="
        <LANGINFOS>
          <LANGINFO lang="*" label="Product-Name" description="Insert the Product-N
          <LANGINFO lang="DE" label="Name des Produkts" description="Tragen Sie den
          <LANGINFO lang="EN" label="Product-Name" description="Insert the Product-
        </LANGINFOS>
      </CMS_INPUT_TEXT>
```

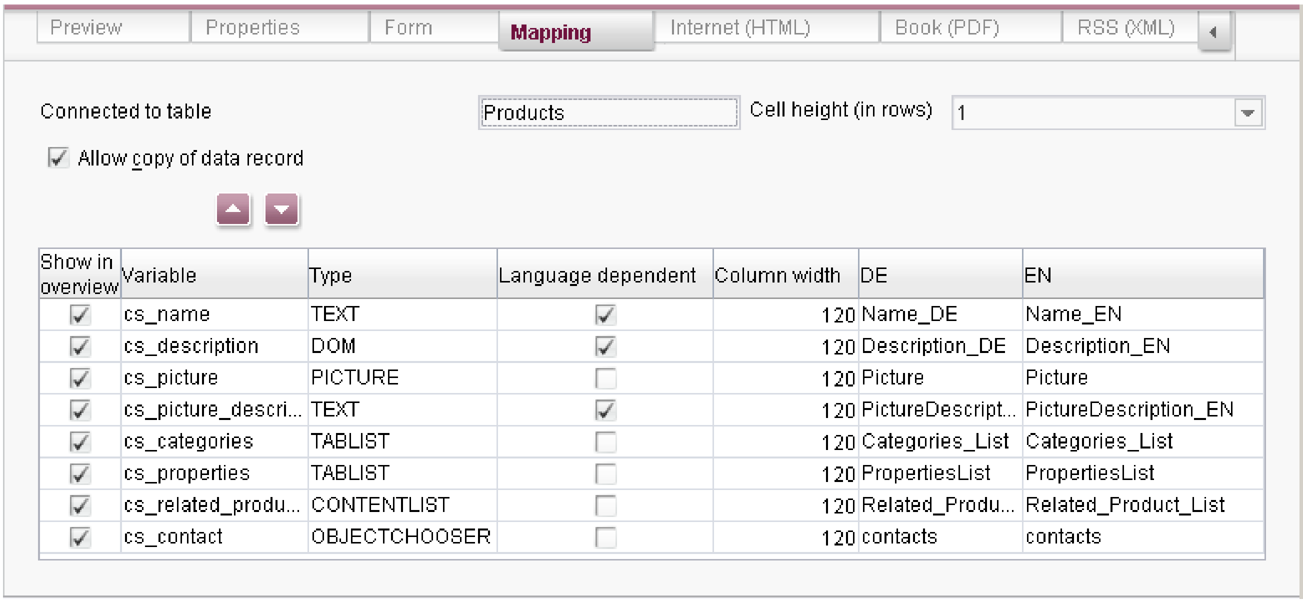
Figure 2-95: Table template – "Form" tab

*The "FirstSpirit Online Documentation" contains a list of all available input elements. The input elements are explained with all their attributes and a schematic example under the menu item **Template development – Forms**.*

2.12.5 Table templates – Mapping tab (up to and including V4.0)

The input elements and table columns are then linked with each other in the mapping area (Chapter 2.12.5 page 133).

This area is used to define the input components via which the data records are inserted in the database tables. Each input component defined (in the Form tab) is assigned a table column.



Connected to table: Products

Cell height (in rows): 1

☑ Allow copy of data record

Show in overview	Variable	Type	Language dependent	Column width	DE	EN
☑	cs_name	TEXT	☑	120	Name_DE	Name_EN
☑	cs_description	DOM	☑	120	Description_DE	Description_EN
☑	cs_picture	PICTURE	□	120	Picture	Picture
☑	cs_picture_descri...	TEXT	☑	120	PictureDescript...	PictureDescription_EN
☑	cs_categories	TABLIST	□	120	Categories_List	Categories_List
☑	cs_properties	TABLIST	□	120	PropertiesList	PropertiesList
☑	cs_related_produ...	CONTENTLIST	□	120	Related_Produ...	Related_Product_List
☑	cs_contact	OBJECTCHOOSER	□	120	contacts	contacts

Figure 2-96: Table template – "Mapping" tab (new Look&Feel, view V4.1)

Figure 2-96 shows all fields and columns available from FirstSpirit V4.1. They are described in Chapter 2.12.6 page 135. The following fields and columns are available up to FirstSpirit V4.0:

Variable: This column contains the name of the variables as defined in the table template form (Chapter 2.12.2 page 132).

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

Language dependent: If the input component in the form tab is defined in multiple languages, this is indicated by a tick in this column.

Column width: This field is used to enter the width of the column in pixels, with which it is displayed later in the Content Store.

Language (DE/EN): This field is used to select the table column in which the content of the input element is to be transferred. A separate column exists for each project language. If the input component is language-independent, the same table column must be selected here for each language. If input components are language dependent, a separate table column must exist for each language, into which the value is transferred.

2.12.6 Table templates – Mapping tab (from V4.1)

From FirstSpirit version 4.1 additional configuration options are available in comparison with them of version 4.0 (see Chapter 2.12.5 page 133). They are displayed in Figure 2-96.

Connected to table: In this field the table is shown for which the mapping settings are valid.

Cell height (in rows): Data sets can be displayed in the related Content source (see *FirstSpirit Manual for Editors*, Chapter 5) multiline. This combobox can be used to define how many rows a cell or a data set should have in the data overview (maximum: 10). In this way, thumbnails of pictures can be displayed in the overview too.

Allow copy of data record: If this checkbox is activated (default setting) existing data sets can be copied in the respective Content source by the editor. If the checkbox is deselected only "empty" new data sets can be created, the icon ⧉ is then disabled.

▲ ▼ Each row of this list corresponds to a column in the data overview of the related Content source. Clicking on these icons the selected row can be moved one position up or down, the related column in the data overview will be moved one position to the left or right. In this way, more important columns can be moved to the left. The order of the columns can be modified by the editor manually, but when the view is refreshed the order will be reset to the setting on this tab. In contrast, the order which is set here does not have any effects on the data entry.

Show in overview: Deselect the checkboxes in this column to hide table columns in the data overview of the respective Content source, for example to improve the clarity if there are many columns. In contrast, the order which is set here does not have any effects on the data entry.

2.12.7 Table templates – Template sets tabs

These tabs are used to specify the subsequent appearance of the individual data records, entered with the help of this table template, on the website or in other output channels.

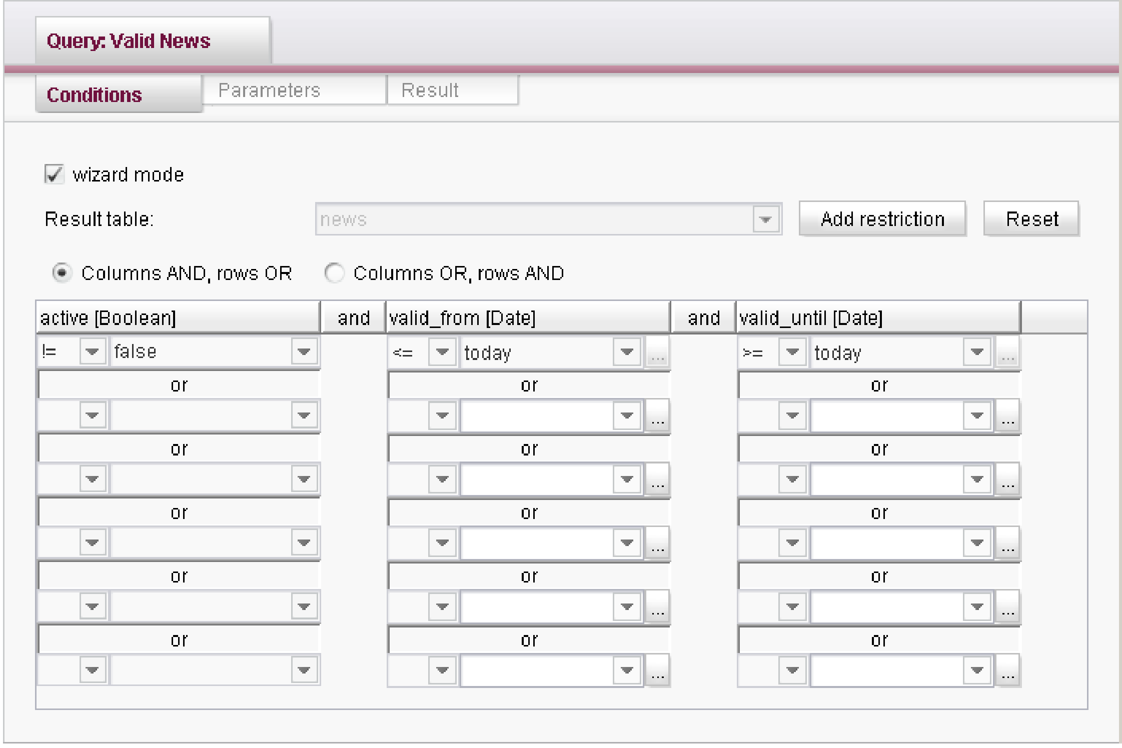
The tab for table templates is identical to the tab for page templates with the same name and can be edited in the same way.

For information on the "Template sets" tab, see Chapter 2.5.4 page 87.

2.12.8 Query – Conditions tab

Several queries can be created for each content schema to limit the number of data records for subsequent output. The conditions a data record must fulfil in order to be included in the results list are specified in these queries.

In the "Conditions" tab a graphic editor can be used to define the required filter criteria for a query in so-called "Wizard Mode". Several rules can be defined, which then affect the display of the matching data records in the "Result" tab.



☑ wizard mode

Result table: news

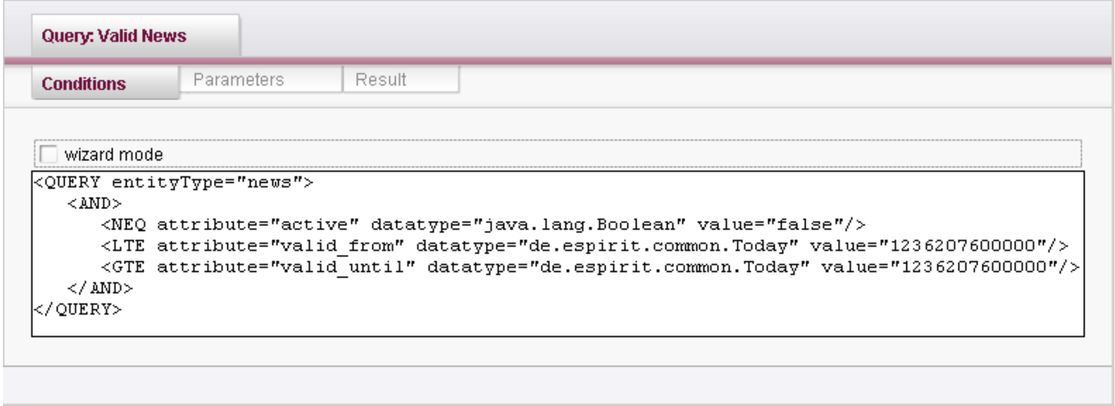
● Columns AND, rows OR ○ Columns OR, rows AND

active [Boolean]		valid_from [Date]		valid_until [Date]
!= false	and	<= today	and	>= today
or		or		or
or		or		or
or		or		or
or		or		or
or		or		or

Figure 2-97: Query – "Conditions" tab (Wizard Mode) (new Look&Feel)

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

> ! *Tags and parameters which can be used for direct coding of queries can be looked-up in the FirstSpirit Online Documentation: section "Query part (QUERY)" in the chapter "Function contentSelect" ("Template development" / "Template syntax" / "Functions" / "in the header" / "contentSelect"").*



□ wizard mode

```
<QUERY entityType="news">
   <AND>
      <NEQ attribute="active" datatype="java.lang.Boolean" value="false"/>
      <LTE attribute="valid_from" datatype="de.espirit.common.Today" value="1236207600000"/>
      <GTE attribute="valid_until" datatype="de.espirit.common.Today" value="1236207600000"/>
   </AND>
</QUERY>
```

Figure 2-98: "Conditions" tab (no Wizard Mode) (new Look&Feel)

If changes are made to the query which cannot be displayed in Wizard mode, the query is automatically adjusted (in the editor), as soon as Wizard mode is enabled again.

Result table: A table from the FirstSpirit schema for which output restrictions are to be entered can be selected here. Once this selection has been made, this field is disabled. The selection can only be removed from the whole query using the "Reset" button (see below).

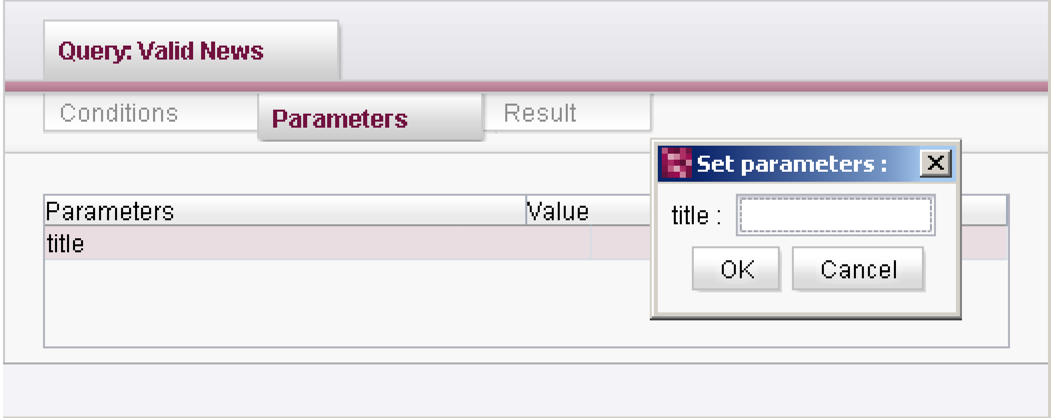
Add restriction: Press this button to add a new condition. A window opens in which a specific column of the selected result table can be selected as the new reference. The restricting condition can then only be defined for this reference. The specific values which must be fulfilled can be entered in the condition column in the bottom part of the window. To do this, the required comparative operator for the condition is selected in the left-hand field. In the right-hand field, either a specific comparable value can be entered or a parameter identifier can be specified for the comparable value. This parameter is then queried for each execution so that the specific comparable value does not have to be defined until the execution.

Reset: All conditions defined to date and query results are removed. A result table can be selected again. A confirmation prompt appears before the reset so that data cannot be inadvertently deleted.

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

Columns OR, rows AND: If this option is selected, the collated results of all columns is output, duplicate data records are skipped. The individual rows of a column condition are connected by an AND operation.

2.12.9 Query – Parameter tab



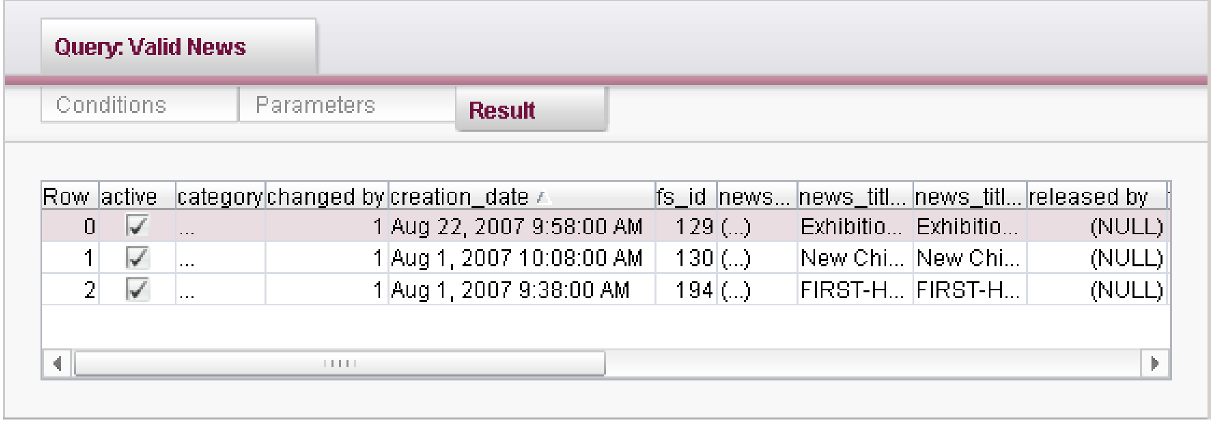
Parameters	Value
title	

Figure 2-99: Query – "Parameters" tab (new Look&Feel)

All parameters used in this query are listed in this area. If necessary, the parameters can be set now in the **Value** column. I.e. the corresponding query parameters are assigned values. With each execution, these values are then used for the query.

2.12.10 Query – Result tab

The result data records which result due to the conditions in the query and the values assigned to the query parameters are output in this area.



Row	active	category	changed by	creation_date	fs_id	news...	news_titl...	news_titl...	released by
0	✓	...	1	Aug 22, 2007 9:58:00 AM	129	(...)	Exhibitio...	Exhibitio...	(NULL)
1	✓	...	1	Aug 1, 2007 10:08:00 AM	130	(...)	New Chi...	New Chi...	(NULL)
2	✓	...	1	Aug 1, 2007 9:38:00 AM	194	(...)	FIRST-H...	FIRST-H...	(NULL)

Figure 2-100: Query – “Result” tab (new Look&Feel)

2.13 Workflows

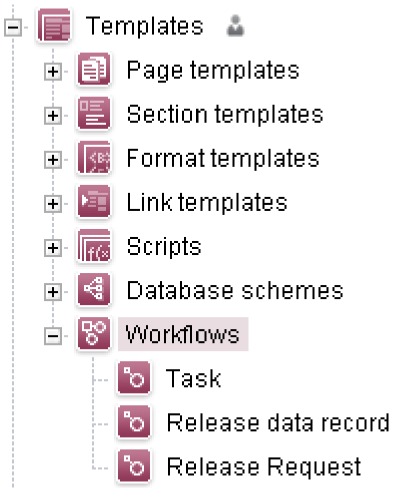


Figure 2-101: Template Store tree view – Workflows

A workflow is a sequence of tasks which are worked through in a fixed, specified structure. Both due dates and authorised groups of people can be defined for the respective tasks. Workflows integrated in FirstSpirit are the task setting and the release request.

For details of modelling, configuration and execution of workflows, see Chapter 4 page 152 ff.

3 Data sources in FirstSpirit

FirstSpirit has efficient mechanisms for linking databases. Within the editing environment, the linked databases are called content or data sources. The data records managed in the content sources can be integrated in the web pages and seamlessly edited in FirstSpirit, without leaving the editing environment (see Chapter 3.5 page 149).

The linking of databases is available for a large number of databases and is executed via the JDBC drivers provided by the database manufacturers. Each database manufacturer implements their own internal structure to manage the data stored in the database server (DBMS). These internal structures in conjunction with the security and maintenance requirements of internal company operation have implications for the form and configuration of the databases' connection with FirstSpirit.

For further information on linking and configuring databases in FirstSpirit, see "FirstSpirit Manual for Administrators", Chapter 4.8 "Database connection“.

The following chapters are intended to support the template developer on choosing the correct connection and to explain the concepts for working with data sources (content) in the FirstSpirit JavaClient:

Chapter 3.1 defines the terms used, as these can have different meanings within the area of databases depending on the context.

Chapter 3.2 and 3.3 deal with the layer types used in FirstSpirit for the database connection. The choice of layer type has diverse effects on subsequent operation, but cannot be easily changed and should therefore be carefully considered first.

Chapter 3.5 describes the concept of content (data) sources in the FirstSpirit JavaClient.

General recommendations for updating templates on content are discussed in Chapter 3.6.

3.1 Terms

Many misunderstandings arise in dealing with the linking of content (data) sources in FirstSpirit due to a large number of terms used, some with ambiguous meanings. The manufacturers of the databases to be linked also tend to use their own terminologies, which overlap with the FirstSpirit terminology. Therefore the terms used in this document are clarified first:

Layer: This term is used to describe the connection configuration in FirstSpirit to a Database Management System (DBMS). In FirstSpirit a layer can be assigned to several FirstSpirit schema*ta* (see below). From FirstSpirit Version 4.0, the layer types used are "Multiproject layer" and "SingleProjectLayer":

- Multiproject layer: This layer type contains an explicit definition of the DB schema used in the layer definition. In this case, all tables of the FirstSpirit schemata which use this layer are stored in the given DB schema. This layer type behaves differently under FirstSpirit Version 3.1 (see Chapter 3.2.1 page 143) and FirstSpirit Version 4.x (see Chapter 3.2.2 page 145). In FirstSpirit Version 4.x, a MultiProjectLayer should only be assigned to exactly one FirstSpirit schema (see Chapter 3.2 page 143).
- Single project layer: This layer type is a new introduction in FirstSpirit Version 4.0 and does not contain any explicit definition of the DB schema to be used. FirstSpirit automatically creates a separate DB schema for each FirstSpirit schema. The database user given in the layer requires extensive database permissions for this (see Chapter 3.3 page 147).

! *For information about the use of the layer names from FirstSpirit Version 4.2 see Chapter 3.4 page 148.*

FirstSpirit schema: This term describes the structures and templates of data sources (content) described in FirstSpirit. FirstSpirit schemata (or “schemes”) therefore contain both tables and their foreign key relationships and the templates for their generation. The table structure and data records of the FirstSpirit schemata are deposited in a DBMS within a *DB schema* (see below) (see Chapter 3.5 page 149). Exactly one layer is always assigned to each FirstSpirit schema (see Chapter 3.2 page 143 and Chapter 3.3 page 147).

DB schema: This term describes the logical area within the database in which the tables are filed ("tablespace"). Each table within this area must have a unique name.

The term "database" is also frequently used as a technical term in DBMS. In the layer configuration, FirstSpirit simply calls this "schema" (plural form “schemes” or “schemata”).

3.2 Multiproject layers

The following chapters describe the use of MultiProjectLayers and discuss the relevant differences between FirstSpirit versions 3.1 and 4.0:

- Use of Multiproject layers in **FirstSpirit Version 3.1**
  (see Chapter 3.2.1 page143)
- Use of Multiproject layers in **FirstSpirit Version 4x**
  (see Chapter 3.2.2 page145)

For information about the use of the layer names from FirstSpirit Version 4.2 see Chapter 3.4 page 148.

3.2.1 Multiproject layer in FirstSpirit Version 3.1

In FirstSpirit Version 3.1, there is only one layer type, the so-called Multiproject layer. In FirstSpirit Version 3.1, this is capable of holding the tables of several FirstSpirit schemata (including from different projects). A mechanism renames the tables, ensuring that even tables with the same name from different FirstSpirit schemata do not result in conflicts within the DB schema. If a name overlap occurs in the table name in the DB, FirstSpirit Version 3.1 appends a number, which is increased if overlapping occurs again. A table in the FirstSpirit schema called "category" then becomes, e.g. "category1" in the DB schema.

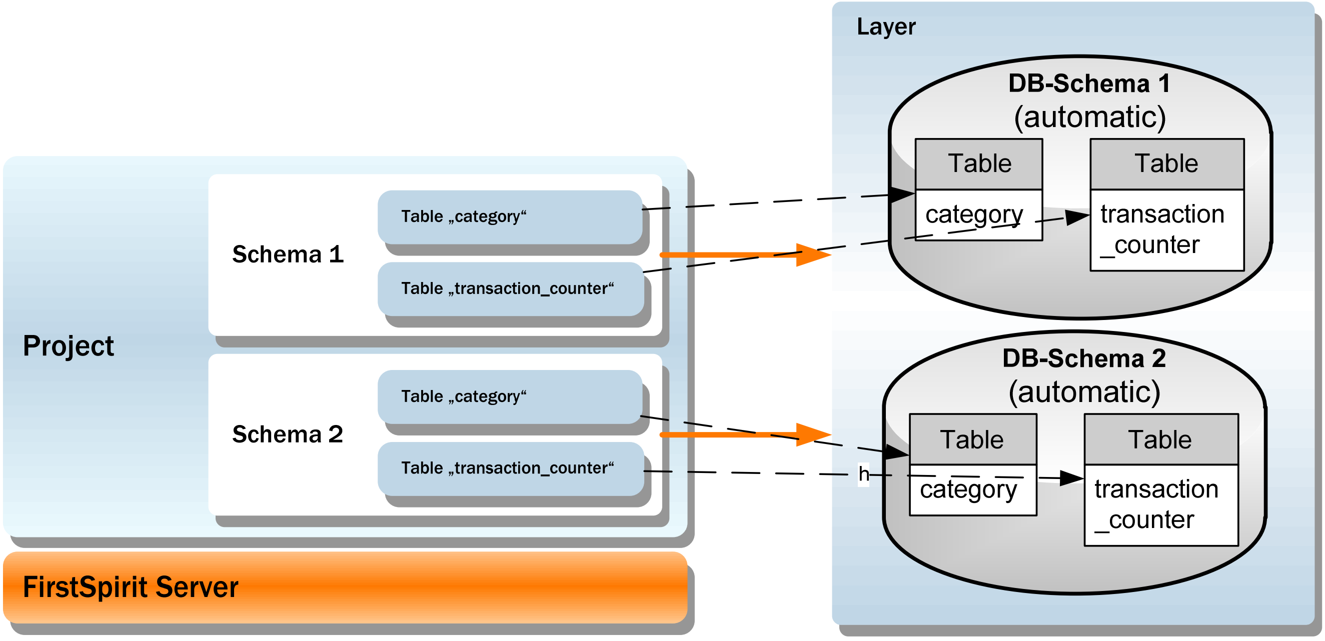


Figure 3-1: Multiproject layer in FirstSpirit Version 3.1

FirstSpirit memorises this assignment in the internally managed schema definition and can therefore correctly manage the data filed in the data source.

> **!** *Despite the convenient mechanisms of the Multiproject layer it is advisable to explicitly assign a separate DB schema to each FirstSpirit schema.*

But in practice, the FirstSpirit schemata at least of a project are frequently placed on a single DB schema. The reason for this is often that, unlike the recommended procedure, database administrator support is not required. With the recommended procedure, the database administrator must first create the corresponding DB schema for each FirstSpirit schema in the DBMS.

However, the mechanism of assigning several DB schemata to one MultiProjectLayer also led to unmanageable and unclear DB schemata in FirstSpirit Version 3.1 and was therefore not continued in FirstSpirit Version 4.0. Therefore, a Multiproject layer in FirstSpirit Version 4.x can no longer be assigned to several FirstSpirit schemata, without provoking conflicts in the database (see Chapter 3.2.2 page 145). In FirstSpirit Version 4.0, the SingleProjectLayer was introduced as a replacement for the flexible assignment of FirstSpirit schemata to DB schemata (cf. Chapter 3.3).

3.2.2 Multiproject layer in FirstSpirit Version 4.0

The so-called Multiproject layer also exists under FirstSpirit Version 4.0. Its name has been retained from the historical use under FirstSpirit Version 3.1 (cf. Chapter 3.2.1), although the mechanism for avoiding conflicts if tables with the same name occur in different FirstSpirit schemata no longer exists under FirstSpirit Version 4.0. As a replacement, the Single project layer was introduced in FirstSpirit Version 4.0 (cf. Chapter 3.3).

> ! *If the Multiproject layer in FirstSpirit Version 4.0 is assigned to several FirstSpirit schemata, a conflict occurs in the event of tables with the same name within the FirstSpirit schemata, because these are assigned to the same table in the DB schema (cf. Figure 3-2).*

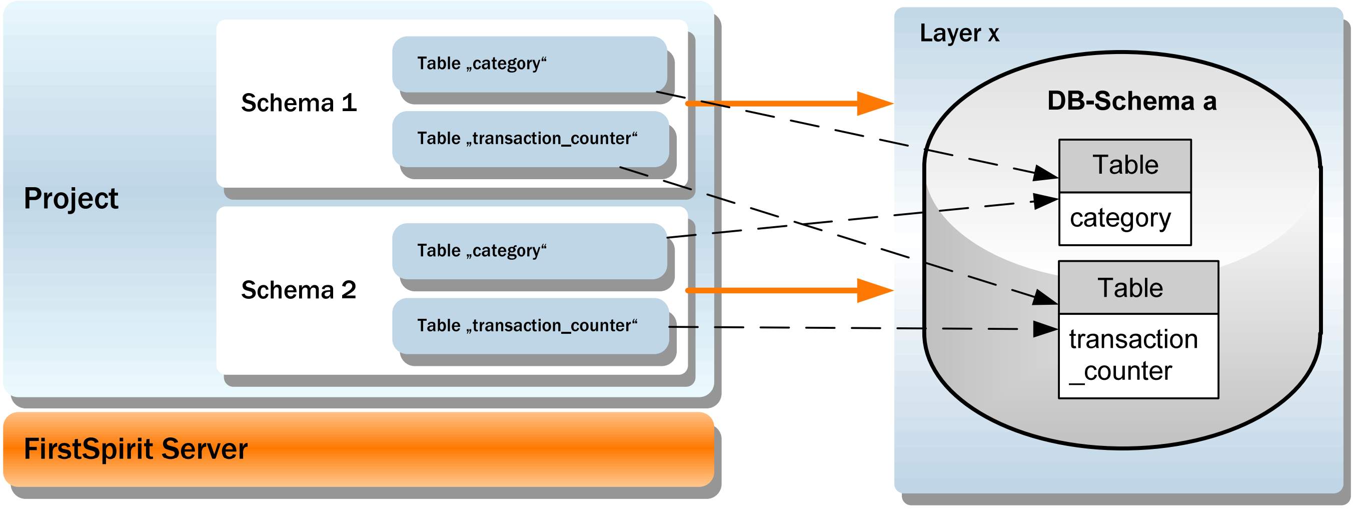


Figure 3-2: Problematic use of a Multiproject layer

A special case here is the system table "transaction_counter", which is created as a concealed table for each FirstSpirit schema. Here, FirstSpirit Version 4.0 tries to remove the aforementioned conflict by transferring the tables into one table.

> ! *We always advise against mixing two FirstSpirit schemata in one DB schema. Multiproject layers should **always** be assigned to one FirstSpirit schema only.*

Correct use of Multiproject layers is shown in Figure 3-3. A separate Multiproject layer is created for each FirstSpirit schema and therefore a separate DB schema is filed in the corresponding tables.

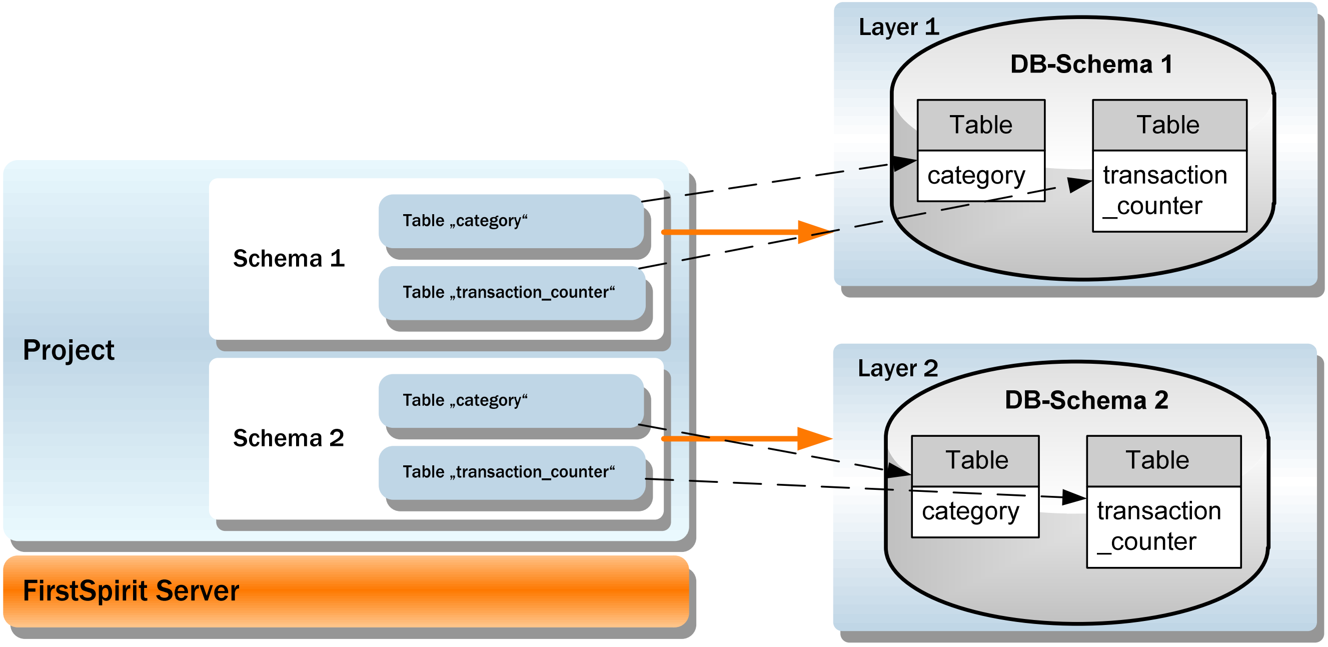


Figure 3-3: Correct use of separate Multiproject layers

3.3 Single project layers

The Single project layer was introduced with FirstSpirit Version 4.0 to enable FirstSpirit schemata to be created in FirstSpirit Version 4.x without the assistance of a database administrator.

Unlike the Multiproject layer, with the Single project layer an explicit DB schema is not given for saving the tables in the layer definition. FirstSpirit creates the DB schemata belonging to the FirstSpirit schemata independently in the DBMS. The name of the DB schemata is made up of the schema and project ID (cf. Chapter 2.2.1.9 page 31).

For use of a Single project layer, the user named in the layer requires the necessary permissions to create DB schemata in the DBMS. In many DBMS this is only possible with permissions similar to those of a database administrator.

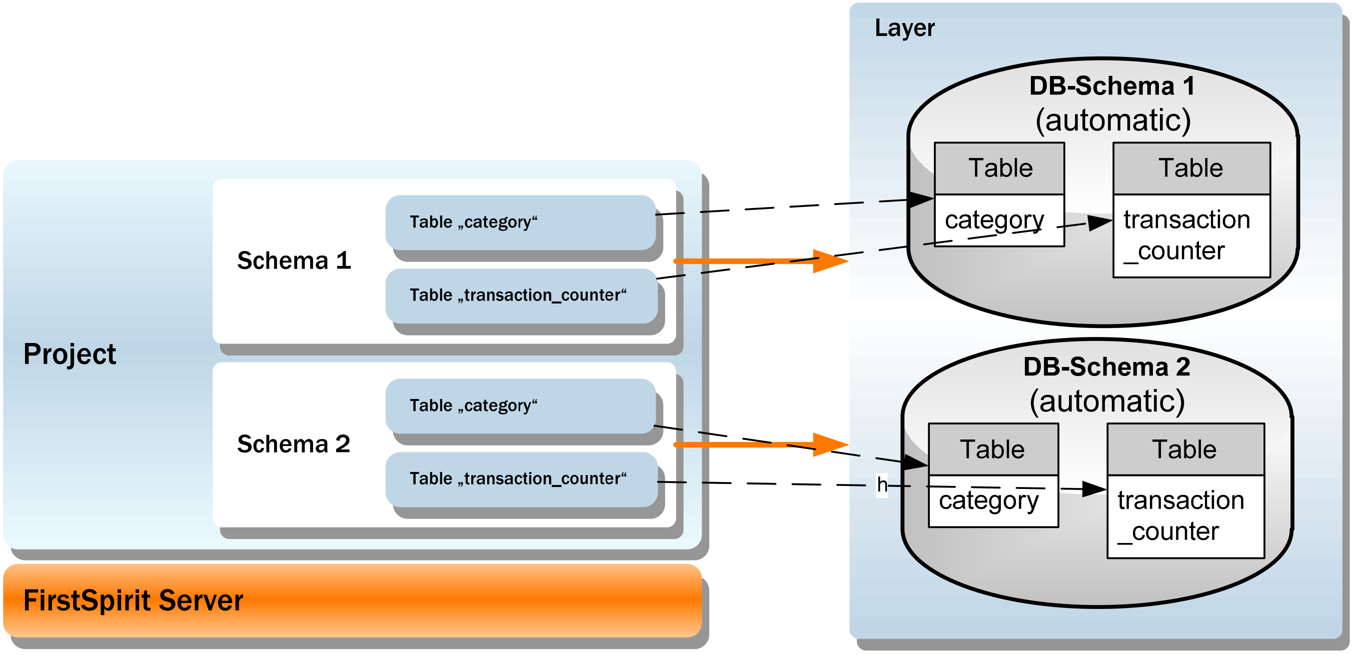


Figure 3-4: Single project layer under FirstSpirit Version 4.0

For information about the use of the layer names from FirstSpirit Version 4.2 see Chapter 3.4 page 148.

3.4 Layer types and their names from FirstSpirit Version 4.2

The name of the layer types for the database link in FirstSpirit 4.0 led to misunderstandings during use in the past. Therefore, the name is changed with the introduction of FirstSpirit Version 4.2. The familiar function is retained – it is not necessary to adjust existing projects.

Version 4.0 and 4.1	from Version 4.2	Description
Multi project layer	**Default layer**	This layer type contains an explicit definition of the DB schema used in the layer definition. In this case, all tables of the FirstSpirit schemata which use this layer are stored in the given DB schema. In a FirstSpirit project to which default layers only are assigned, a FirstSpirit user cannot create any new additional schemata. Only the FirstSpirit administrator can add further default layers to the project. A default layer should always be assigned to precisely one FirstSpirit schema only.
Single project layer	**DBA layer**	This layer type is a new introduction in FirstSpirit Version 4.0 and does not contain any explicit definition of the DB schema to be used. FirstSpirit automatically creates a separate DB schema for each FirstSpirit schema. With this, FirstSpirit users are now also able to create further schemata. However, for most DBMS, this requires comprehensive DBA rights (DBA = Database Administrator).

3.5 Data (content) sources in the FirstSpirit JavaClient

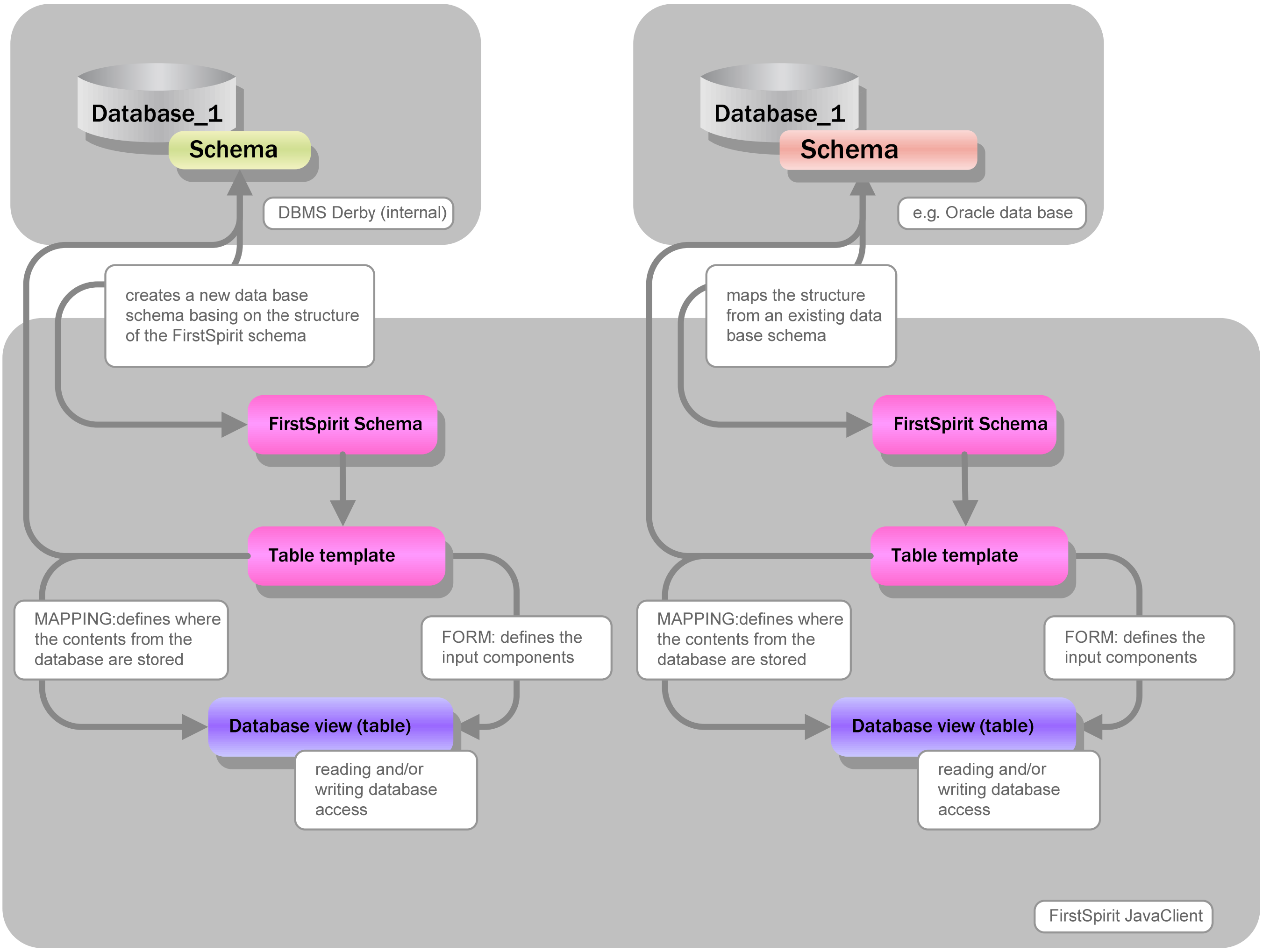

Database_1
Schema
DBMS Derby (internal)
creates a new data base schema basing on the structure of the FirstSpirit schema
FirstSpirit Schema
Table template
MAPPING:defines where the contents from the database are stored
FORM: defines the input components
Database view (table)
reading and/or writing database access
Database_1
Schema
e.g. Oracle data base
maps the structure from an existing data base schema
FirstSpirit Schema
Table template
MAPPING:defines where the contents from the database are stored
FORM: defines the input components
Database view (table)
reading and/or writing database access
FirstSpirit JavaClient

Figure 3-5: Concept – Schemes, table templates, views of databases

FirstSpirit schema: The FirstSpirit JavaClient can be used either to create a new, empty database schema (content schema) (see Chapter 2.2.1.9 page 31) or to create a database schema (content schema) from an existing database (see Chapter 2.2.1.10 page 35).

After a new schema has been created, the graphic editor in the FirstSpirit JavaClient can be used to create the tables required in the selected database and to relate them to each other (see Chapter 2.12.1 page 126). The columns which are to be subsequently entered by the editor must be given for each table. A column with the necessary primary key is automatically generated when the table is created.

Instead of generating an empty schema node (cf. Chapter 2.2.1.9 page 31), a new schema node can also be created in the FirstSpirit project on the basis of the existing tables and relationships of a database (see Chapter 2.2.1.10 page 35).

Depending on the settings of the project administrator for the configured database, the changes within a schema in the JavaClient, for example, inserting a table, can be accepted in the physical database ("Sync") or can be prevented ("no Sync").

Table template: A table template can be created (below the schema node) for each table modelled within the schema. These table templates are used to define the input components via which the editor can subsequently enter data in the corresponding tables and via which input element the editor can accept data of a reference table (see Chapter 2.12.4 page 133)). The "Mapping" tab can also be used to assign the content entered via the input component to a database table of the physical database (see Chapter 2.12.5 page 133). The mapping therefore defines the storage location of the content in the database. The template sets tab can be used to define the appearance of the data records for generation in the individual output channels (see Chapter 2.12.6 page 135).

Queries: In addition, queries can be created for each database schema (see Chapter 2.12.8 page 136). Restrictions are made in these queries, which are then used to evaluate the result table. The restrictions made are then taken into account when the data records of a table are output.

View of a database: Within the Content Store of FirstSpirit, the editors work on a "view" of the database. To this end, a table is created with a link to the database table. The data in this table is displayed in tabular form. Depending on the project administrator's settings for the configured database, the editors can either access the database content with read access only and output this sorted on a page, for example as the result of a query ("Content Projection"), or also access with write access and therefore insert new content in the database. If write access is allowed, new data records can be inserted or existing data records can be changed. To do this, the input elements defined in the table template are available to the editor (see Chapter 2.12.4 page 133).

3.6 Template update on data (content) sources

The "Template update" function can be used to exchange templates between FirstSpirit projects (see Chapter 2.3.2 and Chapter 2.3.3). The template update is capable of both updating the templates and of creating new templates. FirstSpirit schemata are handled as complete, self-contained template objects. This means that, apart from the schema definition, all table templates and queries are also included in the update of a FirstSpirit schema. It is not possible to update individual parts of these template objects only.

When an existing FirstSpirit schema is updated the template update adopts the layer definition of the existing FirstSpirit schema. However, if the template update creates a new FirstSpirit schema, the layer definition of the original project is adopted. This is the required behaviour in rare cases only.

! *We recommend using the export/import function in the context menu to transfer FirstSpirit schemata from one project into another. In this case you can explicitly select which layer should be used. In addition, there is also the option of transferring the data stored in the data source (see Chapter 2.3.4.6 page 61 .*

4 Workflows

A workflow is a sequence of tasks which are worked through in a fixed, specified structure. Both due dates and authorised groups of people can be defined for the respective tasks. Workflows integrated in FirstSpirit are the task setting and the release request.

With the help of a graphic workflow editor, project-specific workflows can be created in the Template Store (see Chapter 4.2 page 160).

Instances of these workflows can then be started, context-bound, on each element within the FirstSpirit project or without context via the FirstSpirit menu bar. Each instance of a workflow must be run through according to the rules defined in the workflow.

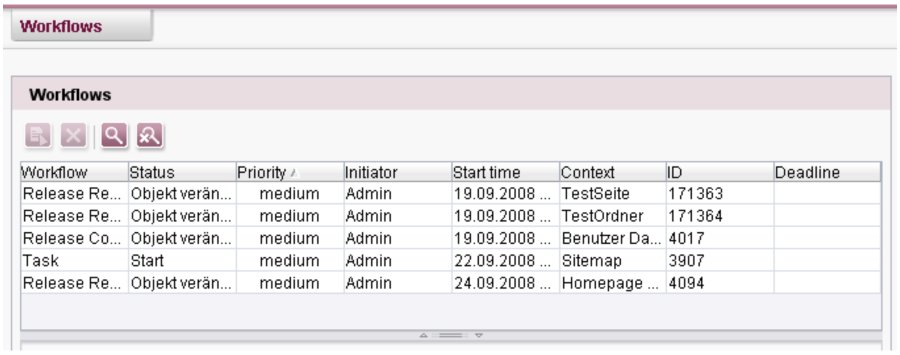
The "Workflows" root node in the Template Store contains an overview of all open or already closed workflows (instances) within the project (see Chapter 4.1 page 152), whereby a filtered view is also possible depending on various search criteria (see Chapter 4.1.1 page 154). Tasks can be edited (see Chapter 4.1.2 page 156) and closed again (see Chapter 4.1.3 page 159) within the overview.

For further information on starting and passing on workflows, see "FirstSpirit Manual for Editors", Chapter 12 "Workflows in the FirstSpirit JavaClient“ and "FirstSpirit Manual for Editors (WebClient)".

4.1 Overview

An overview of all open or already closed workflows (instances) within the project is displayed on the "Workflows" root node in the Template Store.

✓ 4.1 This view has been enhanced to include a search function. Whereas before it was only possible to display all workflows or tasks on the "Workflows" root nodes in the Template Store, a filtered view is now possible too (by workflow, element ID, user, modifier, etc.) (see Chapter 4.1.1 page 154). (This function is not released until FirstSpirit Version 4.1.)



Workflows

Workflow	Status	Priority	Initiator	Start time	Context	ID	Deadline
Release Re...	Objekt verän...	medium	Admin	19.09.2008 ...	TestSeite	171363	
Release Re...	Objekt verän...	medium	Admin	19.09.2008 ...	TestOrdner	171364	
Release Co...	Objekt verän...	medium	Admin	19.09.2008 ...	Benutzer Da...	4017	
Task	Start	medium	Admin	22.09.2008 ...	Sitemap	3907	
Release Re...	Objekt verän...	medium	Admin	24.09.2008 ...	Homepage ...	4094	

Figure 4-1: Workflows overview

Click the icon to open the task list for editing the task (see Chapter 4.1.2 page 156).

Click the icon to close the selected task (see Chapter 4.1.3 page 159).

Click the icon to open the "Search for tasks" dialog for defining a task search filter (see Chapter 4.1.1 page 154). ✓ 4.1

Click the icon to cancel the filtered display and to display the whole view instead (see Chapter 4.1.1 page 154) ✓ 4.1 .

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

Workflow: Name of the workflow which has been started.

Status: Status of the current instance of the workflow.

Priority: The current priority defined for editing or dealing with the task.

Initiator: Login name of the modifier who started the workflow.

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

Context: If the workflow was started on an element, for example a page or a medium, this element is displayed. Double-click the row to switch the context directly to the corresponding element in the tree view.

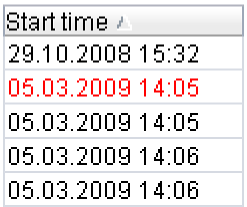
ID: If the workflow was started on an element, for example a page or a medium, the ID of the element is displayed. Double-click the row to switch the context directly to the corresponding element in the tree view.

Deadline: If the current task has been scheduled with a fixed date (deadline), the deadline is displayed here.

If a (context-dependent) task within the table is double-clicked, the focus in the JavaClient changes directly to the element in the tree view on which the task was started.

Multiple selections can be made within the table by simultaneously pressing the SHIFT and CTRL key.

Sort by column content: Click the respective column heading to change the way in which the entries are sorted in the table. The first time any column heading is clicked the entries are sorted in ascending order, if it is clicked again they are sorted in descending order (according to column content). The sort is indicated by an ▲ icon behind the column heading:



Start time ▲
29.10.2008 15:32
05.03.2009 14:05
05.03.2009 14:05
05.03.2009 14:06
05.03.2009 14:06

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

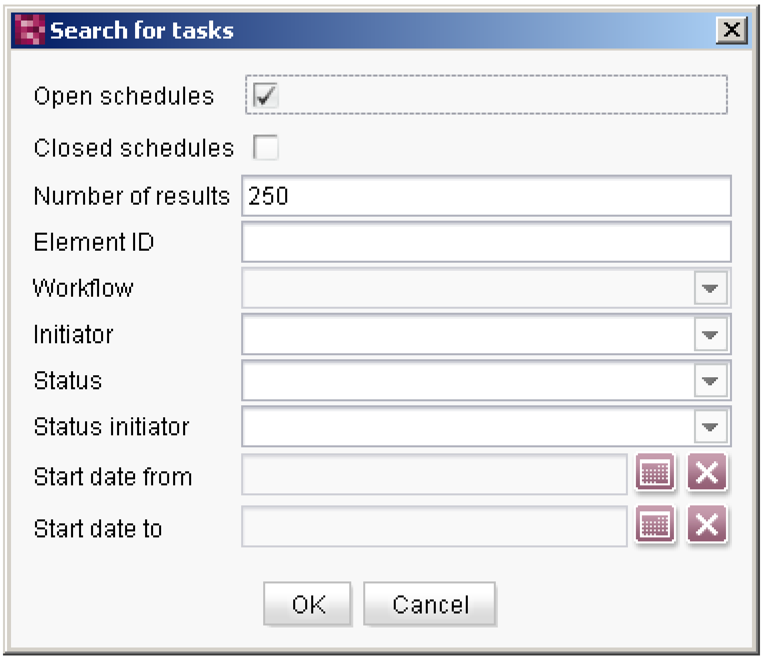
A third click cancels the sorting.

4.1.1 Search for tasks (filtered overview) (from V4.1)

This function is released for FirstSpirit Version 4.1 and higher only.

The search function of workflows in the FirstSpirit JavaClient has been enhanced. Whereas before it was only possible to display all workflows or tasks on the "Workflows" root nodes in the Template Store, a filtered view is now possible too (by workflow, element ID, user, modifier, etc.).

The different filter options can be set using the "Search for tasks" dialog. The dialog is opened by clicking the 🔍 icon:



Search for tasks

Open schedules ☑

Closed schedules □

Number of results 250

Element ID

Workflow

Initiator

Status

Status initiator

Start date from

Start date to

Figure 4-3: "Search for task" dialog for filtering the view

Open schedules: All "open tasks" are displayed. Open tasks are all those which have not yet reached the end status (of the workflow).

Closed schedules: All "closed tasks" are displayed. Closed tasks are all those which have reached the end status (of the workflow).

Number of results: The number of tasks found which match the filter criteria entered can be limited to a maximum number of results. If more results match the search criteria than allowed as a maximum, the most up to date results are no longer shown.

Element ID: Unique ID of the object on which the workflow was started. If it is a context-less workflow, an empty field is displayed.

Workflow: Name of the workflow which has been started. Depending on the "Preferred display language" in the "Extras" menu, either the unique reference name or the language-dependent display name of the workflow is displayed.

Initiator: Login name of the modifier who started the workflow. The search for substrings is supported. This means the search result does not have to exactly match the search term. Instead, all results which contain the search term are displayed.

Status: Status of the current instance of the workflow. The search for substrings is supported. This means the search result does not have to exactly match the search term. Instead, all results which contain the search term are displayed.

Status initiator: Login name of the modifier who switched the current instance of the workflow to its current status. The search for substrings is supported. This means the search result does not have to exactly match the search term. Instead, all results which contain the search term are displayed.

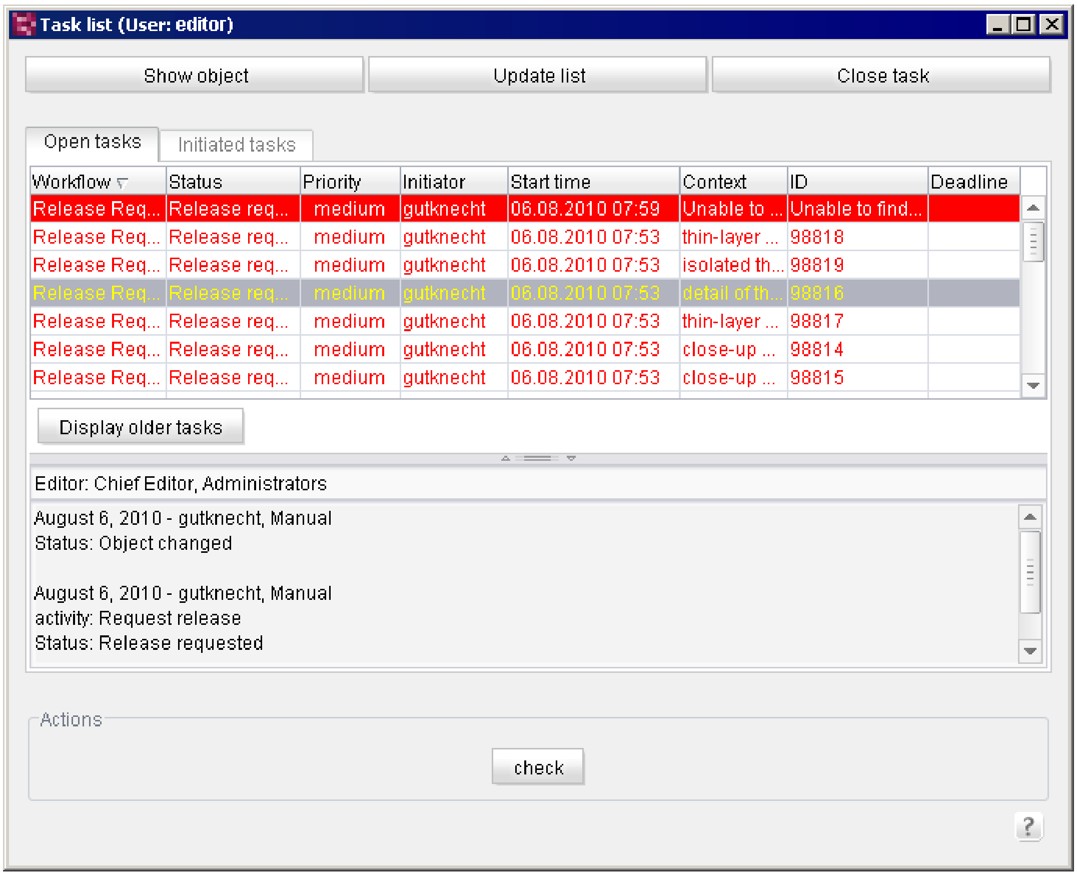
Start date from: The 📅 icon can be used to open the date selection component. A date for the search start date can be entered here. The date on which a workflow was started is always decisive.
If a start date only is entered, all workflows with the currently given date are sought.

Start date to: The 📅 icon can be used to open the date selection component. A date for the search end date can be entered here. The date on which a workflow was started is always decisive.

OK Click this button to filter the tasks according to the criteria entered. Click the icon to cancel the filtered display and to display the whole view instead.

4.1.2 Edit tasks

Click the icon in the "Workflows overview" dialog (see Figure 4-1) to open the task list. The task selected within the overview is also marked in the tasks list. Provided the user has the necessary permissions to switch the workflow, the transitions in the bottom part of the task list are displayed directly.



Workflow	Status	Priority	Initiator	Start time	Context	ID	Deadline
Release Req...	Release req...	medium	gutknecht	06.08.2010 07:59	Unable to ...	Unable to find...	
Release Req...	Release req...	medium	gutknecht	06.08.2010 07:53	thin-layer ...	98818	
Release Req...	Release req...	medium	gutknecht	06.08.2010 07:53	isolated th...	98819	
Release Req...	Release req...	medium	gutknecht	06.08.2010 07:53	detail of th...	98816	
Release Req...	Release req...	medium	gutknecht	06.08.2010 07:53	thin-layer ...	98817	
Release Req...	Release req...	medium	gutknecht	06.08.2010 07:53	close-up ...	98814	
Release Req...	Release req...	medium	gutknecht	06.08.2010 07:53	close-up ...	98815	

Editor: Chief Editor, Administrators

August 6, 2010 - gutknecht, Manual
Status: Object changed

August 6, 2010 - gutknecht, Manual
activity: Request release
Status: Release requested

Figure 4-4: Task list

For performance reasons, from 4.2R4, only 25 tasks are initially displayed in the task list (Ctrl + T or "Tasks" menu / "Tasks list") on the "Open Tasks" and "Initiated Tasks" tabs. If more tasks exist, they can be shown using the **Display older tasks** button.

Coloured markings indicate for example if the logged-in user is selected as editor of the task directly or because of his group membership (red lettering), if the logged-in user is not selected as editor (black lettering) or if it is an invalid task (red background colour).

Invalid tasks, e.g. produced because an object on which a workflow is active is deleted, are visualised with a red background in the tasks list from 4.2R4. These cannot be set to the next status, they can only be closed using the "Close task" button. If the task can be repaired, e.g. the deleted object for which the workflow still exists is restored, the "Repair task" button appears. This is used to reset the task, the status colour and write protection.

Multiple selections can be made by simultaneously pressing the SHIFT and the CTRL key (all tasks can be selected using the key combination CTRL + A). If several tasks within the list are selected, they can be passed on together in one step (see Chapter 4.11.4 page 238).

The task list can also be opened using the "Tasks" menu or by clicking the 📋 icon in the FirstSpirit toolbar.

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

4.1.3 Close tasks

Under certain conditions, it can be necessary to close an open task although the end status has not yet been reached. A task can be closed by clicking the ☒ icon in the "Workflows overview" dialog (see Figure 4-1).

The function corresponds to the "Close task" button available within the task list.

Multiple selections can be made by simultaneously pressing the SHIFT and the CTRL key (all tasks can be selected using the key combination CTRL + A). If several tasks within the list are selected, they can be deleted together in one step.

Before the tasks are deleted a confirmation prompt appears.

! *Closed tasks cannot be restored.*

4.2 Modelling workflows

4.2.1 Create a workflow

New, project-specific workflows are created using the context menu on the "workflows" root node in the Template Store or on a folder within this node. Click the "Create new workflow" entry to create a new workflow within the tree view.

A graphic editor for modelling a new workflow opens in the right-hand side of the Editing window. As a default, a start state is displayed there with a transition to the first activity of the workflow and an end state.

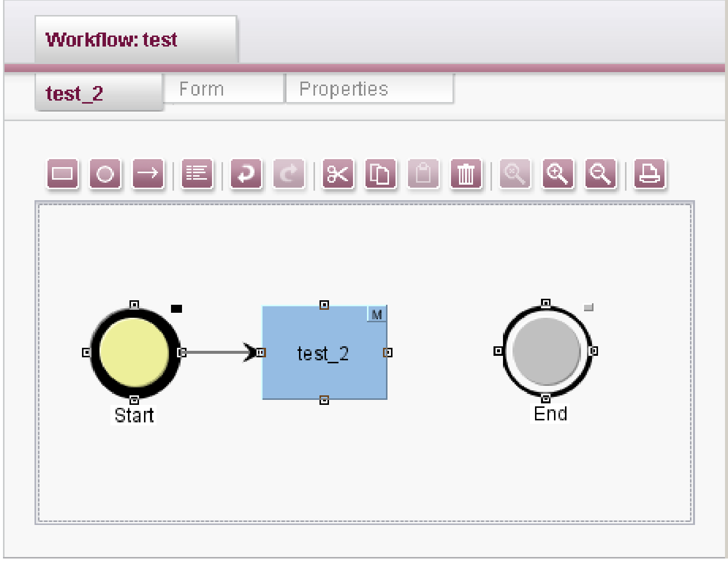


Figure 4-5: Initial status after creating a new workflow

The workflow can now be modelled within the editor by adding further statuses, activities and transitions (see Chapter 4.2.3 ff.).

Each workflow must begin with a start state and end with an end state.

The editor is used either via the toolbar (see Chapter 4.2.2 page 161) or via a context menu, which can be activated in any position of the editor.

4.2.2 Toolbar of the workflow editor

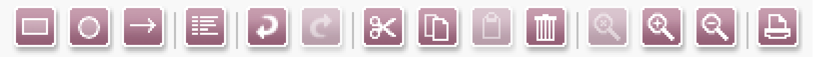


Figure 4-6: Toolbar of the workflow editor

Creates new activity (A)

Create new activity: A new activity is created by clicking the icon (or the keyboard shortcut A) in the editor (see Chapter 4.2.3.2 page 163).

Creates new status (S)

Create new status: A new status is created by clicking the icon (or the keyboard shortcut S) in the editor (see Chapter 4.2.3.1 page 162).

Creates new transition (T)

A new transition is created by clicking the icon (or the keyboard shortcut T) in the editor (see Chapter 4.2.3.3 page 163).

Change properties, click this icon to open the properties window of the activated workflow element.

Cut element, click this icon to cut all the selected elements of the workflow editor and copy it into the temporary memory. (Several elements can be selected by dragging a frame with the mouse.)

Copy element, click this icon to copy all the selected elements of the workflow editor into the temporary memory.

Paste element, click this icon to paste the elements copied into the temporary memory into the workflow editor.

Delete, this icon can be used to remove an element from the workflow process.

Zoom 1:1; this icon can be used to display all the elements of the workflow editor in their original size again.

Zoom in; this icon can be used for enlarged display of the workflow editor's elements.

Zoom out; this icon can be used for reduced size display of the workflow editor's elements.

Print, this icon (or the keyboard shortcut Ctrl + P) can be used to print out the workflow graphic. A window opens for the print settings (see Chapter 4.2.8 page 168).

4.2.3 Elements of the graphic workflow editor

Three different object types are available within the editor, with which the new workflows can be modelled and configured:

▪ States or status	(see Chapter 4.2.3.1 page 162).
▪ Activities	(see Chapter 4.2.3.2 page 163).
▪ Transitions	(see Chapter 4.2.3.3 page 163).

4.2.3.1 State / Status

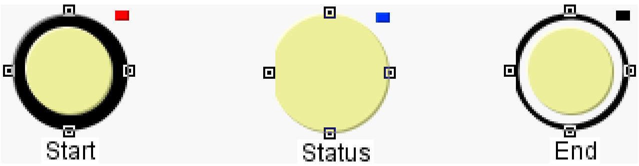


Figure 4-7: Status (states) in the workflow editor

States, also called status, are represented by circles. A state is the result of an (automatic or manual) activity. States indicate the status a workflow can currently have (i.e. the state it is in).

Creates new status (S)

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

- a start state (has outgoing transitions only),
- an end state (has incoming transitions only)
- or a normal status (has incoming and outgoing transitions).

The display of the different types is highlighted in the editor by a dark border (in start and end status) (see Figure 4-7).

4.2.3.2 Activity

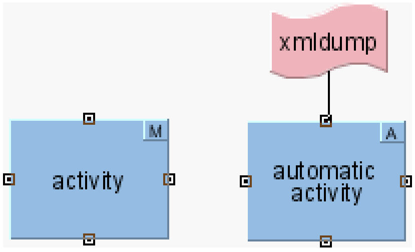


Figure 4-8: Activities in the workflow editor

Activities are represented by rectangles. An activity consists of the performance of a task (e.g. "Check") and the initiation of an action (e.g. clicking the "Direct release" button).

An activity can either be executed manually by a user, or automatically by a script (see Chapter 4.5.4 page 184).

Manual activities are identified in the editor by an "M" in the top right-hand corner (see Figure 4-8 – left-hand activity), automatic activities are identified in the editor by an "A" in the top right-hand corner (see Figure 4-8 – right-hand activity).

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

- Manually (by a modifier)
- Automatically (by a script).

4.2.3.3 Transition

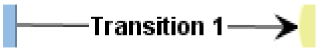


Figure 4-9: Transition in the workflow editor

Transitions are represented by arrows. Transitions form the connection between an activity and a status. The permissions for a workflow model are defined here. Cancelling an action results in the previous status (before connecting the transition). The cancellation does not have to be modelled separately in the workflow.

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

4.2.4 Keyboard shortcuts in the Workflow Editor

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

4.2.5 Editor accessibility features

An element is selected by simply clicking it within the editor. This element can be moved to the required place in the editor by left-clicking and keeping the mouse button pressed. Incoming and outgoing transitions follow the moved element.

The mouse can be used to drag a border around several elements. In this way, several elements can be moved, cut or copied simultaneously.

If a status is selected in the workflow editor, the **New activity** function causes the new activity to be automatically connected to this status by a transition. Analogous to this, if an activity is selected, the **New status** function creates a transition between the activity and the new status.

Transitions are automatically always straight connections from the source to the target. Control points (knots) can be inserted on a transition, above all to make the representation of loops clearer. The connection between two control points is a straight line, but any number of control points can be inserted.

To insert a control point, the transition must be right-clicked at the required position. If a control point is right-clicked, this control point is removed again.

A control point can be moved to the required position within the editor with a left-click and by keeping the mouse button pressed.

4.2.6 Modelling rules

- Each workflow has exactly one *start state*.
- The start state can follow *exactly one outgoing transition*. As a selection cannot take place in the start state, the first outgoing transition is always taken into consideration.
- Transitions represent a targeted (directional) connection between exactly one source and exactly one target element.
- The source and target element of a transition can only be states and activities, not other transitions.
- Transitions can always only exist between *one* state and *one* activity, never between two states or two activities.
- States and activities can have any number of incoming and outgoing transitions (exceptions: start state and end state).
- States and activities should always have one unique (for the workflow) name.
- Transitions *may* have a name, this *must* then be unique in relation to its start element.
- Each workflow as a fixed, defined set of end states; at least one *end state* must be defined.
- An end state may not have *any outgoing* transitions.

4.2.7 Examples of modelling rules

- A state is always followed by an activity. The state and activity are connected by "transitions" and require a unique name. A name can be allocated for transitions, it must then be unique in relation to its start element.

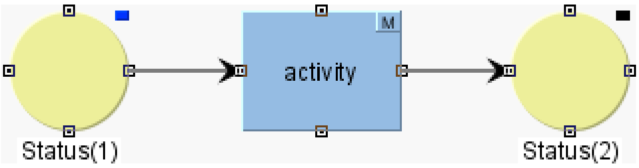
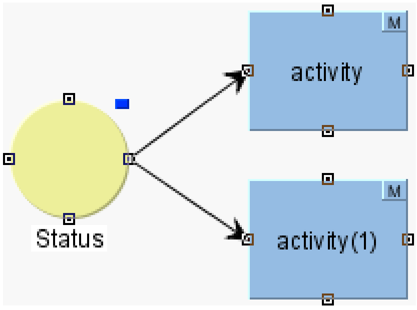


Figure 4-10: Status and activities modelling rule

- Several activities can take place in a state. Equally, several activities can lead to a state.



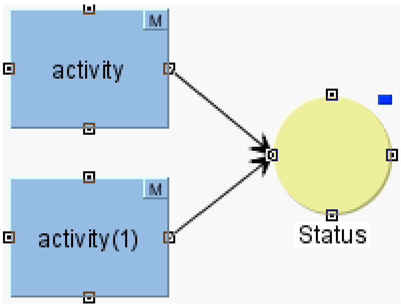
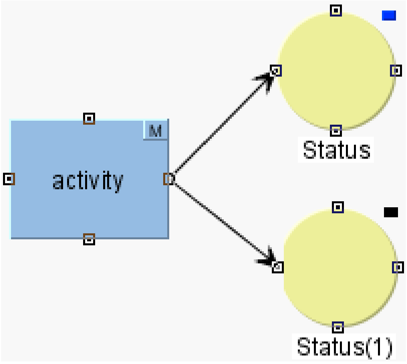


Figure 4-11: One status, several activities modelling rule

- An activity can lead to several states. Equally, several states can initiate one activity.



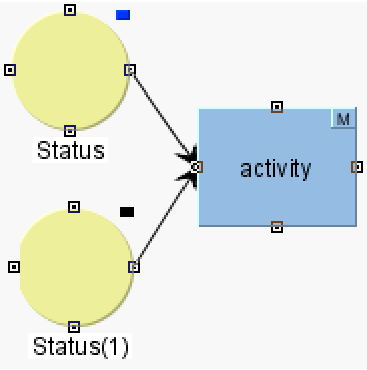


Figure 4-12: Several statuses, one activity modelling rule

- A script can only be attached to one (automatic) activity, whereby the connection line is not directional.

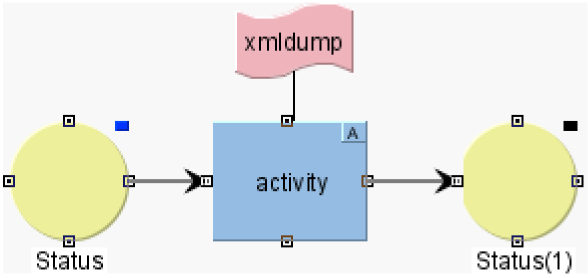


Figure 4-13: Activities and scripts modelling rule

4.2.8 Print preview for workflow models

Print A print preview of the modelled workflow can be requested within the workflow editor by pressing the Print button (or using the keyboard shortcut Ctrl + P) (see Chapter 4.2.2 page 161). A window opens for the print settings.

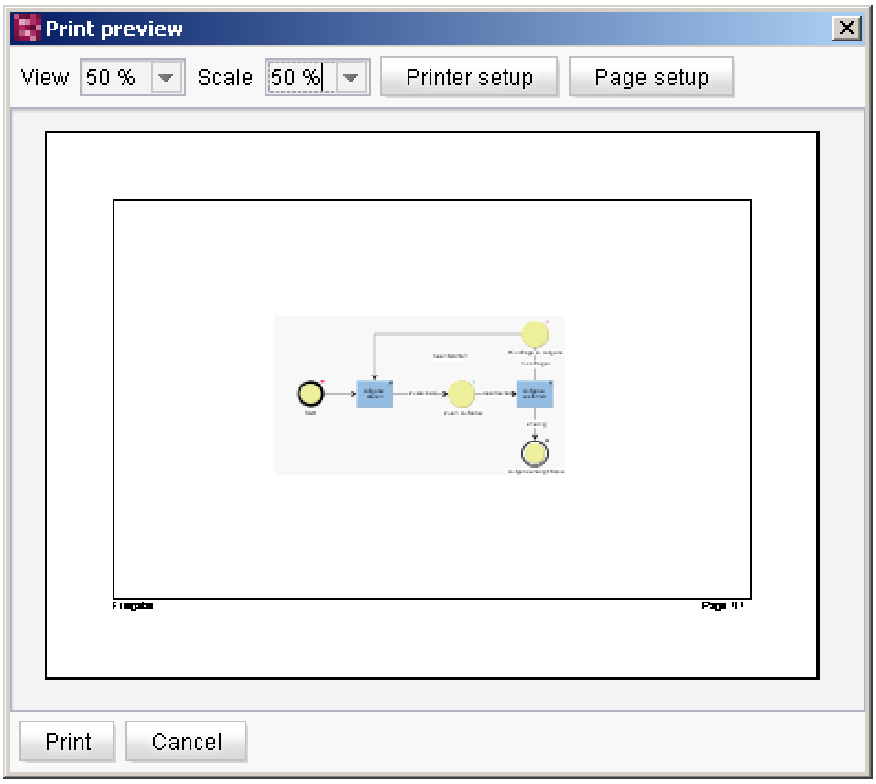

View 50 %
Scale 50 %

Figure 4-14: Print preview

View: The combobox can be used to adjust the percentage size of the pages in the preview window.

Scale: The combobox can be used to select the percentage size of the workflow model on the print preview page. If the display is large (does not fit on one page), several preview pages are displayed.

Printer setup: Click this button to open a window in which the print settings can be made.

Page setup: Click this button to open a window in which several settings can be made for the printed pages.

Print: Click this button to start printing.

Click this button to cancel printing.

4.3 Error handling within workflows

4.3.1 General error handling

<u>On starting:</u> If an exception occurs when the workflow is started, for example, because the user does not have permission to connect the transitions of a workflow, the workflow is not started on the object.

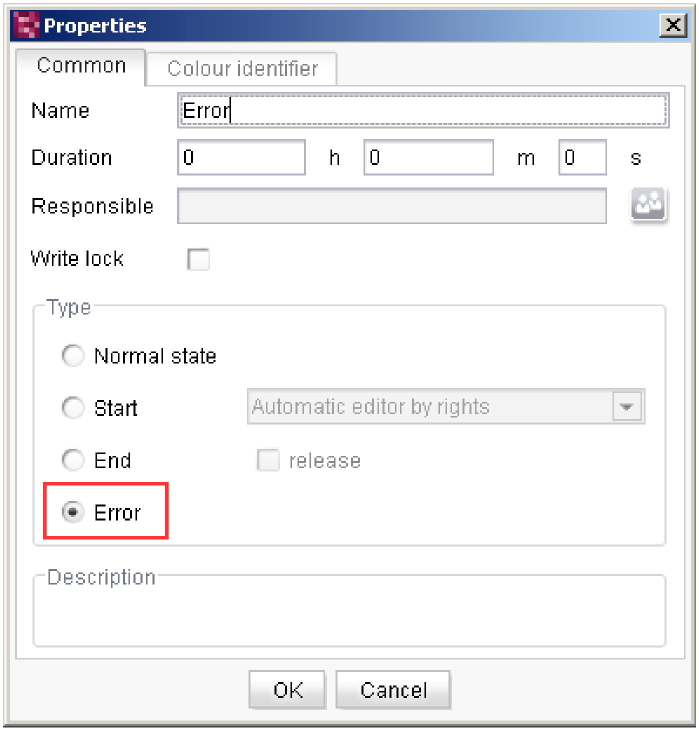
<u>On executing transitions:</u> A different situation exists if the workflow has already been started and an error or an exception occurs while a transition is being executed. In this case, the status before executing the transition, i.e. the last "error-free" status is retained. If an error status is defined within the workflow model, the element is in error status after the exception occurs (see Chapter 4.3.2 page 169).

4.3.2 Error status (from V4.1)

This function is released for FirstSpirit Version 4.1 and higher only. Screenshots are therefore displayed in the new "LightGray" Look & Feel. The display in the "Classic" Look & Feel can differ slightly.

There are many reasons for the occurrence of exceptions while executing a workflow, for example an incorrect configuration in the workflow's model or a script error in an appended script. An optional error station is available within the modelling of workflows to reliably intercept these errors and to prevent an instance of the workflow from being in an inconsistent state after connecting a transition.

To this end a normal status is simply added to the model. The type "Error" must now be enabled in the "Properties" dialog of the status:



Properties

Name: Error

Duration: 0 h 0 m 0 s

Responsible:

Write lock: □

Type

○ Normal state

○ Start Automatic editor by rights

○ End □ release

● Error

Description

Figure 4-15: Configure error status

The status is then indicated in the model by a red boundary:

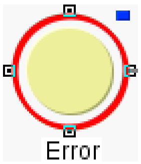


Figure 4-16: Error status in the model

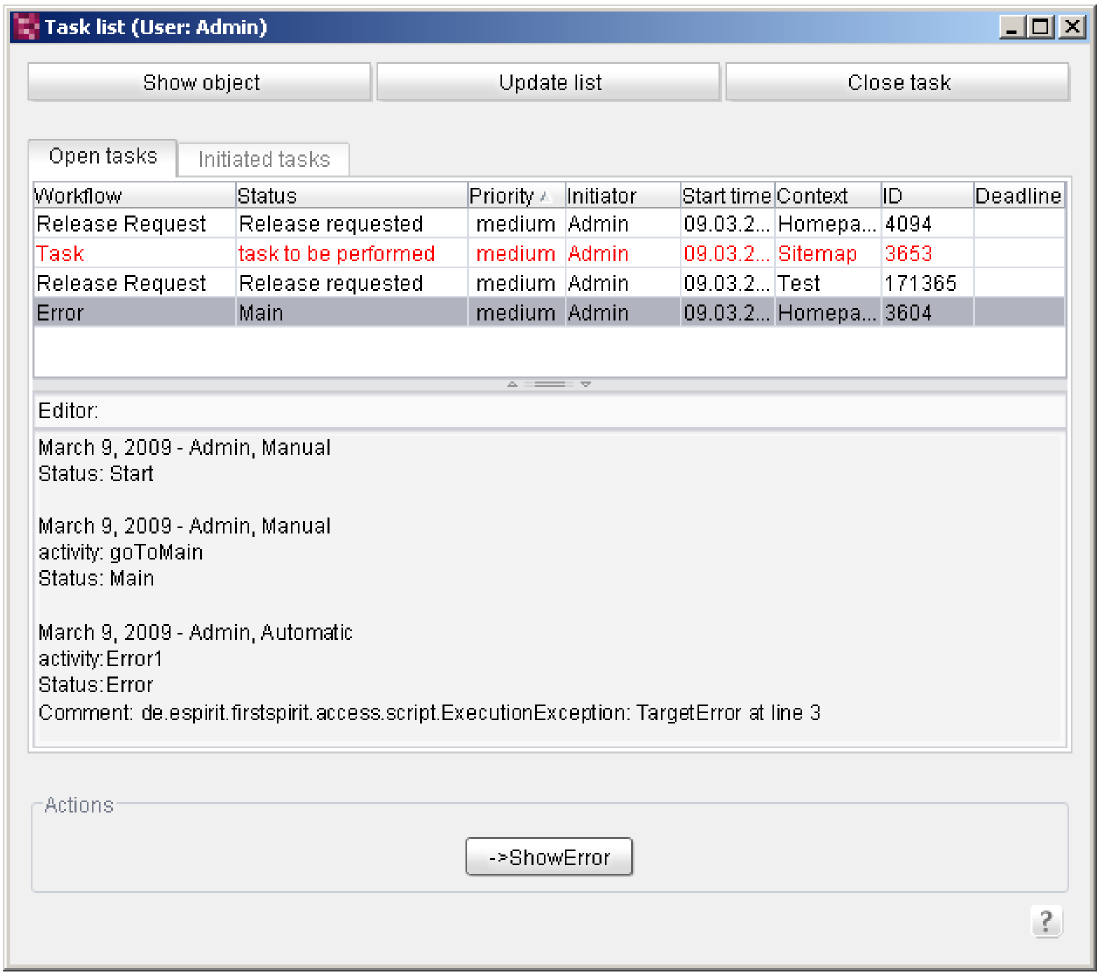
The error status may not be switched via a transition, i.e. analogous to the start status it has outgoing transitions only. Error handling within the workflow is modelled via these outgoing transitions (see Figure 4-18).

If an exception now occurs at any point within the workflow, the workflow's instance immediately reaches the error status.

The error status intercepts all exceptions which occur within the implementation of the workflow, including those which are not handled within the workflow. Examples of handled and unhandled exceptions are described in Chapter 4.3.3.

After the error is resolved the workflow can be forwarded to the following status (according to the workflow model).

The task list provides an overview of all instances of the workflow which have been incorrectly executed:



Workflow	Status	Priority	Initiator	Start time	Context	ID	Deadline
Release Request	Release requested	medium	Admin	09.03.2...	Homepa...	4094	
Task	task to be performed	medium	Admin	09.03.2...	Sitemap	3653	
Release Request	Release requested	medium	Admin	09.03.2...	Test	171365	
Error	Main	medium	Admin	09.03.2...	Homepa...	3604	

Editor:

March 9, 2009 - Admin, Manual
Status: Start

March 9, 2009 - Admin, Manual
activity: goToMain
Status: Main

March 9, 2009 - Admin, Automatic
activity:Error1
Status:Error
Comment: de.espirit.firstspirit.access.script.ExecutionException: TargetError at line 3

Figure 4-17: Task list showing tasks with error status

The tasks can be sorted by their current status by clicking the table heading "Status".

Each workflow can only have one error status. If a status is defined as an error status, although an error status already exists in the workflow model, the first status is automatically reset to "normal" type.

4.3.3 Example: Workflow "Error" (from V4.1)

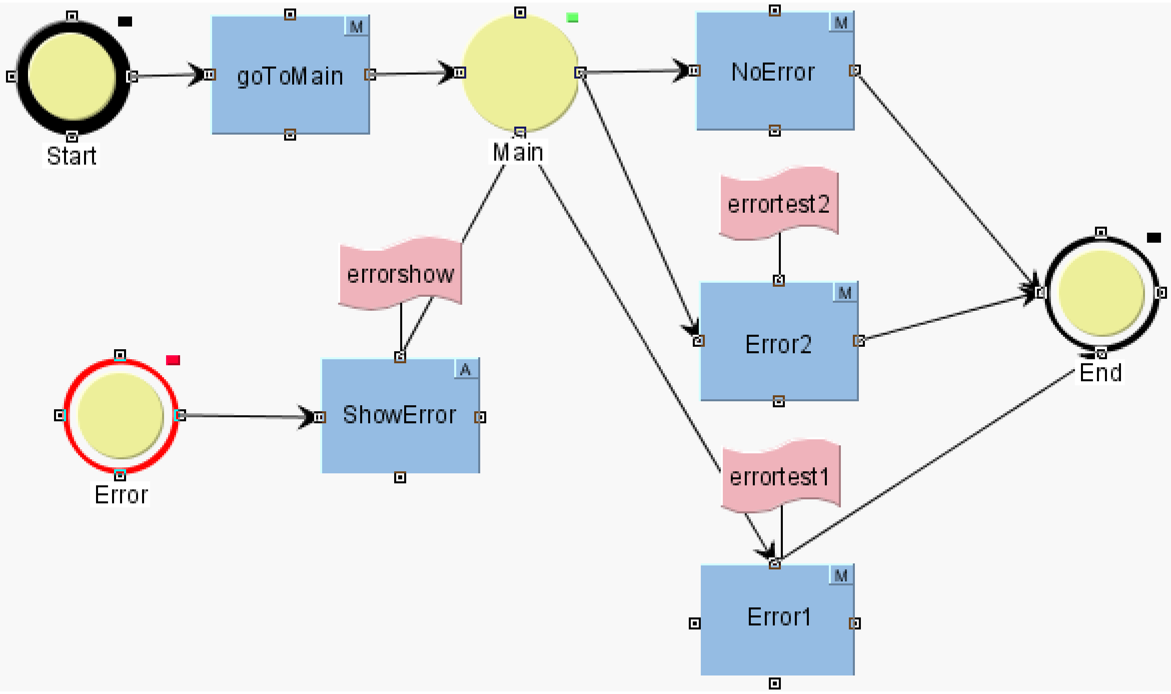


Figure 4-18: Example of workflow "Error"

The workflow consists of the "errortest" workflow and the corresponding scripts: "errorshow", "errortest1" and "errortest2". The workflow is made available for importing into the Template Store ("Workflows" node) in the form of a compressed Zip file.

Script: errortest1:

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

The first script "errortest1" throws an unhandled IllegalStateException. This exception is not handled in the workflow; the result is "error" status instead of "end" status.

Script errortest2:

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

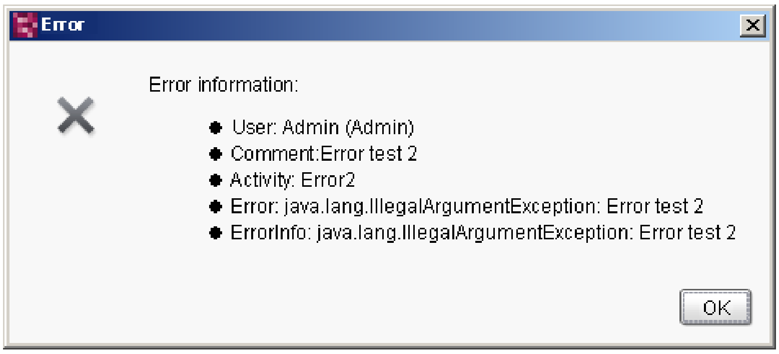
The second script "errortest2" shows the error handling within a script. If an exception occurs, the instance of a workflow is switched directly to error status via `context.gotoErrorState(...)`.

Script errorshow:

```
import de.espirit.firstspirit.common.gui.*;
import de.espirit.firstspirit.access.*;

errorInfo = context.getTask().getErrorInfo();

if (errorInfo != null) {
text = new StringBuilder("<html>Error information:<br>");
text.append("<ul>");
text.append("<li>User: " + errorInfo.getUserLogin() + " (" +
errorInfo.getUserName() + ")");
text.append("<li>Comment " + errorInfo.getComment());
text.append("<li>Activity: " + errorInfo.getErrorActivity());
text.append("<li>Error: " + errorInfo.getThrowable());
text.append("<li>ErrorInfo: " + errorInfo.getErrorInfo());
text.append("</ul>");
CMSDialog.showErrorDialog(text.toString());
} else {
CMSDialog.showInfoDialog("No error information available.");
}
context.doTransition("->Main");
```

An unpublished (untested) API function ("CMSDialog") is used within the example. But the call can be replaced by the Java Swing class `JOptionPane`, e.g.:
`JOptionPane.showMessageDialog(null,"Hello " + userName);`

The "errorshow" script shows the error information via an error dialog. If an error occurs, the dialog is automatically opened by the workflow within the scope of the error handling. The dialog contains relevant information for correcting the error (e.g. the user who started the workflow, the activity which led to the error, etc.):



Error information:

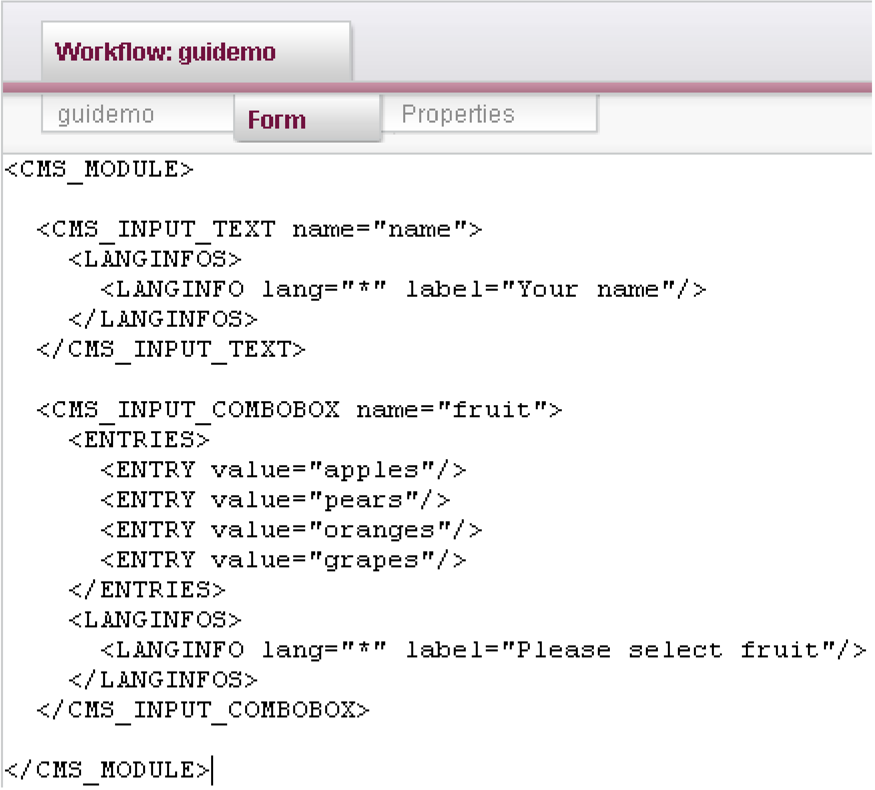
- User: Admin (Admin)
- Comment:Error test 2
- Activity: Error2
- Error: java.lang.IllegalArgumentException: Error test 2
- ErrorInfo: java.lang.IllegalArgumentException: Error test 2

Figure 4-19: Dialog with relevant error information

After showing the dialog, the workflow instance is automatically switched back to "main" status:

4.4 Form support for workflows (form)

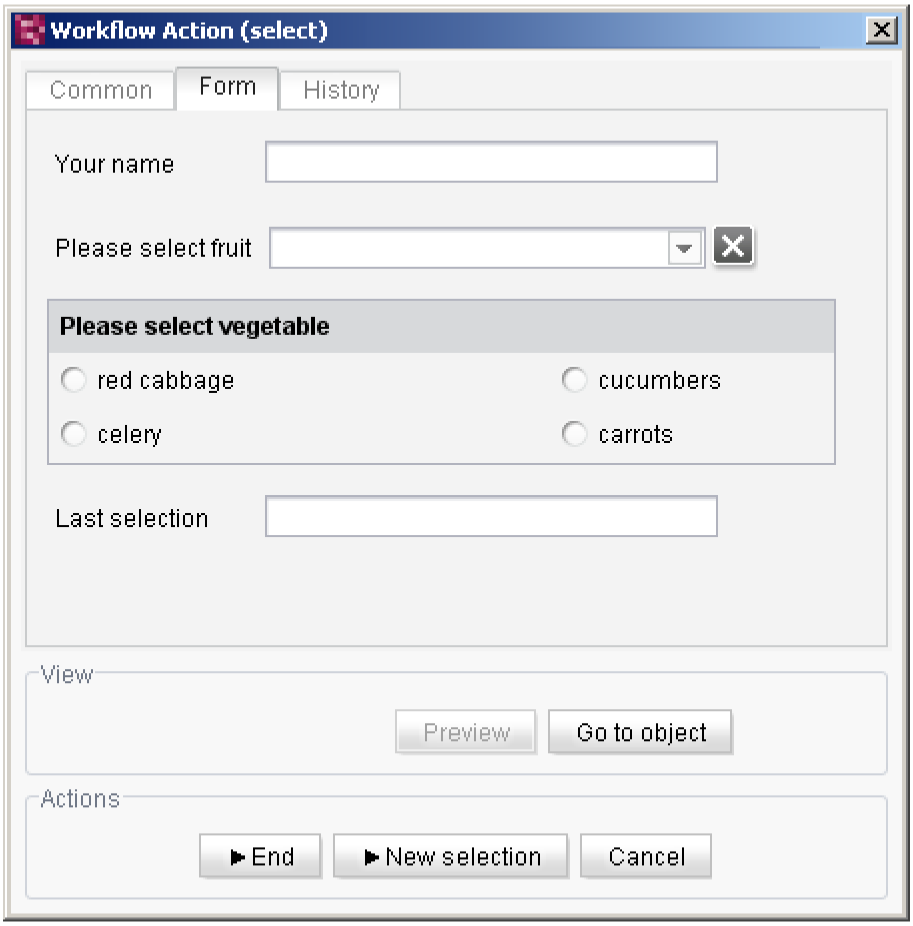
Forms can be used within workflows to enter content. The forms are defined in the workflow within the "Form" tab:



```
<CMS_MODULE>

  <CMS_INPUT_TEXT name="name">
    <LANGINFOS>
      <LANGINFO lang="*" label="Your name"/>
    </LANGINFOS>
  </CMS_INPUT_TEXT>

  <CMS_INPUT_COMBOBOX name="fruit">
    <ENTRIES>
      <ENTRY value="apples"/>
      <ENTRY value="pears"/>
      <ENTRY value="oranges"/>
      <ENTRY value="grapes"/>
    </ENTRIES>
    <LANGINFOS>
      <LANGINFO lang="*" label="Please select fruit"/>
    </LANGINFOS>
  </CMS_INPUT_COMBOBOX>

</CMS_MODULE>
```

Figure 4-20: Form tab (workflow model)

During the execution of the workflow, the modifier can enter values using the input components defined in the form area:



Workflow Action (select)

Your name

Please select fruit

Please select vegetable

○ red cabbage

○ cucumbers

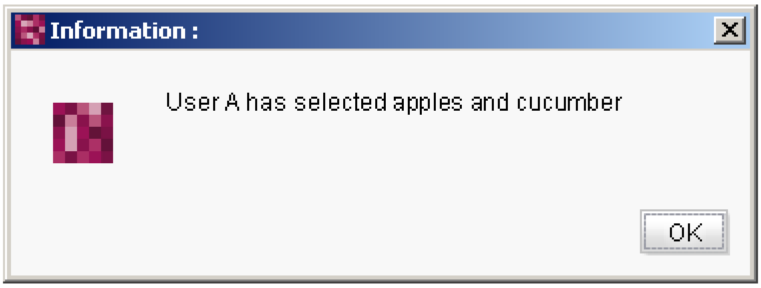
○ celery

○ carrots

Last selection

Figure 4-21: Form within the execution

The stored values can be output again within the workflow at a later date:



Information :

User A has selected apples and cucumber

Figure 4-22: Information dialog with form content

4.4.1 Example: Workflow "GUI"

A "guitest" script is executed on the activity to display the forms within the workflow example.

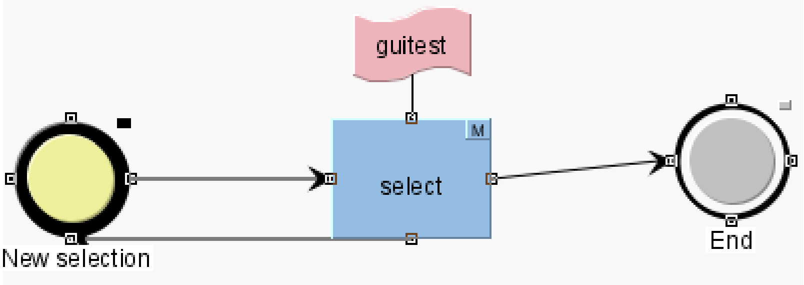


Figure 4-23: "GUI" workflow example

Script "guitest":

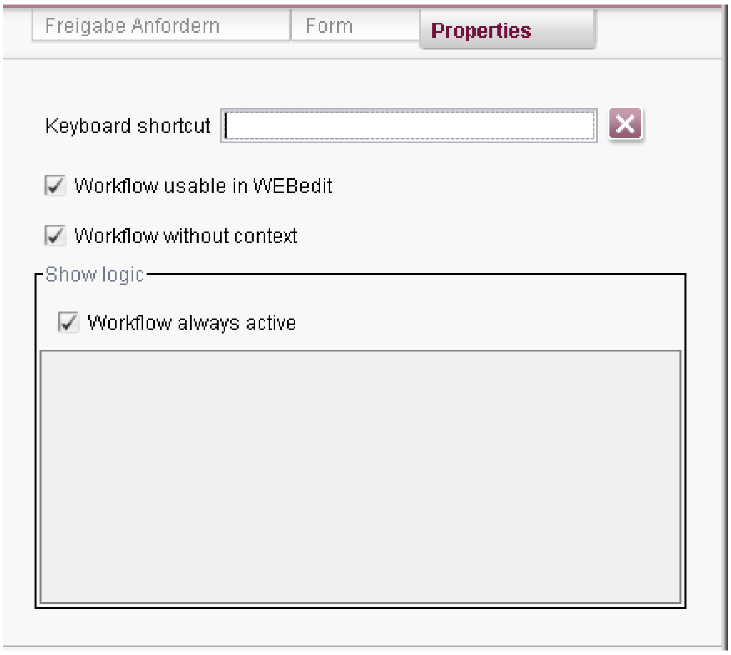
```
//!Beanshell
import de.espirit.firstspirit.common.gui.*;
import de.espirit.firstspirit.access.editor.*;

se = context.getStoreElement();
context.showActionDialog();
transition = context.getTransitionParameters();
data = context.getData();
if (transition.getTransition() != null) {
           // display selected values
name = data.get("name").getEditor().get(EditorValue.SOLE_LANGUAGE);
fruit = data.get("fruit").getEditor().get(EditorValue.SOLE_LANGUAGE);
vegetable =
data.get("vegetable").getEditor().get(EditorValue.SOLE_LANGUAGE);
           // save selected values
lastSelection = data.get("lastSelection").getEditor();
lastSelection.set(EditorValue.SOLE_LANGUAGE, name + ", " + fruit + ", " +
vegetable);
CMSDialog.showInfoDialog(name + " has selected " + fruit + " and " +
vegetable);
           // do transition
context.doTransition(transition.getTransition());
} else {
CMSDialog.showInfoDialog("You have not selected any transition.");
}
```

❗ *An unpublished (untested) API function ("CMSDialog") is used within the example. But the call can be replaced by the Java Swing class `JOptionPane`, e.g.:*

```
JOptionPane.showMessageDialog(null,"Hello " + userName);
```

4.5 Properties of a workflow (configuration)

4.5.1 General properties



Freigabe Anfordern | Form | **Properties**

Keyboard shortcut

☑ Workflow usable in WEBedit

☑ Workflow without context

Show logic

☑ Workflow always active

Figure 4-24: Properties tab (workflow model) (new Look&Feel)

✓ 4.1 **Keyboard shortcut:** In this field, a unique keyboard shortcut can be defined for each workflow. In this case the workflow no longer has to be started or switched via the context menu or the "Tasks" menu but instead can be opened directly via the defined keyboard shortcut. The cursor must be located within the field to define a new keyboard shortcut. It is then sufficient to enter the required key combination via the keyboard. The input is then copied into the input field. Text input is not possible. To change the keyboard shortcut, position the cursor in the field again and then call the new key combination. Press the "Esc" key to delete a defined keyboard shortcut for the workflow.

! *This function is released for FirstSpirit Version 4.1 and higher only.*

! *Keyboard shortcuts can only be used for context-bound workflows.*

Workflow usable in WebEdit: If the checkbox is selected the workflow can be executed not only in the JavaClient, but also in the WebClient.

! *NO dialogs can be shown within a script in the WebClient. If, within the workflow's script, a call of the form: CMSDialog.showErrorDialog(...) or JOptionPane.showMessageDialog(...) is used, an error will occur if the workflow is executed in the WebClient.*

Workflow without context: If the checkbox is selected, the workflow can be started without a context on one (or several) objects. The standard "task" workflow can, for example, be started without context.

Show logic: Show logic can be used to show or hide workflows depending on specific properties (see Chapter 4.5.2 page 179).

4.5.2 Show logic for workflows (from V4.1)

! *This function is released for FirstSpirit Version 4.1 and higher only.*

Show logic can be assigned in the "Properties" tab of a workflow in the Template Store. It can be used to show or hide workflows depending on specific properties. The show logic relates to the starting of the workflow only (not its visibility within the Template Store). If the show logic prevents the starting of a workflow, for example at a specific time for for a specific group, this workflow is no longer displayed via the context menu (for context-bound workflows) and via the "Tasks – Start Workflow" menu function (for context-less workflows).

Show logic is realised for a specific project using a BeanShell script. In this way, specific view options can be stored for each workflow.

Possible applications:

- Workflows may only be executed within a specific period (e.g. Mondays from 8.00 – 9.00 a.m. only)
- Workflows may only be executed by a specific user or a specific group (cf. Figure

4-24 "Editors" group). This case can also be realised by configuring the permissions for execution of a workflow, but that is only possible with context-bound workflows. Viewing and hiding can be realised for context-less workflows using the view logic.

- Workflows may be viewed for certain elements only, e.g. picture media only. Depending on the number of picture media, configuring via the permissions to execute a workflow on the individual elements would be correspondingly extensive. In this case it is therefore easier to realise this using the workflow's view logic.

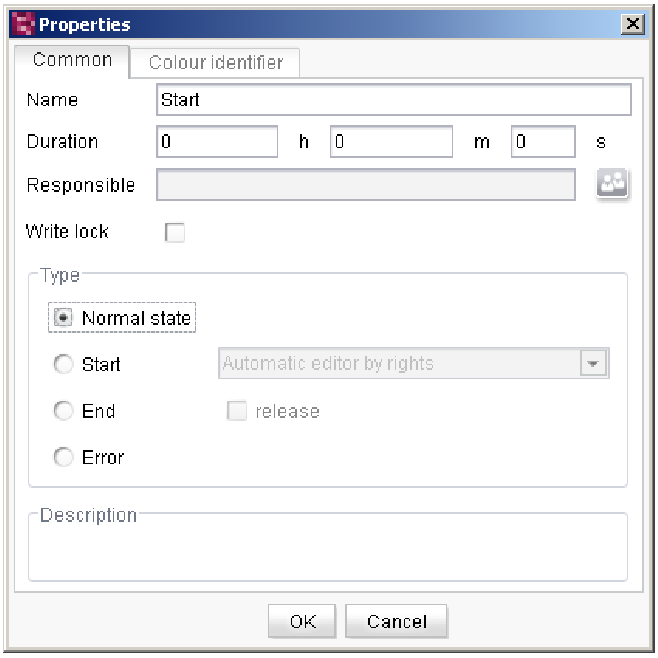
Workflow always active: This checkbox can be selected if the show logic is to be disabled. In this case the workflow is always shown, regardless of the show logic. The deposited show logic is no longer evaluated, but is retained and can be reactivated by deselecting the checkbox.

> **!** *If it is a context-bound workflow, in addition to the view logic, the permission to start the workflow on the element is also evaluated. If the user does not have permission to start the workflow the workflow is not displayed, regardless of the show logic.*

4.5.3 Properties of a status

If a status is selected within the workflow editor, the "Properties" window can be requested by clicking the icon, using the context menu, with the key combination Alt + Enter or by double-clicking (see Chapter 4.2.2 page 161). Settings can then be made for the selected element using the "Common" and "Colour identifier" tabs.

4.5.3.1 Common tab



Properties

Name: Start

Duration: 0 h 0 m 0 s

Responsible:

Write lock □

Type

● Normal state

○ Start

○ End

○ Error

Automatic editor by rights

□ release

Description

Figure 4-25: Properties of a status (common)

Name: A unique name for the selected state must be entered in this field (character limit: <= 40 characters).

Duration: A duration, during which a workflow can remain in the current state before a message is sent to the responsible users or groups, can be entered here.

Responsible: The responsible users or groups who are to be notified if the duration is exceeded are listed in this field. Click the 👥 icon to open another window in which the responsible persons can be selected from a list.

For details of group or user selection, see FirstSpirit Manual for Editors, Chapter 13.2.4 "Change authorised groups / users".

Write lock: If this option is enabled, edit mode is locked for the corresponding object for as long as it has this status (see Chapter 4.7 page 208).

Type: The current state can be defined here as a start or end node. Each workflow requires exactly one **Start state** and at least one **End state** (see also Chapter 4.2.3.1 page 162).

- **Normal state:** Default setting, applies to all states which are not start or end states.

- **Start:** Describes the state of an object with which the workflow is started. The start state is used to select the authorised users who may switch the future status of a current workflow instance (see Chapter 4.6 page 194)
    - Manual editor (per action)
      (see Chapter 4.6.2.1 page 198)
    - Automatic editor by rights
      (see Chapter 4.6.2.2 page 198)

- **End:** Describes a possible state in which an object can be after the workflow has finished. It is also possible to define whether the object is to be released as soon as this end state is reached.

- **Error:** (from V4.1) This type is used for error states (see Chapter 4.3 page 169).

Description: An explanatory comment on the current state can be entered in this field. This comment is displayed as a tooltip in the workflow editor.

From FirstSpirit Version 4.2 language-dependent display names and descriptions can be added by means of the fields **Display name** and **Description**. These are the *editing languages* (not the project languages). Editing languages are defined for a project by the project administrator and can then be configured by the editor using the "Extras – Preferred Display Language" menu. The display name is displayed to the editor, for example, in the workflow dialogues (labelling of the buttons of the transition dialogue, Help and History tabs), as entries of the context menu for starting/switching workflows, the description is used as tooltip and on the Help tab. If no display name is defined, the unique name will be displayed. If there is no description, the text of the field **Comment** is displayed.

4.5.3.2 Colour Identifier tab

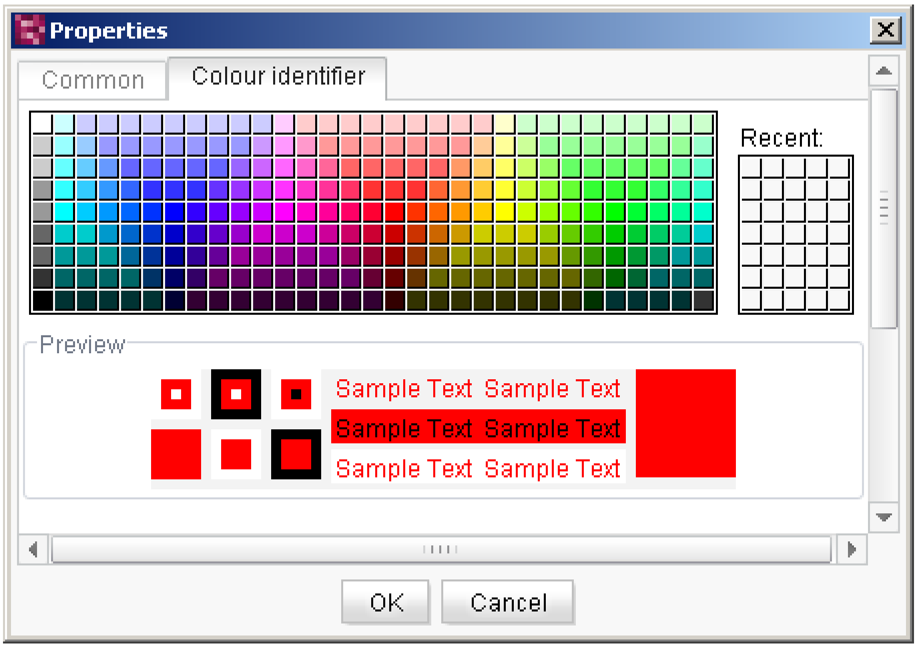


Figure 4-26: Properties of a status (Colour identifier)

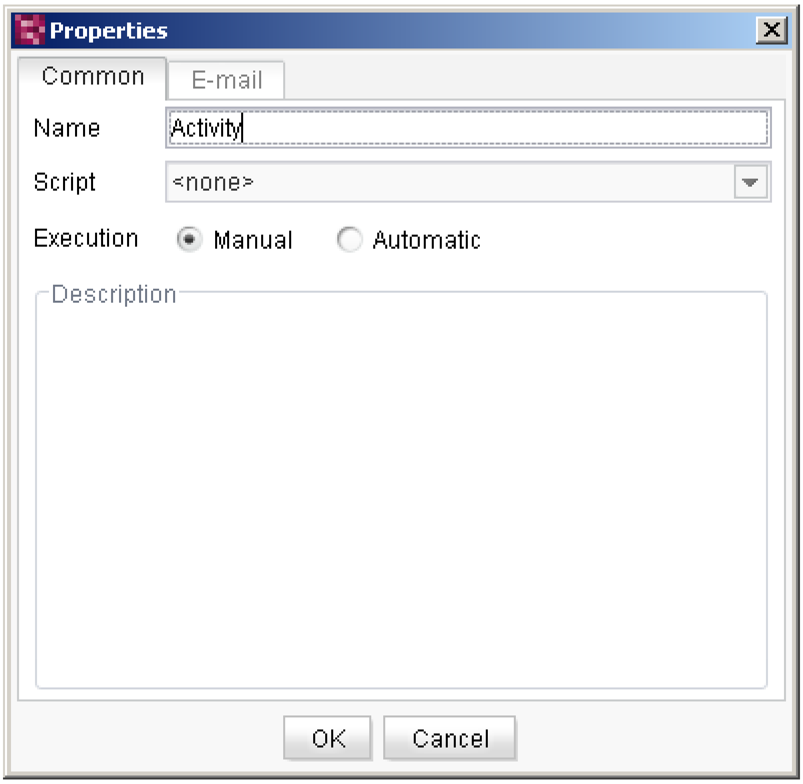
The required colour identifier for the current state can be selected in this tab using the colour schema. The object in the tree structure of the FirstSpirit client (on which the workflow was started) is highlighted by this colour as soon as the instance of the workflow has reached the corresponding state.

To make it easier to subsequently find a previously selected colour, all colours which have already been selected once within the workflow are listed in the left-hand part of the window.

4.5.4 Properties of an activity

If an activity is selected within the workflow editor, the "Properties" window can be requested by clicking the icon, using the context menu, with the key combination Alt + Enter or by double-clicking (see Chapter 4.2.2 page 161). Settings can then be made for the selected element using the "Common" and "E-mail" tabs.

4.5.4.1 Common tab



Name: Activity

Script: <none>

Execution: ● Manual ○ Automatic

Figure 4-27: Properties of an activity (common)

Name: A unique name for the selected activity must be entered in this field (character limit: <= 40 characters).

Script: The combobox can be used to select a script (from the project), which is executed as soon as this activity is called. Automatic execution must be selected if the required activity is to be executed by a script (see "Execution").

Execution: Here it is defined whether an activity is to be executed manually by a user or automatically by the system (cf. Chapter 4.2.3.2 page 163):

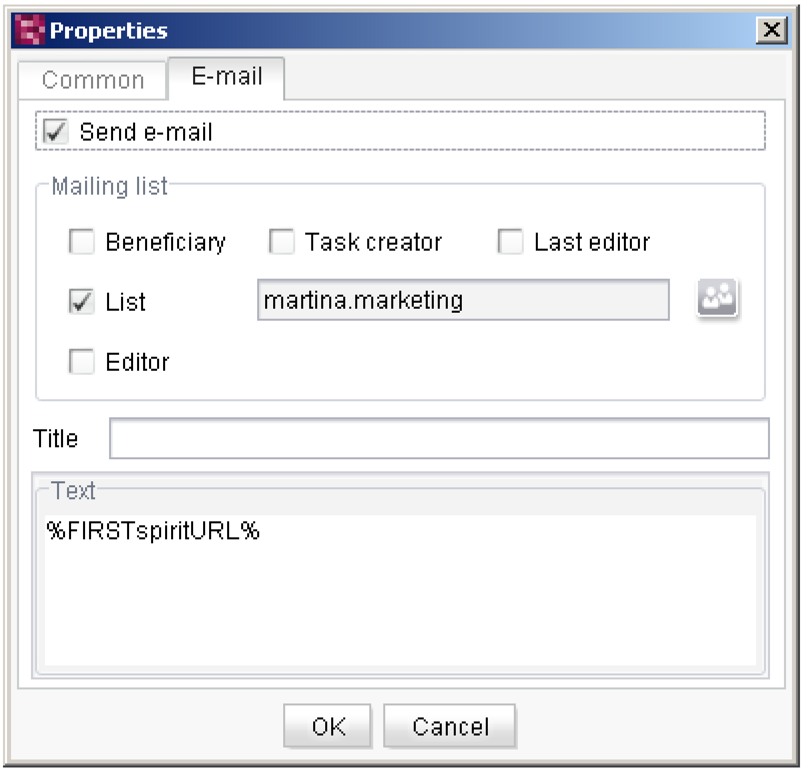
- **Manual:** When a *manual activity* is executed, a dialog appears, with which the workflow (instance) can be forwarded.

- **Automatic:** *Automatic activities* do not expect any user interaction and are executed as soon as one of the upstream states in the model is reached (i.e. the action is triggered by the system and not by the user). An automatic action (and therefore any coupled script) is therefore executed immediately after reaching a state. The script can automatically execute the necessary check as well as advancing the workflow (instance).

Description: An optional explanatory comment for this activity can be entered here.

From FirstSpirit Version 4.2 language-dependent display names and descriptions can be added by means of the fields **Display name** and **Description**. These are the *editing languages* (not the project languages). Editing languages are defined for a project by the project administrator and can then be configured by the editor using the "Extras – Preferred Display Language" menu. The display name is displayed to the editor, for example, in the workflow dialogues (labelling of the buttons of the transition dialogue, Help and History tabs), as entries of the context menu for starting/switching workflows, the description is used as tooltip and on the Help tab. If no display name is defined, the unique name will be displayed. If there is no description, the text of the field **Comment** is displayed.

4.5.4.2 E-mail tab



☑ Send e-mail

Mailing list

□ Beneficiary □ Task creator □ Last editor

☑ List martina.marketing

□ Editor

Title

Text

%FIRSTspiritURL%

Figure 4-28: Properties of an activity (E-mail)

Send e-mail: If the checkbox is selected, an e-mail is sent to the selected recipients (see "Mailing List"), as soon as the activity has been executed.

Mailing list: Here you can select to which persons an e-mail is to be sent.

- **Beneficiary:** Persons who are authorised to forward the workflow into the following status. These permissions are either defined through the permissions to switch the transition directly within the workflow model (see Chapter 4.5.5.2 page 190) and/or through the permissions to switch a transition on an object, on which the workflow instance was started.

- **Task creator:** The user who started the instance of the workflow.

- **Last editor:** The user who switched the workflow instance into its current status.

- **List:** Click the 👥 icon to open another window in which the required persons or groups can be selected from a list.
  For details of group or user selection, see FirstSpirit Manual for Editors, Chapter 13.2.4 "Change authorised groups / users".

- **Editor:** The person currently editing or working on the workflow.

Title: The text of the Subject line in the e-mail is entered here.

Text: The message to be sent to the recipient is entered in this field. The following % expressions can be used as wildcards, which are automatically replaced by the system:

Wildcards for generating context-specific information:

`%FIRSTspiritURL%` = HTTP connection mode (default mode)
`%FIRSTspiritSOCKETURL%` = SOCKET connection mode
`%PAGESTORE_PREVIEW_URL%` = Preview URL of a page from the Page Store
`%SITESTORE_PREVIEW_URL%` = Preview URL of a page reference from the Site Store
`%WF_NAME%` = Name of the workflow
`%CREATOR%` = Creator of the workflow (complete name)
`%LAST_USER%` = last user (editor)
`%LAST_COMMENT%` = last comment
`%NEXT_USER%` = next user (editor)
`%PRIORITY%` = priority
`%DATE%` = deadline (only if set)
`%HISTORY%` = history of the workflow's instance
`%WEBeditURL%` = WebEdit link on the preview of the page

If the wildcards `%FIRSTspiritURL%`, `%FIRSTspiritRMIURL%` or `%FIRSTspiritSOCKETURL%` are entered in the "Text" field, a link (which links to the corresponding node in the project) is generated in the sent e-mail, e.g. for `%FIRSTspiritURL%`:

http://myServer:9999/fs4root/start/FIRSTspirit.jsp?app=client&project=QS_akt&name=vorlage_1&type=Page&id=4394331&host=myServer&port=9999&mode=HTTP

or of `%PAGESTORE_PREVIEW_URL%`:

http://myServer.espirit.de:9999/fs4preview/preview/4238727/page/DE/current/4238731/4394331

The other wildcards can be used to generate further context-specific information on the respective workflow instance, e.g. `%HISTORY%`:

```
Mar 9, 2009 8:53:43 AM - Admin, Manual
activity: Release request
Status: Release requested
Comment: UserB : Please release content
```

Apart from the JavaClient URL (`%FIRSTspiritURL%`), a link to a preview page in WebClient can also be transferred in the text (`%WEBeditURL%`), e.g.:

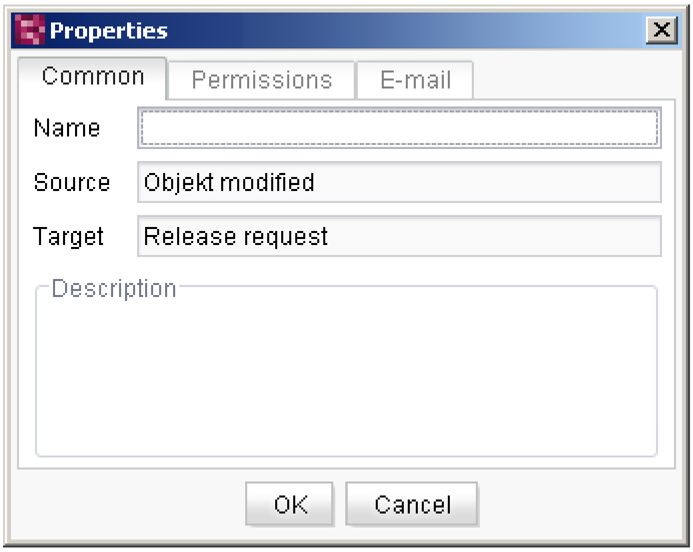
```
http://myServer:9999/WEBedit/Dispatcher?project=333333&id=2222216&language=DE&guiLanguage=DE&action=ExternalPreview&firsteditDir=WEBedit
```

> ! *The wildcard substitution only works if the JNLP servlet has been installed on the system.*

For further 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 Common tab



Name

Source: Objekt modified

Target: Release request

Description

Figure 4-29: Properties of a transition (Common)

Name: A name for the selected transition can be assigned in this field. This name must be unique regarding its source (character limit: <= 40 characters).

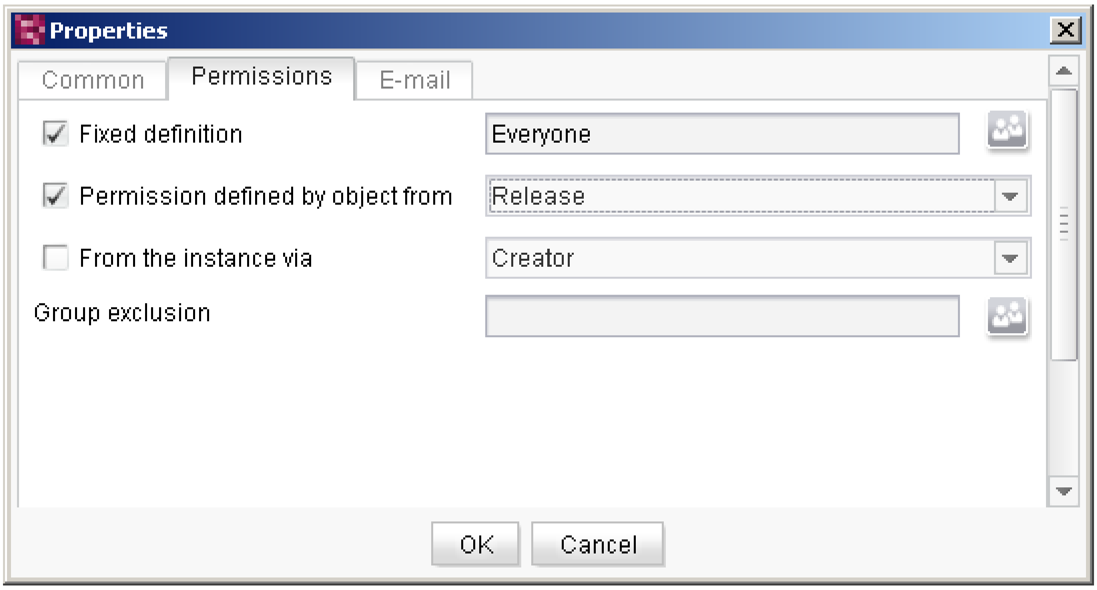
Source: The source from which the transition starts is automatically displayed in this field.

Target: The target to which the transition points is automatically displayed in this field.

Description: An explanatory comment on the current transition can be entered in this field.

From FirstSpirit Version 4.2 language-dependent display names and descriptions can be added by means of the fields **Display name** and **Description**. These are the *editing languages* (not the project languages). Editing languages are defined for a project by the project administrator and can then be configured by the editor using the "Extras – Preferred Display Language" menu. The display name is displayed to the editor, for example, in the workflow dialogues (labelling of the buttons of the transition dialogue, Help and History tabs), as entries of the context menu for starting/switching workflows, the description is used as tooltip and on the Help tab. If no display name is defined, the unique name will be displayed. If there is no description, the text of the field **Comment** is displayed.

4.5.5.2 Permissions tab



☑ Fixed definition: Everyone

☑ Permission defined by object from: Release

□ From the instance via: Creator

Group exclusion:

Figure 4-30: Properties of a transition (Permissions)

Fixed definition: If this option is selected, the authorised users of this transition are then specified by a fixed definition. The field lists the responsible users and/or groups who are allowed to switch this transition. Click the icon after the responsible persons (fixed definition field) to open another window in which the responsible persons can be selected from a list of project groups or users.

Permission defined by object from If this option is selected, the authorised users result from the permissions definition in the tree structure of the FirstSpirit client. This field can be used to select the permission the user must have on the object under consideration to be able to perform this transition.

From the instance via If this option is selected, the authorised users result from the current instance of the workflow. The creator of the instance or the last editor can be selected from this field. The option "last editor of target activity" is only available on outgoing transitions of states and can only be used if the workflow contains a loop, so that an activity can be passed through more than once, for example:

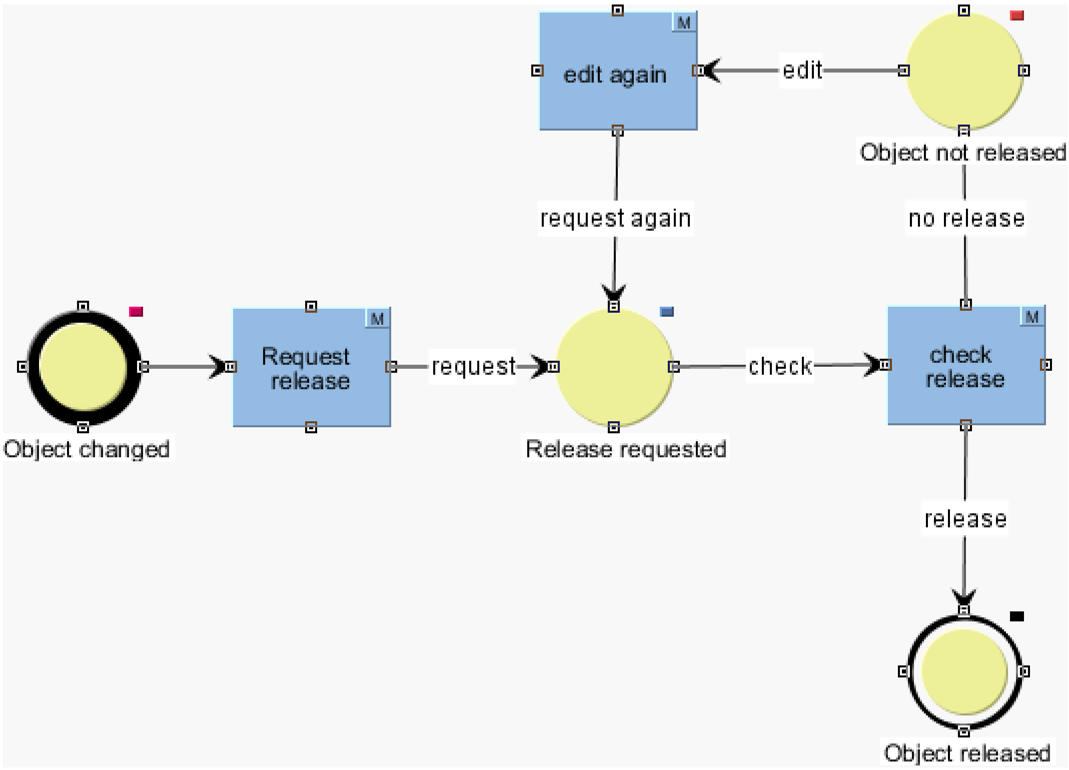
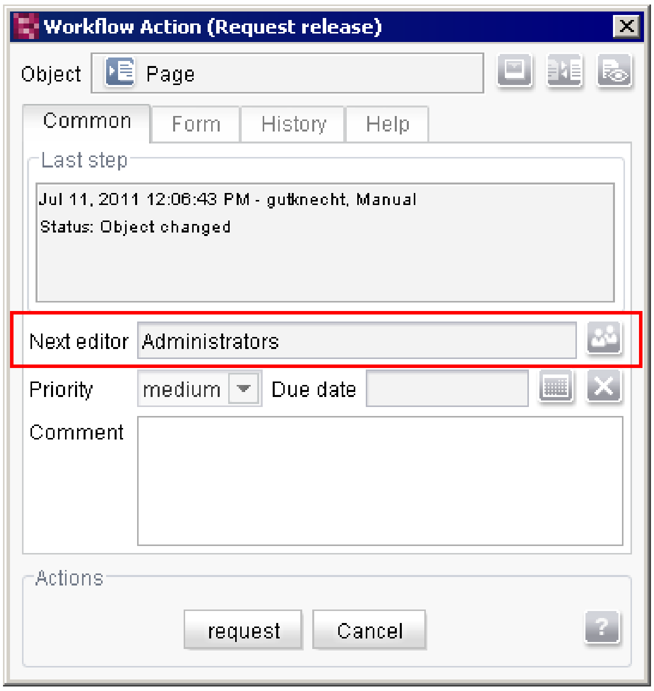


Figure 4-31: Default workflow "Release"

The activity "check release" in Figure 4-31, for example, would be in this case the target activity, i.e. an activity to which a status is pointing. If the option "last editor of target activity" would be selected on the transition "check", only a user who has effected this transition already once can effect this transition again.

Group exclusion: Groups who are **not** to be shown in the field "Next editor" of the workflow dialog "Workflow Action" can be selected here:



Workflow Action (Request release)

Object: Page

Last step

Jul 11, 2011 12:06:43 PM - gutknecht, Manual
Status: Object changed

Next editor: Administrators

Priority: medium

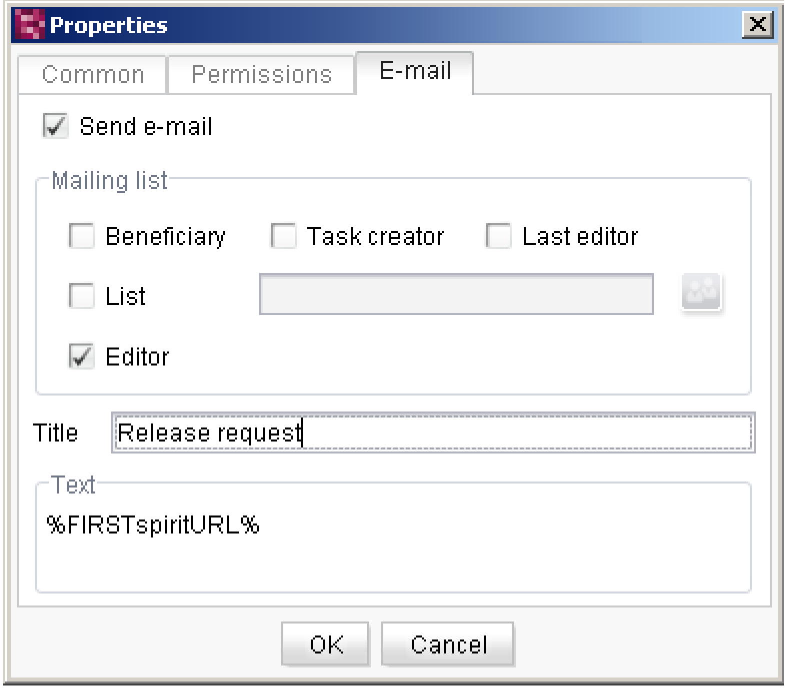
Due date

Comment

Figure 4-32: Preselection "Next editor"

However, the groups which are selected via the function "Group exclusion" can still be selected in the above shown dialog (Figure 4-32) using the icon 👥. Furthermore, the selection of "Group exclusion" has also effects on the sending of e-mails.

4.5.5.3 E-mail tab



☑ Send e-mail

Mailing list

□ Beneficiary □ Task creator □ Last editor

□ List

☑ Editor

Title: Release request

Text

%FIRSTspiritURL%

Figure 4-33: Properties of a transition (E-mail)

Send e-mail: selecting this option causes an e-mail to be sent to the selected recipient as soon as this transition has been executed.

E-mail transmission and wildcard substitution are carried out analogous to the description of e-mail transmission in Chapter 4.5.4.2, page 186.

4.6 Permissions configuration for workflows

Permissions for executing workflows are a special type of editorial permissions which refer to the workflows within a project only.

The permissions configuration can be defined, context-dependent, directly on the object on which the workflow instance is started or in the Template Store with general validity within the workflow model:

- *General permissions configuration* for starting and switching a workflow in the Template Store (for all instances) (see Chapter 4.6.1 page 194)
- *Context-dependent permissions assignment* for starting a workflow on individual objects, sub-trees and stores (for individual instances – depending on the object on which the workflow is started) (see Chapter 4.6.3 page 199)
- *Context-dependent permissions assignment* for connecting individual transitions of a workflow ("special permissions") on individual objects, sub-trees and stores (for individual instances – depending on the object on which the workflow is started) (see Chapter 4.6.4 page 201)

Apart from the actual permissions configuration, the authorised modifiers of a workflow (instance) can be limited by the editor (on editing an activity), provided this has been configured by the template developer of the workflow (see Chapter 4.6.2 page 195).

The effects of permissions definition in the JavaClient are described by way of an example in Chapter 4.6.5 (page 203 ff.)

4.6.1 General rights (permissions) configuration via the Template Store

Within the Template Store, the general permissions configuration for starting or switching a workflow are set via the permissions assigned to the individual transitions. This ensures that each individual activity can be performed by authorised users only. The permissions dialog is opened by double-clicking the transition in the workflow model. The permissions for connecting the transition can be assigned in the "Permissions" tab (see Chapter 4.5.5.2 page 190).

Overwriting transition permissions: The permissions which are defined within the workflow model are evaluated for all instances of the workflow. Therefore, in this way, generally valid permissions configurations can be defined for the workflow. However, these permissions can be overwritten for (context-dependent) workflows. Transition permissions for individual objects, sub-trees or stores can be overwritten using the "Permission Assignment" dialog on the respective objects (see Chapter 4.6.3 page 199 and Chapter 4.6.4 page 201).

Context-dependent transition permissions: Apart from the fixed definition permissions of a group or a user for connecting a transition, the transition permissions can also be assigned, context-dependent, within the Template Store. In this case, the transition permissions must be selected "From the instance via" (see Chapter 4.5.5.2 page 190). For example, if "last editor" is selected here, the editor is automatically the only person to receive the permission to connect the transition, which switched the workflow instance into the current status.

Linking with the editorial permissions: Apart from the option of determining the permissions on a context-dependent basis from the workflow instance (see above), the editorial permissions can also be linked to the transition permissions (also context-dependent). In this case, the transition permissions option "Permission defined by object via" must be selected (see Chapter 4.5.5.2 page 190). For example, if the editorial permission "Release" is selected here, only the modifier who has permission to "Release" on the object on which the workflow instance was started, automatically receives permission to connect the transition.

4.6.2 Change or block modifier preselection

The preselection of authorised "modifiers" is displayed to the executing editor of the workflow in the "Editor" field within the activity dialog. "Editors" are all groups or users who have permission to connect future transitions of the workflow. The permissions defined on outgoing transitions with future status are taken into account (see Chapter 4.6.1 page 194).

- Example (workflow model):

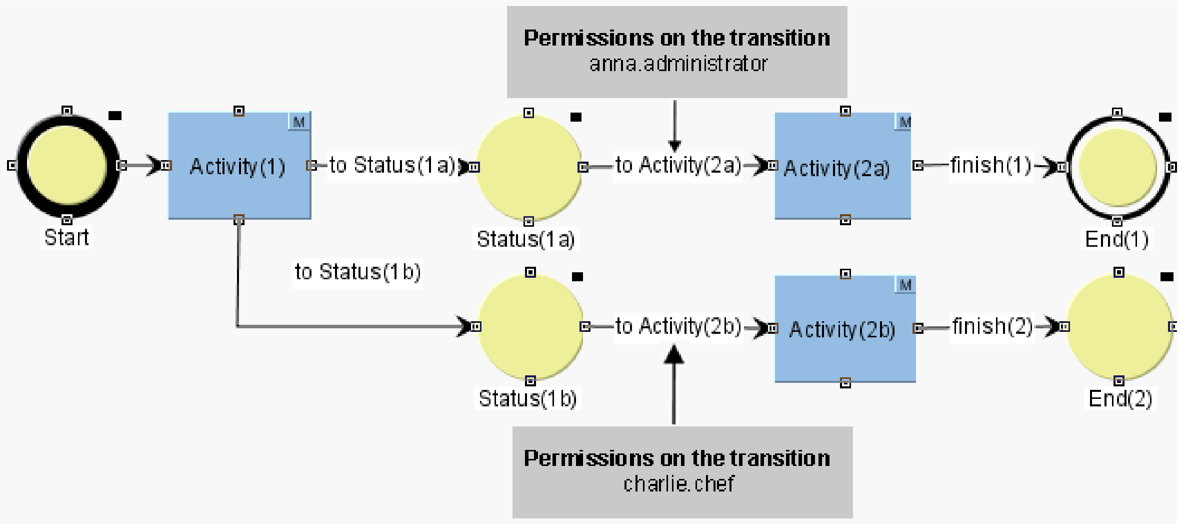
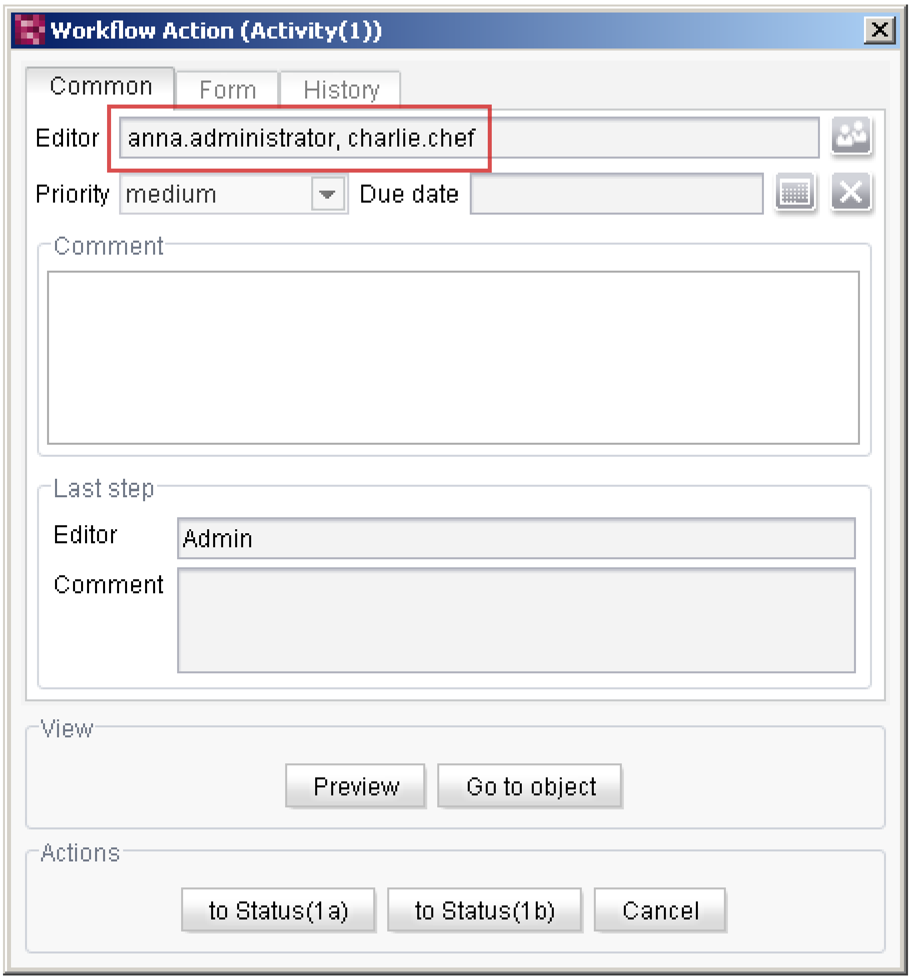


Figure 4-34: Workflow model with permission configuration (grey background)

Example description (see Figure 4-34):

- The permissions on the transition "to Activity(2a)" have been assigned a fixed definition for user "anna.administrator".
- The permissions on the transition "to Activity(2b)" have been assigned a fixed definition for user "charlie.chef".

The workflow is now started by the editor. The activity dialog "Activity(1)" opens (see Figure 4-35). There the editor can choose between the two statuses: "Status(1a)" and "Status(1b)". The future modifiers (editors) of the workflow are automatically listed in the "Editor" field. After forwarding the current "Activity(1)" the workflow has either "Status(1a)" or "Status(1b)". Future "Editor" can therefore only be groups or users who have permissions on the outgoing transitions of these two statuses. Therefore, in the example, the "editors" who have permissions to connect the transitions "to Activity(2a)" and to connect the transitions "to Activity(2b)".



Editor: anna.administrator, charlie.chef

Priority: medium

Last step

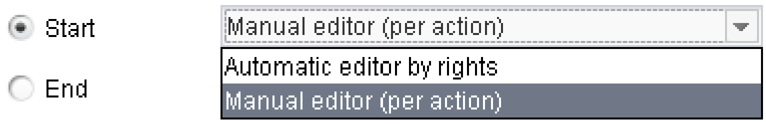
Editor: Admin

Comment:

Figure 4-35: Example – Workflow action "Activity(1)"

This preselection of the future possible modifiers (editors) can be changed by the editor. To do this, the template developer must open the configuration dialog of the start status on the start status of the workflow in the Template Store (see Chapter 4.5.3.1 page 181).

In the "Common" tab you can choose between two options:



● Start

○ End

Manual editor (per action)

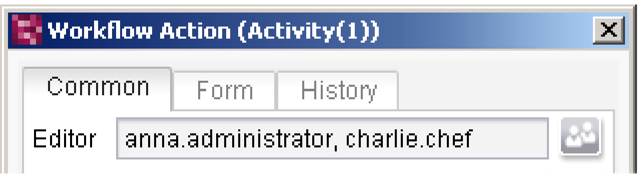
- Automatic editor by rights
- Manual editor (per action)

Figure 4-36: Permission configuration for the start status

- Manual editor (per action) (see Chapter 4.6.2.1 page 198)
- Automatic editor by rights (see Chapter 4.6.2.2 page 198)

4.6.2.1 Manual editor (per action)

The permissions defined within the workflow (on the outgoing transitions of the future status) are evaluated in the "Editor" filed. If the "Manual editor (per action)" option is selected, these modifiers (editors) can be *changed* by the editor. The button for group or user selection in the "Workflow Action" dialog, which is displayed on starting or switching the workflow (instance), is then *active*.



Workflow Action (Activity(1))

Editor anna.administrator, charlie.chef

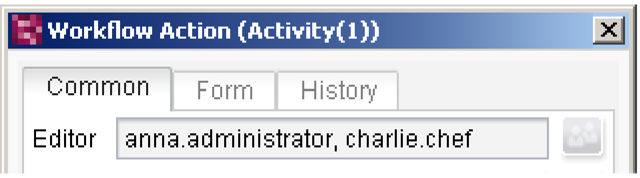
Figure 4-37: Workflow action – manual editor by rights

Click the button to open the dialog for group and user selection with all authorised modifiers. The editor can now restrict this selection to the required users.

The editor can only restrict the modifiers. If the list of modifiers is to be extended, the permissions on the transitions of the workflow model must be adjusted by the template developer.

4.6.2.2 Automatic editor by rights

The permissions defined within the workflow (on the outgoing transitions of the future status) are evaluated in the "Editor" field. If the "Automatic editor by rights" option is selected, these modifiers (editors) *cannot be changed* by the editor. The button for group or user selection in the "Workflow Action" dialog, which is displayed on starting or switching the workflow (instance), is then *inactive*.



Workflow Action (Activity(1))

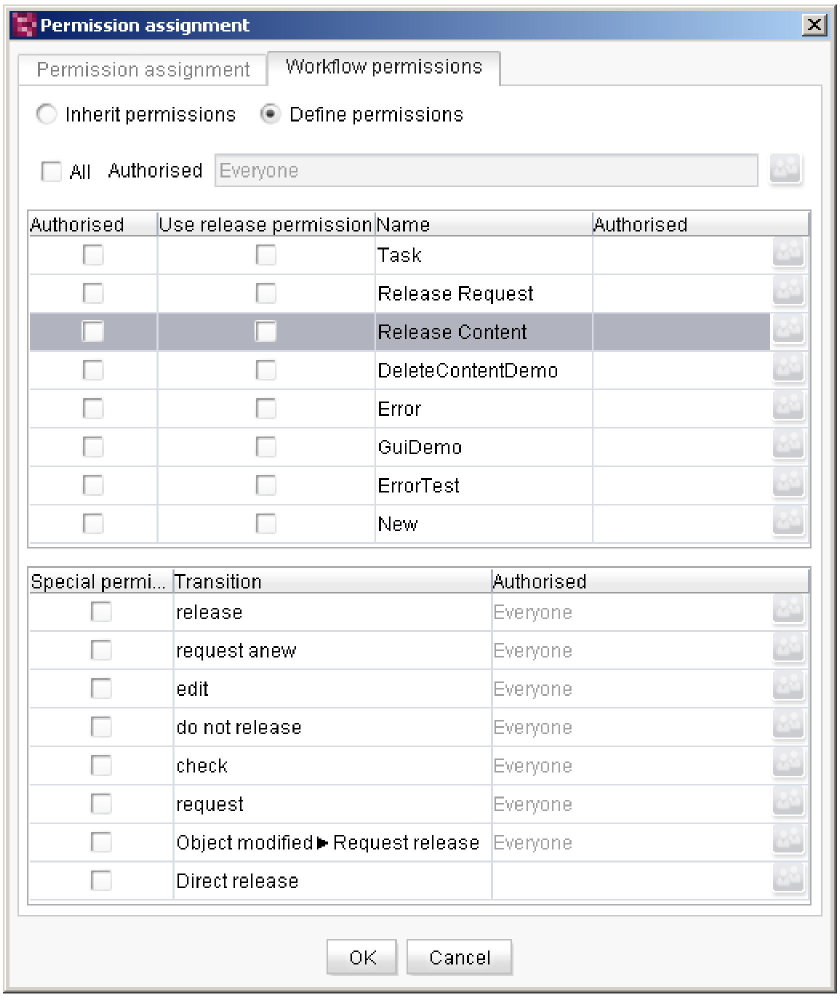
Editor anna.administrator, charlie.chef

Figure 4-38: Workflow action – Automatic editor by rights

4.6.3 Context-dependent permissions for starting a workflow

Context-dependent permissions are assigned in the FirstSpirit JavaClient. Here all areas of the project can be assigned editorial permissions for specific groups or users. Detailed permissions can be assigned for each object, for example for an individual page in the Page Store. These permissions can be inherited hierarchically within the individual stores.

The permissions for the execution of workflows are issued parallel to the editorial permissions for groups and users via the "Permission Assignment" dialog. The "Permission Assignment" dialog is opened via the "Extras – Change Permissions" context menu on the required object within the JavaClient tree structure. In addition to the general "Permission assignment" of editorial permissions there is also a "Workflow permissions" tab:



○ Inherit permissions ● Define permissions

□ All Authorised Everyone

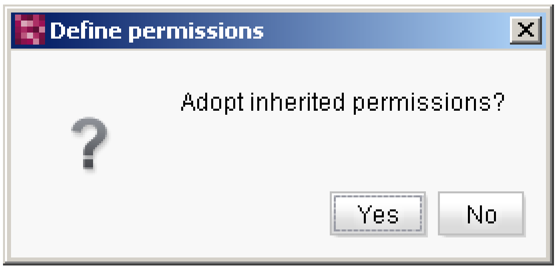
Authorised	Use release permission	Name	Authorised
□	□	Task	
□	□	Release Request	
□	□	Release Content	
□	□	DeleteContentDemo	
□	□	Error	
□	□	GuiDemo	
□	□	ErrorTest	
□	□	New	

Special permi...	Transition	Authorised
□	release	Everyone
□	request anew	Everyone
□	edit	Everyone
□	do not release	Everyone
□	check	Everyone
□	request	Everyone
□	Object modified ▶ Request release	Everyone
□	Direct release	

Figure 4-39: Context-dependent workflow permissions

Inherit permissions: The "Inherit permissions" radiobutton is selected as a default (exception: root nodes). With this setting the "Workflow permissions" are inherited from a higher-level node.

Define permissions: If the "Define permissions" radiobutton is selected, permissions for the workflows can be defined on the node. In the first step the window opens for adopting the inherited permissions.



Define permissions

Adopt inherited permissions?

Figure 4-40: Adopt inherited permissions?

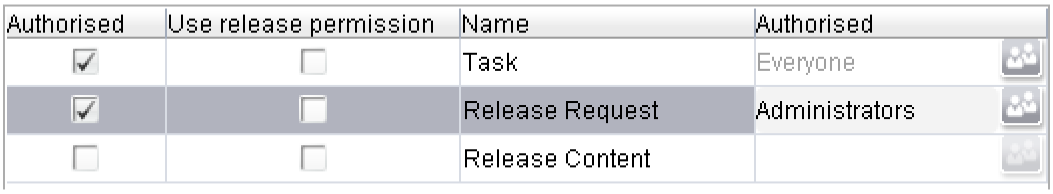
If the permissions are adopted from a higher-level node, the inherited permissions are copied into the table of the workflows. If, on the other hand, the dialog is confirmed with "No" the workflow permissions are reset. The table view is now active, regardless of the selection, i.e. the user can define their own permissions.

All: If the "All" checkbox is selected, all workflows can be started by the "authorised" user on the current node and all hierarchically low-level nodes of the tree structure. In this case, the two tables below it cannot be edited and all the settings made in them have no significance. If the checkbox is not selected, the settings must be individually defined for each workflow.

Authorised: This field lists all users and/or groups who may call a workflow on the current node. If the 👥 icon is clicked, the "Select Groups/Users" window opens. All the project's groups and users are listed. The window can be used to select the authorised groups and individual users.

All the project's workflows are listed in the top table (cf. Figure 4-39). If selected workflows only are to be allowed for a sub-tree, a list of the workflows which can be started by selected users can be created during the permissions definition. If necessary, a different user can be defined for each workflow.

The input options of this table are only enabled if the "All" checkbox is deselected. In this case, the permission for starting a workflow can be issued or prevented for individual workflows:



Authorised	Use release permission	Name	Authorised
☑	□	Task	Everyone 👥
☑	□	Release Request	Administrators 👥
□	□	Release Content	👥

Figure 4-41: Context-dependent permissions for starting individual workflows

Authorised: If the "Authorised" checkbox is selected, all authorised users may start the workflow (see "Authorised" column). The authorisation (permission) is assigned for the current node and all hierarchically lower-level nodes of the tree structure.

Use release permission: If the "Use release permissions" checkbox is selected, the release permissions defined in the "Permissions assignment" tab are evaluated for each user. Important: Contradictions in permissions definition can occur if the checkbox is not selected. A conflict situation can arise if, for example, a user has *no* permission to release on a specific object, but is listed as being authorised in the standard "Release Request" workflow. In such a case, the release would be prevented by the system, but the user cannot see the response (no release) because, as defined, the workflow can be passed on up to the "Issue Release" status. If, on the other hand, the "Use release permissions" checkbox is activated, the release permissions of the user are evaluated at each workflow transition. If contradictions are then found between the editorial permissions (no permission to release) and the permissions in the workflow (e.g. release), these transitions are hidden for the "unauthorised" user. In this case, the user can request the release, i.e. start the workflow, but they can no longer forward the object to the following "Object released" status. The transition required for this is hidden.

Name: This column contains the name of the workflow.

Authorised: This field lists all users and/or groups who may start a workflow on the current node. If the 👥 icon is clicked, the "Select Groups/Users" window opens. All the project's groups and users are listed. The window can be used to select the authorised groups and/or individual users.

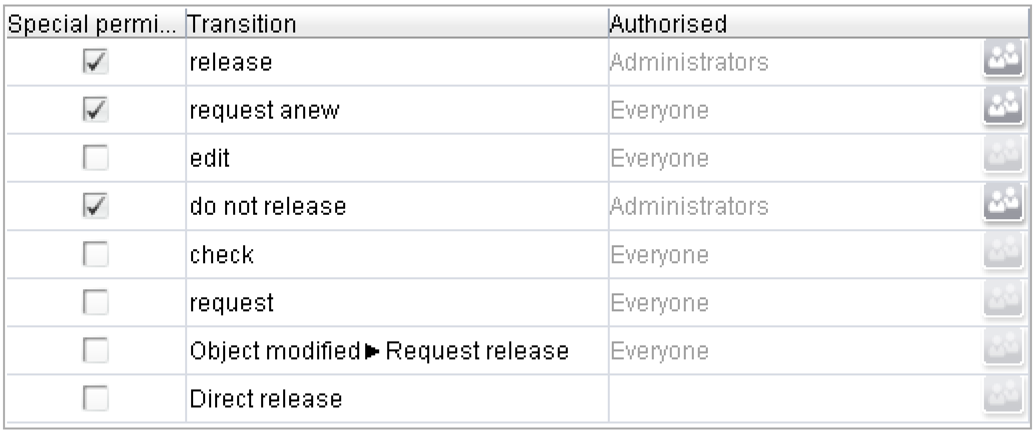
For further information on editorial permissions, see FirstSpirit Manual for Editors, Chapter 13.

4.6.4 Context-dependent permissions for switching a workflow

The context-dependent permissions assigned to date in Chapter 4.6.3 (page 199 ff.) solely relate to the permission to start a workflow. However, so-called context-dependent "Special permissions" can also be defined for the workflow's individual transitions.

If a specific activity in an individual node is to be performed by another user, this can be defined using the context-dependent special permissions. To do this, the required workflow must first be selected in the top table (see Figure 4-41). All the transitions of the selected workflow are then listed in the bottom table for the definition of special permissions (see Figure 4-42). Authorised users can then be defined for the required workflow transition.

> ! *If permissions for executing the transition have already been defined via the workflow model within the Template Store (see Chapter 4.6.1 page 194), this definition (context-dependent "Special permissions") overwrites the existing permissions (from the workflow model).*



Special permi...	Transition	Authorised
☑	release	Administrators
☑	request anew	Everyone
□	edit	Everyone
☑	do not release	Administrators
□	check	Everyone
□	request	Everyone
□	Object modified ▶ Request release	Everyone
□	Direct release	

Figure 4-42: Context-dependent special rights for connecting a transition

Special permissions: If the tick is set in this column, the permissions assigned in the workflow for this transition are ignored on this node. Instead, the permissions listed in this place in the permissions column apply to this transition.

Transition: This column lists the names of the transitions. If a name has not been assigned for a transition in the workflow, the names of the transition's source and target appear here.

Authorised: All users and/or groups who may execute this transition are listed in this field. The transition permissions listed here are adopted from the workflow model (see Chapter 4.5.5.2 page 190), but are overwritten if the "Special permissions" checkbox is selected (default setting: "Everyone" group).

If the 👥 icon is clicked, the "Select Groups/Users" window opens. All the project's groups and users are listed. The window can be used to select the authorised groups and/or users.

4.6.5 Effects of the permissions configuration

Transition permissions are either defined generally via the workflow model (see Chapter 4.6.1 page 194) or context-dependent for individual objects or sub-trees (see Chapter 4.6.3 page 199 and Chapter 4.6.4 page 201).

The effects are identical for both permission definitions:

Transitions which lead to an activity, authorise the user to call and perform this activity via the context menu of the corresponding object.

Transitions which lead to a state, authorise the user to switch this state in the activity dialog.

Example (workflow model):

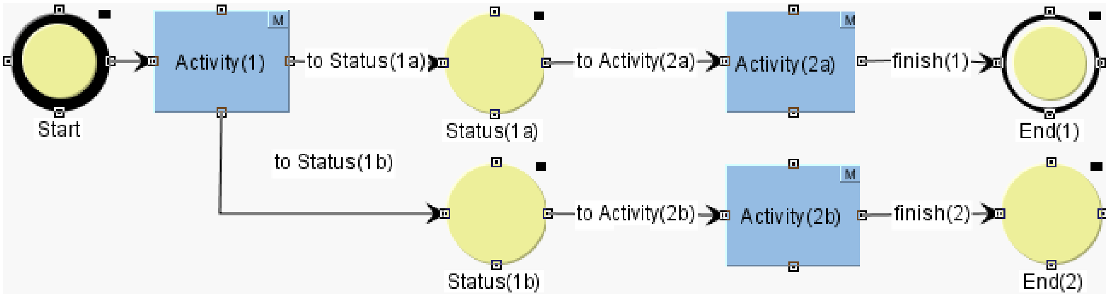
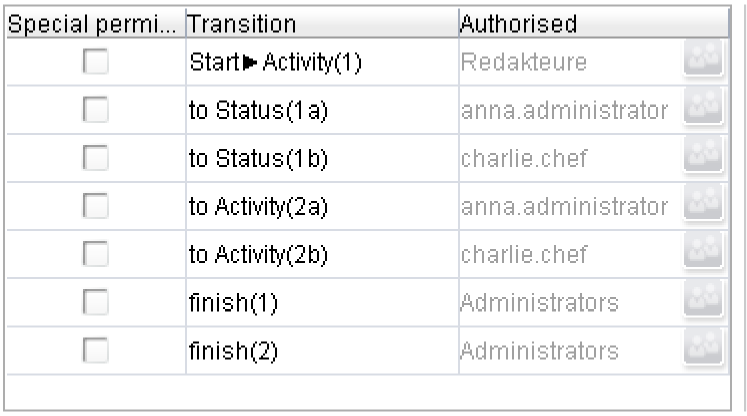


Figure 4-43: Workflow model example

Example of permissions definition (defined through the workflow model):



Special permi...	Transition	Authorised
□	Start▶Activity(1)	Redakteure
□	to Status(1a)	anna.administrator
□	to Status(1b)	charlie.chef
□	to Activity(2a)	anna.administrator
□	to Activity(2b)	charlie.chef
□	finish(1)	Administrators
□	finish(2)	Administrators

Figure 4-44 Example of permissions definition through 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:

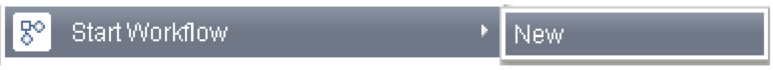
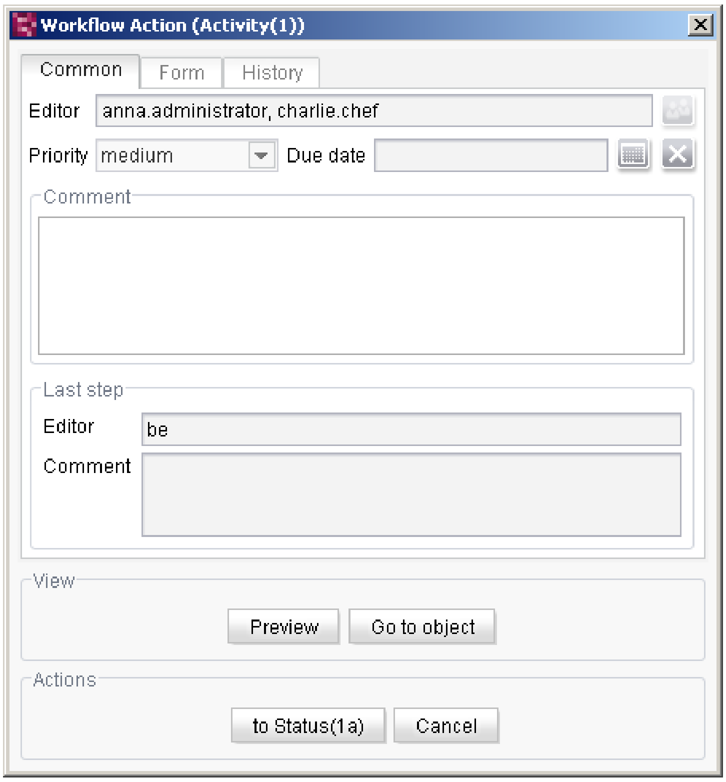


Figure 4-45: Start the workflow via the context menu:

2. The "Activity(1)" dialog provides the option to switch the workflow into "Status(1a)" or "Status(1b)". The button for switching "Status(1a)" is shown for the "Editors" group only (see Figure 4-46), the button for switching "Status(1b)" is shown for the "Administrators" group only (see Figure 4-47).



Editor: anna.administrator, charlie.chef

Priority: medium

Due date:

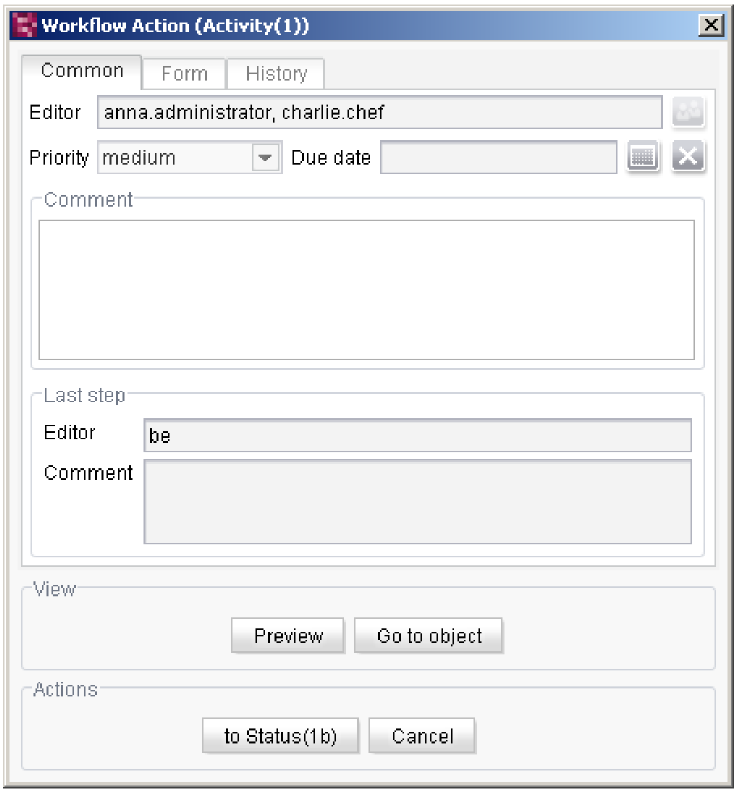
Comment

Last step

Editor: be

Comment

Figure 4-46: Activity dialog for switching the transition "to Status(1a)"



Workflow Action (Activity(1))

Editor: anna.administrator, charlie.chef

Priority: medium

Due date:

Comment:

Last step

Editor: be

Comment:

Figure 4-47: Activity dialog for switching the transition "to Status(1b)"

3. If the workflow instance has Status(1a), the user anna.administrator may select the following transition "to Activity(2a)" from the context menu. "Activity(2a)" dialog is displayed.

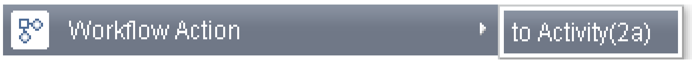


Figure 4-48: Switch the workflow via the context menu

4. If the workflow instance has Status(1b), the user charlie.chef may select the following transition "to Activity(2b)" from the context menu. "Activity(2b)" dialog is displayed.

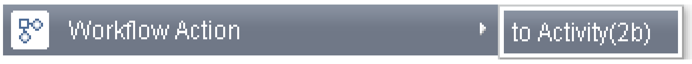
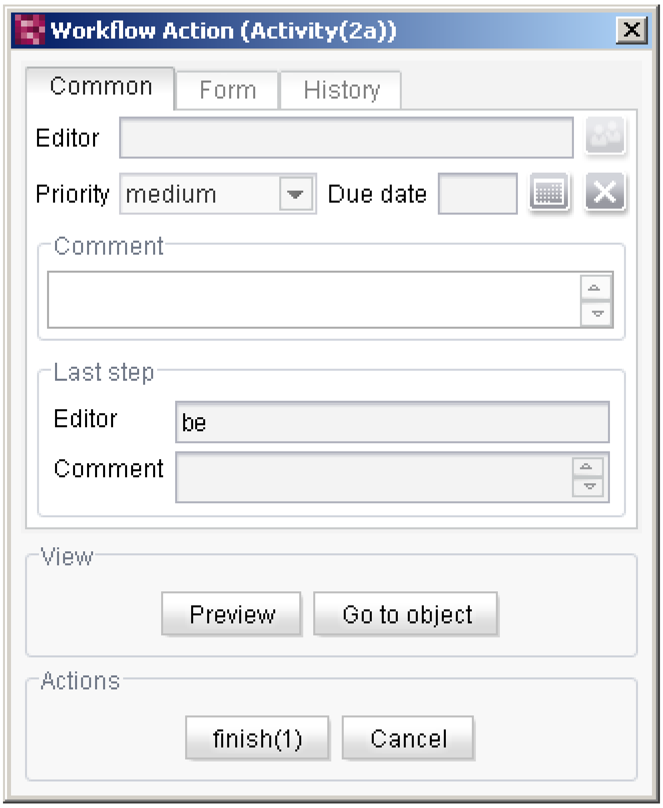


Figure 4-49: Switch the workflow via the context menu

5. The "Activity(2a)" dialog provides the option to exit the workflow with status "End(1)". The button for switching "finish(1)" is displayed for user "anna.administrator" only.
(In this case, the field with the future "Editors" (modifiers) is empty, because it is the last transition (see Chapter 4.6.2 page 195)).

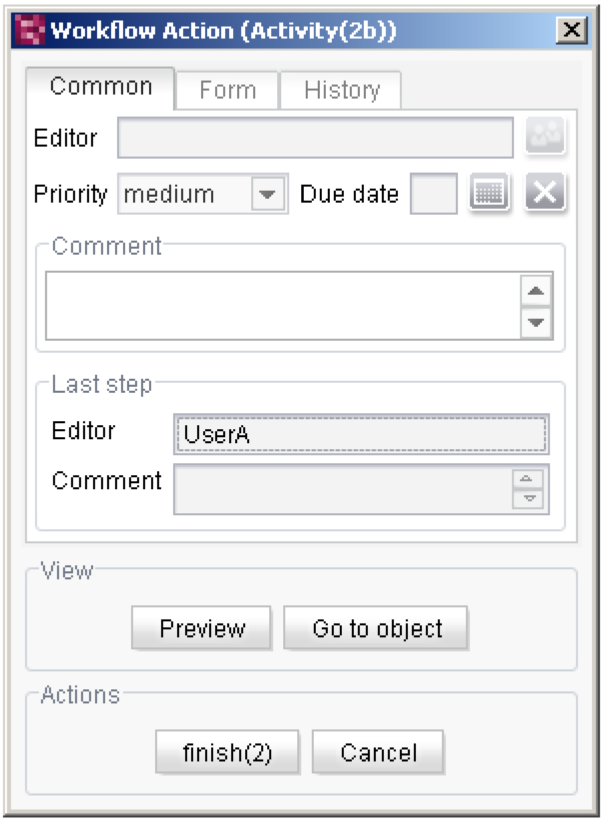

Workflow Action (Activity(2a))

Priority: medium

Last step Editor: be

Figure 4-50: Activity dialog for switching the transition "finish(1)"

6. The "Activity(2b)" dialog provides the option to exit the workflow with status "End(2)". The button for switching "finish(2)" is displayed for user charlie.chef only.
   (In this case, the field with the future "Editors" (modifiers) is empty, because it is the last transition (see Chapter 4.6.2 page 195)).



Editor

Priority medium

Due date

Comment

Last step

Editor UserA

Comment

View

Actions

Figure 4-51: Activity dialog for switching the transition "finish(2)"

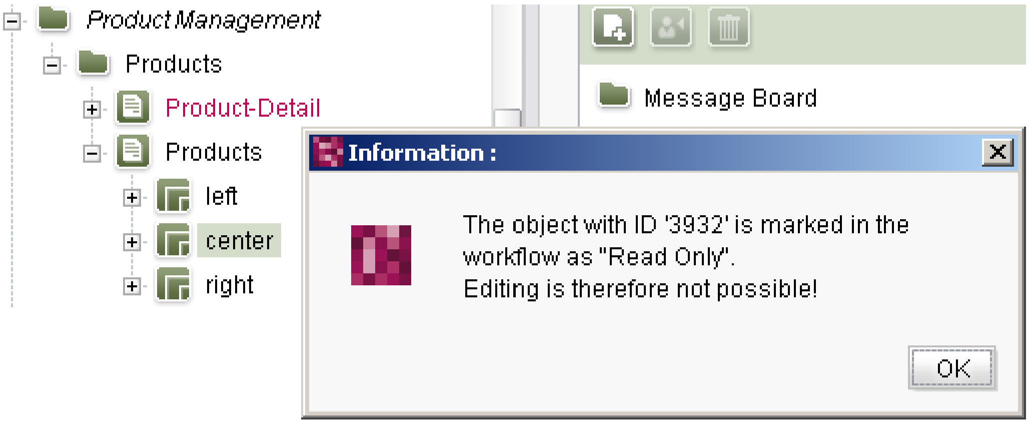
If a user or a group has the permission to connect a transition which leads to an activity dialog, this group or user should also have the permission to connect the transition into at least one following status. Otherwise the activity dialog only contains a button for cancelling the action. In this case, the permissions should be checked and, if necessary, redefined.

4.7 Write protection within workflows

4.7.1 General information

When a (context-bound) workflow is started, the element on which the workflow was started can be assigned write protection (write lock) (see Chapter 4.5.3.1 page 181). This write lock should prevent an element from being changed by another modifier while a workflow is running.

Write lock due to currently running instances of a workflow:



The object with ID '3932' is marked in the workflow as "Read Only".
Editing is therefore not possible!

Figure 4-52: Write lock on lower-level objects

The write lock affects both the current object and all lower level objects of the currently running workflow instance. In the example from Figure 4-52, a write lock has been set on the "Product Management" folder by a current workflow. If another user tries to lock this folder, they are informed that the element cannot be edited at the present time. The same message appears if the user tries to lock the "Products" page or any object below the "Product Management" folder.

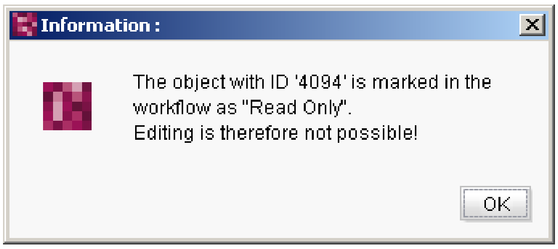
The write lock is set irrespective of whether the workflow uses a script or not and regardless of which actions are executed on the element concerned.

4.7.2 Write lock in case of creating and moving objects

Several actions can be executed within the FirstSpirit JavaClient without the object being in edit mode. This change to the edit concept should ensure that parallel working (with many users) works as smoothly as possible in large projects too. For example, the whole sub-tree is no longer requested for editing an element, only the object to be changed at this time. Therefore, it is also possible to create or move an element without requesting a write lock on the parent node first (edit mode).

However, as workflows involve potentially critical actions (e.g. the release of an object), a workflow's write lock also prevents the creation or moving within the currently running instance of the workflow.

For example, if an editor attempts to add a section to a page on which a workflow has been started, the following error message is displayed:



Information :

The object with ID '4094' is marked in the workflow as "Read Only".
Editing is therefore not possible!

Figure 4-53: Write lock on a page (due to a workflow)

4.7.3 Write lock within scripts

For several actions which are executed via the Access API of FirstSpirit, a write lock is necessary on the element concerned, e.g.:

- Recursive deletion of elements in the project
- (Recursive) release of elements in the project

One problem which now arises in practice is that of setting a write lock in a script (on an element or sub-tree - API call `setLock(true, false)` or `setLock(true)`), if a write lock (due to the workflow – see Chapter 4.5.3.1) already exists on the element due to the starting of the workflow. In this case, the workflow's write lock prevents the setting of the "normal" write lock on the element.

However, it is not necessary to set the write lock for simple delete or release actions within the workflow, as the element concerned is automatically locked by the workflow when the transition is connected.

The situation is different if the deletion or release is to be executed recursively, i.e. on a sub-tree of the project. In this case, a recursive write lock has to be set on the complete sub-tree and this is only possible if the workflow's write lock is cancelled. To do this, the (automatically set) write lock is temporarily removed through the status of the workflow and is reset when the delete or release option is finished (via the script). The precise procedure is described by way of an example in Chapter 4.10.1.

4.8 Using scripts in workflows

Scripts are a powerful tool for implementing customer-specific wishes within FirstSpirit workflows. As already mentioned in the description of the elements of the workflow editor, scripts within workflows can only be tied to activities (see Chapter 4.5.4.1 page 184). An activity can either be executed manually by a user, or automatically by a script (see Chapter 4.2.3.2 page 163).

The result of an activity always relates to the instance of a workflow. It is either a change in state into a subsequent state which can be reached from the activity or the retention of the current state (corresponds to the "Cancel" semantics in the activity dialog). This also applies to scripts which are coupled to this activity. The script must therefore ensure "itself" that a transition is performed in the following state.

In the workflow model, the activities with which the script is linked can either be defined as "manual" or as "automatic" – in both cases it can be useful to use a script.

! *If scripts are used within workflows, NO automatic evaluation of the editorial permissions occurs (e.g. on release). These permissions must be suitably linked to the transition permissions within the workflow (see Chapter 4.5.5.2 page 190).*

4.8.1 Automatic activities and scripts

Automatic activities do not expect any user interaction and are executed as soon as one of the upstream states in the model is reached (i.e. the action is triggered by the system and not by the user). An automatic action (and therefore the coupled script) is therefore executed immediately after reaching a state. An automatic action without script can be used, for example to automatically send an e-mail within the workflow (see Chapter 4.5.5.3 page 193).

! *The use of automatic actions can cause potential endless loops to be formed. This situation is identified by the FirstSpirit workflow interpreter, execution of the relevant workflow instance is quit and an error message appears.*

4.8.2 Manual activities and scripts

In this case, execution of the action is started by a user. If a script is not available, the standard form for workflows is displayed to the user with all the transitions authorised for them ("Activity dialog"). As soon as the action is assigned to a script, this dialog display is no longer automatic. If an activity dialog is to be displayed to the user, this must be executed via the script (see Chapter 4.8.3 page 211 and Chapter 4.8.4 page 213).

4.8.3 Workflow context

Among other things, the following methods are available for the script context for workflows:

```
Transition showActionDialog();
```

Task: Display of the activity dialog (in most cases relevant for "manual" actions only). Returns the transition selected by the user as a "Transition" object. Important: the actual transition is NOT performed (for example, see Chapter 4.8.4 page 213).

```
void doTransition(firstspirit.workflow.model.Transition transition)
```

Task: Execution of the given transition. This can, e.g. be a transition selected by the user or another transition available (and allowed) in this action. If a transition is selected which is not allowed an error message appears (for example, see Chapter 4.8.4 page 213).

```
void doTransition(String transitionName)
```

Task: Execution of the transition given with name. If a name was not assigned to the transition in the model, a name in the form"->"+“Name of the target state" is automatically formed, which can be given here (for example, see Chapter 4.8.5 page 216).

```
Transition[] getTransitions()
```

Task: Determines the set of all transitions available for the current user in the current state (for example, see Chapter 4.8.4 page 213).

```
Data getData();
```

Task: A form can be assigned to a workflow model. This form is displayed to the editor in the activity dialog and they can enter or change data (see Chapter 4.4 page 174). With this method the script has access to the content of the form and can, if necessary, also make changes (see Chapter 4.4.1 page 176).

```
Map getSession()
```

Task: (Apart from the form) a special data structure (Java Map) is assigned to each instance of a workflow, which enables a script to save and if necessary to change its own instance state. As this state is part of the workflow instance, it is available to all scripts run through during the instance's life cycle. This method (of instance-related data) can therefore not be used to exchange between scripts (for example, see Chapter 4.8.5 page 216).

Examples:

List of all possible transitions with permissions starting from the current action:

```
//!firstspirit.scripting.BeanshellWrapper

transitions = context.getTransitions();
print("Anzahl Transitionen:" + 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));
  }
}
```

Status management in workflow instances (counter):

```
//!firstspirit.scripting.BeanshellWrapper

state=context.getSession();
v=state.get("test");
if(v==null) v=0;
state.put("test",++v);
```

Generate an instance for each 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!");}
}
```

Further methods are given in the FirstSpirit Access API.

4.8.4 Example: Issue of messages in workflows

Messages can be issued (output) to the executing user within a workflow. The issue of messages is realised via scripts within the workflow. The script shows a dialog to the modifier who performs the relevant action within a workflow. This dialog can display certain information from the workflow context (see Chapter 4.8.3 page 211).

Example: Workflow "Message":

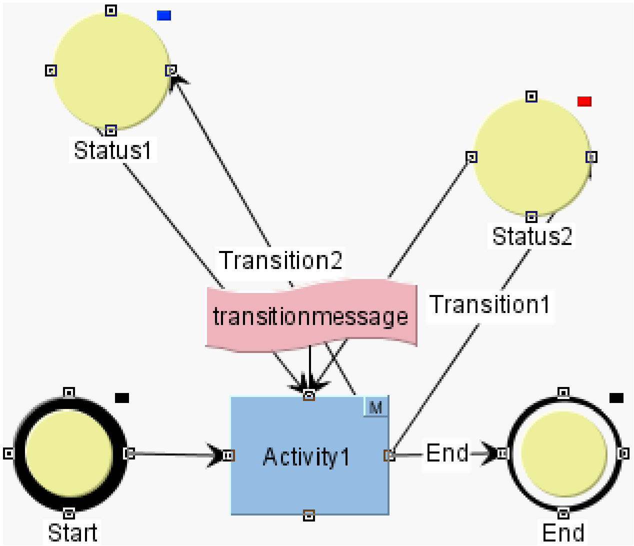
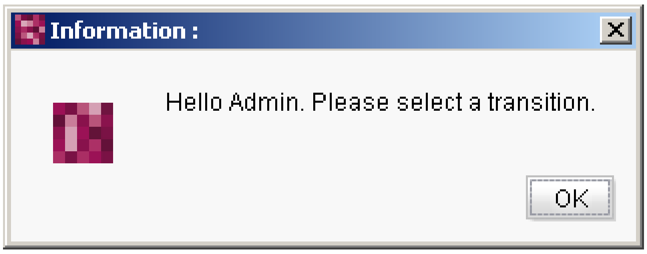


Figure 4-54: Example of workflow "Message"

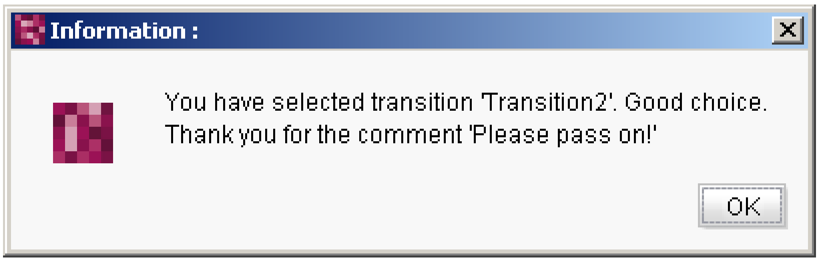
In this workflow example, an information dialog with the output "Hello $USER" is shown before and after the connection of a transition:



Hello Admin. Please select a transition.

Figure 4-55: First information dialog

After the transition has been connected, a further information dialog appears with the output "You have selected transition $TRANSITION. Thank you for the comment $KOMMENTAR":



You have selected transition 'Transition2'. Good choice.
Thank you for the comment 'Please pass on!'

Script "transitionMessage":

```
//!Beanshell
import de.espirit.firstspirit.common.gui.*;


userName =
context.getGuiHost().getUserService().getUser().getLoginName();
CMSDialog.showInfoDialog("Hello " + userName + ". Please select a
transition.");


context.showActionDialog();
transition = context.getTransitionParameters();


if (transition.getTransition() != null) {
CMSDialog.showInfoDialog("You have selected transition '" +
transition.getTransition() + "'. Good choice.\nThank you for the comment
'" + transition.getComment() + "'");
context.doTransition(transition.getTransition());
} else {
CMSDialog.showInfoDialog("You have not selected any transition.");
}
```

The information displayed to the user within the dialogs is obtained in the script via the context of the workflow (`WorkflowScriptContext`), e.g. the transition parameters (see script example):

```
context.getTransitionParameters();
```

The call:

```
CMSDialog.showInfoDialog("Hello " + userName + ". Please select a
transition.");
```

or the call:

```
JOptionPane.showMessageDialog(null,"Hello " + userName + ". Please select
a transition.");
```

can be used to realise dialogs within the script.

An unpublished (untested) API function ("CMSDialog") is used within the example. But the call can be replaced by the Java Swing class `JOptionPane`, e.g.:
`JOptionPane.showMessageDialog(null,"Hello " + userName);`

NO dialogs can be shown within a script in WebClient. If, within the workflow's script, a call of the form:
CMSDialog.showErrorDialog(...) or JOptionPane.showMessageDialog(...) is used, an error will occur if the workflow is executed in WebClient. .

4.8.5 Example: Persistent content within workflows

Content about the session can now also be saved within workflows and read out again after a transition has been connected.

Example: Workflow "Counter":

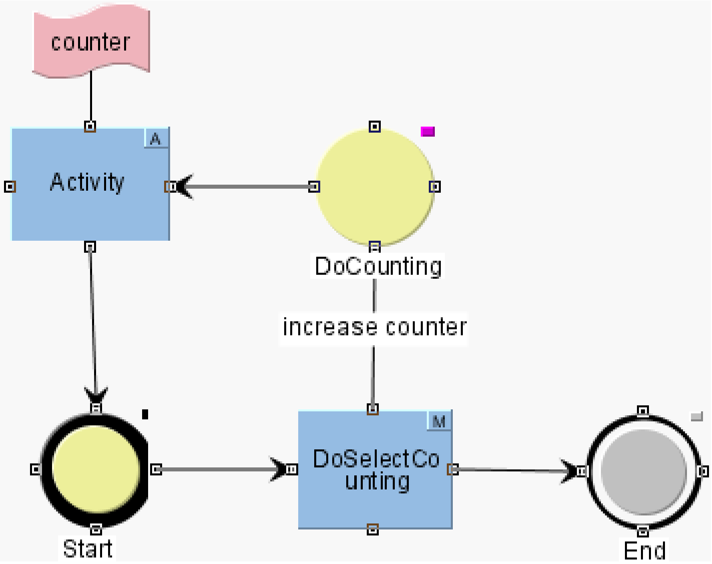
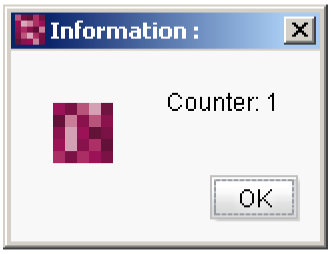


Figure 4-56: Example of workflow "Counter"

Within the "DoSelectCounting" activity, a counter can be increased by the value 1 each time the workflow is executed. The counter value is saved and is increased again by the value 1 the next time the workflow is started. The value is displayed to the modifier within an information dialog:


Information :

Counter: 1

Figure 4-57: Value of the counter

Script "Counter":

```
//!Beanshell
import de.espirit.firstspirit.common.gui.*;

session = context.getSession();
counter = session.get("counter");
if (counter == null) {
counter = new Integer(1);
}
CMSDialog.showInfoDialog("Counter: " + counter);
session.put("counter", new Integer(counter + 1));
context.doTransition("->Start");
```

An unpublished (untested) API function ("CMSDialog") is used within the example. But the call can be replaced by the Java Swing class `JOptionPane`, e.g.:
`JOptionPane.showMessageDialog(null,"Hello " + userName);`

4.9 Delete via a workflow (from V4.1)

This function is released for FirstSpirit Version 4.1 and higher only.

4.9.1 Introduction (from V4.1)

A project-specific workflow can be created and tied directly to the delete controls available to date (menu bar buttons, context menu entry) to delete elements in the FirstSpirit JavaClient and in the FirstSpirit WebClient. Instead of simply deleting an object, for example a page, a more complex delete function can be made available via the workflow, for example the additional deletion of dependent objects of a page (for demo workflow, see Chapter 4.9.3).

Deleting via a workflow is only available if the project has been configured accordingly by the project administrator.

The new workflow is then started within the client using the familiar control elements. The individual tasks of the workflow appear, as usual, in the task list (see Chapter 4.9.2 page 218).

If deletion via a workflow is configured within a project, the permissions configuration for the workflow must be adjusted. The conventional editorial permissions for deleting, which are defined for a user or a group, only take effect if the permissions configuration is adjusted accordingly in the workflow (see Chapter 4.9.4 page 220).

4.9.2 Delete via a workflow in the JavaClient (from V4.1)

If the deletion of elements in the project has been tied to a workflow, the workflow can be started or forwarded in the JavaClient using the conventional control for deleting. The following controls are available for this:

- Select element and click <Del> key.
- Select element and click the "Delete" context menu entry

- Select element and click the icon in the icon bar

Analogous to the multiple selection of workflows, it is possible to delete a set of objects at the same time via a workflow (see Figure 4-58 and Chapter 4.11 page 236).

! *The workflow can only be started if no workflows have been started to date on one of the selected objects and the user has the relevant permissions for executing the workflow. Otherwise the relevant controls are disabled.*

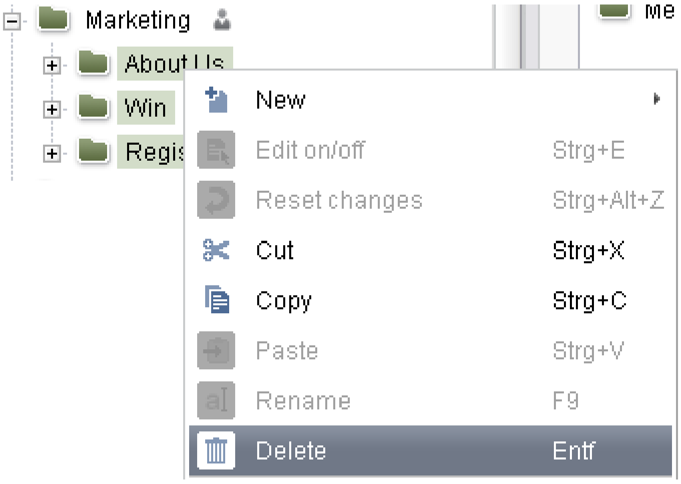


Figure 4-58: Multiple selection on deleting via a workflow

! *The "Remove" permission is also evaluated if elements are deleted via a workflow. If a user has permission to switch the workflow but NOT permission to remove elements, the workflow can be started (context menu entry "Delete" is activated), but the element cannot be deleted. The transition which deletes the element is not displayed to this user.*

4.9.3 Delete via a workflow in the WebClient (from V4.1)

If the deletion of elements in the project has been tied to a workflow, the workflow can be started or forwarded in the WebClient using the conventional control for deleting.

The following controls are available for this:

- Delete Select the element in the store overview in the WebClient and click the "Delete" button.

-

Open Quick Edit bar and click the "Delete Page" button

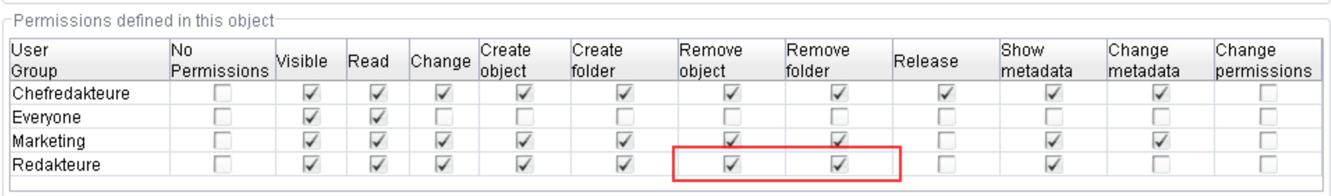
> ! *The workflow can only be started if no workflows have been started to date on one of the selected objects and the user has the relevant permissions for executing the workflow. Otherwise the relevant controls are disabled.*

> ! *NO dialogs can be shown within a script in the WebClient. If, within the workflow's script, a call of the form: CMSDialog.showErrorDialog(...) or JOptionPane.showMessageDialog(...) is used, an error will occur if the workflow is executed in the WebClient.*

4.9.4 Permissions configuration (from V4.1)

Permissions are assigned in the FirstSpirit JavaClient. Here, all areas of the project can be assigned permissions for specific groups or users (see Chapter 4.6 page 194).

Permissions to delete elements (without workflow) are usually defined via the so-called editorial permissions. Editorial permissions are defined for a user or a group on the respective element. Certain permissions can be assigned in this way for all editorial work. Apart from "Visible" or "Change" these also include, for example, the permission to "Remove object" and the permission to "Remove folder".



Permissions defined in this object

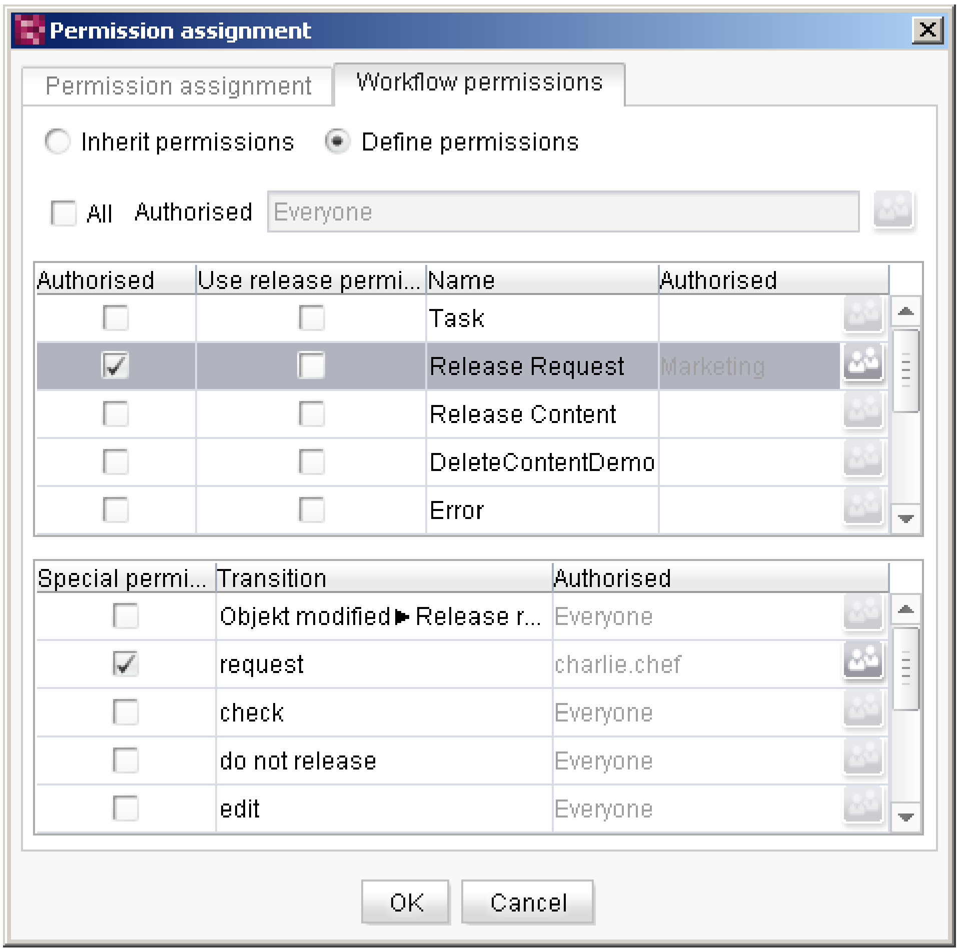
User Group	No Permissions	Visible	Read	Change	Create object	Create folder	Remove object	Remove folder	Release	Show metadata	Change metadata	Change permissions
Chefredakteure	☐	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☐
Everyone	☐	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐
Marketing	☐	☑	☑	☑	☑	☑	☑	☑	☐	☑	☑	☐
Redakteure	☐	☑	☑	☑	☑	☑	☑	☑	☐	☑	☐	☐

Figure 4-59: Editorial permissions "Remove object" and "Remove folder"

For further information on editorial permissions, see FirstSpirit Manual for Editors, Chapter 13.1.

These editorial permissions do not take effect automatically if deleting is tied to a workflow. If these permissions are to be evaluated, the permissions configuration must be adjusted in the workflow first (see Figure 4-61).

If deleting in a project is handled via a workflow, the permissions configuration must be moved to the workflow. The permissions to execute workflows are assigned at the same time as the editorial permissions for groups and users in the "Permission assignment" dialog within the stores in the FirstSpirit JavaClient:



○ Inherit permissions ● Define permissions

□ All Authorised Everyone

Authorised	Use release permi...	Name	Authorised
□	□	Task	
☑	□	Release Request	Marketing
□	□	Release Content	
□	□	DeleteContentDemo	
□	□	Error	

Special permi...	Transition	Authorised
□	Objekt modified ▶ Release r...	Everyone
☑	request	charlie.chef
□	check	Everyone
□	do not release	Everyone
□	edit	Everyone

Figure 4-60: Rights to execute workflows

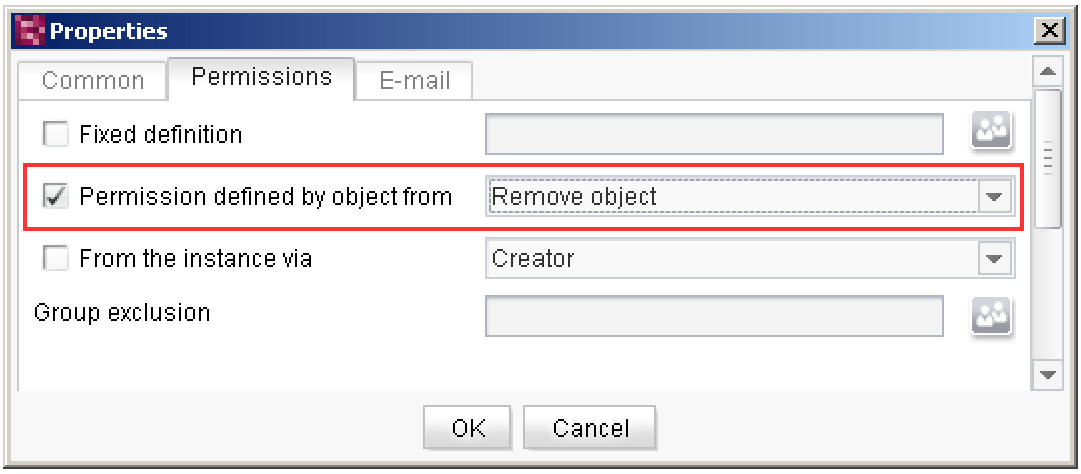
The permissions defined for the execution of workflows in the top dialog area ("authorised") solely refer to the starting of the respective workflow (see Chapter 4.6.3 page 199).

The permissions for executing a transition (from one step in the workflow to the next step) are either:

- defined by the template developer in the workflow (see Chapter 4.6.1 page 194)
- by the assignment of "Special permissions" for the individual steps of a workflow (see Figure 4-60) (see Chapter 4.6.4 page 201)

For further information on permissions for executing workflows, see FirstSpirit Manual for Editors, Chapter 13.2.

If the conventional editorial permissions ("Remove folder", "Remove object") and the permissions for executing the workflow are to be linked to each other in a suitable way, the permissions configuration within the workflow should be adjusted. Within a workflow, permissions are issued in the individual transitions (see Chapter 4.5.5.2 page 190). This ensures that each individual activity can be performed by authorised users only. The permissions dialog is opened by double-clicking the transition in the workflow model. The permissions for connecting the transition can be assigned in the "Permissions" tab:



□ Fixed definition

☑ Permission defined by object from: Remove object

□ From the instance via: Creator

Group exclusion

Figure 4-61: Link editorial permissions and transition permissions

If the "Permission defined by object from" option is selected, the authorised users result from the editorial permissions defined in the tree structure of the JavaClient. This field can be used to select the permission the user must have on the object under consideration to be able to perform this transition. If the "Remove object" or "Remove folder" permission is selected here, when the workflow is started the system examines whether the user has the permission to "Remove" on the element. The permissions are then evaluated analogous to conventional deletion without a workflow.

<u>Special case "Delete objects":</u> If an object is deleted via a workflow in combination with the permission configuration via the "Permission defined by object from", a

special case takes effect. If the element is deleted, the permissions can no longer be determined from the object. In the example from 4.9.5, the object and therefore the permissions defined on the object are no longer available on the last transition "end". In this case: "anything is always allowed" on a deleted object. If this is not wanted, the permissions configuration for the corresponding transitions must be changed (e.g. to "fixed definition" groups or users).

> ! *If "Special permissions" for switching a workflow to the next level are defined on an element (see Figure 4-60), the permissions defined by the template developer for this workflow are overwritten as a result.*

4.9.5 Example: Workflow "Delete" (from V4.1)

The workflow for deleting elements consists of the workflow and the corresponding "clientdelete" script (for deleting individual objects) and "serverdelete" script (for deleting sub-trees).

If the "clientdelete" action is executed, the element is locked by the corresponding script and is then deleted. After the deletion the workflow is forwarded to the following "End" status.

If the "serverdelete" action is executed, the element is recursively locked by the corresponding script. The "serverdelete" action can therefore be used not only to delete an individual element, but sub-trees also. Here the deletion is performed via a ServerHandle, which returns a results report and in case of an error throws an exception.

After the successful deletion the workflow is forwarded to the following "End" status. The following applies to both actions: in case of an error, the workflow is not switched to end status but instead is switched to an error status modelled in the workflow.

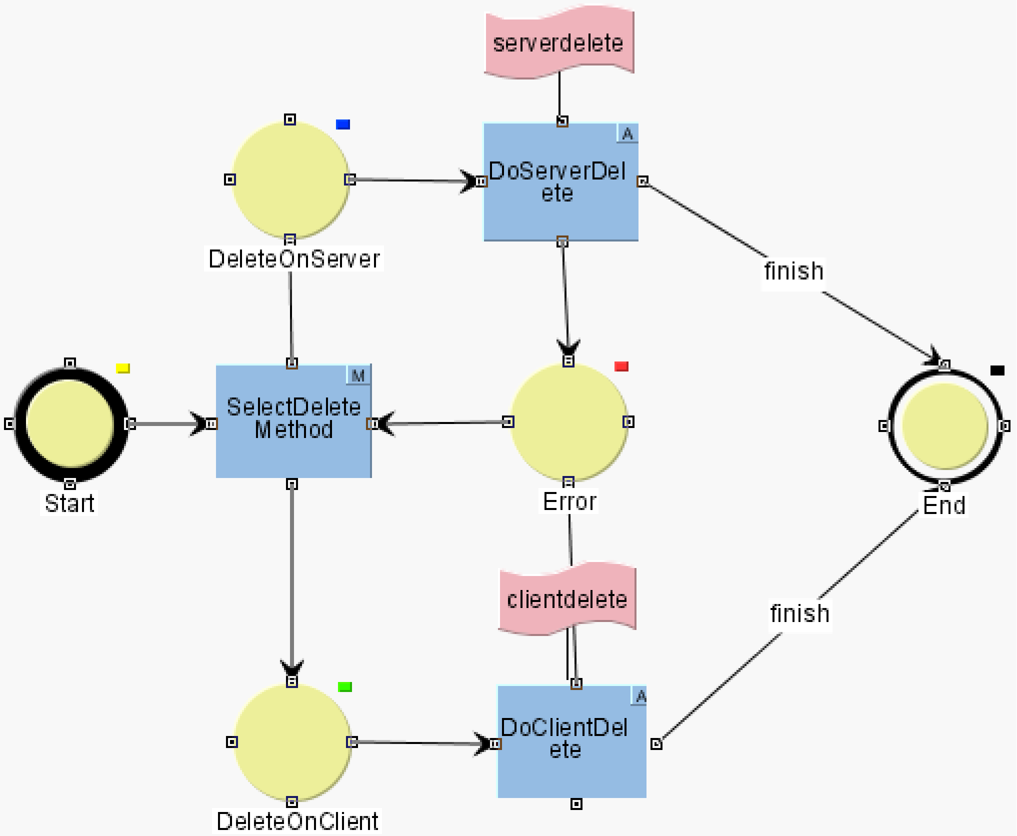


Figure 4-62: "Delete" workflow example

Script "clientdelete":

```
//!Beanshell
import de.espirit.firstspirit.common.gui.*;
se = context.getStoreElement();
try {
se.delete();
context.doTransition("->End");
} catch (Exception ex) {
CMSDialog.showErrorDialog(null, "Error while deleting: " + ex,
ex);
context.getSession().put("error", ex.toString());
context.doTransition("->Error");
}
```

An unpublished (untested) API function ("CMSDialog") is used within the example. But the call can be replaced by the Java Swing class `JOptionPane`, e.g.:
`JOptionPane.showMessageDialog(null,"Hello " + userName);`

Script "serverdelete":

```
//!Beanshell
import de.espirit.firstspirit.common.gui.*;

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()) {
          CMSDialog.showErrorDialog("The following elements could
not be deleted: " + notDeleted);
}
if (parent != null) {
          parent.refresh();
          context.getGuiHost().gotoTreeNode(parent);
}
if (!se.isDeleted()) {
          se.setLock(true, false);
}
context.doTransition("->End");
} catch (Exception ex) {
```

```
CMSDialog.showErrorDialog(null, "Error while deleting: " + ex,
ex);
context.getSession().put("error", ex.toString());
context.doTransition("->Error");
}
```

4.9.6 Example: Workflow "ContentDeleteDemo" (from V4.1)

Apart from deleting individual elements and sub-trees (see Chapter 4.9.5 page 223), it is also possible to use a workflow to delete structured data.

To do this, a differentiation must be made within the workflow (in the script) between normal elements ("StoreElements", e.g. pages, media, page references) and "entities".

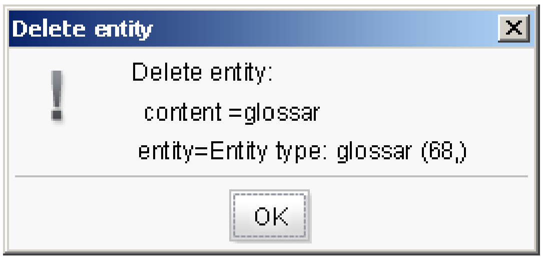
In the script, this information is obtained from the context of the workflow (`WorkflowScriptContext`) (see Chapter 4.8.3 page 211):

```
workflowable = context.getWorkflowable()
```

The getWorkflowable() method returns whether the element on which the workflow was started is a StoreElement, for example a medium, or an entity, in the form of a data record (see script example). For example, the output of the script can be adjusted accordingly:

```
if (workflowable instanceof ContentWorkflowable) {
...
} else {
...
}
```

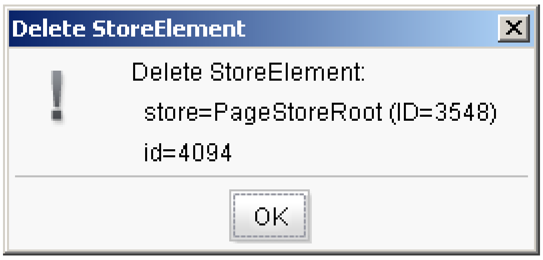
In the example the output is controlled depending on the context on which the workflow was started. If the delete function is started on a data record, the script delivers the following output:



Delete entity:
content =glossar
entity=Entity type: glossar (68,)

Figure 4-63: Delete entity

On a "StoreElement" it delivers the output:



Delete StoreElement

Delete StoreElement:
store=PageStoreRoot (ID=3548)
id=4094

Figure 4-64: Delete StoreElement

In this example the deletion also takes place directly via the WorkflowScriptContext (see Chapter 4.8.3 page 211):

```
workflowable.delete();
```

Here the delete method is called on the `Workflowable` object and not, as in the example from Chapter 4.9.5, on the `StoreElement`. This delete method can be used to delete both a `StoreElement` and a data record (`Entity`).

The workflow for deleting entities consists of the workflow and the corresponding "deletecontentdemo" script (for deleting individual entities)

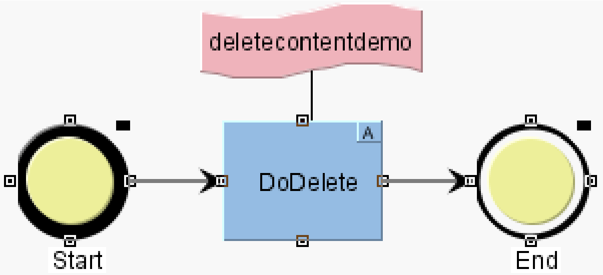


Figure 4-65: Example of "DeleteContentDemo" workflow

Following the successful deletion the workflow is automatically forwarded to the following "End" status.

Script ("deletecontentdemo"):

```
//!Beanshell
import de.espirit.firstspirit.access.*;
import de.espirit.firstspirit.access.store.contentstore.*;

workflowable = context.getWorkflowable();
if (workflowable instanceof ContentWorkflowable) {
message = "Delete entity:\n  content=" +
workflowable.getContent().getName() + "\n  entity=" +
workflowable.getEntity().getKeyValue();

JOptionPane.showMessageDialog(null, message, "Delete entity",
JOptionPane.WARNING_MESSAGE);

} else {

message = "Delete StoreElement:\n  store=" +
workflowable.getStore().getName() + "\n  id=" +
workflowable.getId();

JOptionPane.showMessageDialog(null, message, "Delete
StoreElement", JOptionPane.WARNING_MESSAGE);

}

workflowable.delete();

context.doTransition("->End");
```

4.10 Workflows with complex functions

Very complex functions can be realised by using the modelling of workflows and the use of BeanShell scripts within these models. A good example is the previously described workflow for deleting elements (see Chapter 4.9.5 page 223).

Within the scripts, actions are executed on specific project content via the Access API of FirstSpirit. Unlike the standard functions (e.g. the standard context menu entry "Delete"), the developer of the workflow (or the corresponding scripts) must ensure that all the necessary boundary conditions, for example for deleting an element, are also covered by the script. For most actions, this requires at least one write lock on the element concerned. However, the type of element to be executed and on which elements it is to be executed is decisive. For example, when a simple element is deleted, e.g. a medium, it is not necessary to set a write lock; however, a lock is required for recursive deleting, e.g. a page with sections (cf. Chapter 4.9.5 page 223).

The following chapters deal with the write lock within workflows and explain examples of workflows with complex functions.

4.10.1 Example: "RecursiveLock" workflow

This workflow for recursive locking of sub-trees consists of the workflow and the corresponding "lockrecursive" script.

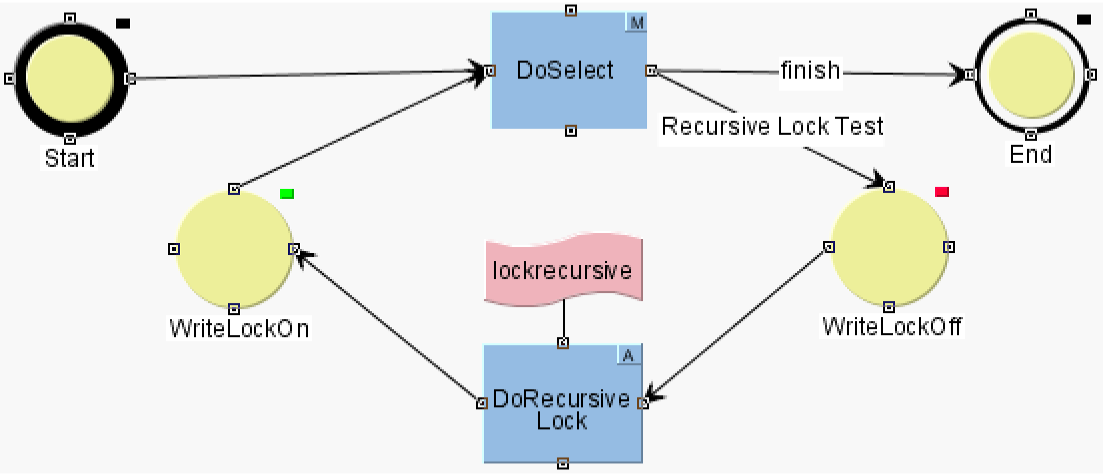
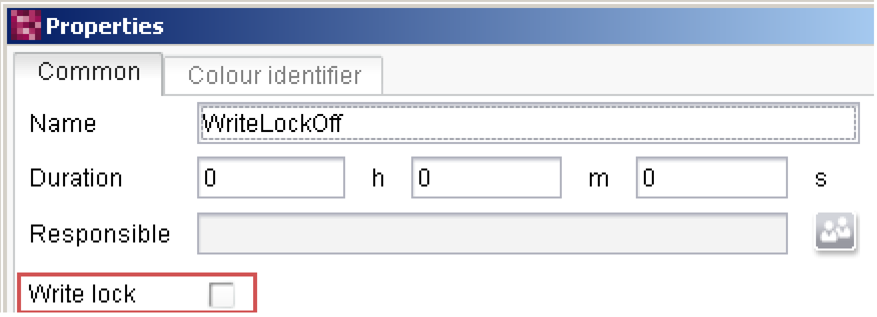


Figure 4-66: "RecursiveLock" workflow example

Within the script, a recursive write lock is executed on the element on which the workflow was started. In order for this to succeed, the automatically set write lock of the workflow must be cancelled first. To do this, the write lock is removed first in the "WriteLockOff" status:



Name: WriteLockOff
Duration: 0 h 0 m 0 s
Responsible:
Write lock ☐

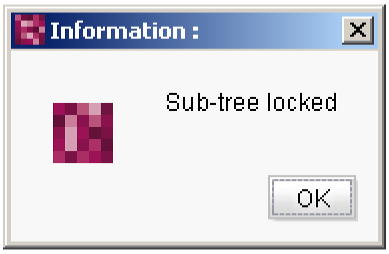
Figure 4-67: Remove write lock via the workflow's status

The "Write Lock" checkbox is deselected, the workflow's write lock is now cancelled on connecting the "Recursive Lock Test" transition. The following "DoRecursiveLock" action is executed automatically and is linked to the "lockrecursive" script.

The script can now be used to set a recursive write lock on the element:

```
// set recursive lock
se = context.getStoreElement();
se.setLock(true);
CMSDialog.showInfoDialog("Sub-tree locked");
```

The elements are recursively locked, a dialog is displayed to the editor with the message "Sub-tree locked":



Sub-tree locked

Figure 4-68: Write lock set on the sub-tree

When the message is confirmed, execution of the script is continued, the recursive write lock on the sub-tree is cancelled:

```
// reset recursive lock
se.setLock(false);
```

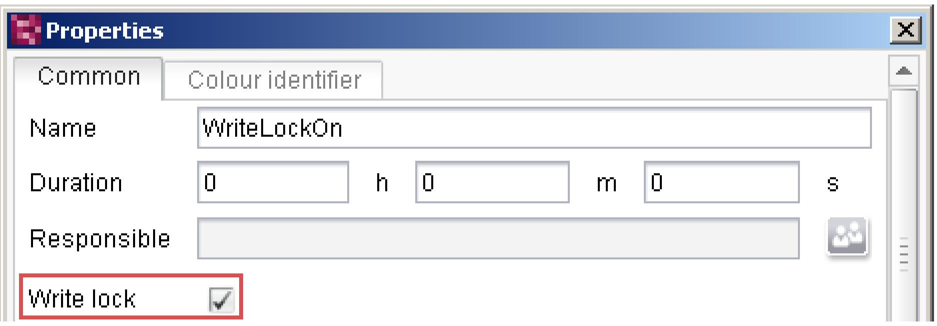
In the next step the simple write lock must be restored on the element:

```
// non recursive lock, normal state during workflow
```

```
se.setLock(true, false);
context.doTransition("->WriteLockOn");
```

This is necessary to enable connection of the following transition within the workflow.

The standard write lock of the workflow must then be restored. To this end, the "write lock" checkbox is re-selected in "WriteLockOn" status:



Name: WriteLockOn
Duration: 0 h 0 m 0 s
Responsible:
Write lock ☑

Figure 4-69: Set write lock via the workflow's status

Script "lockrecursive":

```
//!Beanshell
import de.espirit.firstspirit.common.gui.*;

// set recursive lock
se = context.getStoreElement();
se.setLock(true);
CMSDialog.showInfoDialog("Sub-tree locked");

// reset recursive lock
se.setLock(false);

// non recursive lock, normal state during workflow
se.setLock(true, false);
context.doTransition("->WriteLockOn");
```

! *An unpublished (untested) API function ("CMSDialog") is used within the example. But the call can be replaced by the Java Swing class `JOptionPane`, e.g.:*
`JOptionPane.showMessageDialog(null,"Hello " + userName);`

4.10.2 Example: "RecursiveRelease" workflow

This workflow for recursive release consists of the workflow and the corresponding "serverrelease" script.

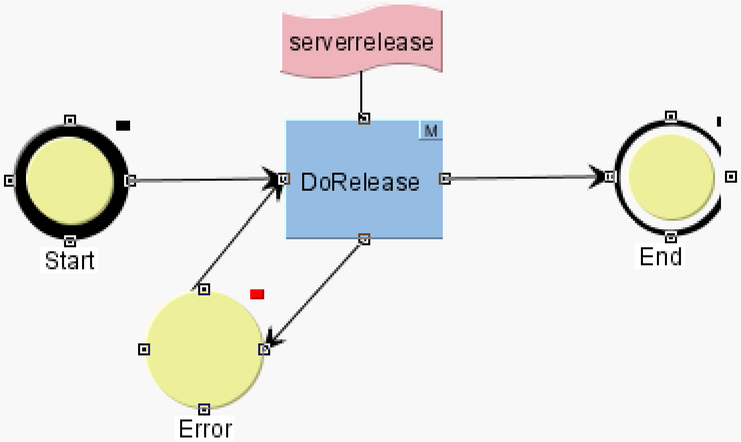


Figure 4-70: "RecursiveRelease" workflow example

The element on which the workflow was started and all appended elements are to be recursively released within the workflow.

The server side release used within the "serverrelease" script controls both the release and the internal setting of the write lock for the element concerned. If elements are found which already have a write lock the server-side release cannot be executed. These elements can be called via the return value of the server-side release (in test mode only):

```
handle.getProgress(true).getLockFailedElements()
```

Therefore, a recursive write lock does not have to be set for the server-side release via the script. However, in order for the release to be given, there must be no write locks set by the workflow. The write lock on the element is therefore first cancelled via the script:

```
    se.setLock(false, false);
```

Then the server-side release is executed by calling the method:

```
AccessUtil.release(IDProvider releaseStartNode, boolean checkOnly,
boolean releaseParentPath, boolean recursive,
IDProvider.DependentReleaseType dependentType)
```

In the example, the following transfer parameters are set for the server-side release:

```
handle = AccessUtil.release(se, false, false, true,
de.espirit.firstspirit.access.store.IDProvider.DependentReleaseTyp
e.DEPENDENT_RELEASE_NEW_AND_CHANGED);
```

Transferred parameters:

`releaseStartNode`: Start node for the release

`checkOnly`: If the value `true` is transferred the specific release is only tested.

`releaseParentPath`: If the value `true` is transferred, the complete parent chain of the object to be released is determined and all previously never released objects are also released.

`recursive`: If the value `true` is transferred, all children elements of the object to be released are determined and recursively and are also released.

`dependentType`: This parameter is used to determined and release dependent objects of the object to be released. For example, if a medium is referenced on one page, this medium can also be directly released on the specific release of the page. The following dependencies can be taken into account:

- `DEPENDENT_RELEASE_NEW_AND_CHANGED`: new and changed dependent objects are taken into account.
- `DEPENDENT_RELEASE_NEW_ONLY`: new created (never yet released objects) only are taken into account
- `NO_DEPENDENT_RELEASE`: dependent objects are not taken into account and if necessary must be released separately (default setting).

The different release options can be combined with each other in any way necessary to realise extensive release within a short time. However, under certain circumstances the release of all objects involved in the release process may not be wanted in all cases and should therefore be executed circumspectly.

For further information on server-side release, see "FirstSpirit Manual for Developers (Part 2: Advanced)".

Returned parameters:

```
ServerActionhandle<? extends ReleaseProgress,Boolean >
```

The server-side release returns a `ServerActionHandle`, which contains all information about the release process.

Within the script example, the result of the release process is queried first:

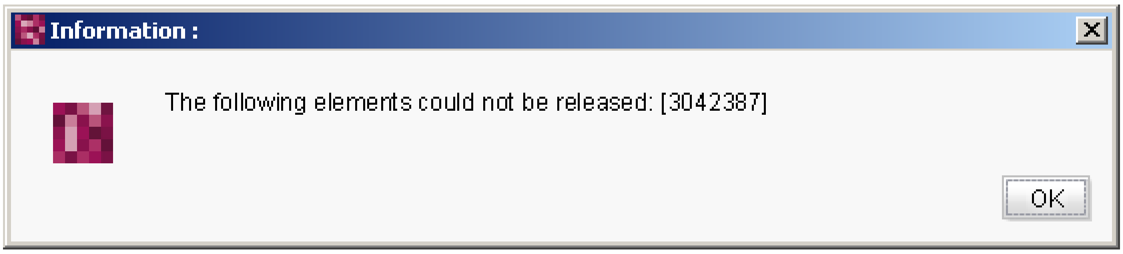
```
handle.getResult();
handle.checkAndThrow();
```

The errors during the release are then examined. If elements cannot be released, for example, because a write lock existed on the element or the modifier did not have the relevant permissions to release an element, these can be called 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 modifier the elements which could not be released:

```
if (!notReleased.isEmpty()) {
         CMSDialog.showErrorDialog("The following elements could
not be released: " + notReleased);
}
```



The following elements could not be released: [3042387]

Figure 4-71: Error message – Unreleased elements

> ! *The error message is displayed in test mode only ("`checkOnly`").*

Script: "serverrelease":

```
//!Beanshell
import de.espirit.firstspirit.common.gui.*;
import de.espirit.firstspirit.access.*;
import de.espirit.firstspirit.access.store.*;

se = context.getStoreElement();
try {
se.setLock(false, false);
handle = AccessUtil.release(se, false, false, true,
de.espirit.firstspirit.access.store.IDProvider.DependentReleaseTyp
e.DEPENDENT_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()) {
          CMSDialog.showErrorDialog("The following elements could
not be released: " + notReleased);
}


se.refresh();
context.getGuiHost().gotoTreeNode(se);
se.setLock(true, false);
context.doTransition("->End");


} catch (Exception ex) {
CMSDialog.showErrorDialog(null, "Error during release: " + ex,
ex);
context.getSession().put("error", ex.toString());
context.doTransition("->Error");
```

An unpublished (untested) API function ("CMSDialog") is used within the example. But the call can be replaced by the Java Swing class `JOptionPane`*, e.g.:*

`JOptionPane.showMessageDialog(null,"Hello " + userName);`

4.11 Multiple selection of workflows (from V4.1)

! *This function is released for FirstSpirit Version 4.1 and higher only.*

4.11.1 General information on multiple selection in FirstSpirit

The ability to make multiple selections of elements is available within many dialogs in FirstSpirit. This means a large number of elements can be selected and edited in a simple way. For example, multiple selection is possible within the tree view in the FirstSpirit JavaClient (see Figure 4-72). This enables several elements to be selected on which a certain action (e.g. move, copy, delete) can then be executed on them all at the same time.

Multiple selections can be made by simultaneously pressing the SHIFT and CTRL key. In addition, the CTRL + A key combination can be used to select all visible elements of a store (within the tree view) or all elements within a table (e.g. within the Task List).

Within the tree view, the multiple selection of elements is restricted to the respective current store. Therefore, if an element is already selected, e.g. in the Page Store, an element from another store cannot then 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 selected. For example, if a folder of the Page Store is not expanded, the pages below it are not included in the selection.*

4.11.2 Multiple selection of workflows (from V4.1)

Multiple-selection of workflows enables a workflow to be started and switched on a set of objects.

The required objects for this can be selected within the tree view (see Chapter 4.11.1 page 236). The context menu is then opened as usual and the required workflow is selected:

This function is released for FirstSpirit Version 4.1 and higher only.

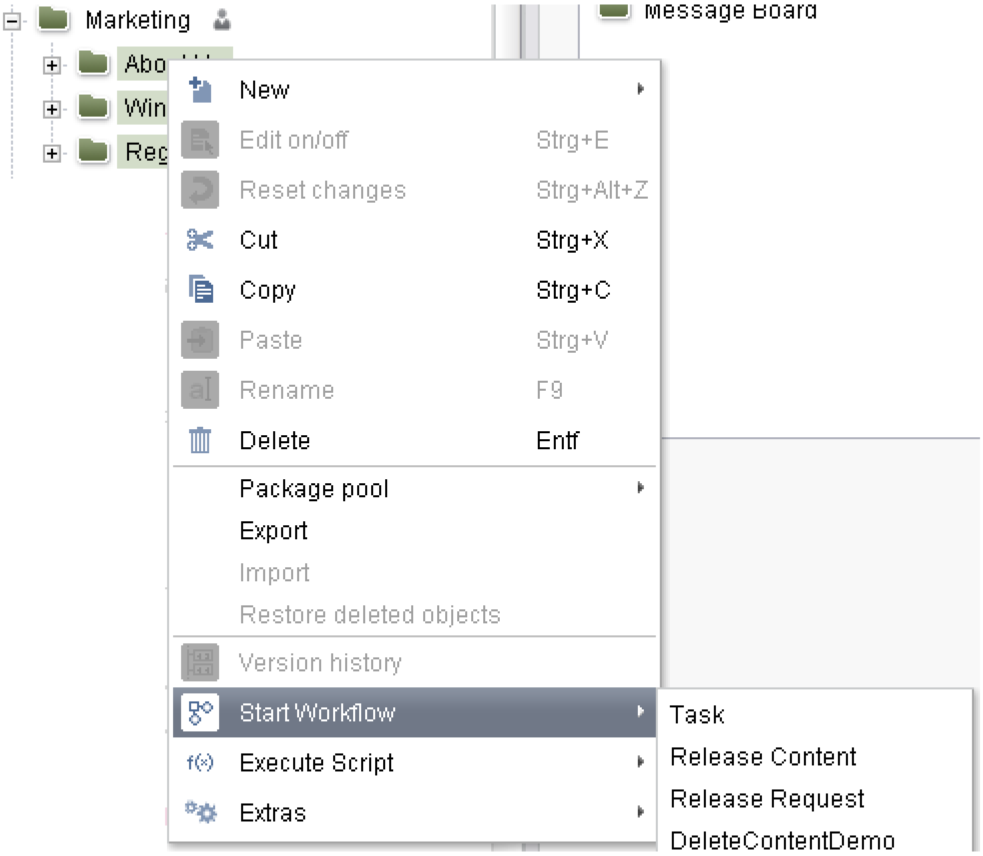


Figure 4-72: Starting a workflow on several elements

4.11.3 Requirements for starting and advancing workflows (from V4.1)

This function is released for FirstSpirit Version 4.1 and higher only.

The starting or switching of a workflow can only take place if all elements have the same workflow status or no workflow has yet been started to date on the selected elements (see Figure 4-72).

With the multiple selection of elements, it is determined for each element, which workflows and which transitions of a workflow can be displayed via the context menu. At the same time it is also, for example, taken into account whether:

- a workflow has already been started on the element,
- the user has the necessary permissions to start the element on this workflow,
- a workflow may be started on this element,
- a workflow has already been started on the elements but the elements have not reached the same workflow status.

If these requirements are not fulfilled for only one element in the multiple selection, the context menu returns the calculation "not available" for all the selected elements.

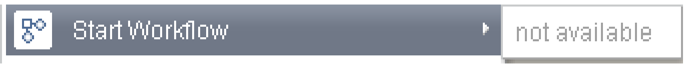


Figure 4-73: Context menu – not available

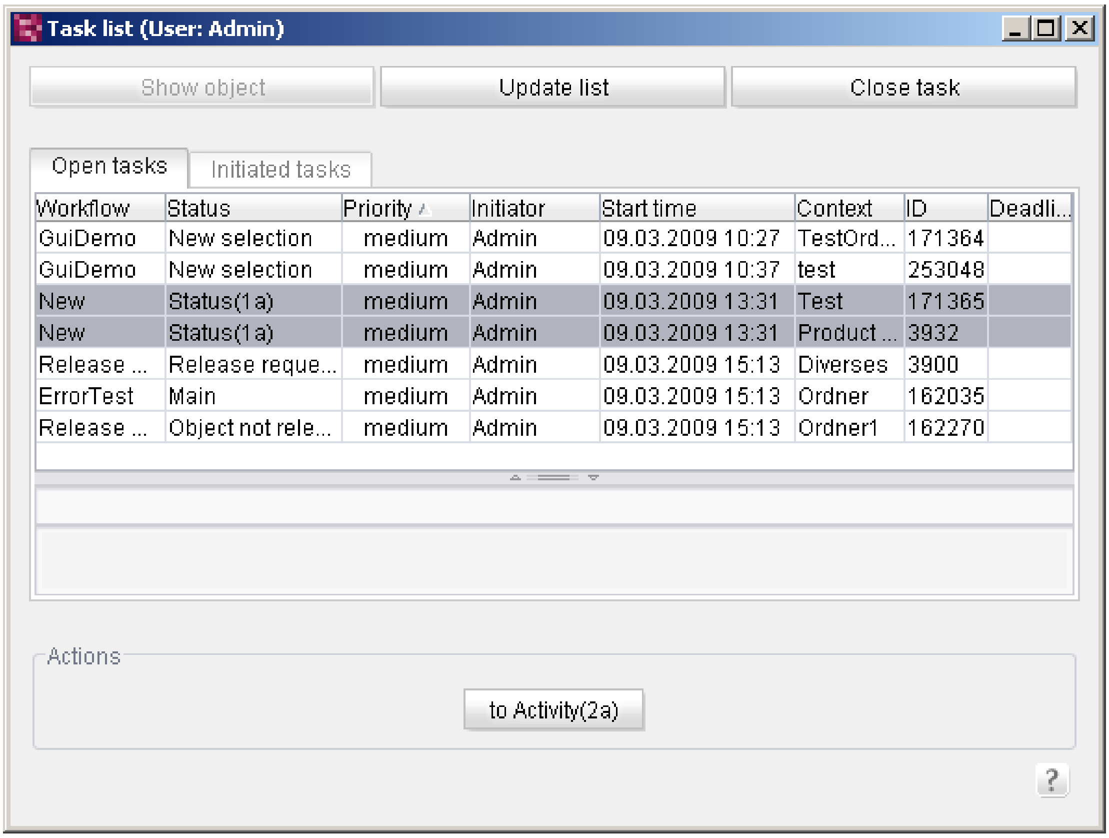
In this case, the multiple selection should be cancelled, the individual elements rechecked and if necessary reselected.

4.11.4 Multiple selection via the task list (from V4.1)

This function is released for FirstSpirit Version 4.1 and higher only.

Apart from multiple selection via the tree view, workflows which have already been started can also be forwarded using the task list (see Figure 4-74) or the "Workflows" overview in the Template Store (see Chapter 4.1 page 152).

The task list displays all not yet completed tasks ("Open tasks") and all started tasks ("Initiated tasks") within a tabular view. Apart from the name of the workflow, the current status of the respective workflow instance is also displayed.



Workflow	Status	Priority	Initiator	Start time	Context	ID	Deadli...
GuiDemo	New selection	medium	Admin	09.03.2009 10:27	TestOrd...	171364	
GuiDemo	New selection	medium	Admin	09.03.2009 10:37	test	253048	
New	Status(1a)	medium	Admin	09.03.2009 13:31	Test	171365	
New	Status(1a)	medium	Admin	09.03.2009 13:31	Product ...	3932	
Release ...	Release reque...	medium	Admin	09.03.2009 15:13	Diverses	3900	
ErrorTest	Main	medium	Admin	09.03.2009 15:13	Ordner	162035	
Release ...	Object not rele...	medium	Admin	09.03.2009 15:13	Ordner1	162270	

Figure 4-74: Multiple selection via the task list

Several tasks can be selected within the tabular task list. Provided these tasks have the same workflow and the same status and the user has the necessary permissions to execute the workflow for the selected elements, the possible actions are displayed in the bottom part of the task list.

In this way the tasks can be forwarded in parallel (i.e. at the same time). Unlike starting via the tree view, several elements from different stores can also be selected in the task list and switched in parallel.

4.11.5 Multiple selection via the "workflows" overview (from V4.1)

This function is released for FirstSpirit Version 4.1 and higher only.

Apart from the task list, a further overview of all tasks started to date exists on the "Workflows" node within the Template Store (see Chapter 4.1 page 152). Unlike the task list, the tasks here can be filtered by specific search criteria and even already closed tasks can be displayed (see Chapter 4.1.1 page 154).

Several tasks can be selected within the overview. Direct advancing of the workflows is not possible here; however, clicking the "Edit" button opens the task list (see Chapter 4.1.2 page 156). The elements previously selected in the overview are now directly selected in the task list and can be forwarded from there (see Chapter 4.11.4 page 238).

Apart from editing the overview can also be used to close several selected tasks (see Chapter 4.1.3 page 159).

5 Document Groups

5.1 Introduction

5.1.1 Objective

The concept of FirstSpirit fulfils the paradigm of separation of structure, content and display of a website. The individual areas can be changed independently of each other and content can be reused at any time. For example, content in the form of pages with variable section lists are maintained in the Page Store. These pages are then grouped together in the Site Store to form structures (e.g. page groups, menu level).

This basic structure is appropriate for the generation of websites and individual PDF documents, because exactly one results document (e.g. an HTML or a PDF file) is generated for each FirstSpirit page.

But in several cases, this "1:1" correspondence is not wanted, for example for creating the PDF print version of an extensive company description. A single PDF document is to be generated for this, which contains a complete sub-tree of the Site Store. In this case, several FirstSpirit pages are grouped together to form a single document. Simple implementation of this case is made possible by the concept of the FirstSpirit document group.

5.1.2 Concept

Document groups can be created within the FirstSpirit Site Store. Unlike the other elements of the Site Store (e.g. menu levels), a document group can be selected as a link target, but it is not part of the navigation. (For this reason, document groups can also not be selected as the startpage of a menu level.)

A document group can contain page references and menu levels of the Site Store. The advantage of the support of menu levels is that new pages are "automatically" added to the document group when they are transferred into the selected menu level of the Site Store.

As other templates usually have to be used to generate a document from a set of individual documents, it is necessary to either define a specific template for the document group or to adjust the existing templates so that both individual documents and document groups are generated correctly (see Chapter 5.3.4 page 247).

A further special feature of the document group is the optical restriction to a fixed presentation channel. This means a document group is generated as a PDF only (i.e. only 1x and for all presentation channels). Through this enhancement, it is simultaneously possible to realise a "few PDF documents", without having to generate a complete presentation channel (see Chapter 5.3.5 page 249).

5.2 Configuration

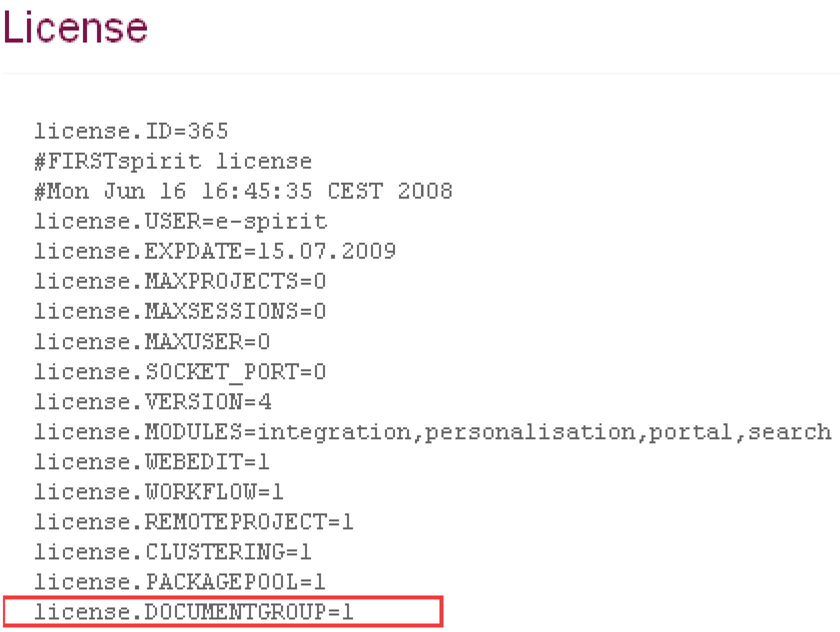
5.2.1 Check licence file

Use the "FirstSpirit – Configuration – Licence" menu of the FirstSpirit Server Monitoring to display the valid FirstSpirit functions of the `fs-license.conf` licence file. The `license.DOCUMENTGROUP` parameter must be set to the value `1` for use of the function (see Figure 5-1).

If not, a new valid licence can be requested from the manufacturer and added in the blue part of the window. The new licence file can be saved by clicking the Save button.

Manipulating the `fs_license.conf` results in an invalid licence. If changes become necessary, please contact the manufacturer.

When adding a new configuration file `fs_license.conf` it is not necessary to restart the server. The file is automatically updated on the server.



License

```
license.ID=365
#FIRSTspirit license
#Mon Jun 16 16:45:35 CEST 2008
license.USER=e-spirit
license.EXPDATE=15.07.2009
license.MAXPROJECTS=0
license.MAXSESSIONS=0
license.MAXUSER=0
license.SOCKET_PORT=0
license.VERSION=4
license.MODULES=integration,personalisation,portal,search
license.WEBEDIT=1
license.WORKFLOW=1
license.REMOTEPROJECT=1
license.CLUSTERING=1
license.PACKAGEPOOL=1
license.DOCUMENTGROUP=1
```

Figure 5-1: Display of the licence file parameters (Server-Monitoring)

5.3 Using document groups in the FirstSpirit JavaClient

5.3.1 Create new document groups

Document groups are created using the context menu of the Site Store (or using the keyboard shortcut Ctrl + D):

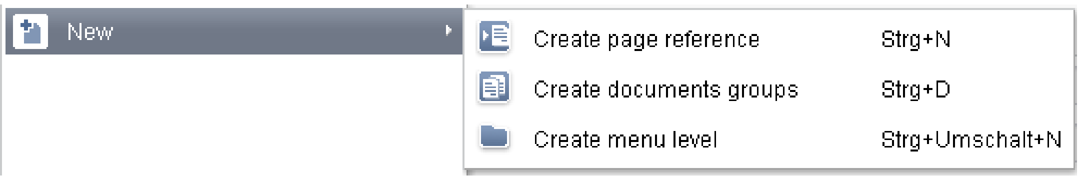
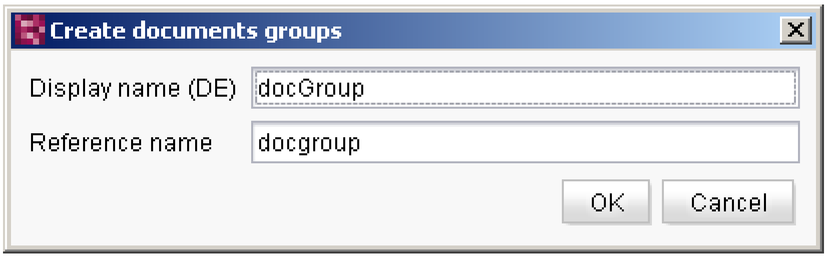

New
Create page reference Strg+N
Create documents groups Strg+D
Create menu level Strg+Umschalt+N

Figure 5-2: Create new document group (context menu)

A dialog window for creating a new document group opens:



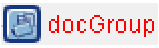
Display name (DE): docGroup

Reference name: docgroup

Figure 5-3: Create a new document group

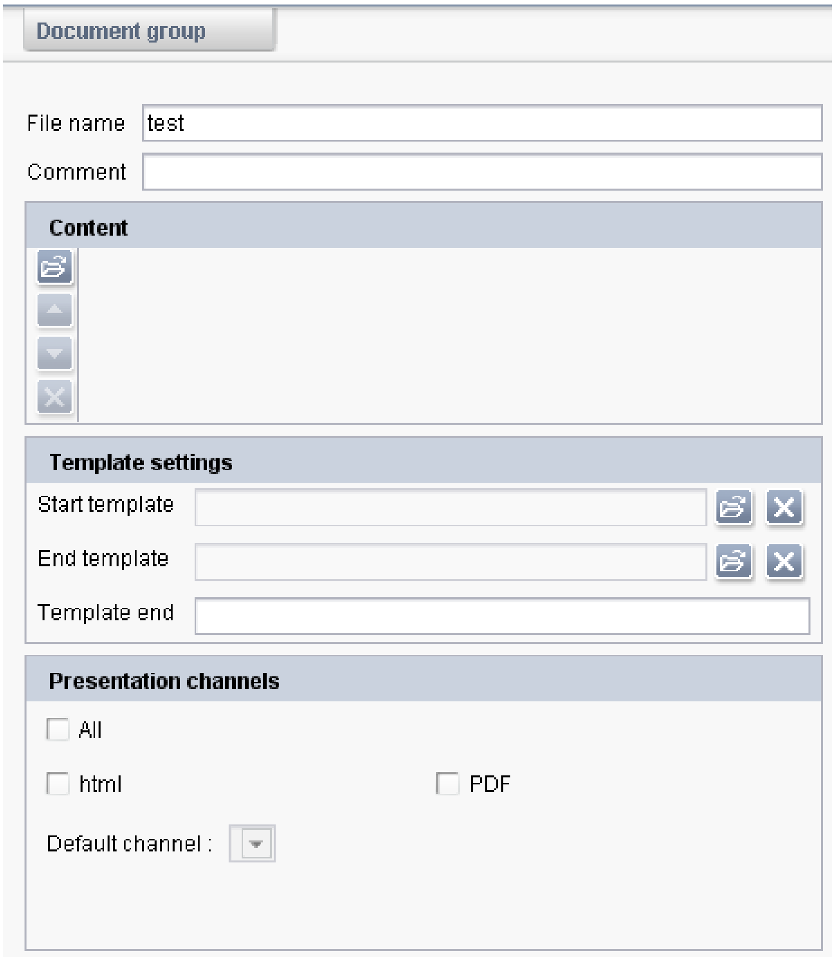
Language-dependent display names and the unique reference name for the new document group can be entered here.

The document group is then displayed in the tree structure with the following symbol:



5.3.2 Define properties

Further configuration options for the document group are available in the right-hand editing window:



Document group

File name test

Comment

Content

Template settings

Start template

End template

Template end

Presentation channels

□ All

□ html □ PDF

Default channel :

Figure 5-4: Properties of the document group

File name: The file name of the document group can be defined in this field. The generated file (e.g. a PDF file) is filed on generation under this name in the generation directory (default procedure: The reference name of the document group is adopted, but can be changed).

Comment: An additional name for a document group can be assigned in this field. The comment can be used, for example, to display a title heading in the generated PDF file.

The content of the document group can be selected from the project's Site Store (see Chapter 5.3.3 page 246).

In addition, the template settings (see Chapter 5.3.4 page 247) and the presentation channels for generation of the document group can be adjusted (see Chapter 5.3.5 page 249).

5.3.3 Manage content of the document groups

The content of the document group can be managed in this area:

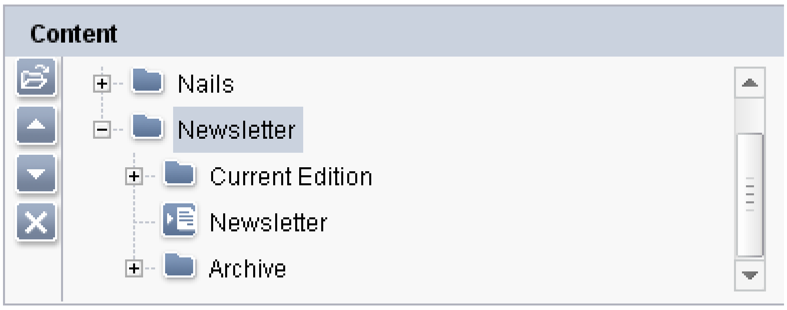


Figure 5-5: Content of a document group

📂 click this icon to add new elements from the project's Site Store to the document group.

☒ use this icon or the "Del" key to remove selected elements again.

> *If a page reference is deleted from the Site Store, which is already being used in a document group, after it has been deleted the name of the page reference no longer appears in the editing window of the document group, but instead the text "--Broken Link--". This text indicates that the integrated page reference is no longer available in the project. The node concerned can be removed from the document group using the* ☒ *button.*

↑ ↓ this icon can be used to change the order of the elements in the document group – it must be noted that this is only possible on the highest level (see Figure 5-5). If menu levels from the Site Store are copied into the document group, the sub-menus can be opened (with a double-click). However, the lower level elements cannot be changed (sorting) or deleted. It is merely a viewing function (see Figure 5-6).

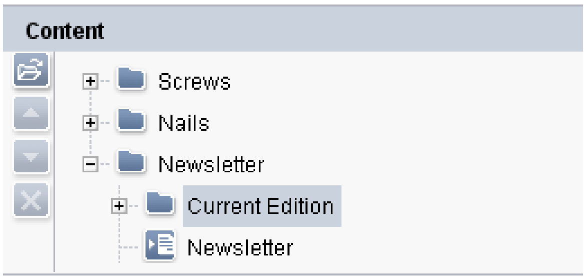
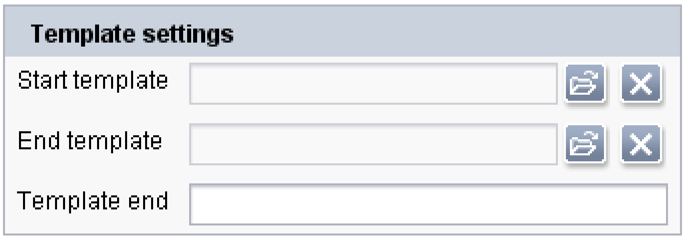


Figure 5-6: Content of a document group (lower level)

5.3.4 Template settings for document groups

This area is used to define which template is to be rendered *before the first page* and *after the last page* of the document group. In this way, structures which only appear once (for example, tables of content) can be generated.



Template settings

Start template

End template

Template end

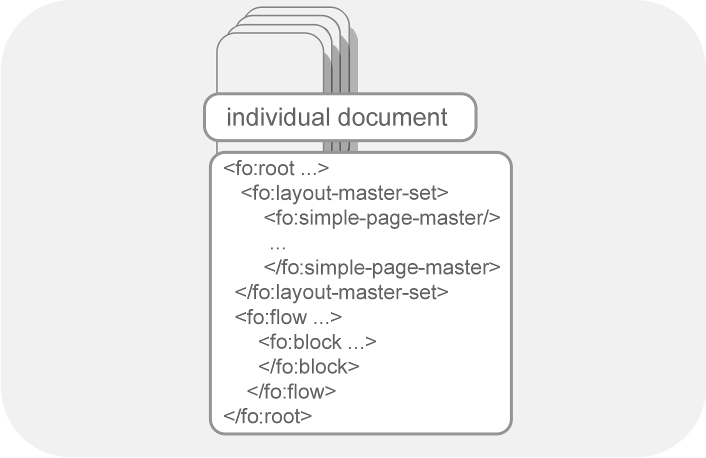
Figure 5-7: Template settings for document groups

Start template: Page template which is to be rendered before the first page of the document group, e.g. a table of contents.

End template: Page template which is to be rendered after the last page of the document group, e.g. a glossary.

Template end: In addition, a template suffix can be defined. The template to be rendered (a part of the document group) is replaced by an extended template.

Background: The individual parts of the document group possibly contain information which is only required once or are only allowed to exist once within the document group. For example, the PDF output channel contains an Information page template for generation of the PDF (FOP). If a page (based on this page template) is generated in the PDF output channel, this information is necessary for the individual page (Figure 5-8).



```
<fo:root ...>
   <fo:layout-master-set>
       <fo:simple-page-master/>
         ...
       </fo:simple-page-master>
   </fo:layout-master-set>
   <fo:flow ...>
      <fo:block ...>
      </fo:block>
   </fo:flow>
</fo:root>
```

Figure 5-8: Example FOP information of an individual document

However, in a document group, which is made up of several pages, the FOP information is only required once (and not for each individual part of the page group). Errors occur during the PDF generation if this information is contained more than once. For this application case, the FOP information can be moved into the start template (opening tags) and the end template (closing tags). The FOP information within the page page template can then be removed (see Figure 5-9). This ensures that the PDF channel creates a PDF during the generation and all content of the document group (without inner lying FOP information) is created within an FOP information.

To this end, a new template can be created (based on the original template). The new template must be named with the original name and a freely selectable suffix. This suffix can be entered as a "template end" in the template settings of the document group. The page template to be rendered is now automatically replaced by the document group's new page template when the document group is generated.

Recommended is a suffix in the following form: `_[A..z][any character string]`.

Background: As the reference names within a project have to be uniquely assigned, templates created with an existing reference name are automatically assigned consecutive numbering. I.e. if a suffix in the form `_[1..9]` is entered manually, it can result in overlapping with the automatic numbering.

Another option is to adjust the original page template so that the FOP information is written for individual documents, but is suppressed for document groups, e.g. by a query within the template:

```
$CMS_IF(!#global.docgroup)$<?xml version="1.0" encoding="UTF-8"?>
<of:root xmlns:fo="http://www.w3.org/1999/XSL/Format" ...
...
$CMS_END_IF$
     $CMS_VALUE(#global.page.body("center"))$
$CMS_IF(!#global.docgroup)$
                  </of:block>
           </of:flow>
     </of:page-sequence>
</of:root>
$CMS_END_IF$
```

In this case, a start and end template are still needed, however not a new page template for generation of the document group.

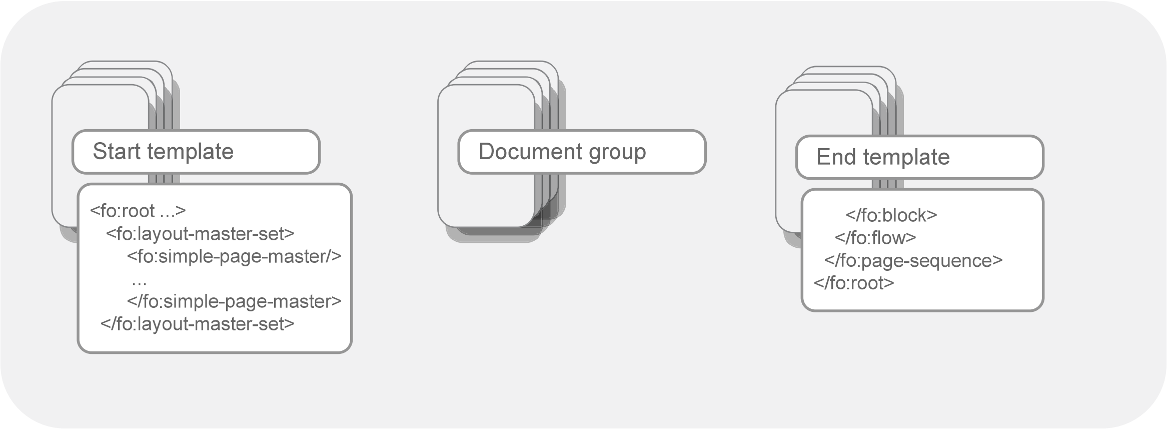
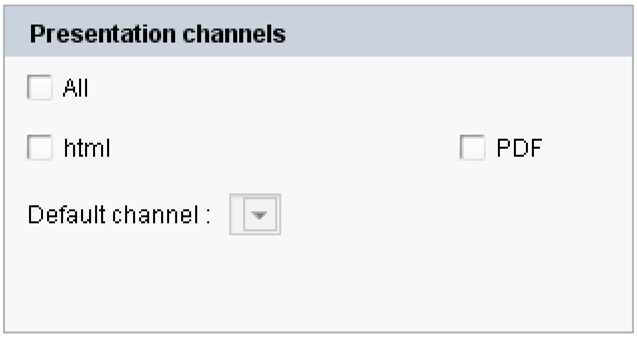


Figure 5-9: Example of FOP information of a document group

5.3.5 Presentation channels for generating document groups

This area is used to define the presentation channels for which the document group is to be generated. A presentation channel is not selected in the default setting. The "All" setting results in the document group being generated separately for each presentation channel. Optionally, selected presentation channels only can be generated. If more than one presentation channel is selected, but not "All", a default channel must be defined. This default channel is then the link target used if a link from a presentation channel points to the document group for which the document group is not generated.


Presentation channels

□ All

□ html □ PDF

Default channel :

Figure 5-10: Presentation channels for generation of a document group

The "Presentation channels" area is characterised by another special feature: all presentation channels are displayed here – i.e. including the "inactive" presentation channels. This enables a document group to be generated on the basis of a presentation channel which is not part of the "normal" generation. Therefore, use of document groups enables, for example, a few PDF documents to be inserted into a project, without having to generate a completely new channel.

5.4 Template development

5.4.1 System objects

So-called system objects can be used to access information, data and objects within the templates. System objects are always context-dependent. Several specific system objects are available within the generation context, with which the generation of documents can be controlled from document groups.

System objects can be invoked wherever method invocations can be used: Each system object begins with the symbol # and is extended by the respective name of the system object. Different methods can be invoked depending on the type of system object.

In general system objects can be read-accessed only. However, there are several methods which can also modify system objects or can affect the system performance. These include, e.g. the output of a document group's elements during generation (cf. `#docGroup.pageFirst`).

The specific system objects for document groups are briefly described here first, before examples of several applications are shown in the following chapter (see Chapter 5.5 page 254).

For further information on system objects, see FirstSpirit Online Documentation.

#global.docgroup: The system object `#global.docgroup` is available within the generation context for access to information on document groups. `#global.docgroup` returns the value "true" during generation of a document group.

#docGroup: The system object refers to the ("virtual") root node of the current document group. `#docGroup` provides various methods via dot notation with which, for example, a table of contents can be generated (see Chapter 5.5.2):

- childs
  Returns a list of the document group's child nodes. Each child is in turn of the same type as `#docGroup`. The list of children contains nodes of the type "page reference" first, followed by nodes of the type "menu level".

- depth
  Returns the depth of the node in the document group (e.g. "0" for the highest level). By way of comparison: Nodes of the `#docGroup.childs` list have depth "1".

- isFolder
  Indicates whether the node is of the type "menu level" (value "true") or not (value "false").

- isPageRef
  Indicates whether the node is of the type "Page Reference" (value "true") or not (value "false").

- index
  Returns the node's index in the "childs" list of the parent node.

- parent
  Returns the parent node of the element. If the `docGroup` is already the root node, `null` is returned.

- selected
  Returns the value "true", if the node itself or a child node is currently being written in the output.

- label
  Returns the heading of the node; for the root this equals the name of the document group, for menu levels the menu name and for page references the sitemap name.

- chapter
  Returns the chapter number, e.g. a text in the form "2.3.1".
- section
  Returns the chapter number; however, unlike `chapter`, with `section` only folders are included in the chapter numbering.
- root
  Returns the virtual root element (equals the value of `#docGroup`).
- pageFirst
  The value should be set to "true" in the start template (using the expression `$CMS_SET(...)$`), if pages are to be output before folders (see Chapter 5.3.4 page 247). Output within the template can be achieved with the expression `$CMS_VALUE(...)$`.

#docNode: The system object `#docNode` returns an element (e.g. a page reference) from the document group during generation of the document group. The referenced object is of the same type as the return value of the system object `#docGroup`.

5.4.2 Context variables

Apart from the system objects, other specific context variables exist for working with document groups. These context variables are not written in the generation context by the system, but can be specifically used by the template developer, for example to adjust the templates for generation of the document group (see Chapter 5.3.4 page 247).

PREFIX / SUFFIX: The variables PREFIX and SUFFIX can be used by the template developer in the start template. If these variables are set, the content in front of (PREFIX) and/or after (SUFFIX) each node is shown.

They are used, e.g. to realise headings (see Chapter 5.5.1 page 254), navigations to the next / previous / higher level chapter or for the generation of several pageflows in XML-FO documents for PDF generation.

> ! *The SUFFIX variable for menu levels is not shown until after the last child node of the menu level.*

5.4.3 Start and end template

In principle, these templates only contain the frame for generating a valid document of the selected presentation channel (see Chapter 5.3.4 page 247).

Example: Start template in HTML

```
<CMS_HEADER></CMS_HEADER>
<html>
  <body>
```

Example: End template in HTML

```
<CMS_HEADER></CMS_HEADER>
</body>
</html>
```

If necessary, special functions can also take be used at this point, e.g. the creation of a "clickable" table of contents of the document group (cf. Chapter 5.5.2 page 255).

5.5 Application examples

5.5.1 Example: Chapter headings

A simple way of realising chapter headings is to use the page-specific context variable PREFIX (cf. Chapter 5.4.2).

This variable can be used to define FirstSpirit expressions *before* the actual content of each page reference or menu level:

```
<CMS_HEADER>
</CMS_HEADER>
$CMS_SET(PREFIX)$
  <br />
  <hr />
  <h$CMS_VALUE(#docNode.depth)$>
    Chapter&nbsp;$CMS_VALUE(#docNode.chapter)$&nbsp;
    $CMS_IF(!#docNode.label.isEmpty)$
      <a name="$CMS_VALUE(#docNode.label.convert2)$">
        $CMS_VALUE(#docNode.label.convert2)$
      </a>
    $CMS_END_IF$
  </h$CMS_VALUE(#docNode.depth)$>
  <hr />
$CMS_END_SET$
```

With this, the following variables are defined and written in the context:

PREFIX: Defines the structure of the chapter heading including anchors for the table of contents.

In this variable definition, the following context variables are used, which refer to the respective current element of the document group during the evaluation (cf. Chapter 5.4.1):

Variable	**Function**
#docNode.depth	Depth of the current document object in the tree of the document group
#docNode.chapter	Page reference/menu level numbering
#docNode.label	Heading of the node

5.5.2 Example: Table of contents

The table of contents is usually needed once at the beginning of a document. The template fragment shown below should therefore be copied into the start template (cf. 5.3.4 page 247):

```
<!-- Headline -->
<h1>
  <a name="summary"></a>
  $CMS_IF(#global.language.abbreviation == "EN")$
    Summary
  $CMS_ELSE$
    Summary
  $CMS_END_IF$
  <br />
</h1>

<!-- Recursive loop -->
$CMS_SET(renderEntry)$
  $CMS_FOR(child, entries)$
    $CMS_FOR(void, [0..(child.depth - 1)])$
        &nbsp;&nbsp;&nbsp;
    $CMS_END_FOR$
    $CMS_VALUE(child.chapter)$
    &nbsp;
    $CMS_IF(!child.label.isEmpty)$
      <a href="#$CMS_VALUE(child.label.convert2)$">
        $CMS_VALUE(child.label.convert2)$
      </a>
      <br />
    $CMS_END_IF$
    $CMS_IF(child.childs.size > 0)$
      $CMS_SET(entries, child.childs)$
      $CMS_VALUE(renderEntry)$
    $CMS_END_IF$
  $CMS_END_FOR$
$CMS_END_SET$

$CMS_SET(entries, #docGroup.childs)$
$CMS_VALUE(renderEntry)$
<br/>
```

The table of contents is introduced by the heading:

```
<!-- Headline -->
<h1>
  <a name="summary"></a>
  $CMS_IF(#global.language.abbreviation == "EN")$
    Summary
  $CMS_ELSE$
    Summary
  $CMS_END_IF$
  <br />
</h1>
```

This is followed by the recursive execution:

```
<!-- Recursive loop -->
$CMS_SET(renderEntry)$
  <!-- All children (same level) -->
  $CMS_FOR(child, entries)$
    <!--Indent -->
    $CMS_FOR(void, [0..(child.depth - 1)])$
        &nbsp;&nbsp;&nbsp;
    $CMS_END_FOR$

    <!--Chapter -->
    $CMS_VALUE(child.chapter)$
    &nbsp;
     <!--Label -->
    $CMS_IF(!child.label.isEmpty)$
      <a href="#$CMS_VALUE(child.label.convert2)$">
        $CMS_VALUE(child.label.convert2)$
      </a>
      <br />
    $CMS_END_IF$

    <!--Recursive loop abort condition -->
    $CMS_IF(child.childs.size > 0)$
      $CMS_SET(entries, child.childs)$
      $CMS_VALUE(renderEntry)$
    $CMS_END_IF$
  $CMS_END_FOR$
$CMS_END_SET$
```

With it, all children of a level are considered and the chapter number and corresponding labelling is obtained for each element in the document group. To this end, the chapter name is output (from the page reference or the menu name) and is directly assigned a link:

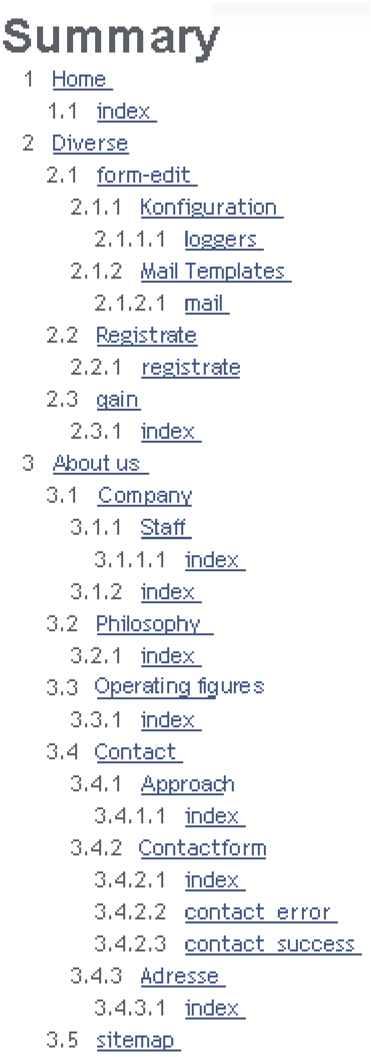
```
<a href="#$CMS_VALUE(child.label.convert2)$">
        $CMS_VALUE(child.label.convert2)$
      </a>
```

Using the "convert2" function on the page group text causes illegal characters to be replaced by the conversion rule.

Finally, the abort condition for the recursion:

```
   <!–Recursive loop abort condition →
    $CMS_IF(child.childs.size > 0)$
      $CMS_SET(entries, child.childs)$
      $CMS_VALUE(renderEntry)$
    $CMS_END_IF$
```

The result is displayed as follows:



Summary

1 Home
1.1 index
2 Diverse
2.1 form-edit
2.1.1 Konfiguration
2.1.1.1 loggers
2.1.2 Mail Templates
2.1.2.1 mail
2.2 Registrate
2.2.1 registrate
2.3 gain
2.3.1 index
3 About us
3.1 Company
3.1.1 Staff
3.1.1.1 index
3.1.2 index
3.2 Philosophy
3.2.1 index
3.3 Operating figures
3.3.1 index
3.4 Contact
3.4.1 Approach
3.4.1.1 index
3.4.2 Contactform
3.4.2.1 index
3.4.2.2 contact error
3.4.2.3 contact success
3.4.3 Adresse
3.4.3.1 index
3.5 sitemap

Figure 5-11: Example of document group with numbering and labelling

5.5.3 Example: Jump to table of contents

Especially in longer documents, it is frequently desirable to be able to jump to the table of contents at the end of a chapter.

In this example this function is achieved using a page-specific SUFFIX. The SUFFIX variable can be used to define FirstSpirit expressions *after* the actual content of each page reference or menu level. In the example, the SUFFIX is output after rendering a page reference:

```
$CMS_SET(SUFFIX)$
  $CMS_IF(#docNode.isPageRef)$
    <a href="#summary">
```

```
      $CMS_IF(#global.language.abbreviation == "EN")$
        to Summary
      $CMS_ELSE$
        go to summary
      $CMS_END_IF$
    </a>
  $CMS_END_IF$
$CMS_END_SET$
```

5.5.4 Example: Local page references

For the preparation of links in document groups, it can be useful to decide whether a link remains within the document group or points outside the document group, e.g. to generate an internal PDF link.

The following expressions can be used for this:

```
$CMS_VALUE(#docGroup.contains("gain_1"))$
$CMS_VALUE(#docGroup.contains(myVar))$
```

To this end, either the Uid ("Unified Identifier") is transferred as a string or the page reference is transferred directly as an "object". The return value of the ".contains(...)" expression is a Boolean value, which e.g. can be used for an If statement.

6 Tracking Changes using Revision Metadata (from V4.1)

6.1 Introduction (from V4.1)

FirstSpirit provides an option for tracking changes using the FirstSpirit Access-API. Access to the metadata of a revision is possible via specific API functions (see Chapter 6.3 page 261). The revision metadata contains information about the type (what changes have taken place?) and the scope (which elements were changed?) of change in the project. The information made available through the revision's metadata is very fine grained. For example, if changes are made to the contents, it is possible to determine which properties of an element have been changed, e.g. whether the content was changed in a certain editing language, whether a child element has been added or removed or whether certain attributes have been changed, e.g. the permissions for the corresponding element.

The enhanced revision information can be used to determine all changes which have taken place within a project, from a specific revision up to a specific revision.

This information can be accessed via the FirstSpirit Access-API, e.g. by means of BeanShell script.

The following chapter describes methods for obtaining one or several revisions of a project which are to be examined for changes (see Chapter 6.2 page 260) and methods for determining the corresponding change information (see Chapter 6.3 page 261). Different metadata information is available depending on the respective change type (see Chapter 6.3.1 page 261).

In addition, detailed examples are described for using change tracking in the project.

The first example determines all database changes which have taken place since the last deployed revision of a project (see Chapter 6.4 page 263).

The second example determines content changes which have taken place within the project between a start revision to be defined and an end revision to be defined (see Chapter 6.5 page 268).

6.2 Get revisions (from V4.1)

FirstSpirit operates with a revision-based repository. A special technique is used to manage the change in data over time: the so-called revision management.

A revision can be thought of as a kind of "snapshot" of the whole repository at a specific point in time. Unlike a version which usually only relates to a single object, the complete state of all objects in the repository is described in a revision.

Revisions are described by consecutive numbering (revision ID), whereby there is always precisely one current revision for the whole repository. If a repository is edited, all the changes made are linked with a new revision number. The revision number results from the last current revision number of the whole repository increased by one. All unchanged objects keep their old revision numbers. If an object is changed it is not overwritten in the repository but instead is inserted as a new object (with a higher revision number).

To determine the period during which certain changes have taken place within the project, the corresponding revision of the repository must be obtained first.

The revision can be directly obtained via the project. Either the required unique revision ID can be transferred, e.g.:

```
project.getRevision(revisionId);
```

or the date of the required revision can be transferred, e.g.:

```
project.getRevision(context.getStartTime());
```

The transferred date does not have to be uniquely assigned to a revision. Any date value can be transferred. If a revision exists on this date it is returned, otherwise the method returns the next smallest revision.

A selection of revisions within a certain period can be made available using the method:

```
project.getRevisions(Revision from, Revision to, int maxCount,
Filter<Revision> filter);
```

Two revisions are transferred. The first revision ("from") defines the bottom revision limit and the second revision ("to") defines the upper revision limit. Apart from these two revisions, all revisions with a higher revision ID than the bottom revision limit and a lower revision ID than the upper revision limit are returned.

The respective most up to date revision can be obtained using:

```
 getRevision(new Date());
```

```
start = project.getRevision(context.getStartTime());
end = project.getRevision(new Date());
revisions = project.getRevisions(start, end, 0, null);
```

Optionally, the "maxCount" parameter can also be transferred, which limits the number of returned revisions to a maximum value and – also optionally – a filter for further limitation.

6.3 Determine changes within a revision (from V4.1)

The revisions (see Chapter 6.2 page 260) can then be used to obtain the metadata with enhanced information on the changes:

```
revision.getMetaData();
```

The metadata manages different information, which depends on the type of respective change (see Chapter 6.3.1 page 261). Not only language-dependent content changes of an element are taken into account but also structural changes (e.g. move) or a change to the element attributes (e.g. name, permission definition, etc.) (see Chapter 6.3.2 page 262).

6.3.1 Determine change type (from V4.1)

The changes which have taken place within a revision can be obtained using:

```
metaData.getOperation();
```

The delivered revision operation (`RevisionOperation`) for example returns information on the change type (`RevisionOperation.OperationType`):

```
operation.getType();
```

Different change types are available for different project content.

The following types of change are possible for content of the type `IDProvider`:

▪ *CREATE*	a new object has been created in the project
▪ *MODIFY*	an object in the project has been changed
▪ *MOVE*	an object in the project has been moved
▪ *DELETE*	an object in the project has been deleted
▪ *RELEASE*	an object in the project has been released

The corresponding revision operation (e.g. `ModifyOperation`) returns an object of the type `BasicElementInfo` with further information on the object concerned (e.g. the unique identifier).

The following types of change are possible for content of the type `Entity`:

- *CONTENT_COMMIT* database content has been changed

The corresponding revision operation (e.g. `ContentOperation`) returns an object of the type `EntityInfo` with further information on the data records concerned (e.g. the ID of the data record or the ID of the corresponding database schema).

6.3.2 Determine changed elements (from V4.1)

Further information on the changes can be called depending on the respective change operations, for example, which data records were released within the project (for operation type: *CONTENT_COMMIT*):

```
operation.getReleasedEntities();
```

or, for example, which new content has been created within the project (for operation type: *CREATE*):

```
operation.getCreatedElement()
```

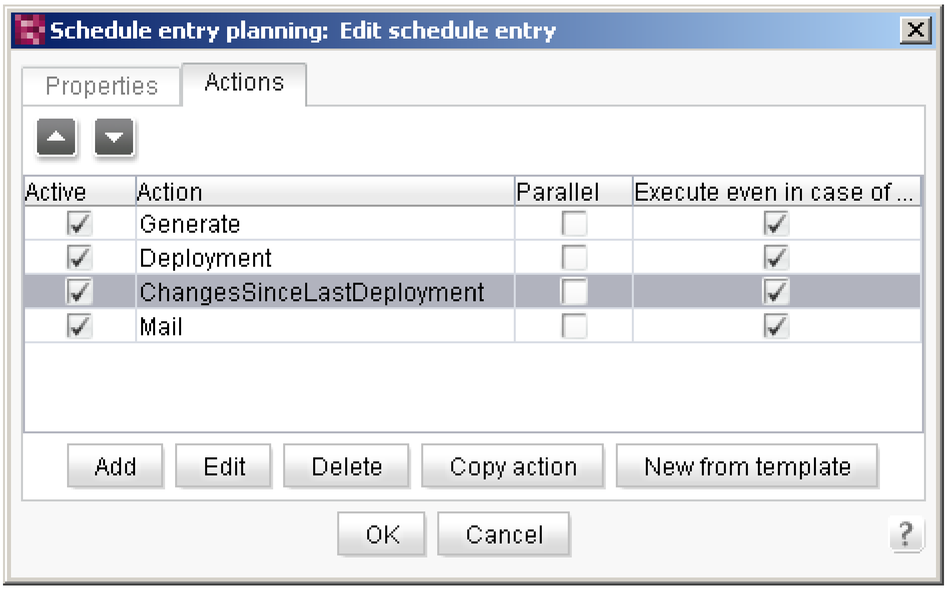
Further methods are given in the examples of the next two chapters (see Chapter 6.4 and Chapter 6.5).

For an overview of all available methods, see FirstSpirit Access-API documentation[5]*.*

[5] About the FirstSpirit Online documentation in Template Development – FirstSpirit API

6.4 Changes since the last deployment (from V4.1)

The first example is used to display the changes which have been made within the project since the last deployment. Specifically, it involves a project in which experts' reports are managed using the Content Store of FirstSpirit. An overview of the change to the data records is now to be made available each time the project is deployed. A post-deployment script is created within the deployment schedules first to determine the changes:



Schedule entry planning: Edit schedule entry

Active	Action	Parallel	Execute even in case of ...
☑	Generate	□	☑
☑	Deployment	□	☑
☑	ChangesSinceLastDeployment	□	☑
☑	Mail	□	☑

Figure 6-1: Post deployment script configuration

The script first determines the ID of the revision which was current at the time of the last project deployment:

```
task = context.getTask();
lastExecutionRevisionId = (Long) context.getVariable(task.getName() +
".revision");
if (lastExecutionRevisionId != null) {
    context.logInfo("revision of last execution=" +
lastExecutionRevisionId);
    revId = lastExecutionRevisionId.longValue();
}
```

Then, all revisions of the project since the last deployment are obtained. The revision with the revision ID just obtained is used as the lower revision limit ("startRev"). The current revision at the time the deployment starts 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);
```

Within a loop, all determined revisions are then examined for changes:

```
checkChanges(revisions) {
    for (revision : revisions) {
          metaData = revision.getMetaData();
          operation = metaData.getOperation();
          if (operation != null) {
                type = operation.getType();
                switch (type) {
                      case OperationType.CONTENT_COMMIT:
                      ..
                      ..
                      break;
                }
          }
    }
```

At the same time, only changes to database content – i.e. of the operation type CONTENT_COMMIT – are to be taken into account here, namely only the new data records created and the changed data records of a specific database table.

```
createdEntities = operation.getCreatedEntities();
releasedEntities = operation.getReleasedEntities();
```

are first used to determine all new generated and all released data records. This selection is then limited to a specific database table (here: `MyEntityTypName`):

```
ENTITY_TYPE = "MyEntityTypName";
if (ENTITY_TYPE.equals(created.getEntityTypeName())){
..
}
if (ENTITY_TYPE.equals(released.getEntityTypeName())) {
..
}
```

In complete form:

```
case 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 IDs of the changed and released data records determined ("created" and "released") are used to obtain the required information (e.g. the expert report numbers) and then save it for further use.

```
context.setProperty("created", createdList);
context.setProperty("updated", updatedList);
```

The values saved using `context.setProperty(..)` are only persistent within the current schedule, i.e. can continue to be used in a subsequent action within the schedule with `context.getProperty(..)`. In this example, the content is further used within the mail template in the following "Mail" action (cf. Figure 6-1):

```
Hello,
$CMS_SET(created, #context.getProperty("created"))$$CMS_SET(updated,
#context.getProperty("updated"))$
new$CMS_IF(created != null)$($CMS_VALUE(created.size)$)$CMS_END_IF$ and
modified$CMS_IF(updated != null)$($CMS_VALUE(updated.size)$)$CMS_END_IF$
certificates have been published on
http://www.certificates-online.com


$CMS_IF(created.size > 0)$New certificates:
=====================
$CMS_FOR(entity, created)$ * $CMS_VALUE(entity.CertificateNo)$
($CMS_VALUE(entity.Date.format("dd.MM.yy"))$) - $CMS_VALUE(entity.carNo)$
$CMS_END_FOR$$CMS_END_IF$
```

```
$CMS_IF(updated.size > 0)$Updated certificates:
=====================
$CMS_FOR(entity, updated)$ * $CMS_VALUE(entity.CertificateNo)$
($CMS_VALUE(entity.Date.format("dd.MM.yy"))$) - $CMS_VALUE(entity.carNo)$
$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@certificates-online.com
```

When a deployment schedule is executed, the template now generates an e-mail with the new generated and changed content:

Example (mail):

```
Hello,

new(4) and modified(2) certificates have been published on
http://www.certificates-online.com

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@certificates-online.com
```

Content which has been saved using `context.setVariable(..)`, is also persistent beyond the execution of the current schedule run (unlike the saving of content using `context.setProperty(..)`). This option is used in the example, to save the revision at the time of the current schedule:

```
context.setVariable(task.getName() + ".revision",
new Long(endRev.getId()));
```

When the next schedule is started this information can then be used to obtain the revision ID, which was current at the time of the last deployment of the project:

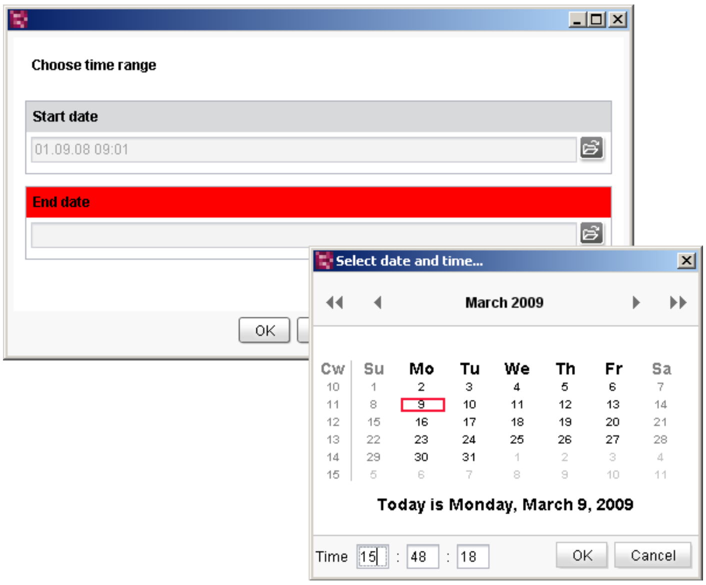
```
lastExecutionRevisionId = (Long) context.getVariable(task.getName() +
".revision");
```

If necessary, the complete script and the templates described here can be requested from the FirstSpirit Helpdesk.

6.5 Changes between two revisions (from V4.1)

In the second example the revisions can be conveniently selected via a GUI. To this end, a new script must be created first in the project's Template Store.

Input components for selecting a start and end date for the required revision limits can be configured in the form area of the script:



Choose time range

Start date: 01.09.08 09:01

End date:

Select date and time...

March 2009

Cw	Su	Mo	Tu	We	Th	Fr	Sa
10	1	2	3	4	5	6	7
11	8	9	10	11	12	13	14
12	15	16	17	18	19	20	21
13	22	23	24	25	26	27	28
14	29	30	31	1	2	3	4
15	5	6	7	8	9	10	11

Today is Monday, March 9, 2009

Time 15 : 48 : 18

Figure 6-2: GUI for selecting the revision limits

If necessary, the XML file for configuring the form area can be requested from the FirstSpirit Helpdesk.

Analogous to the first example, the relevant revisions can then be obtained using the selected data:

```
data = context.showGui();
if (data != null) {
    context.logInfo("data=" + data);
    from = data.get("from").getEditor().get(null);
    to = data.get("to").getEditor().get(null);

    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);
    }
```

The changes which have taken place within these revisions are determined in `checkChanges`. In this example the changes to project content of the operation type DELETE and CREATE are taken into account (within the Page Store). In addition, changes to the project's database content are also determined here:

Change by removing elements in the Page Store:

```
case OperationType.DELETE:
    deleteRoot = operation.getDeleteRootElement();
    if (deleteRoot.getStoreType() == Type.PAGESTORE) {
    // include only pagestore
    context.logInfo("found delete in pagestore (deleted node=" +
deleteRoot.getUid() + ") in revision=" +
getRevisionString(revision));
    }
    break;
```

Change by adding elements in the project:

```
case 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;
```

Change by creating, deleting, changing or releasing database content:

```
case 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
..
```

7 Server-Side Release

Apart from release via a workflow, all objects in FirstSpirit can be released on the server side via the Access API. To this end, methods exist to define the different release settings for an object. In this way, the specific release can be used to release other objects dependent on the current object, e.g. the complete parent chain or the child elements of the object to be released.

In general, a differentiation is made between the following release options:

- Standard release (see Chapter 7.1 page 272):
  Release for the object to be released including additional, defined release options for the standard case. These pre-defined release options differ depending on the object. For example, the standard release options of a page in the Page Store, are also used to release the lower level sections and the parent elements which have never been released. The standard release of a page reference in the Site Store on the other hand only takes into account the page reference itself. The standard release options cannot be changed.
- Specific release (see Chapter 7.2 page 273):
  Release for the object to be released including optional release options which are defined by the user. The different release options can be combined with each other in any way necessary to realise extensive release within a short time. However, under certain circumstances the release of all objects involved in the release process may not be wanted in all cases and should therefore be executed circumspectly.

7.1 Standard release

This option is used to execute the release for the current object (e.g. page or folder of the Page Store), including defined, object-dependent release options for the standard case.

Direct release of an object is executed using the following API method:

```
AccessUtil.release(IDProvider toRelease, boolean checkOnly)
```

Transferred parameters:

`toRelease`: Element to be released

`checkOnly`: If the value `true` is transferred the standard release is only tested. The objects to be released are not transferred into release status. Instead, the standard release is run through, e.g. to uncover errors before the real release of an object.

Returned parameters:

```
ServerActionhandle<? extends ReleaseProgress,Boolean >
```

The server-side release returns a `ServerActionHandle`, which contains all information about the release process and, for example, the status of the release or the log info.

7.2 Specific release

Depending on the defined release parameters, the specific release takes into account even more (dependent) objects within the release process.

- **Ensure accessibility (parent chain):** starting from the selected object, all new (never released) higher level nodes are also released (see Chapter 7.2.4 page 281). This option is useful, for example, if a new page has been created below a new folder in the Page Store and are both to be released together. Unlike the recursive release, other new pages below the folder were not released. While the combination of this option with the "recursive release" option remains limited to the current store (see Chapter 7.2.5 page 284), in combination with the "dependent release" option it also affects the parent chains of the dependent objects and therefore other stores (see Chapter 7.2.6 page 286).
- **Release recursive:** starting from the selected object, all lower level nodes are also released. This selection is useful, for example, if many pages below a folder in the Page Store have been changed and now all the changes are to be released together. This options remains limited to the current store (cf. Chapter 7.2.1 page 276).
- **Release new dependent objects only:** starting from the selected object, all objects dependent on the selected object (e.g. a medium used in a picture input component) and which have never been released yet (new created objects) are also released. If this release option is combined with other options (e.g. release of the parent chain), the dependent release also affects other objects and stores involved in the release process.
- **Release new and changed dependent objects:** starting from the selected object, all objects dependent on the selected object (e.g. a medium used in a picture input component) are also released. Both objects which have never been released (new created objects) and objects which have been edited again in the meantime following a release (changed objects) are taken into account. If this release option is combined with other options (e.g. release of the parent chain),

the dependent release also affects other objects and stores involved in the release process.

Specific release of an object is executed using the following API method:

```
AccessUtil.release(IDProvider releaseStartNode, boolean checkOnly,
boolean releaseParentPath, boolean recursive,
IDProvider.DependentReleaseType dependentType)
```

Transferred parameters:

`releaseStartNode`: Start node for the release

`checkOnly`: If the value `true` is transferred the specific release only is tested. The objects to be released are not transferred into release status. Instead, the defined release options are run through, e.g. to uncover errors before the real release.

`releaseParentPath`: If the value `true` is transferred, the complete parent chain of the object to be released is determined and all previously never released objects are also released. If the `releaseParentPath=false` option is set, the parent chain is not releassed, but the elements to be released are added to the release child list of hte parent node. The following applies:

- In the case of *changed* parent nodes: The object to be released can be reached in release status. However, the parent element is not released.
- In the case of *new* parent nodes: The object to be released cannot be reached in release status as the parent node has never been released. This can result in invalid references in the release status (see Chapter 7.2.4 and Chapter 7.2.5).

`recursive`: If the value `true` is transferred, all children elements of the object to be released are determined and recursively and are also released. If the value `false` is transferred the child elements are not taken into account in the release (see Chapter 7.2.1, Chapter 7.2.3 and Chapter 7.2.5).

`dependentType`: This parameter is used to determined and release dependent objects of the object to be released. For example, if a medium is referenced on one page, this medium can also be directly released on the specific release of the page. The following dependencies can be taken into account (see Chapter 7.2.2, Chapter 7.2.3 and Chapter 7.2.6):

- `DEPENDENT_RELEASE_NEW_AND_CHANGED`: new and changed dependent objects are taken into account.

- `DEPENDENT_RELEASE_NEW_ONLY`: new created (never yet released objects) only are taken into account

- `NO_DEPENDENT_RELEASE`: dependent objects are not taken into account and if necessary must be released separately (default setting).

The different release options can be combined with each other in any way necessary to realise extensive release within a short time. However, under certain circumstances the release of all objects involved in the release process may not be wanted in all cases and should therefore be executed circumspectly.

The server-side release is therefore explained in the following chapters using several examples.

Returned parameters:

```
ServerActionhandle<? extends ReleaseProgress,Boolean >
```

The server-side release returns a `ServerActionHandle`, which contains all information about the release process.

7.2.1 Recursive release

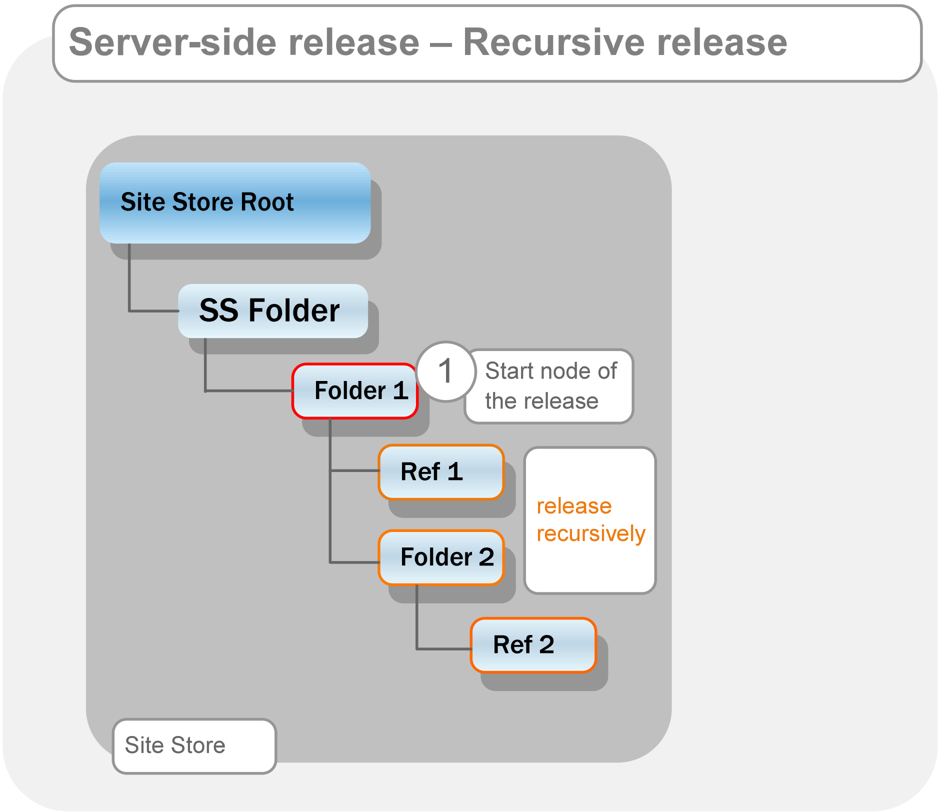


Figure 7-1: Server-side release – recursive

The following parameters have been set to call `AccessUtil.release(...)`:

```
releaseStartNode:  Folder 1
releaseParentPath: false
boolean recursive:  true
DependentReleaseType: NO_DEPENDENT_RELEASE
```

The selected start node for the release is the menu level "Folder 1".

Recursive release: The `recursive` option is evaluated at the starting point of "Folder 1" release. The recursive release *solely* affects the child elements of the release starting point. In the example from Figure 7-1, the child elements "Ref 1", "Folder 2" and "Ref 2" are therefore released by the option.

Recursive releases of other dependent elements are not executed - even in combination with other release options. The recursive release therefore does *not* affect the release of child element dependent objects in other stores.

7.2.2 Dependent release

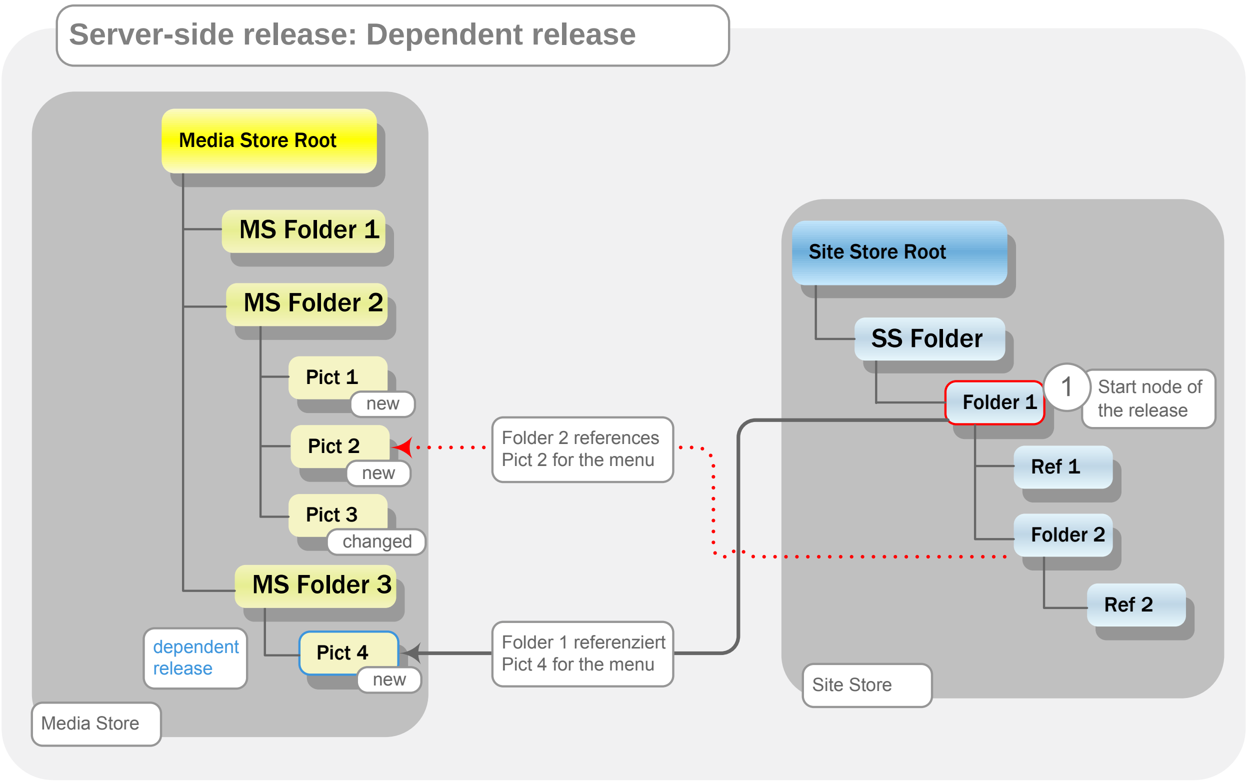


Figure 7-2: Server-side release – release only new or new and changed

The following parameters have been set to call `AccessUtil.release(...)`:

```
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 menu level "Folder 1".

Dependent release: The `DEPENDENT_RELEASE_NEW_ONLY` and `DEPENDENT_RELEASE_NEW_AND_CHANGED` options affect *all* dependent objects in the Page Store, Site Store and Media Store. This release option therefore not only concerns the start node but also all objects considered during the release process. In the example from Figure 7-2, the option examines and releases all outgoing references of the menu level "Folder 1". If the dependent release only is activated (without recursive release) "Pict 4" only would be released (see Figure 7-2); however, if other release options are activated, the release can be considerably more extensive (see Chapter 7.2.3 page 279, Chapter 7.2.6 page 286 and Chapter

7.2.7 page 288).

❗ *All outgoing references for the dependent release are completely taken into account in one direction only. If all dependent objects are to be included in the release process, the release must therefore take place in a certain order (see Chapter 7.2.8 page 290).*

7.2.3 Dependent release with recursive release

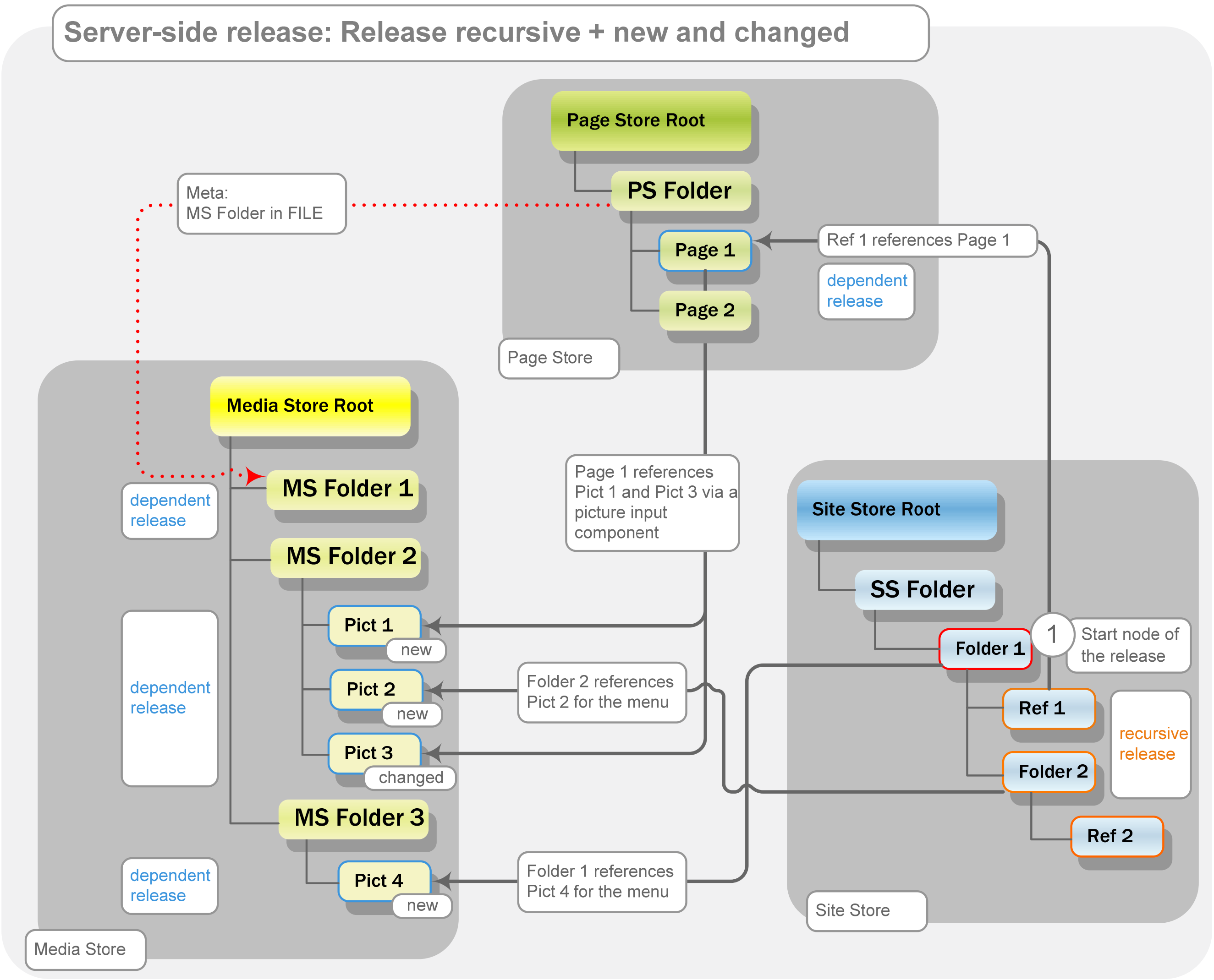


Figure 7-3: Server-side release – release recursive and dependent

The following parameters have been set to call `AccessUtil.release(...)`:

```
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 menu level "Folder 1".

Dependent release and recursive release If the `DEPENDENT_RELEASE_NEW_ONLY` or `DEPENDENT_RELEASE_NEW_AND_CHANGED` options are combined with the `recursive` option, the dependent release also affects all objects which lie below the start node. Therefore, in the example from Figure 7-3, not only the outgoing references of the menu level "Folder 1" are examined (cf. Chapter 7.2.2 page 277), but also the outgoing references of the child objects located under it:

- In relation to the example, "Ref 1" which has a reference in the Page Store and is located below "Folder 1" is therefore also examined. The page reference "Ref 1" is released by the recursive release, the page "Page 1" is also released through the dependent release.
- The menu level "Folder_2", which has become part of the release process due to the recursive release option, has a reference to a medium in the Media Store. The recursive release releases the folder "Folder 2" and the lower level page reference "Ref 2". The dependent release also releases the referenced medium "Pict 2".
- The page "Page 1" which was dependently released, also has outgoing references in the Media Store. The referenced media "Pict 1" and "Pict 3" are also dependently released.

Other dependent or recursive objects are no longer considered, as they are not covered by any of the release options.

! *All outgoing references for the dependent release are completely taken into account in one direction only. If all dependent objects are to be included in the release process, the release must therefore take place in a certain order (see Chapter 7.2.8 page 290).*

7.2.4 Ensure accessibility (parent chain)

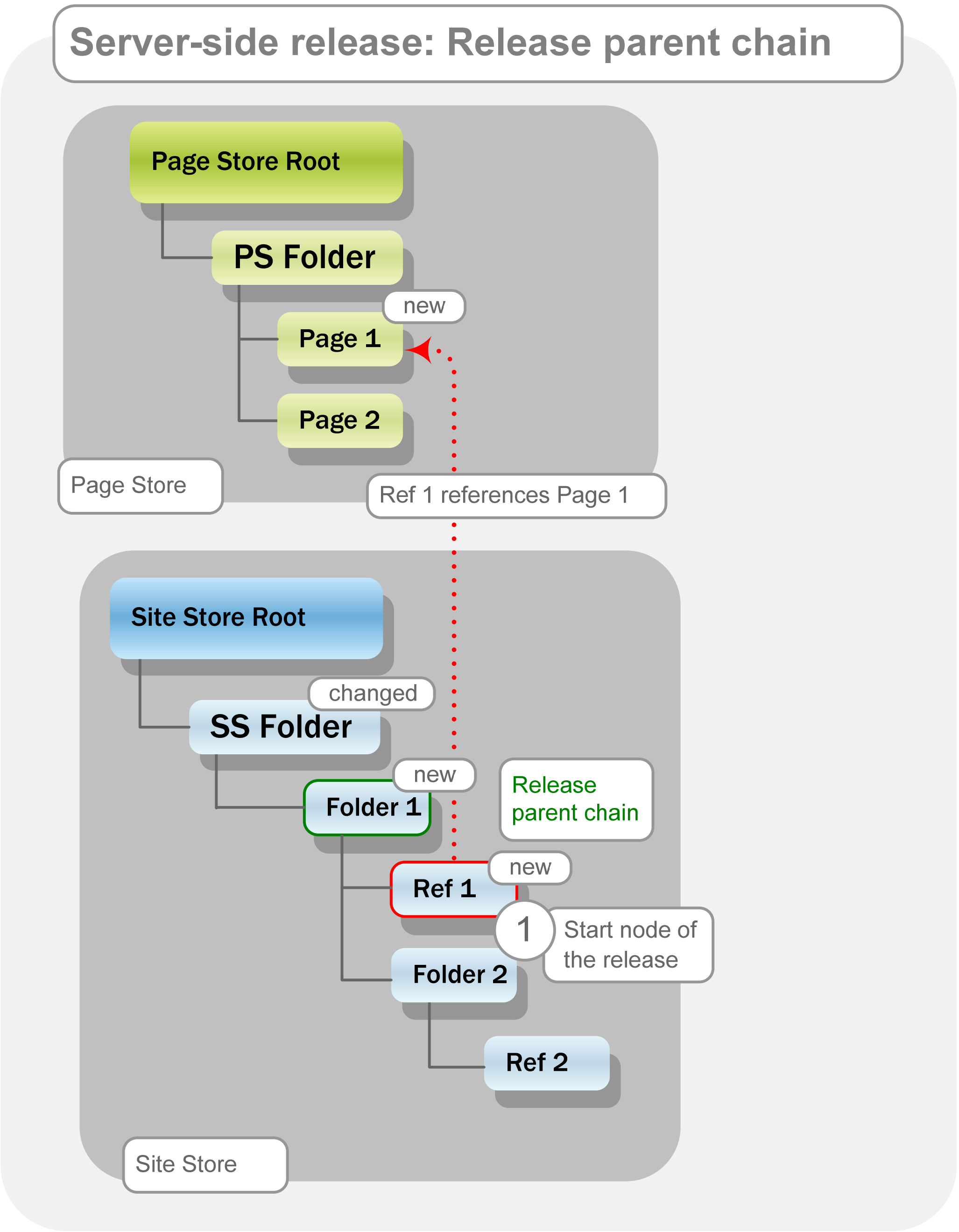


Figure 7-4: Server-side release – Ensure accessibility (parent chain)

The following parameters have been set to call `AccessUtil.release(...)`:

```
releaseStartNode:  Ref 1
releaseParentPath: true
boolean recursive:  false
DependentReleaseType: NO_DEPENDENT_RELEASE
```

The selected start node for the release is the page reference "Ref 1".

Release parent chain: Starting from the start node of the release "Ref 1", the complete parent chain of the object is considered up to the root node of the store. The `releaseParentPath` option releases all nodes of the parent chain, which *have never been released*. In specific terms, this means objects which have already been released (changed objects), are not released by the `releaseParentPath` options, not even if they have changed by an addition, for example, by adding a page page reference (see Figure 7-4).

If the `releaseParentPath=false` option is set, the parent chain is not released, but the elements to be released are added to the release child list of the parent node. The following applies:

- In the case of *changed* parent nodes: The object to be released can be reached in release status. However, the parent element is not released.
- In the case of *new* parent nodes: The object to be released cannot be reached in release status as the parent node has never been released. This can result in invalid references in the release status.

Background: If a page reference is to be released, although the editor does not have permission to release within the higher menu level, the page reference should still be included in the release status. Any content changes within the menu level (e.g. other references), should however not be released by the `releaseParentPath` option (see Figure 7-5).

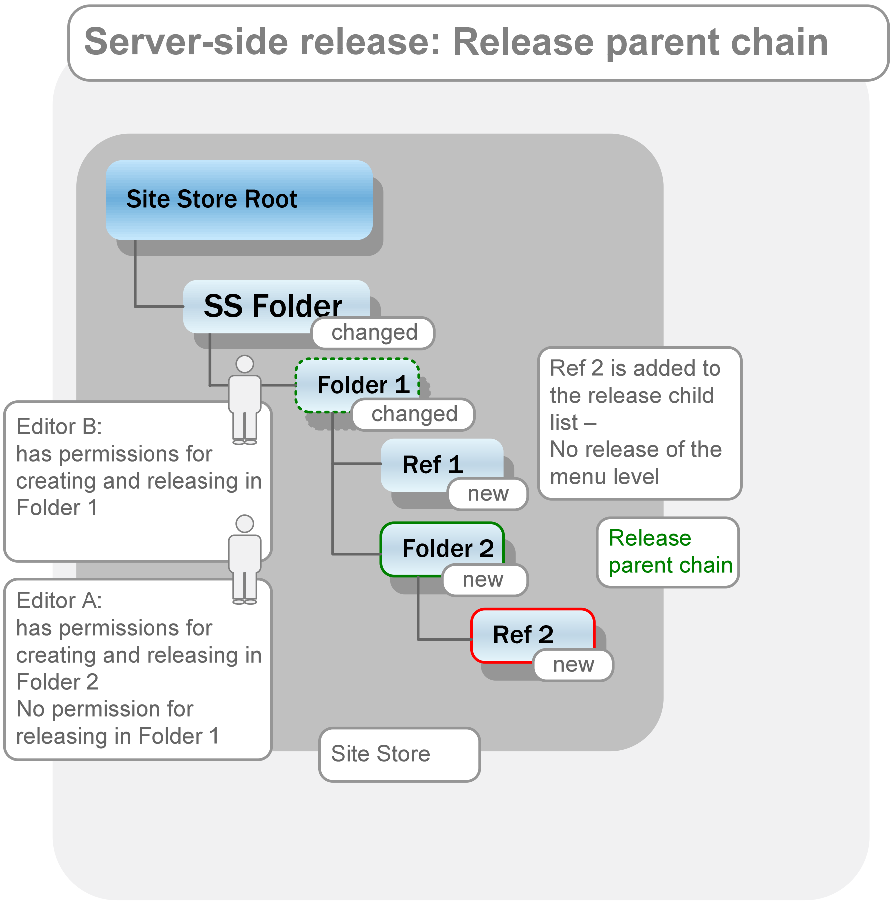

Server-side release: Release parent chain
Site Store Root
SS Folder
changed
Folder 1
changed
Ref 1
new
Folder 2
new
Ref 2
new
Ref 2 is added to the release child list – No release of the menu level
Release parent chain
Editor B: has permissions for creating and releasing in Folder 1
Editor A: has permissions for creating and releasing in Folder 2 No permission for releasing in Folder 1
Site Store

Figure 7-5: Server-side release – Release parent chain (release child list)

In the example from Figure 7-5, a release of page reference "Ref 2" by "Editor A" released both the newly created page reference and the newly created menu level "Folder 2". Menu level "Folder 1" is *not* released. "Folder 2" is however added to the release child list of menu level "Folder 1" using the `releaseParentPath` option so that the new menu level "Folder 2" (and therefore the page reference "Ref 2") can be reached in release status. Page reference "Ref 2" can therefore be reached within the release status, but not the newly created page reference "Ref 1". If "Editor B" now releases the page reference "Ref 1", this is also added to the release child list of the menu level "Folder 1". As "Folder 1", as a changed object, can already be reached via the release child list of the folder "SS Folder" which has also been changed, the release is therefore closed. The two menu levels ("Folder 1" and "SS Folder") are not released by the option, as they are not newly created objects.

7.2.5 Ensure accessibility (parent chain) and release recursively

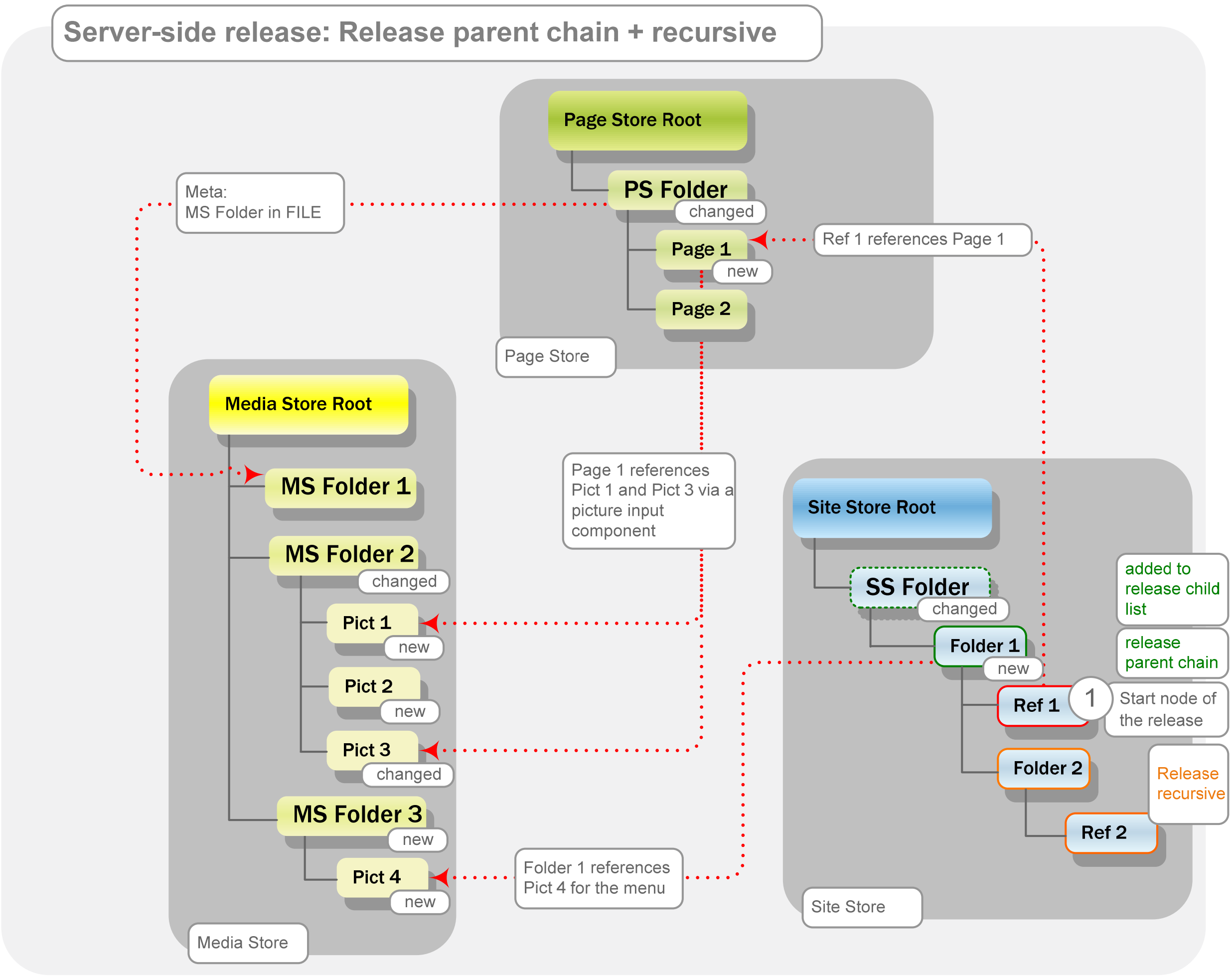


Figure 7-6: Server-side release – Ensure accessibility (parent chain) and release recursively

The following parameters have been set to call `AccessUtil.release(...)`:

```
releaseStartNode:  Ref 1
releaseParentPath: true
boolean recursive:  true
DependentReleaseType: NO_DEPENDENT_RELEASE
```

The selected start node for the release is the page reference "Ref 1".

Ensure accessibility (parent chain) and recursive release: Starting from the start node of release "Ref 1", the complete parent chain of the object is considered up to the root node of the store. The `releaseParentPath` option releases all nodes of the parent chain, which have *never been released* (cf. Chapter 7.2.4 page 281). In addition, all child elements of the starting point are released recursively (cf. Chapter

7.2.1 page 276). Using the example in Figure 7-6, it can be clearly seen that this release is limited to the Site Store, as no dependencies are taken into account here (unlike Figure 7-7). With this release it must be noted that defective references result if a new object referenced in the Site Store has been created, i.e. in the example, "Page 1" and the media "Pict 1" and "Pict 4". The current configuration in the example (cf. Figure 7-6) will therefore result in an error within the release, as the referenced page "Page 1" has never been released. If the references refer instead to objects which have already been released once ("changed"), the last released versions of each of the objects is referenced. In this case the release from the example could be successfully executed.

7.2.6 Ensure accessibility (parent chain) and dependently release

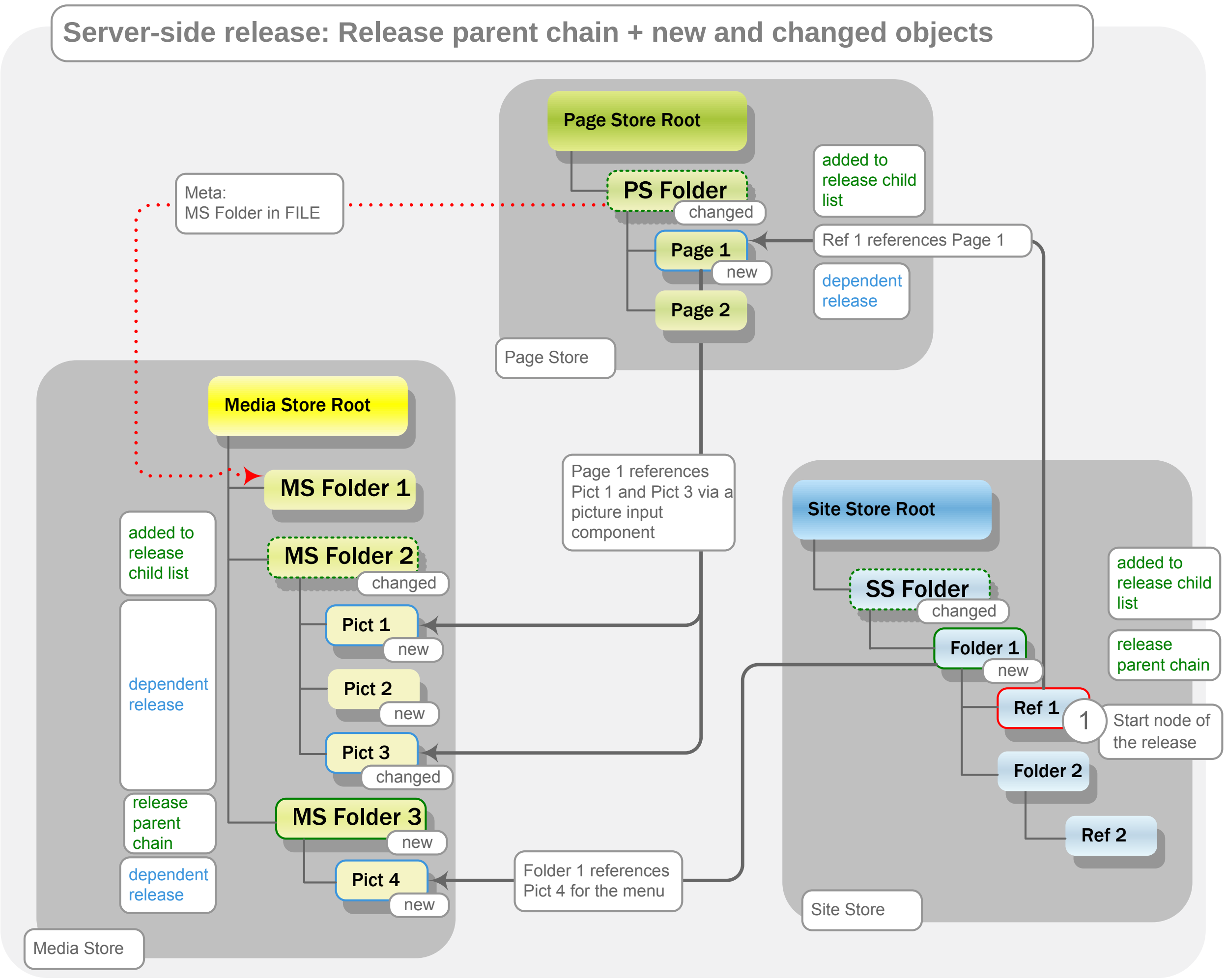


Figure 7-7: Server-side release – Ensure accessibility (parent chain) and dependent objects

The following parameters have been set to call `AccessUtil.release(...)`:

```
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 the page reference "Ref 1".

Ensure accessibility (parent chain) and dependent release: If the `DEPENDENT_RELEASE_NEW_ONLY` or `DEPENDENT_RELEASE_NEW_AND_CHANGED` options are combined with the `releaseParentPath` option, the dependent release affects both the current start node and the parent chain on release of elements which have never been released. This means, for example for the release of a page reference, that the page referenced there is released. The whole parent chain for the referenced page is now run through also and elements which have never been released are sought. These elements are also released. The same applies to dependent objects in the Media Store.

- The whole parent chain is run through for the page reference "Ref 1". Objects there which have never been released are released, i.e. in the example the new menu level "Folder 1", but not the changed menu level "SS Folder".
- The menu level "Folder 1" has an outgoing reference in the Media Store. The dependent release also releases the medium "Pict 4".
- The whole parent chain is now run through for the medium "Pict 4" and all never released objects are released. In the example, only the new media folder "MS Folder 3" is released.
- On release of the page reference "Ref 1", the referenced page "Page 1" is released.
- The whole parent chain is now run through for the page "Page 1" and all never released objects are released. In the example, this does not apply to any objects, as the parent node "PS Folder" has already been released once. Dependent objects of the folder "PS Folder" are therefore not taken into account in the dependent release.
- But the page "Page 1" which was dependently released, still has outgoing references in the Media Store. The referenced media "Pict 1" and "Pict 3" are also dependently released.
- The parent chain is now also examined for the two media. As the common parent node "MS Folder 2" has only changed, no release is executed here.

! *All outgoing references for the dependent release are completely taken into account in one direction only. If all dependent objects are to be included in the release process, the release must therefore take place in a certain order (see Chapter 7.2.8 page 290).*

7.2.7 Ensure accessibility (parent chain), release recursively and dependently

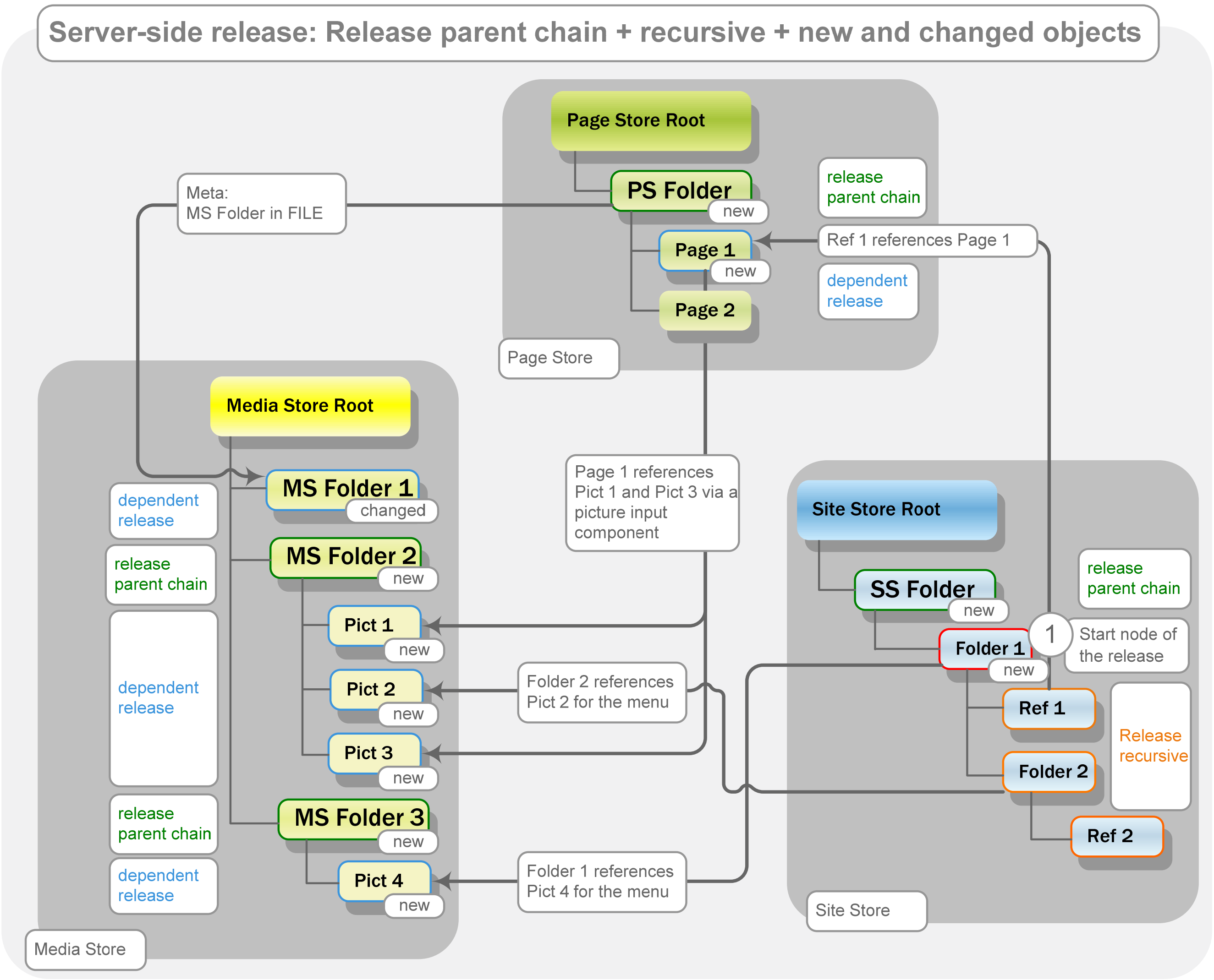


Figure 7-8: Server-side release including all options

The following parameters have been set to call `AccessUtil.release(...)` :

```
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 menu level "Folder 1".

Ensure accessibility (parent chain), release recursively and release dependently: The most comprehensive release is executed if all release options are combined with each other. In this case, both never released elements of the parent chain and all elements below the start node are released. In addition, the dependent objects of all nodes affected by the release process are released and the whole parent chain is examined there and if applicable is released. Unlike the release of the parent chain, the recursive release does not affect the dependent objects. Using the example in Figure 7-8, it is clear that the release approximately affects all the displayed objects – only "Page 2" is not affected:

- The `recursive` option is evaluated at the starting point of the release "Folder 1". The recursive release *solely* affects the child elements of the release's starting point. In the example from Figure 7-8, the child elements Ref 1, Folder 2 and Ref 2 are therefore released by the option.
- All outgoing references of the released objects are released. In the example, this is the objects "Pict 4" (through the reference within the menu level "Folder 1"), "Page 1" (through the page reference "Ref 1"), "Pict 2" (through the reference within the menu level "Folder 2")
- The outgoing edges of these released objects are examined again and released. In the example, these are the media "Pict 1" and "Pict 3" (through the reference within the page "Page 1").
- The complete parent chains of all released elements are now examined and all never released parent nodes are released. In the example, these are 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").
- The dependent objects of the released parent nodes are now released. In the example, this is the "MS Folder 1" (through the reference in "PS Folder"). Unlike the `releaseParentPath` release option, "MS Folder 1" is released even if it has only been "changed", i.e. had already been released once.

! *All outgoing references for the dependent release are completely taken into account in one direction only. If all dependent objects are to be included in the release process, the release must therefore take place in a certain order (see Chapter 7.2.8 page 290).*

7.2.8 Order for the release

All outgoing references for the dependent release are completely taken into account in one direction only, to prevent cyclical dependences in the release.

If all dependent objects are to be included in the release process, the following order must be adhered to:

- Release in the Site Store includes outgoing references in the Page Store
- Release in the Site Store includes outgoing references in the Media Store
- Release in the Page Store includes outgoing references in the Media Store

The following are not taken into account:

- Release in the Page Store includes *no* outgoing references in the Site Store
- Release in the Media Store includes *no* outgoing references in the Site Store or in the Media Store

Further cases in which dependent objects are shown in the reference graph but not taken into account for a dependent release:

- Page→Page reference: Page with the input component CMS_INPUT_PAGEREF in which a page reference is used.
  ⇨ Only the page will be released, but not the dependent page reference.
- Page→Medium: Page basing on a page template in which the reference to a medium is hard coded, for example `$CMS_REF(media:"XXX")$` in the HTML channel.
  ⇨ Only the page will be released, but not the dependent medium.
- Medium→Medium: In a css file (Parse file: yes) another Medium (for example a picture) is referenced hard coded. Both media are not released yet.
  ⇨ If the css file is released ("Specific release -> Release dependent objects "), the referenced picture will not be released at the same time.
- Page with LINK/DOM editor→Page reference: Both referenced objects are not yet released.
  ⇨ The referenced medium (picture) is released at the same time, but not the referenced page reference.
- Page→Data set: Page with input component CMS_INPUT_CONTENTLIST / FS_LIST etc., in which data sets are referenced.
  ⇨ Object of the Content Store will not be released.

! *Under certain circumstances, cyclical dependencies can occur, which cannot be automatically released and therefore have to be triggered manually.*
Example: *2 pages exist in the Page Store ("Page 1" and "Page 2") each with a section and a section reference to the section on the other page:*
-- Page 1
-- Section A
-- Section reference to Section B of Page 2
-- Page 2
-- Section B
-- Section reference to Section A of Page 1

If the section references have not yet been released, neither Page 1 nor Page 2 can be automatically released in this constellation. To release the pages, one of the section references must be deleted first to remove the cyclical dependency, e.g. "Section reference to Section B of Page 2". Page 2 can then be released. The section reference must then be established again, then Page 1 can also be released.

In some cases, dependent objects are shown in the reference graph but not released at the same time as the object on which the dependent release workflow is started.

8 WebEdit

WebEdit has been developed as a supplement to the FirstSpirit JavaClient editing system. WebEdit mode provides a browser-based interface for fast and uncomplicated input and maintenance of editorial content. The authors can immediately use the diverse functions of the FirstSpirit editing environment, because unlike the installation of the FirstSpirit JavaClient, the web browser is used for WebEdit and therefore no other software (Java environment (JRE)) is required. From a technical point of view, WebEdit operates purely on the basis of HTML and JavaScript.

WebEdit is usually used if authors want to change existing content very quickly, without having to familiarise themselves with the far-reaching functions of the FirstSpirit editing environment. To keep use and user prompting in WebEdit as simple and understandable as possible, the WebEdit interface does not provide the full functional scope of the FirstSpirit editing environment. Functions which are not part of the editorial work, for example, defining and changing workflows or editing templates, are therefore not a component part of WebEdit.

For details of the functional scope of WebEdit and restrictions within the WebEdit versions 4.0 and 4.1, see "FirstSpirit Release Notes".

Further documentation about the subject WebEdit can be found in the FirstSpirit Online Documentation under "Template development – WebEdit".

8.1 Requirements for the use of WebEdit

The requirements for the use of WebEdit are described in the following documentation:

System requirements:

- FirstSpirit Technical Data Sheet

Project requirements:

- FirstSpirit Manual for Administrators ("Project prerequisites when using WebEdit" Chapter)

Browser compatibility:

- FirstSpirit Technical Data Sheet

Configuration:

- FirstSpirit Manual for Administrators:
  Chapter "WebClient Configuration"
  Chapter "WebEdit as local project application (from V4.1)
  Chapter "Activate scripts in WebEdit" Chapter
  Chapter “Prevent ‘directory browsing’”
  Chapter "Configure WebEdit tree presentation"
  Chapter "Configure workflows in WebEdit"
  Chapter “Area: WEBedit configuration”

For details of browser configuration for editing editorial content, see:

- FirstSpirit Manual for Editors (WebClient):
  Chapter "Browser Configuration"
- FirstSpirit Manual for Administrators:
  Chapter "Browser Configuration for the Use of WebEdit"

8.1.1 Use of WebEdit without adjusting the templates

> ! *WebEdit can be used without changing the project templates up to FirstSpirit Version 4.0 inclusively, but this is not recommended. From FirstSpirit-Version 4.1 the use of WebEdit without adjusting the templates is no more supported!*

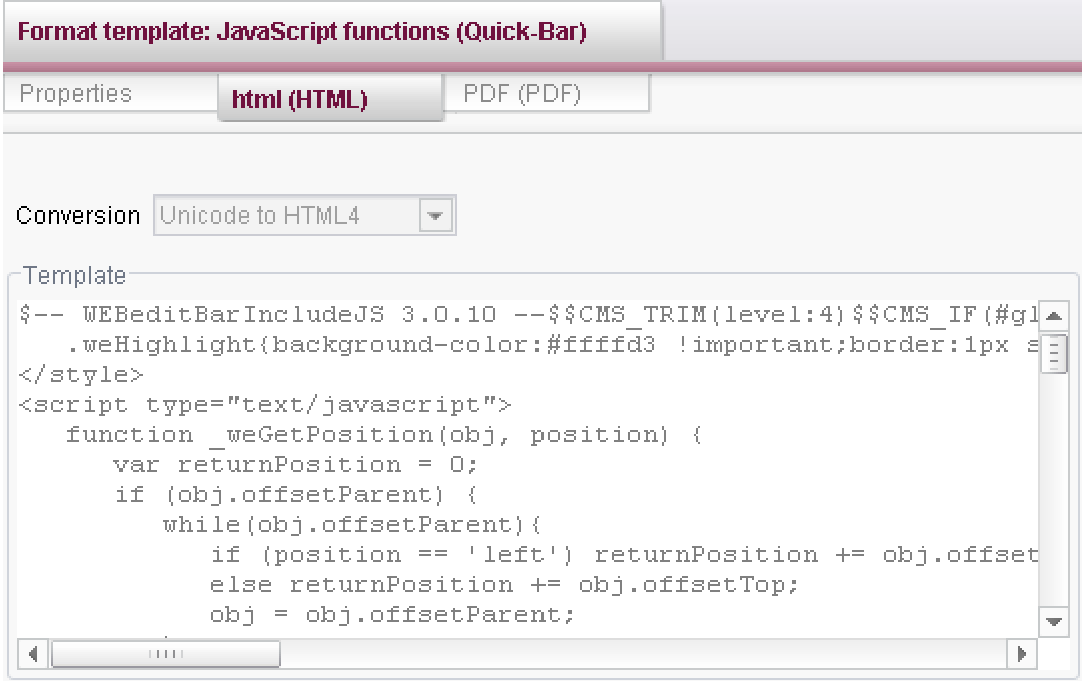
If WebEdit is used without adjusting templates the following restrictions apply:

- WebEdit mode is controlled solely using the WebEdit toolbar (i.e. changing existing data requires more work operations than using the Quick Edit bar or the WebEdit icons).
- Links may not use any explicit targets such as "_top" or "_parent".

8.1.2 Use of WebEdit with adjustment of the templates (recommended)

If the templates are to be used both for WebEdit and within the JavaClient (for example for a WebEdit preview), adjustments can be made within the templates to integrate controls for direct editing in the page preview. These control elements (cf. FirstSpirit Manual for Editors (WebClient)) enable the pages or certain parts of pages to be edited within the preview directly in the browser.

To this end, the format templates for the required elements must be created first in the project:



Conversion: Unicode to HTML4

```
$-- WEBeditBarIncludeJS 3.0.10 --$$CMS_TRIM(level:4)$$CMS_IF(#gl
   .weHighlight{background-color:#ffffd3 !important;border:1px s
</style>
<script type="text/javascript">
   function _weGetPosition(obj, position) {
      var returnPosition = 0;
      if (obj.offsetParent) {
         while(obj.offsetParent){
            if (position == 'left') returnPosition += obj.offset
            else returnPosition += obj.offsetTop;
            obj = obj.offsetParent;
```

Figure 8-1: WebEdit control format template

These can then be integrated in a suitable position of the page and/or section templates, e.g. in the HTML presentation channel, using the call:

```
$CMS_RENDER(template:"WEBeditIncludeJS")$
```

! *When specifying the UID, note upper/lower case (case sensitive).*

By adjusting the templates, the same templates can be used for the WebClient and the JavaClient.

The template adjustment must always be made for the template set configured for this project by the project administrator (cf. "FirstSpirit Manual for Administrators"). Either an existing template set (e.g. HTML channel) or a separate template set (e.g. WebEdit channel) can be used for WebEdit.

> ! *When adjusting the templates, make sure that the layout of the pages is correct both with the inserted controls (e.g. in the WebEdit standalone view) and in the view without controls (e.g. in a preview from the JavaClient).*

8.2 Functional scope of WebEdit

- WebEdit button on the start page
- Login and project selection
- Simplified editing due to new controls (Quick Edit bar) at page and section level: The new Quick Edit controls group together frequently required functions at page and section level in the form of a fold-out menu. The following functions are available:
  - Page: New menu, New page, New section, Edit page, Start or Forward workflow, Edit metadata, Delete page, Help
  - Section: New section, Edit section, Move up, Move down, Edit metadata, Delete section, Help

> ! *The Quick Edit bar will be no more supported from FirstSpirit Version 5.0. It will be replaced by the Easy-Edit functionality (see Chapter 8.3 page 305).*

- Easy-Edit (from FirstSpirit Version 4.2): explanation about the use of Easy-Edit see Chapter 8.3 page 305 and FirstSpirit Online Documentation [6].
- Themes for WebEdit: The "Look & Feel" of all central WebEdit components, such as controls, icons, fonts, etc, can be configured using Themes. The creation of new "WebEdit Themes" is no longer supported from FirstSpirit Version 4.0. The Theming function of WebEdit will be dropped with the release of WebEdit Version 5.0. Support for the creation of new themes for WebEdit is therefore no longer planned. The FirstSpirit standard themes ("default", "xp") and the specific SAP theme ("sap") can however continue to be used in FirstSpirit Version 4.1.
- Introduction of Content Outlining: In conjunction with the Quick Edit bar at section level, it is now possible to interactively select the content area to which an operation relates if the Quick Edit menu is displayed. This requires a DIV

[6] FirstSpirit Online Documentation – Chapter: ../Template development/WebEdit/Easy-Edit

container to be added in the template.
Note: This functionality is not supported by FirstSpirit Version 4.2 because it can cause conflicts with the Easy-Edit functionality.

- Introduction of a new WYSIWYG editor with:
  - Undo/Redo
  - Support for lists
    Create new and extend nested lists is NOT possible
    Editing nested lists is possible
  - "Drag & Drop" for selected text
  - Full-screen edit mode
- Introduction of context-sensitive, multi-lingual Online Help
- Improvement of interfacing for workflows. Certain standard workflows can be stored for the status values "New", "Changed" and "Delete" (in the Quick Edit bar), which are recommended to the editor or are executed with priority. In addition, from FirstSpirit Version 4.1, a project-specific workflow can be tied directly to the previous controls for deleting (menubar buttons, context menu entry) elements (see Chapter 4.9 page 218). The evaluation order for the Quick Edit bar starts with the stored standard workflows (highest priority) and ends with the standard "Delete" function (lowest priority).
- Redesign of the input components CMS_INPUT_LINKLIST and CMS_INPUT_CONTENTAREALIST
- Internal scalability optimisations: Performance and main memory requirements
- Improved multi-user synchronisation
- WebEdit as project locale web application: With this feature WebEdit can now be combined for single projects for example with a project spezific personalisation configuration (see FirstSpirit Manual for Administrators, Chapter "WebEdit as local project application (from V4.1)“).
- Support for the permission definition component CMS_INPUT_PERMISSION with the following restrictions:
  - No support for scripts
  - (to date) only 1 can be used per page
  - No conflict visualisation
- Support for the most important input components in the Page and Content Store:
  - DOM editor: The DOM editor supports WYSIWYG for format templates and bold/italic and underline. Support for internal and external links and links on elements from the Content Storevia appropriate link editors. Any existing

  configuration is adopted from the form.
  - DOM table editor: Convenient insertion of rows and columns. Each cell can be separately edited in the DOM editor.

- Generic link editors: In FirstSpirit Version 4.2 the configuration options for links are enhanced by the introduction of generic link editors. The use of the related input components is restricted in WebEdit.
- Enhancements for Input components and selection dialogues:
  - Preview in picture input components
  - Display of elements in the Media Store
  - Fading effects when loading new contents (e.g. when generating a preview)
  - Additional control elements can be displayed in layers and iFrames into the preview.
- Remote Media function: The multiple definition of remote projects is supported in WebEdit too – with restrictions:
  - The definition of more than one remote project (<REMOTE...>) and the definition of categories (<CATEGORY...>) is supported for WebEdit only for the input components CMS_INPUT_FILE and CMS_INPUT_ PICTURE.
  - If more than one remote project is configured for an input component each with the parameter uploadFolder only the first *uploadFolder* is taken into account in WebEdit. Further *uploadFolder* of other remote projects are ignored in WebEdit.
- Page Store:
  - Editing folders (i.e. create/delete/rename)
  - Edit pages (i.e. create/delete pages and create/delete/sort sections)
- Media Store:
  - Editing folders (i.e. create/delete/rename)
  - Create/delete pictures/files
  - Automatic assignment of reference names
- Content Store:
  - Overview table (incl. paging, but without sort)
  - Create/delete data records
  - Search: "Simple search" and "saved search" incl. parameters
- Site Store:
  - Editing folders (i.e. create/delete/rename)

- Create/delete/rename pictures and files (language-independent only)

- Support for multi-lingualism
- Support for workflows
- Automatic, unique assignment of reference names: The WebEdit functions "New" and "Create" automatically convert non-unique reference names (or reference names with special characters). References names which are the same are automatically assigned as unique names by appending a consecutive number (e.g. by appending "_1").
- Enhancement of the WWW rollout: More files are automatically updated when the server is started.
- Optimisation of the transition in workflows: If only one activity can be switched, this activity is selected directly and the transition dialog appears.

8.2.1 New in Version 4.1

Development in Version 4.1 focuses on the user. The functions planned within the scope of "WebEdit 5.0" development have already been implemented in the FirstSpirit JavaClient. Comprehensive revision of the WebClient is not planned until Version 5.

However, several new functions have already been realised for WebEdit 4.1. In particular, the option to configure WebEdit as a local project web application is new (see FirstSpirit Manual for Administrators, "WebEdit as local project application" chapter). This means that WebEdit can now, for example, be combined with a project-specific personalisation configuration.

The following functions have been supported in WebEdit to date:

- Media restrictions (with restrictions – see Chapter 8.2.2 page 299)
- Linking workflow to the "Delete" function
  (see Chapter 4.9 page 218)
- Language enhancement: Apart from German, English, French, Spanish and Russian, the FirstSpirit applications JavaClient and WebClient are now also available in Italian. The language setting of the menu labelling, context menus and dialogs can be selected via the FirstSpirit start page.

8.2.2 New in Version 4.2

The WebClient has been comprehensibly redesigned for WebEdit 4.2. To this end (among other things), the "xp" theme, which can be configured for a project by the project administrator, has been revised.

An important new function of WebEdit 4.2 is “Easy-Edit“. This will be described in Chapter 8.3 from page 305, a detailed description of the configuration options can be found in the FirstSpirit Online Documentation[7].

From FirstSpirit Version 4.2, the configuration options for links have been considerably enhanced by the introduction of so-called “generic link editors”. For this purpose new input components have been introduced among other things. The functionality of the generic link editors is alos supported by WebEdit, but with restrictions in comparison with the use in JavaClient. For information for developers about using the functionality see FirstSpirit Online Documentation, Chapter “Link templates”[8], for information about restrictions in WebEdit see FirstSpirit Online Documentation, Chapter “WebEdit”[9].

For further new functions see *FirstSpirit Release Notes Version 4.2* and *FirstSpirit Online Documentation*[10].

[7] FirstSpirit Online Documentation - Chapter: ../Template development/WebEdit/Easy-Edit

[8] FirstSpirit Online Documentation - Chapter: ../Template development/Link templates/Generic link editors

[9] FirstSpirit Online Documentation - Chapter: ../Template development/WebEdit/Restrictions

[10] FirstSpirit Online Documentation - Chapter: ../Template development/WebEdit

8.2.3 Restrictions

At the present time (09-2009), the following restrictions must be noted in WebEdit mode:

- "New Window" projects: When new windows are opened, there is no WebEdit bar.
- WebEdit preview: It is possible that the preview is not up-to-date, for example, if changes are made to the templates. In this case, a current preview can be requested manually.
- Despite extensive optimisations within the scope of implementation of WebEdit, very large projects and/or slow network connections sometimes result in substantial delays (see below).
- Tree view: The "Locked" state is not displayed in the tree view of the stores. On the other hand, the coloured marking for the status of a workflow on an object is shown.
- Support for move operations within the tree is not implemented. However, it is possible to change the order of the sections below a page (or the order of the menu levels in the Site Store).
- The following functions for increasing the usability of FirstSpirit JavaClient are not available in WebClient (see also FirstSpirit Release Notes 4.2):
  - Multi-Tabbing (horizontal tabs for convenient editing of several workspaces)
  - Breadcrumb navigation (display of the path from the Store root up to the current element above the form area)
  - Individual display of the stores (tree view reduced to one Store)
  - Integrated preview
  - Content highlighting
  - Restore settings on restart
- In addition, the following functions which were implemented as new functions in Version 4.2 of FirstSpirit JavaClient, are not available in WebEdit:
  - Multi-lingualism of content areas: Instead of the language-dependent display names which are displayed in JavaClient, the reference name is displayed in WebClient.
  - Media galleries
  - Importing MS Word documents

- Drag & Drop FirstSpirit objects (e.g. from the local file system into the JavaClient and vice versa, between two workspaces, from the search dialog)
  - Status display for objects
  - Quick text search
  - Project homepage
  - New start dialog
- Site Store: No complete support for page groups. All page references created using the Quick Edit bar are located in the "Default" page group.
- Content Store:
  - The "advanced search" and the full text search are not available.
  - Saved queries can only have parameters of the type "String". (In the JavaClient, Boolean, integer, double and date types are also supported.)
  - Data sets can not be displayed in multi-line-view.
- Media Store:
  - From FirstSpirit Version 4.1 uploading media in the Media Store of the FirstSpirit JavaClient and the FirstSpirit WebClient can be limited to specific file sizes and formats. As the file selection dialog available in the browser for uploading media is not an independent implementation of FirstSpirit, but instead is permanently integrated in each browser (e.g. Firefox, Mozilla, Internet Explorer, Opera), filtering (as in the JavaClient) is technically not possible in the WebClient. The files are therefore not filtered until after the upload and any error message on exceeding the media restrictions defined in the project configuration is issued to the user.
- Input components:
  - Pre-configuration for input components is supported in individual cases only.
  - CMS_INPUT_TEXTAREA: The length restriction has no effect.
  - CMS_INPUT_DOM: No support for links to custom link editors.
  - CMS_INPUT_DOM: Limited support for lists. Nesting within a list is not possible.
  - CMS_INPUT_DOM: No length restriction.
  - CMS_INPUT_DOMTABLE: No support for cell formatting (format templates are possible, however, no cell-specific attributes).

- The following are not planned for WebEdit:
  - Changes in the Template Store (i.e. all types of templates, workflows and schemata CANNOT be edited via WebEdit.)
  - Definition of variables in the Site Store.
  - Resolutions in the Media Store.
  - Support for permission assignment.
  - Support for document groups.
  - CMS_INPUT_DOMTABLE: No merging and splitting of cells.

> ! *For information about restrictions in the functionality of FirstSpirit WebEdit please see also FirstSpirit Online Documentation*[11]*.*

[11] FirstSpirit Online Documentation – Chapter: ../Template development/WebEdit/Restrictions

8.2.4 Implementing WebEdit input components

Supported input components: The following input components were implemented within the scope of the WebEdit implementation:

- CMS_INPUT_TEXT: single line text.
- CMS_INPUT_TEXTAREA: multiple line text without formatting.
- CMS_INPUT_DOM: formatted text with format templates and links (including database links, but no support for lists, limited support for inline tables (from FirstSpirit Version 4.2, see Chapter 2.8 page 95).
- CMS_INPUT_DOMTABLE: Table with formatted text including format templates and links (WITHOUT cell merging and formatting, incl. DOM restrictions).
- CMS_INPUT_COMBOBOX: Simple selection from a dropdown list.
- CMS_INPUT_RADIOBUTTON: Simple selection from a displayed list.
- CMS_INPUT_CHECKBOX: Multiple selection from a displayed list.
- CMS_INPUT_PICTURE: Media selection
- CMS_INPUT_FILE: File selection
- CMS_INPUT_PAGEREF: Page reference selection
- CMS_INPUT_LINKLIST: List of links
- CMS_INPUT_NUMBER: Numbers
- CMS_INPUT_DATE: Date (without data selection dialog)
- CMS_INPUT_CONTENTAREALIST: List of sections
- CMS_INPUT_LINK: Ability to create and edit links.
- CMS_INPUT_LIST: Selection from a set of pre-defined list entries.
- CMS_INPUT_TOGGLE: Switch between two pre-defined values.
- CMS_INPUT_SECTIONLIST: List of all existing sections of a page.
- CMS_INPUT_PERMISSION: Permissions definition for user permissions. Separate configuration is required for use of the permission filter within a WebEdit environment (for further information, see FirstSpirit Manual for Administrators). Restrictions to use of the component exist in WebEdit, for example, the use of validation scripts is not possible.

- CMS_INPUT_OBJECTCHOOSER (from FirstSpirit Version 4.2): Data set selection (restrictions see FirstSpirit Online Documentation[12])
- FS_DATASET (from FirstSpirit Version 4.2): Data set selection (restrictions see FirstSpirit Online Documentation)
- FS_LIST (from FirstSpirit Version 4.2): List of sections (restrictions see FirstSpirit Online Documentation)
- FS_REFERENCE (from FirstSpirit Version 4.2): Reference selection (restrictions see FirstSpirit Online Documentation)

not supported (planned):

- Complex components within the Content Store (CMS_INPUT_TABLIST, CMS_INPUT_CONTENTLIST)

not supported:

- CHART components: Chart graphics
- FONT: Picture generation from Windows fonts

8.2.5 Implementing WebEdit design elements

Supported design elements: The following design elements in the form area were implemented within the scope of the WebEdit implementation

- CMS_COMMENT: Comment on individual parts within the form area of a page or section template.
- CMS_GROUP: Graphic grouping of input components as a group. In WebEdit, unlike JavaClient, the grouped elements can only be displayed as a tab in the top part of the form area (not on the page).
- CMS _LABEL: For specifying additional labelling on each page or on each section.

[12] FirstSpirit Online Documentation – Chapter: ../Template development/WebEdit/Restrictions

8.3 Easy-Edit (from V4.2)

The "Easy-Edit" function has been introduced to enable direct editing of sections within the preview page without using separate windows. It can therefore replace the Quick Edit function at section level (see Chapter 8.5.1 page 316).

Existing FirstSpirit projects do not have to be migrated. Easy-Edit is an additional function, i.e. existing projects can initially continue to be used as usual without adjustments (and therefore without Easy-Edit).

To use the "Easy-Edit" function, the templates of a project must be adjusted first. Special templates, the Easy-Edit format templates, are used for this. They can be combined with the WebEdit format templates and, just like these, are already included in the FirstSpirit scope of supply. The Easy-Edit format templates are created in the project together with the WebEdit format templates using the FirstSpirit server and project configuration, if the project property "Use WebEdit" is enabled. Apart from the format templates, a new media folder "WebClient Media (EasyEdit)" is created in the project for the Easy-Edit function.

! *For release projects, the "WebClient Media (EasyEdit)" folder and the media contained in it must be manually released in the project. On disabling the project property "Use WebEdit", the format templates and the "WebClient Media (EasyEdit)" media folder are removed from the project. In this case, release on the higher level folder of the Media Store takes place automatically.*

The Easy-Edit format templates must be added to the page, section and/or table templates in which the Easy-Edit function is to be used.

! *A precise description of the format templates is given in the FirstSpirit Online Documentation*[13].

[13] FirstSpirit Online Documentation – Chapter: ../Template development/WebEdit/Easy-Edit

The technologies used for the "Easy-Edit" function intervene in the HTML source code of a FirstSpirit project far more than the WebEdit functions used to date. For this reason, it is not possible to guarantee that "Easy-Edit" can be used without changes to the project HTML. Use of JavaScript must at least be enabled within the browser settings; however, other changes to the browser configuration through to adjustments in the project may be necessary.
Furthermore, problems can occur if JavaScript Frameworks are used in projects, as Easy-Edit uses the JavaScript Framework MooTools. Manual adjustments may also be necessary.

The "Easy-Edit" function already provides new user prompting concepts in WebEdit 4.2, which will be continued with the launch of WebEdit 5.0. However, in WebEdit 5.0, the technologies on which these are based will be implemented in a completely new way with the help of the GWT framework[14]*. In order to keep the migration work for projects as small as possible, it is planned to keep template changes to the Easy-Edit function of Version 4.2 compatible with Version 5.0. However, this cannot be guaranteed at the present time.*

[14] GWT – Google Web Toolkit

8.4 WebEdit format templates

The WebEdit standard format templates can be installed and updated by the project administrator via the FirstSpirit server and project configuration. These format templates can then be referenced and used within the project templates.

The scope of supply of FirstSpirit includes the following WebEdit format templates:

`WEBeditBarIncludeJS`	WebEdit basic functions for the Quick Edit buttons.
`WEBeditEditAttribute`	Editing an input component.
`WEBeditEditContent`	Edit a data record.
`WEBeditEditSectionAttributes`	Edit all input components of a section.
`WEBeditIncludeJS`	WebEdit basic functions for the Inline buttons.
`WEBeditQuickBar`	Display the Quick Edit buttons at page or section level.
`WEBeditScripts`	Enables up to three script buttons to be shown in the WebEdit toolbar.
`WEBeditSelectPicture`	Edit a picture.
`WEBeditSwitch`	Switch edit mode (old)
`WEBeditSwitch2`	Switch edit mode (new)

The WebEdit format templates are located in the Template Store in the "WebClient Format Templates" node:

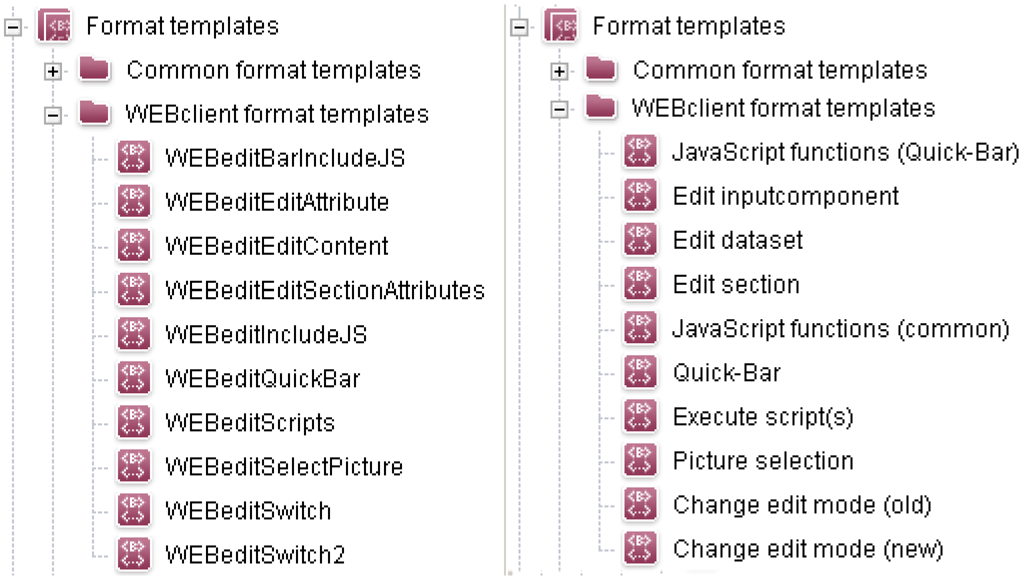


Figure 8-2: WebEdit format templates (reference names/display names)

The display of names in the tree depends on the "Extras" – "Preferred display language" setting (see Figure 8-2).

The instruction $CMS_RENDER(...)$ is used within a template to integrate the content of a format template. The WebEdit format templates therefore have to be integrated within the required page and section templates using $CMS_RENDER(...)$:

```
$CMS_RENDER(template:"TEMPLATE", OPTIONS)$
```

For further information on format templates or the $CMS_RENDER(...)$ instruction, see FirstSpirit Online Documentation.

Note: The optional parameters `WEBeditExternal` and `WEBeditMode` of the templates can be alternatively defined project-wide in the Site Store.

When using templates, note that `WEBeditIncludeJS` must be used in the <HEAD>-area of each the project's page templates!

8.4.1 WEBeditBarIncludeJS

`WEBeditBarIncludeJS` is used to make the WebEdit function "Quick Edit" availalbe to a page. The "`WEBeditBarIncludeJS`" template must be used in each page template of the project in which "Quick Edit" is to be used. In addition, each page template must contain `WEBeditIncludeJS` .

```
$CMS_RENDER(template:"WEBeditIncludeJS")$
$CMS_RENDER(template:"WEBeditBarIncludeJS")$
```

8.4.2 WEBeditEditAttribute

`WEBeditEditAttribute` requires the mandatory parameter `name` for specifying the input components (e.g. "st_text"). In addition, the parameter `tooltip` can be used to assign an alternative tooltip.

```
$CMS_RENDER(template:"WEBeditEditAttribute",name:"st_subheadline")$
```

Mandatory parameters:

`name:` Name of the GUI component ("name" attribute of the <CMS_INPUT_...> component), e.g. `name:"st_text"` or `name:"st_headline"`.

Optional parameters:

`tooltip:` Specification of a different tooltip for the display when the cursor is moved over the link, e.g. tooltip: “Edit heading".

8.4.3 WEBeditEditContent

`WEBeditEditContent` does not require any mandatory parameters. The optional the parameter `tooltip` can be used to assign an alternative tooltip. The optional parameter `WEBeditExternal` can be used to display the WebEdit buttons even in generated and deployed statuses ("1" display in generated status also, "0" display within the preview only).

The Uid of the table template and the ID of the currently rendered data record are normally used. The optional parameter `content` can be used to change the default Uids of the table template used `($CMS_VALUE(ContentName)$)`. The optional parameter `index` can be used to change the default data record ID used `$CMS_VALUE(#row.getId())$)`.

```
$CMS_RENDER(template:"WEBeditEditContent")$
```

Optional parameters:

`tooltip:` Specification of a different tooltip for the display when the cursor is moved over the link, e.g. `tooltip: "Edit Heading"`.

`WEBeditExternal:` If the value here is set to "1", the WebEdit buttons are displayed even in the generated status, if value "0" is set the WebEdit buttons are hidden in the generated status.

`content`: Specification of the Uid of the table template, e.g.: `content:"glossar.glossary"`.

`index`: Specification of the data record ID in the schema, e.g. `index: "128"` or `index:#row.getId()`

8.4.4 WEBeditEditSectionAttributes

`WEBeditEditSectionAttributes` does not require any mandatory parameters. The optional the parameter `tooltip` can be used to assign an alternative tooltip.

```
$CMS_RENDER(template:"WEBeditEditSectionAttributes")$
```

Optional parameters:

`tooltip`: Spcification of a different tooltip for the display when the cursor is moved over the link, e.g. `tooltip: "Edit Heading"`.

8.4.5 WEBeditScripts

`WEBeditScripts` has six optional parameters: `script1`, `scriptTooltip1`, `script2`, `scriptTooltip2 sript3` and `scriptTooltip3`.

```
$CMS_RENDER(template:"WEBeditScripts"
[,script1:"PARAMETER",scriptTooltip1:"TOOLTIP"]
[,script2:"PARAMETER",scriptTooltip2:"TOOLTIP"]
[,script3:"PARAMETER",scriptTooltip3:"TOOLTIP"])$
```

The parameters `script1` to `script3` are used to transfer a character string of parameters which define which parameters are to be taken into account for the script. Within this character string the key and value are separated by the equals sign ("=") and a key/value pair is separated by the "&" symbol. There is NO "&" symbol in front of the first key/value pair. `scriptTooltip1` to `scriptTooltip3` are used to indicate the tooltips for the buttons in the WebEdit toolbar, for example "Execute script".

In WebEdit the keys `script`, `id`, `store` and `templateset` are evaluated in a parameter character string, whereby `script` is a mandatory key and must contain the unique name of the script to be executed. Other keys are made available to the script context.

With `templateset`, either the name or the number of the script's presentation channel must be given. The `id` and `store` keys indicate in which store (e.g.

store=mediastore) and on which node (e.g. id=12345) the script is to be executed. These two keys must always be given together.

Example:

```
$CMS_RENDER(template:"WEBeditScripts",script1:"script=myTestScript&tem
plateset=html&store=mediastore&id=23456&parameter1=value1"
```

In the example, the content of the script's "html" channel with the unique name “myTestScript”, is executed in the Media Store on the node with the ID "23456". The parameter `parameter1` is available in the script context with the value `value1`.

Optional parameters:

`script1:` Specification of the parameter URL for the first script button, e.g. script1:“script=myScript".

`scriptTooltip1:` Tooltip for the first script button in the WebEdit toolbar, e.g. scriptTooltip1:“My 1st script".

`script2:` Specification of the parameter URL for the second script button, e.g. script2:“script=myScript".

`scriptTooltip2:` Tooltip for the second script button in the WebEdit toolbar, e.g. scriptTooltip2:“My 2nd script".

`script3:` Specification of the parameter URL for the third script button, e.g. script3:“script=myScript".

`scriptTooltip3:` Tooltip for the third script button in the WebEdit toolbar, e.g. scriptTooltip1:“My 3rd script".

Parameter URL:

`script=VALUE:` Specification of the unique name of the script to be executed, e.g. script=myScript.

`templateset=VALUE:` Specification of the presentation channel of the script to be executed, either via the channel number or the channel name, e.g. templateset=0 or templateset=html.

`store=VALUE&id=VALUE:` ID and store of the node on which the script is to be executed, e.g. store=mediastore&id=23884.

`NAME=VALUE:` Other variables, which are to be made available in the script context, e.g. date=03.04.2005&editor=Franz

8.4.6 WEBeditSelectPicture

WEBeditSelectPicture has the same mandatory and optional parameters as WEBeditEditAttribute: `name` for the name of the input component and `tooltip` as an optional parameter (cf. Chapter 8.4.2).

```
$CMS_RENDER(template:"WEBeditSelectPicture",name:"st_picture")$
```

Mandatory parameters:

`name:` Name of the GUI component ("name" attribute of the <CMS_INPUT_...> component), e.g. `name:"st_picture".`

Optional parameters:

`tooltip:` Specification of a different tooltip for the display when the cursor is moved over the link, e.g. `tooltip:"Edit picture"`.

8.4.7 WEBeditIncludeJS

`WEBeditIncludeJS` is used to make the WebEdit standard functions available in a page, therefore the parameter must be used in each page template of a project.

```
$CMS_RENDER(template:"WEBeditIncludeJS")$
```

8.4.8 WEBeditQuickBar

`WEBeditQuickBar` is used to add the Quick Edit bar at page/section level. `WEBeditQuickBar` has no mandatory parameters. The following optional parameters can be specified: `barOrientation`, `highlightContainer`, `highlightClass`, `extended`, `wfNew`, `wfChanged`, `wfDelete` and `wfForce`. `barOrientation` is used to specify the fold-out direction of the Quick Edit bar. `hightlightContainer` and `highlightClass` define an area to be highlighted and assign a CSS class to it. `extended` can be used to affect the initial fold-out behaviour of the Quick Edit bar (folded out/extended or folded up/retracted). The parameters `wfNew`, `wfChanged` and `wfDelete` are used to define the recommended workflows for the object status. `wfForce` can be used to force execution of a recommended workflow. A detailed explanation is given in the next chapter.

```
$CMS_RENDER(template:"WEBeditQuickBar",barOrientation:"left",highl
ightContainer:"hc" + #global.page.id)$
```

Optional parameters:

`tooltip:` Specification of a different tooltip for the display when the cursor is moved over the link, e.g. `tooltip:"Edit"`.

`barOrientation:` Specification of the fold-out direction of the Quick Edit bar. If the value "left" is specified, the Quick Edit bar is folded out (dropped down) to the left, if "right" is specified it folds out to the right, e.g. `barOrientation:"left"`. If no parameter is specified the Quick Edit bar folds out to the right.

`extended:` Specification of the initial fold-out behaviour of ONE Quick Edit bar on a preview page, e.g. extended:“true“. "true" means dropped down and "false" means folded up. If the parameter is not specified the Quick Edit bar is folded up (see Chapter 8.5.6 page 318). The parameter can only be used once per page.

`highlightContainer:` ID of the element whose content is to be especially highlighted, e.g. highlightContainer:“section1" (see Chapter 8.5.7 page 318). Examples:

- `highlightContainer: "hc" + #global.id`
- `highlightContainer: "hc" + #global.page.id`
- `highlightContainer: "hc" + #global.section.id`

`highlightClass:` Specification of a CSS class which is to be used for the element which as specified with highlightContainer. If no CSS class is specified, the standard CSS class "weHighlight" is used (see Chapter 8.5.7 page 318). Examples:

- `highlightClass: "weContainer"`
- `highlightClass: "layout"`

`wfNew:` Name of the recommended workflow for "new" status, e.g. `wfNew:"Release request"`.

`wfChanged:` Name of the recommended workflow for "changed" status, e.g. `wfChanged:"Release request"`.

`wfDelete:` Name of the recommended workflow for "Delete" status, e.g. `wfDelete:"Delete"`.

`wfForce:` Force execution of the recommended workflow for a status. "true" means force execution, "false" means execution is not forced. If the parameter is not specified, the execution is not forced, e.g. `wfForce:"true"` or `wfForce:"false"`.

`disableButtons:` Specification of Quick Edit bar buttons (at page level), which are to be concealed in the bar. At present, hiding the buttons using the attribute `disableButtons` is only possible for the "Create Page" (`disableButtons:"newpage"`) and "Extras" (`disableButtons:"extras"`) buttons (cf. Chapter 8.5.4 page 317). The two parameters can also be specified combined (`disableButtons:"newpage,extras"`). These buttons are no longer displayed when the Quick Edit bar is opened.

8.4.9 WEBeditSwitch

The format template `WEBeditSwitch` can be used to realise a jump into a WebEdit preview page ("DeepLink") from a page which is not a preview page.

```
$CMS_RENDER(template:"WEBeditSwitch",guiLanguage:"EN",login:"User_
A",password:"PW_A")$
```

<u>Optional parameters:</u>

`tooltip:`Specification of a different tooltip for the display when the cursor is moved over the link, e.g. `tooltip:"Display page".`

`guiLanguage:` Specification of a valid interface language which is to be used for WebEdit, e.g.`guiLanguage:"EN".`

`login:` Specification of a valid user name for authentication in the WebClient, e.g. `login:"User_A".`

`password:`Specification of a valid password for authentication in the WebClient, e.g. `password:"PW_A".`

8.4.10 WEBeditSwitch2

The format template `WEBeditSwitch2` can be used to realise a jump into a WebEdit preview page ("DeepLink") from a page which is not a preview page. Unlike `WEBeditSwitch`, apart from the conventional authentication ("plain"), other authentication methods can also be used (e.g. authentication using SAP ticket).

```
$CMS_RENDER(template:"WEBeditSwitch2",guiLanguage:"EN",login:"plai
n",user:"User_A",password:"PW_A")$
```

Optional parameters:

`tooltip:` Specification of a different tooltip for the display when the cursor is moved over the link, e.g. `tooltip:"Go to".`

`guiLanguage:` Specification of a valid interface language which is to be used for WebEdit, e.g. `guiLanguage:"EN".`

`user:` Specification of a valid user name for authentication in the WebClient, e.g. `user:"User_A".`

`login:` Specification of a valid authentication procedure for authentication in the WebClient, e.g. `login:"ticket".` Other login options:

▪ `plain`	Plain text authentication
▪ `hash`	Base64 authentication
▪ `ticket`	Authentication using SAP ticket

`ticket:` Specification of a ticket for SAP authentication. The ticket parameter is only required is "ticket" has been defined as the authentication method for the login parameter.

`password:`Specification of a valid password for authentication in the WebClient, e.g. `password:"PW_A".`

8.5 Quick Edit

8.5.1 Use of Quick Edit in FirstSpirit projects

The "Quick Edit" function provides a fast and easy option for performing editorial changes and processes at page and section level.

At present the following operations are available to the editor at page level:

- Create a new menu level
- Create a new page
- Create a new section
- Edit the page in the Page Store.
- Start or forward workflow on the page reference
- Edit the metadata of the page reference
- Delete page
- Help

! *Some of these functions are covered by the Easy-Edit functionality. From FirstSpirit Version 5.0 the Quick Edit bar will no more be supported and replaced by Easy-Edit.*

The operations at section level are:

- Create a new section
- Edit the section in the Page Store.
- Move section up by one position
- Move section down by one position
- Edit metadata of the section
- Delete section
- Help

! *From FirstSpirit Version 5.0 the Quick Edit bar will no more be supported and replaced by Easy-Edit.*

Two steps are necessary to be able to use Quick Edit. First, general functions must be made available at page level (see Chapter 8.5.2 page 317) and secondly Quick Edit buttons must be integrated at page and section level (see Chapter 8.5.3 page

317).

8.5.2 General functions of Quick Edit at page level

All pages of a WebEdit project which are to use the Quick Edit function must be modified. It is advisable to make the settings directly in the page templates. To this end, the general Quick Edit functions must be made available first in the WebEdit presentation channel. In addition to render format template `WEBeditIncludeJS`, the render format template `WEBeditBarIncludeJS` must also be added to the page templates between the opening and closing HTML head tag:

```
...
<html>
  <head>
    <title>Page template</title>
    $CMS_RENDER(template:"WEBeditIncludeJS")$
    $CMS_RENDER(template:"WEBeditBarIncludeJS")$
  </head>
  <body>
...
```

The render format template `WEBeditBarIncludeJS` does not require any additional parameters, just like `WEBeditIncludeJS`.

8.5.3 Integrating the Quick Edit buttons

Quick Edit buttons can be integrated both on page and on section level. A render call must be added to the page or section template for a Quick Edit button:

```
$CMS_RENDER(template:"WEBeditQuickBar")$
```

Whether the page or section operation is displayed in Quick Edit depends on whether the render call is added in the page or section template.

! *ONLY ONE render call may take place for "WEBeditQuickBar" in a page or section template.*

8.5.4 Hiding Quick Edit bar buttons

it is possible to hide certain buttons of the Quick Edit bar (at page level). At present, hiding the buttons using the attribute `disableButtons` is only possible for the "Create Page" and "Extras" buttons (cf. Chapter 8.4.8 page 312). These buttons are no longer displayed when the Quick Edit bar is opened. The render call in the page template must be adjusted to hide the Quick Edit buttons:

Example of hiding the "Extras" button:

```
$CMS_RENDER(template:"WEBeditQuickBar",disableButtons:"extras")$
Extras
```

Example of hiding the "Create page" and "Extras" buttons:

```
$CMS_RENDER(template:"WEBeditQuickBar",disableButtons:"newpage,extras"
)$
```

8.5.5 Orientation of the Quick Edit bar

As the Quick Edit bar overlaps part of the content when it is dropped down and the space for dropping down in one direction can be smaller than the Quick Edit bar, the fold-out direction should be specified for the Quick Edit bar. The parameter for the fold-out direction is called `barOrientation`. The two possible values for `barOrientation` are "right" (Quick Edit bar folds open to the right) and "`left`" (Quick Edit bar folds out to the left). If the parameter is not specified the Quick Edit bar always folds out to the right.

```
$CMS_RENDER(template:"WEBeditQuickBar",barOrientation:"left")$
```

8.5.6 Drop down the Quick Edit bar

In addition to the orientation of the Quick Edit bar, the initial drop down behaviour can also be changed. The fold-out behaviour can be configured for exactly one Quick Edit bar on a page, i.e. for the page template and all section templates used. The name of the attribute is `extended` (see Chapter 8.4.8 page 312). Possible values are "true“ (initially dropped down) and "`false`" (initially folded up). If the parameter is not specified the Quick Edit bar is initially folded up.

```
$CMS_RENDER(template:"WEBeditQuickBar",extended:"true")$
```

8.5.7 Highlighting page areas

If Quick Edit is used in a project, it is not always obvious that a Quick Edit button belongs to the section of a page. For example, a button can belong to the previous or following section. To visualise the affiliation for the editor, page areas or sections can be especially highlighted. There are two attributes for the highlighting `highlightContainer` and `hightlightClass` (see Chapter 8.4.8 page 312).

`highlightContainer` is used to specify a unique ID, which is assigned to an HTML tag whose content is to be highlighted. The system object `#global` can be used to output page-related information. The CMS variables `#global.id` (ID of the node), `#global.page.id` (ID of the page) and `#global.section.id` (ID of the section) can be used to generate a unique ID.

! *No “class" attributes may be defined for the HTML tag whose content is to be defined.*

1st example: HTML tag whose content is to be highlighted:

```
<div>
  $CMS_VALUE(fr_st_text)$
</div>
```

could for example look like this after adding the "id" attribute:

```
<div id="hc$CMS_VALUE(#global.section.id)$">
  $CMS_VALUE(fr_st_text)$
</div>
```

2nd example: Specification of the ID in the render call of "`WEBeditQuickBar`":

```
$CMS_RENDER(template:"WEBeditQuickBar")$
```

The call could, for example, look like this after adding the `highlightContainer` attribute:

```
$CMS_RENDER(template:"WEBeditQuickBar",highlightContainer:"hc" +
#global.section.id)$
```

In the example, the content of the element with the ID:
`"hc" + #global.section.id` would be especially highlighted when the Quick Edit bar is opened. The second attribute `highlightClass` can be used to specify a user-defined CSS class for highlighting. If the parameter `highlightClass` is not specified, the CSS class `weHighlight` is used, which has the following content:

```
.weHighlight{
  background-color:ffffd3 !important;
  border:1px solid #464600 !important;
}
```

A user-defined CSS class is specified as follows:

```
$CMS_RENDER(template:"WEBeditQuickBar",highlightContainer:"hc" +
#global.section.id,highlightClass:"webEditHighlighting")$
```

The user-defined CSS class could, e.g. then look like this:

```
<style type="text/css">
```

```
  .webEditHighlighting {
    background-color:#008000 !important;
  }
</style>
```

! *In the case of websites without tables, problems can occur in Internet Explorer if the CSS attribute "float" is used. If the surrounding area has a width of 100% and if a border is to be added for highlighting, the layout collapses. In most cases it then helps to use another HTML tag or to reduce the width.*

8.5.8 Workflow recommendations

In the Quick Edit bar the status the object has is determined during workflows. A differentiation is made between the following status values: "new", "changed" nd "delete". A workflow can be recommended for each status. The attributes for this are: `wfNew` ("new" status), `wfChange` ("changed" status) and `wfDelete` ("delete" status) (see Chapter 8.4.8 page 312). The name of the workflow must be specified for the attributes, e.g. "Request Release".

In addition, immediate execution of the recommended workflow can be forced with the `wfForce` attribute. The possible values are "true" and "false" (default selection).

```
$CMS_RENDER(template:"WEBeditQuickBar",wfNew:"Release
request",wfForce:"true")$
```

A special workflow is required for dependent release of elements for new elements which were created using the Quick Edit functions "Create menu level" and "Create page". Such a workflow is not included in the WebEdit scope of supply as the configuration of such a workflow can differ greatly from project to project.