FirstSpirit™
Unlock Your Content

FirstSpirit™ CorporateContent

FirstSpirit™ Version 5.1

Version	**1.13**
Status	**RELEASED**
Date	**2015-02-04**
Department	FS-Core
Copyright	2015 e-Spirit AG
File name	CONT51EN_FirstSpirit_CorporateContent

e-Spirit AG

Stockholmer Allee 24
44269 Dortmund | Germany

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

info@e-Spirit.com
www.e-Spirit.com

e-Spirit

Table of contents

1 Introduction

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

The Multisite Management area includes functions that allow the distribution, and thereby the reuse, of FirstSpirit content in SiteArchitect. In the process, the user is conveniently supported by the user interface in an optimal way. This reuse is possible across both project and server boundaries.

The essential use cases here are:

- Reusing editorial content and layouts between different projects (sites or clients)
- Simply reusing specific project solutions
- Supporting the development of quality assurance processes (DQP scenario)

Multisite Management includes the *ContentTransport* and *Corporate Content* functions.

The *Corporate Content* functionality is located on the vertical icon bar in the left area of SiteArchitect under the icon and on the menu bar under the menu item "Corporate Content". The desired project contents are combined in the source project into what are known as packages. These packages can be subscribed to in other FirstSpirit projects (target projects).

ContentTransport functionality is located on the vertical icon bar in the left area of SiteArchitect under the icon. The desired project contents are combined in the source project into what are known as features. The feature combination can then be saved as an archive file and be imported into other FirstSpirit projects (target projects).

The advantage of these functions: Objects can be managed at a central location (source project) and can always be kept at the desired version in other projects (target projects).

1.1 Topic of this documentation

Chapter 2 explains the most important *terms and concepts* for working with packages, features and subscriptions. The chapter offers a general overview of the operation of CorporateContent and helps *first-time users* get started (starting on page 6).

Chapter 3 describes the *configuration settings* on the server. The chapter is only relevant for *administrators* (starting on page 18).

Chapter 4 covers the ContentTransport area with all of the functions for creating, editing and publishing ContentTransport features (starting on page 20).

Chapter 5 covers the Corporate Content area with all of the functions for creating, editing and publishing *packages* (starting on page 38).

2 Terms and concepts

2.1 ContentTransport functionality

Different project content can be collected from all of the stores using Content Transport. In addition to templates, FirstSpirit content can now also be transported. Here, **content from the content store** can be selected for reuse in addition to content from the page, media and site stores.

A feature and the elements it contains always relate to one specific project state. This can be a state such as the release state or one of the past states. FirstSpirit also only transports content from this state via the feature. These past contents can easily be viewed at any time. Both the editing form and the inline preview of the SiteArchitect are displayed with the historical data.

An object explicitly added to a feature is used as the start node. All subordinate objects, including entire folders, are also applied to the feature starting from this start node. The parent chain of the object is also taken into account and implicitly applied to the feature. Necessary and optional dependencies are detected automatically and can be added to the feature manually using the "ContentTransport" store area. Once all desired objects are combined in the feature, the feature can be saved as a compressed zip file and be provided for import into the FirstSpirit target projects.

For more information about limitations and for further notes on the "ContentTransport" functionality please refer to Chapter 4.7 page 38.

2.1.1 Feature combining in ContentTransport

Feature combinations do not have to be closed, i.e. not all of the referenced objects have to be included in a feature, since a link to existing objects is established in the target project. The user interface visualizes these open edges at different spots:

Directly in the **tree view:**
The tree view shows the missing elements of the feature for both the entire feature and starting from elements already in the feature. The missing elements can be added to the feature directly by the user.

In the **graphical display** for the combination

The graphical display provides a complete overview of the elements (open edges) that are in the feature or are still missing. The feature combination is supported optimally by being able to add elements to or remove them from the graphs directly.

2.1.2 Cross-server transport of features

Transporting combinations from Content Transport, called "Features", is possible across server boundaries, for example using external storages (see Chapter 4.8 page 40). Thus, Content Transport also optimally supports DQP scenarios and the associated development and quality assurance process which involve transferring new functions from a development system (D) to a quality assurance system (Q) so that the function can be tested there. After being tested successfully, the function is transferred further to the production system (P) starting from D.

Use schedules for automating the transfer of features between DQP systems (see Chapter 4.9 page 42). Moreover, there is an API available.

2.2 CorporateContent functionality

The FirstSpirit CorporateContent function represents a further development of the previous "package management" function that could be used to distribute templates and content between different projects on a server automatically. Content can be reused conveniently and across projects with FirstSpirit this way. An important aspect when combining packages is that all dependent objects also have to be included in the package.

Packages are created and edited in the source project. The project available for importing packages into other projects is designated the source project. The objects are selected from the source project's project tree. What is known as a start node is defined in the process. All subordinate objects, even entire folders, are transferred to the package starting from this node. If all desired objects have been combined, a new package version is created which is then available for importing to all target projects with a valid subscription.

Likely the most important functional enhancement is the option for creating packages at the last released state. Thus, all of the package's elements no longer have to have been released at the time the version is created.

2.2.1 Package types in CorporateContent

FirstSpirit distinguishes between two package types:

- **Content packages:** Content packages contain objects from the page store, the site store and the media store. They *do not contain templates* or objects from the content store.

- **Template packages:** Objects from the template store are integrated into template packages. In addition, a template package is allowed to contain objects from the content store and the media store. Integrating objects from the media store into a template package should, however, be limited to media that are referenced in the templates directly, such as those used for the layout (cascading style sheets, spacer.gif, logos, etc.). Other media objects continue to belong to a content package.

! *Each object can only be integrated into one respective package!*

2.2.2 Package dependencies in CorporateContent

Different objects are combined into packages. Most objects, excepting objects from the media store, can reference additional objects. For instance, a page from the content store could reference an image from the media store and a template from the template store. The dependencies between objects have to be resolved in order to import objects into different projects successfully. In other words, it must be ensured that all objects referenced in a package are also contained in the package. This is the reason behind strictly separating content and template packages.

A distinction is made between two dependencies in the process:

1. Content-related dependencies:
   The dependencies within a content package are resolved **automatically** using what are known as reference graphs (see section 5.8.1.1, page 113). For instance, which objects a page references for each page that is to be taken over in a package is checked in the process. The referenced objects are then also taken over in the package. If there is an object that is to be transferred to a content package, such as a page reference or a folder from the site store, then all associated pages from the page store are also transfered to the package.
   If referenced objects are already integrated into one content package, they cannot be taken over in another context package since each object is only allowed to be included in

one single package. In this case, a dependency on this dependent content package is established by the system. This is shown when creating a package version or in the version list for a package and in the detailed information for the package. The dependent content packages can then be subscribed to manually (see section 5.6.6, page 93). A content package can have multiple dependent content packages.

2. Dependencies on templates

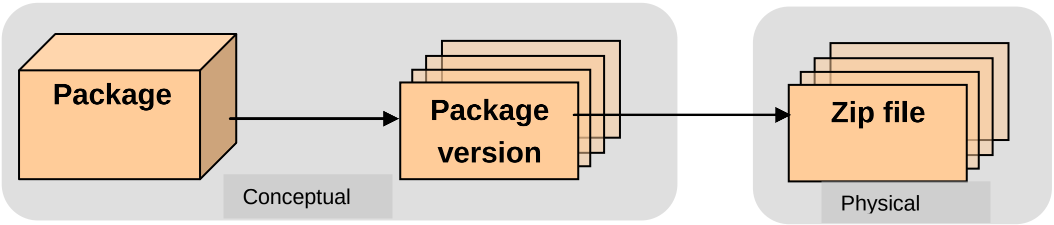
   A content package's dependency on a template **cannot be resolved automatically**. The relationship between a content package and a template package has to be specified in the content package's properties. If there is a dependency between a content package and a template package, a specific sequence has to be followed when creating a version in the master project and when publishing from the master project (see section 5.6.3, page 85).

   Templates can also have dependencies on other templates. These dependencies **cannot be resolved automatically in each instance** since the effects would be very far-reaching in some cases. The package developer should put thought into dependencies and the most effective package structure possible well in advance when packing a template package. The sequence in which objects are added to a package also has to be taken into account in the process. If, for instance, a template has a dependency on a content source, the associated database schema (including table templates and queries) has to be added to the package beforehand.

> ! *For a subscription this means: A package can have a dependency on another package (content or template package). In order to subscribe to a package, all dependent template packages <u>must</u> be subscribed to and all dependent content packages <u>can</u> be subscribed to as well. The import sequence is not arbitrary in this context:*
>
> *Whenever a content package with a **dependency on a template package** is imported, the template package has to be imported first and then the associated content package. If this sequence is not followed, an error message appears and the user can restart the import.*
>
> *Whenever a content package with a **dependency on another content package** is imported, the dependent content package has to be imported first and only then is the content package that contains the references to the dependent objects imported. Errors in the target project may result if this sequence is not followed during the import or if the dependent content packages are not imported.*
>
> *A specific sequence has to be followed for publishing dependent content packages.*

2.2.3 Package definition and package version

A package consists of one or more package versions depending on the specific type (content or template package). Each **package version** has precisely one zip file used for importing into target projects.



The zip file contains all of the data required for the package version and a meta description of the package contents. This meta description is called a **package definition**. The package definition is made hierarchically based on a list of start nodes. These start nodes determine which objects are part of the package. All of the objects below this start node are taken over in the package when it is created. Which objects this exactly entails depends on the structure and contents of the master project.

In addition to the package definition, the dependencies between individual objects have to be taken into account as well (see section 2.2.2, page 9). If there are dependencies between an object contained in a package and another object that is not part of the package, then the dependency is identified automatically using the reference graphs and the referenced object is added to the package even though it is not explicitly part of the package definition. Thus, a package cannot be defined solely via the selected package content. Therefore, a distinction between the package definition and the package version is vital.

Package definition:
Describes the content of a package using the start node from the master project integrated in the package. Referenced objects are not included in this node list and thus are not part of the package definition. The complete content does not result until a *package version* is created using a *package definition*.

Package version:
Contains all of the objects determined using the package definition and all manually referenced objects. Thus a package version provides a complete description of the package contents. Unlike the package definition, which always reflects the most up-to-date content, a package version is only as up-to-date as the date of its last creation.

The package contents change if:

- A new start node is explicitly added, i.e. when a *package definition is modified*.
- An object is implicitly added because it was created from scratch below a start node in the master project (*no change to the project definition)*.

In these two cases, the packages should be refreshed by creating a new package version (see section 5.1.2, page 60). A package version can be released for one or more publication groups.

> ! *If a new object is created below an already integrated start node in an existing package, this object is added to the package automatically and applied in the next package version.*

> ! *Overlap between package contents cannot occur when creating packages. This means, each project node can only belong to exactly one package. Project nodes and objects already used in a package can be identified based on the "ObjectName@PackageName" name extension (in reference names; if the namespace extension has not been disabled, see section 5.1.1.3.1, page 58) and by a package symbol in the project tree. This process increases clarity since otherwise multiple new package versions would have to be created and published at the same time when modifying a single object.*

2.2.4 Publication groups

Creating and publishing packages is a complex task. Incorrect operation can result in problems and conflicts in target projects. Therefore, packages should be thoroughly tested before they are used in a production environment. The concept of a "publication group" was instituted for this purpose (also refer to section 5.6.7, page 95). A publication group is a sort of "marker" that can be assigned to one or more package versions. Packages for specific publication groups can "be released" on master project pages and which publication group a package is intended for is defined on target projects' pages when subscribing to a package. Publication groups are defined server-wide and thus can be used in both master and target projects.

For instance, the following publication groups could be defined:

- Development: For developing packages.
- Test: For projects used for testing packages.
- Production operation: For projects that use a package in a production environment.

The example flow then appears as follows:

Package version	Release for pub. group
Version 0.1	Development
Version 0.2	Development
Version 0.3	Development, Test
Version 0.4	Development, Test
Version 1.0	Production operation
Version 1.1	Development

The "Development" group begins developing a package. The initial 0.1 and 0.2 package versions are only released for this group. The development continues until package version 0.3 is created at some point, which is released for both the "Development" publication group and the "Test" group. This makes this version of the package available to all projects whose subscription was concluded with the "Development" and "Test" publication groups. An automatic or manual update occurs in the target project depending on the project configuration. If the development of the package is concluded, a new package version 1.0 can be released for the "Production operation" group.

As can be seen in the example, multiple versions can be released for one publication group. The package version with the highest package number, i.e. the most up-to-date version, is always used in this case. The package number is unique and is generated when creating a new package version.

2.2.5 Subscription

Subscriptions are created and edited in target projects. Projects that can import packages from a source project are designated as target projects. Only packages defined as "available" in the source project can be subscribed to (see 5.1.1.2, page 51).

A distinction is made between two states for a subscription:

1. **Initialization:** With a subscription, all package contents (such as all of the source project's media files) are initially taken over in the target project and can be edited further by the target project's content editors depending on the package or subscription configuration.

2. **Update:** A new package version has to be created as soon as some aspect of the objects integrated into the package changes in the source project or new objects are to be made available, such as a new image. Each new package version not only contains changes from the preceding version, but also all the modified objects from the preceding version. However, all of the objects contained in the package are no longer replaced when updating a target project with a new package version. Instead, only the newly added and modified objects are replaced.

2.2.5.1 Updating packages in the subscription

A package update can be carried out using two different processes:

1. **Automatic update:** With automatic updating, the decision to update a package is in the hands of the **source project administrator**. From a central location, the administrator initiates the update for all target projects that have a valid subscription to this package by publishing the package (see section 5.6.3, page 85). This is also referred to as a "push" process. Manual intervention on behalf of the person responsible for the target project is not necessary.

2. **Manual update:** With manual updating, the decision to update is in the hands of the **target project administrator**. A new package is made available to the administrator (such as in the package overview, see section 5.6.1, page 75, or in the subscription list) and the administrator can update the project using the new package as needed (see section 5.6.6, page 93). This is also referred to as a "pull" process. The administrative burden is placed on the target projects by manual updating.

Three possible states are feasible during updating:

- An object from the source project is newly created in a target project.
- An object present in a target project is updated with content from the source project.
- A conflict situation occurs (also see section 5.7.4, page 109).

Publication groups have been defined in order to simplify package updating and to avoid errors in production packages (see section 2.2.4, page 13).

2.2.5.2 Subscribing to metadata and project settings templates

In most projects, in addition to standard page templates, there is also a template page for global project settings and for what are known as metadata, if they are used in the project. These templates can be part of a template package and thus can be imported into any target project with a valid subscription. By integrating a project setting template, for example, it is possible to define layout specifications uniformly for headlines and continuous text across an entire project. By integrating metadata, for example, it is possible to work with personalized pages. If these templates are imported into target projects, they can be expanded and adapted to project-specific circumstances there without issue.

! *Imported metadata templates have to be configured in the ServerManager in the project settings under the "Options" item in the "Metadata template" field.*

! *These templates can, in fact, be imported into target projects in both cases, but using them is also not mandatory. This can lead to problems if other packages are based on these project settings or metadata.*

Using CorporateContent only the templates are transfered, not the contents. Please use the ContentTransport functionality for being able to transfer metadata or project settings which have been entered by the editors into other projects (see Chapter 4 page 20).

2.2.5.3 Release

The project-specific concept of release control can also be used for working with packages. Whether the subscribed contents are to be released automatically or not is already determined when subscribing to a package. If **automatic release** is selected, all new or modified objects are released automatically and immediately after being imported, without any action on the part of the target project (for release via workflows see section 2.2.6, page 16). Which objects are modified is not apparent to target project content editors in this case.

In contrast to this procedure, **explicit release** can also be defined. Modified or new objects are shown in red in the target project's project tree in the process and have to be released explicitly by a responsible editor. Advantage: The changes are visible at a glance. This solution does offer more transparency, but would not be very convenient for a larger package scope. For this reason, explicit release can be carried out using a single workflow. When updating a package, a list of the released objects is created at the same time and announced throughout the subscription. All of the objects from this list can then be released using just one workflow (also see section 5.9.3.1, page 131 in this regard).

2.2.6 Integrating workflows and scripts

Packages are usually updated and imported in complex project environments. Integrating workflows into target projects is vital in order to make working as convenient as possible. Specific events are made known to each package in the process. A workflow or script that is started after importing the package can then be assigned to each of these events. Examples of such events include:

- Automatic release: Directly after importing, a workflow is started that releases all new or modified objects automatically, without any action on the part of the target project (see section 2.2.5.3, page 16).
- Resolving a conflict: If a package conflict occurs when importing a package, a workflow that is intended to correct the conflict is started when this event occurs.

- Report function: The report function is particularly interesting for large projects. It creates a log file during import of a package and informs groups of responsible persons about updates.

In the event of a subscription, the assignments created in the package for events are applied by default but can be reconfigured in the target project.

In addition to being run in the target projects, workflows can also be used in the master project. A package update can also basically be initiated using a workflow or a script.

3 Configuration

"CorporateContent" is a license-dependent functionality; this means the "package store" (on the FirstSpirit SiteArchitect menu bar) and the ContentTransport icon (on the FirstSpirit SiteArchitect vertical icon bar) are only shown if a valid license for this functionality is present.

Two steps are needed to activate the functionality:

- Checking the license file and replacing it if necessary (see section 3.1, page 18)
- Activating PackageManagerService (see section 3.2, page 19)

3.1 Checking the license file

The applicable FirstSpirit functions of the license file `fs-license.conf` are shown using the FirstSpirit Monitoring menu "FirstSpirit – Configuration – License". The parameter `license.PACKAGEPOOL` has to be set to a value of `1` for using the "CorporateContent" or "ContentTransport" functions.

If this is not the case, a new valid license can be requested from the manufacturer and be exchanged using FirstSpirit server monitoring.

> ❗ *Tampering with `fs_license.conf` will result in an invalid license. If changes become necessary, please contact the manufacturer.*

The server does not have to be restarted when inserting a new `fs_license.conf` configuration file. The file is updated on the server automatically.

3.2 Starting the "PackageManager" service

The "PackageManager" service on the FirstSpirit server has to be started in the next step. The service can be activated via FirstSpirit Server Monitoring in the "FirstSpirit – Control – Services" area (or via the ServerManager).

The service is started by clicking the "Start" entry. The server does not have to be restarted.

The configuration for automatically starting the service each time the server is restarted can be defined in the "FirstSpirit – Configuration – Services" area.

For configuring using FirstSpirit Server Monitoring, also see the documentation *FirstSpirit Manual for Administrators, Chapter 8*.

4 ContentTransport

The "Content Transport" area is used for creating new features and for editing existing features.

It can be opened using the icon from the vertical icon bar in SiteArchitect. Creating and combining a feature in a source project and updating one in a target project is described in the following sections:

Creating or loading a feature	Section 4.1, page 24
Adding objects to a feature	Section 4.2, page 25
Feature combination	Section 4.3, page 26
Flyout menu	Section 4.4, page 30
Graphical representation of references	Section 4.5, page 31
Updating a feature in a target project	Section 4.6, page 35
Notes on Content Transport	Section 4.7, page 38
Configuring the storage location	Section 4.8 page 40
Automation	Section 4.9, page 42

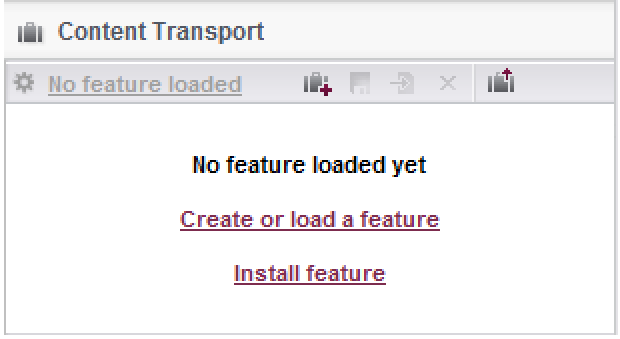


Figure 4-1: Content Transport store

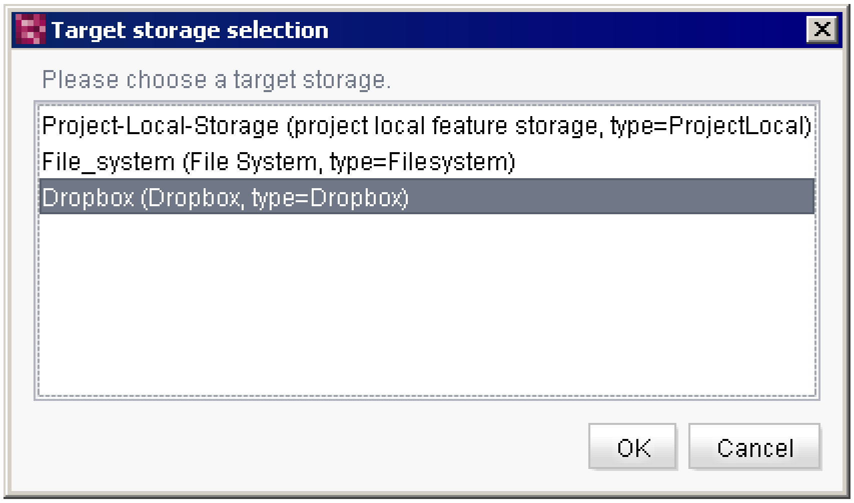
The Content Transport area's icon bar contains entries for creating and editing Content Transport features.

The name of the opened feature is shown after this icon. Clicking the name opens a dialog where the feature name can be modified. If no feature is loaded then "No feature loaded" is displayed.

Create or load a feature; clicking this icon opens a dialog for creating a new feature or loading a feature that already exists (see section 4.1, page 24). This icon is only active if no feature is currently loaded.

Save feature; clicking this icon saves the current feature's object combination. Saving is required in order to enable automatic updating of this feature (refer to section 4.9.1, page 44 for more information).

Create feature Zip file; to use in a target project, the content of the feature is stored in a ZIP file. Using this icon initiates generation of the feature ZIP file. The storage location for the ZIP file can be selected in the following dialog:



Target storage selection

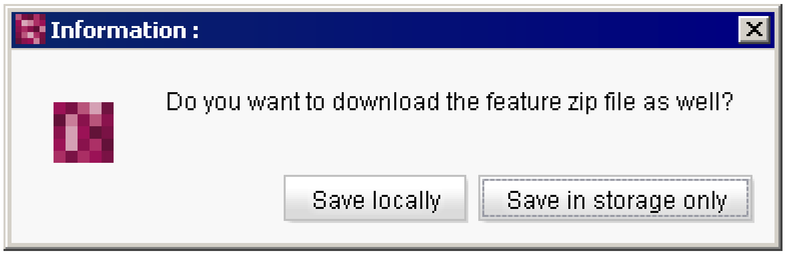
Please choose a target storage.

- Project-Local-Storage (project local feature storage, type=ProjectLocal)
- File_system (File System, type=Filesystem)
- Dropbox (Dropbox, type=Dropbox)

Figure 4-2: Selecting the target storage location

Possible target storage locations can be configured in FirstSpirit ServerManager in the project properties under "Project components"/"FirstSpirit Content Transport Storage App"/"Configuration" or by double-clicking on the project component (refer to section 4.8, page 40). If no other storage location is configured, a default storage location is offered on the local FirstSpirit server ("Project-Local-Storage" storage location).

Clicking "OK" generates the feature ZIP file and saves it to the selected storage location. The user can then select whether the file should also be stored in another, local storage location:



Information :

Do you want to download the feature zip file as well?

Figure 4-3: Saving the ZIP file locally as well

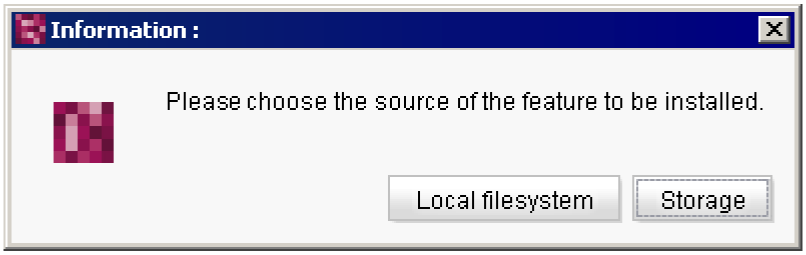
Save locally: The desired target folder for the ZIP file can be selected in following dialog box. The name of the ZIP file is created automatically.

Save in storage only: The ZIP file is only saved in the storage location previously selected.

A message box appears indicating that the feature was saved successfully.

✕ Discard feature; clicking this icon closes the open feature – after affirming a confirmation prompt. Unsaved changes to the object combination are lost.

Install feature; clicking this icon opens a dialog for selecting the source. The call for this function only occurs in the target project.



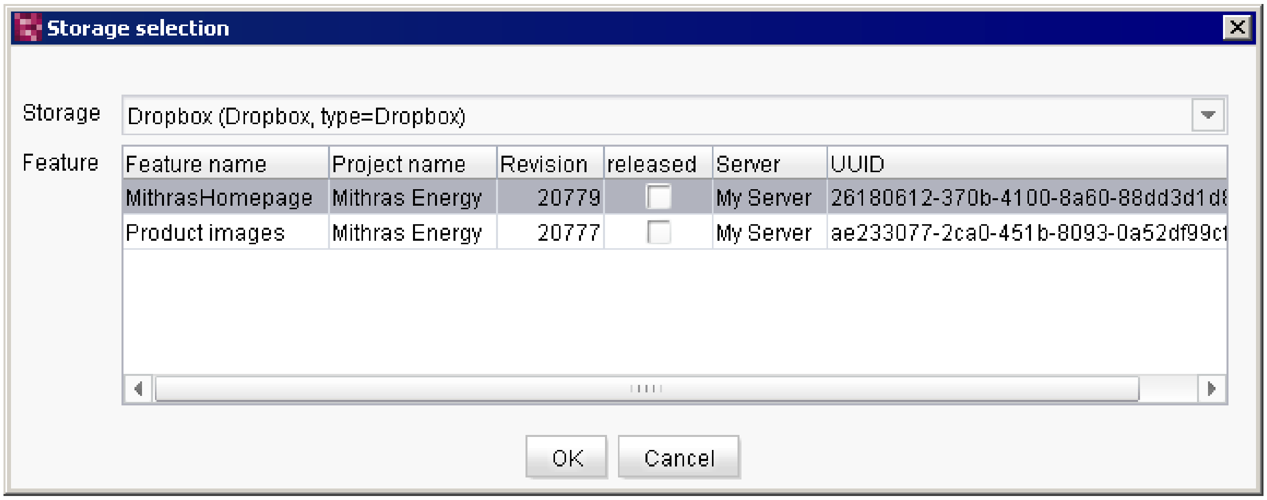
Information :

Please choose the source of the feature to be installed.

Figure 4-4: Selecting a source

Local file system: A dialog opens for selecting a feature ZIP file from the local workstation.

Storage: The feature to be installed can be selected in the following dialog. The storage location from which the feature is to be loaded is selected first. The list is empty if no feature was previously created (see the "Create feature Zip file" function, further up).



Storage: Dropbox (Dropbox, type=Dropbox)

Feature

Feature name	Project name	Revision	released	Server	UUID
MithrasHomepage	Mithras Energy	20779	□	My Server	26180612-370b-4100-8a60-88dd3d1d
Product images	Mithras Energy	20777	□	My Server	ae233077-2ca0-451b-8093-0a52df99c

Figure 4-5: Feature selection

Storage: Storage location selected for feature generation (for configuring storage locations, see section 4.1, page 24). Default storage location on FirstSpirit server: "Project-Local-Storage"

The available features at the particular storage locations are displayed with the following information:

Feature name: Name given to the feature when it was created

Project name: Specifies the name of the source project on the server.

Revision: Feature revision selected when generating the feature.

released: Release state selected when generating the feature.

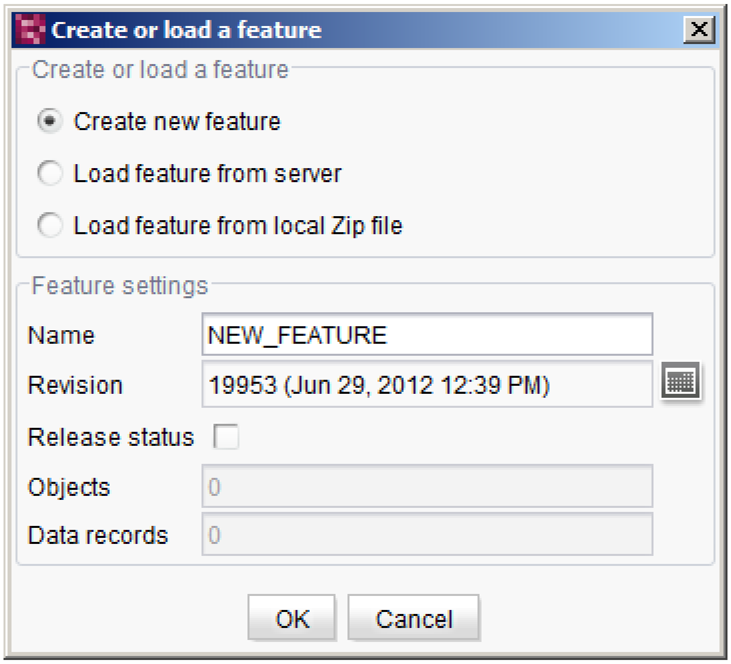
Server: Specifies the name of the server where the feature was created.

UUID: unique ID across servers that was assigned automatically by the system when the feature was generated.

Clicking "OK" installs the selected file in the target project (see section 4.6, page 35).

4.1 Creating or loading a feature

Clicking the icon or the entry "Create or load a feature" in the empty feature area opens a dialog for creating a new feature or for editing or loading an existing feature.



Create or load a feature

Create or load a feature

● Create new feature

○ Load feature from server

○ Load feature from local Zip file

Feature settings

Name: NEW_FEATURE

Revision: 19953 (Jun 29, 2012 12:39 PM)

Release status: □

Objects: 0

Data records: 0

Figure 4-6: "Create or load a feature" dialog

Create or load a feature
The choice of creating a new feature of FirstSpirit objects or loading an already created feature (feature or feature zip file) is made in the upper area of the dialog. Existing feature zip files can be loaded from the local workstation and existing features can be loaded from the FirstSpirit server.

Create new feature: Activating this option creates a new empty feature.

Load feature from server: A dialog for selecting an already existing feature opens. The combo box contains all of the features that have previously been saved on the FirstSpirit server.

Load feature from local Zip file: A dialog for selecting a feature zip file from the local workstation opens.

Feature settings

The settings for the newly created or loaded feature can be edited in the dialog's lower area.

Name: A unique and legible name should be specified when creating a new feature. The name is used to save the feature file on the FirstSpirit server, on the local workstation, or externally.

Revision: A feature always relates to a specific revision of the object (the current revision when being added). The field is used for selecting a maximum revision for all of the FirstSpirit objects contained in the feature. When added to a feature, all objects are stored with the revision in the feature that is directly below this maximum revision.

Release status: If this option is enabled then only the last release state is taken into account for each included object.

Elements: This field specifies how many elements have already been added to the selected feature.

Datasets: This field specifies how many datasets in the selected feature have already been added.

4.2 Adding objects to a feature

4.2.1 Using the tree structure of stores

New objects compatible with Content Transport can be added to a Content Transport feature using the tree structure of the corresponding stores in one of two ways:

- using the object's context menu item **Add to Content Transport feature** or
- using **drag-and-drop** to copy an object to the "Included objects" area (see section 4.3.2, page 27).

The selected object is then added *explicitly* to the feature. Furthermore, all of the selected object's higher level parent elements are *implicitly* added to the list of included objects and all child objects are added *explicitly*.

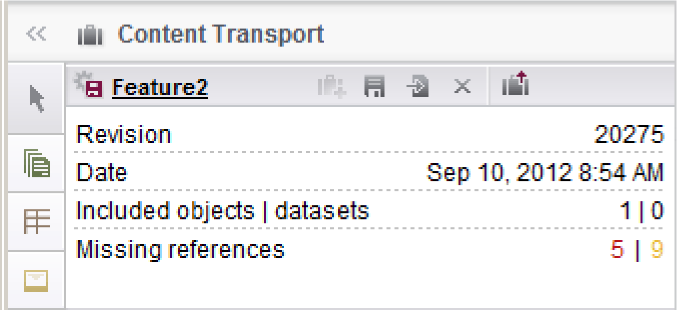
The user works on a view of the datasets in the Data Store when adding datasets. Data sources, filtered data sources (each without contents or datasets) or even individual datasets can be added as objects here.

4.2.2 In the feature combination

Additional objects can also be added to the feature using the areas for required or optional references. The checkbox in front of the respective object just has to be selected for the desired objects, then the objects are included in the feature by clicking the **Add selected** button.

4.3 Feature combination

4.3.1 Overview



Revision	20275
Date	Sep 10, 2012 8:54 AM
Included objects | datasets	1 | 0
Missing references	5 | 9

Figure 4-7: Feature - Overview

Revision: The maximum revision of all included objects is displayed here (with/without release state).

Date: The date and time when the maximum revision was configured is specified here.

Included objects | datasets: The number of objects explicitly added by the user that are present in the feature is specified here.

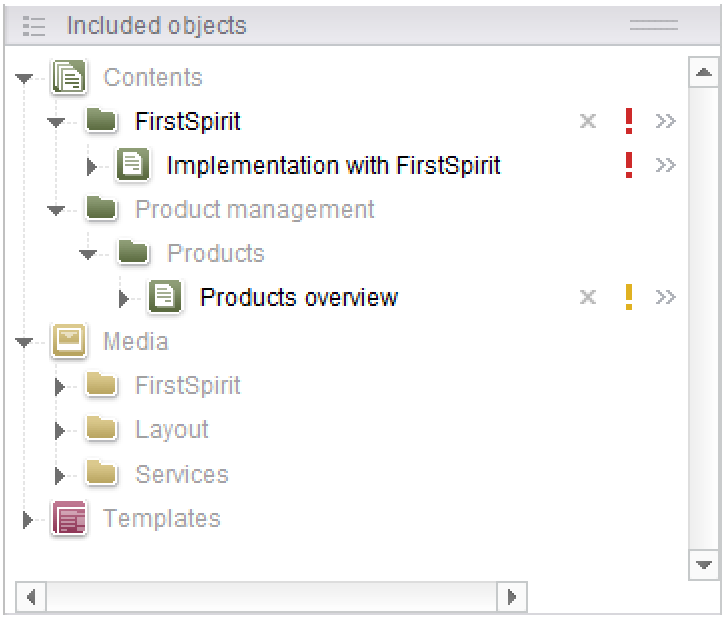
Missing references: How high the number of missing references in the entire feature combination is specified here. The number of absolutely necessary objects is shown here in red; the number of optional objects is in yellow.

> *Missing references always refer to the entire feature – A detailed view of the missing references can be called up using the flyout menu (see section 4.4, page 30).*

4.3.2 "Included objects" area

All objects included for the feature are listed in this area. There is a distinction between explicitly and implicitly added objects. If an object is added explicitly, then all higher level parent objects are implicitly added automatically to the feature as well. Child objects, on the other hand, are added explicitly even though they were not added separately by the user.

- Objects added by the user are shown in normal text; this indicates that they can be removed from the list again by using the × icon.
- Explicit objects that are the child elements of an explicitly included object are also shown in normal text but cannot be removed from the list.
- Implicitly added objects that are at a higher level than an explicitly included object are shown in text with less contrast and cannot be removed from the list either.



Included objects

- Contents
  - FirstSpirit × ! »
    - Implementation with FirstSpirit ! »
  - Product management
    - Products
      - Products overview × ! »
- Media
  - FirstSpirit
  - Layout
  - Services
- Templates

Figure 4-8: Feature – Included objects

Clicking on an object in the list opens a tab with that object's forms in the SiteArchitect edit area for viewing. The object cannot be edited at this point; this is indicated by a clock symbol on the object icon.

Show relation graph; clicking this icon opens a tab in the AppCenter area with a graphical representation of the hierarchical structure and the references (dependencies) of the selected object (see section 4.5, page 31).

× Delete; this icon is only displayed if the associated object was explicitly included by the user. Clicking this icon removes the selected, explicitly added object from the list along with all automatically included child objects, after the user affirms a confirmation prompt. Higher level objects that are not used by other explicitly included objects are also removed.

! Missing optional references; the yellow exclamation mark indicates that the respective object or a child object has missing optional references. The objects are listed in detail in the "Optional missing references" area.

! Missing required references; the red exclamation mark indicates that the respective object or a child object has missing hard references. The objects are listed in detail in the "Required missing references" area.

» Object details; clicking this icon opens a flyout menu with object-specific information (see section 4.4, page 30). Clicking the icon again closes the flyout menu.

4.3.3 "Required missing references" area

All of the objects that are required to install the feature combination in a target project successfully are shown in this area. If all of the required references are found, then this area remains empty.

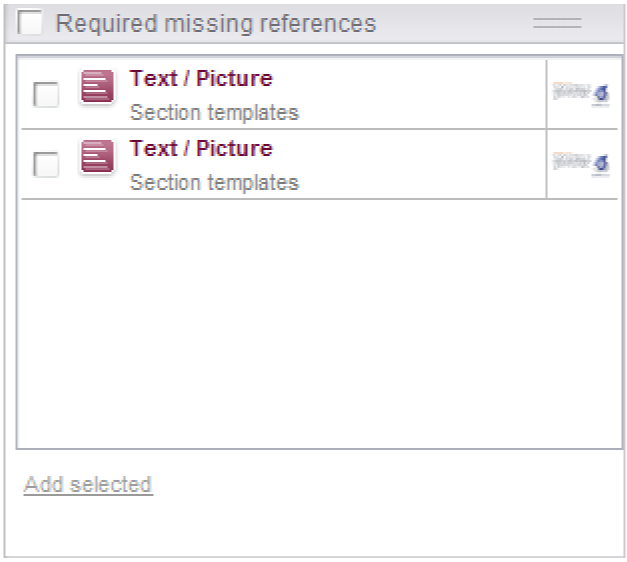


Figure 4-9: Feature – Required missing references

Required objects are displayed in list form, each with a checkbox for selecting each individual object. Clicking on an object in the list opens a tab with that object's forms in the SiteArchitect edit area for viewing. The object cannot be edited at this point; this is indicated by a clock symbol on the object icon.

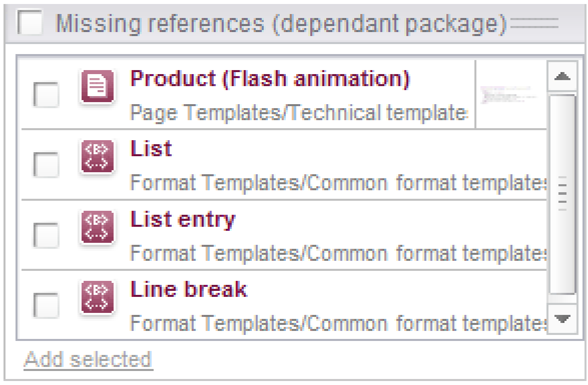
Required missing references: If this checkbox on the top end of the area is selected, then the checkbox for selecting an object is selected for all of the objects in the list.

Clicking on the **Add selected** button integrates all of the objects selected in this area into the feature combination.

Required dependent objects have to be added. But this only applies to an empty project. If missing references are already found in the target project, then these objects do not absolutely have to be added; the feature can be applied regardless.

4.3.4 "Optional missing references" area

All of the objects that are not absolutely required for successful installation in a target project but are desired are shown in this area. If all of the optional dependencies are fulfilled then this area remains empty.



□ Missing references (dependant package)

□ **Product (Flash animation)**
Page Templates/Technical template

□ **List**
Format Templates/Common format templates

□ **List entry**
Format Templates/Common format templates

□ **Line break**
Format Templates/Common format templates

Figure 4-10: Feature – Optional missing references

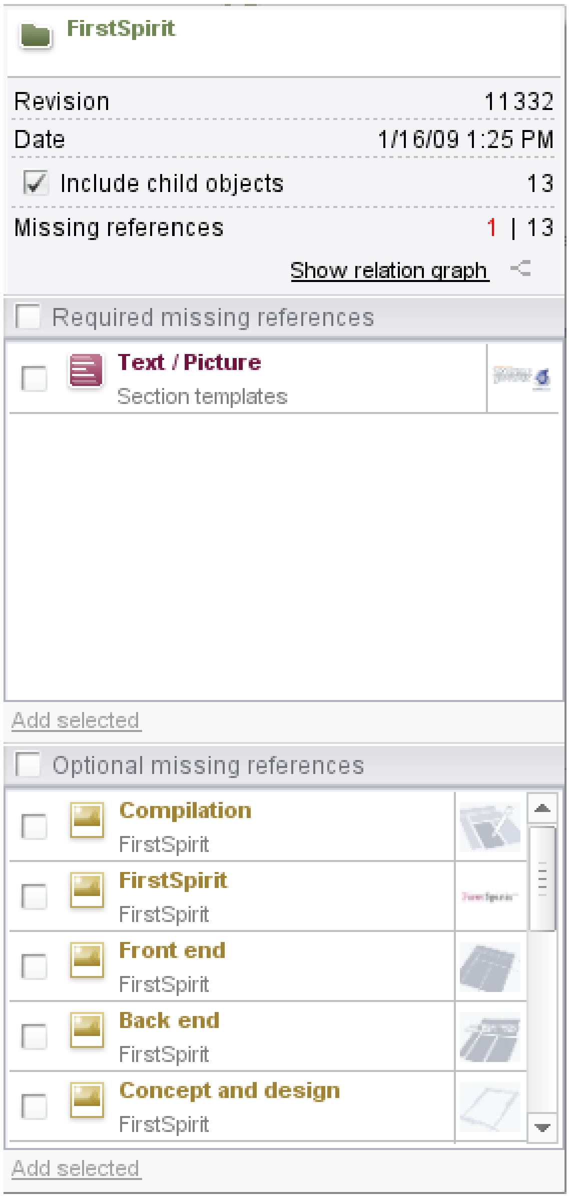
Optional objects are displayed in list form with a checkbox for selecting each individual object. Clicking on an object in the list opens a tab with that object's forms in the SiteArchitect edit area for viewing. The object cannot be edited at this point; this is indicated by a clock symbol on the object icon.

Optional missing references: If this checkbox on the top end of the area is selected, then the checkbox for selecting an object is selected for all of the objects in the list.

Clicking on the **Add selected** button integrates all of the objects selected in this area into the feature combination.

4.4 Flyout menu

The flyout menu contains object-specific information displayed in the same way as the information for the feature combination.



FirstSpirit

Revision 11332

Date 1/16/09 1:25 PM

☑ Include child objects 13

Missing references 1 | 13

Show relation graph

□ Required missing references

□ **Text / Picture**
Section templates

Add selected

□ Optional missing references

□ **Compilation**
FirstSpirit

□ **FirstSpirit**
FirstSpirit

□ **Front end**
FirstSpirit

□ **Back end**
FirstSpirit

□ **Concept and design**
FirstSpirit

Add selected

Figure 4-11: Flyout menu

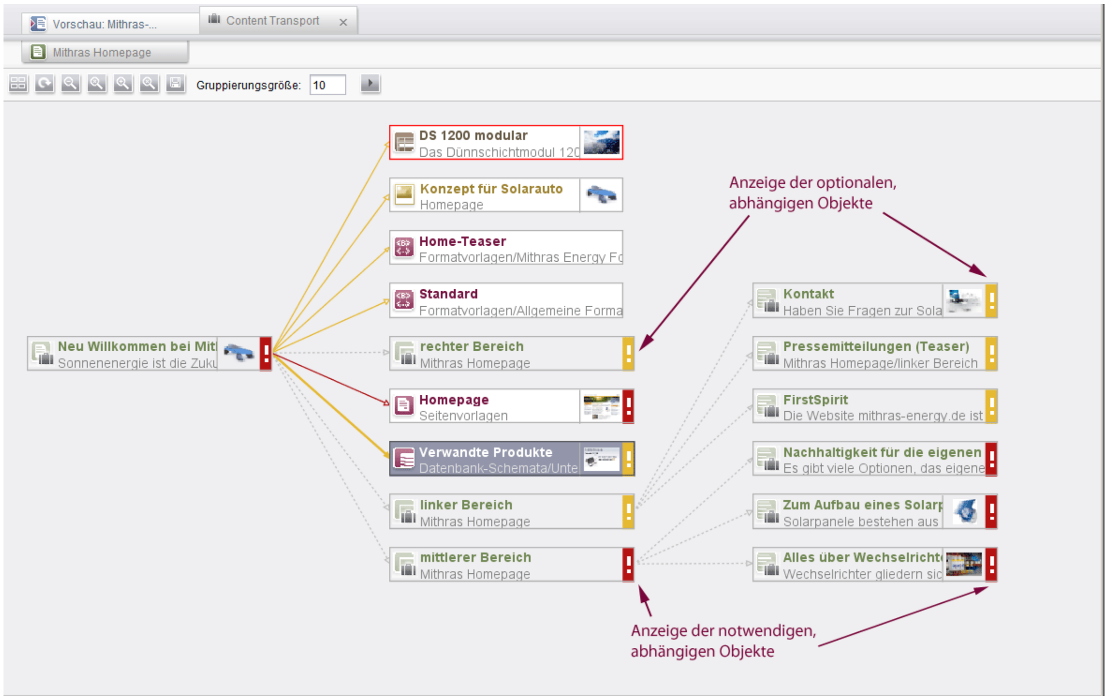
The flyout menu always contains at least a tabular listing of object-specific data the same way as the feature overview.

- Icon and language-dependent display name for the displayed object
- Revision of the object (object-specific state included in the feature) as well as the date and time of the revision.
- "Include child objects" checkbox – If this checkbox is selected, then the missing references for the displayed object are displayed along with all of the object's child objects. If the checkbox is not selected, then only the missing references of the displayed object are displayed.
- Number of child objects
- Number of missing references (required and optional)
- A tab with a graphical representation of an object's references can be displayed using the **Show relation graph** button in the AppCenter area (see section 4.5, page 31).

The Required and Optional missing references areas are the same as the areas in the feature combination with the same name (see sections 4.3.3 and 4.3.4, starting from page 28).

4.5 Graphical representation of dependencies (references)

The graphical representation is used to provide a flexible view of the hierarchical structure and dependencies of a feature's embedded objects. Furthermore, objects can be added/removed in the graphical representation.



Anzeige der optionalen, abhängigen Objekte

Anzeige der notwendigen, abhängigen Objekte

Figure 4-12: Show dependencies

The relation graph is integrated as a reusable tab in the SiteArchitect AppCenter area.

4.5.1 Icon bar

Layout; clicking this icon automatically arranges the displayed objects in a uniform layout. In the process, layout changes made by the user are discarded without prompting.

Update; clicking this icon updates the information displays in the relation graph. Changes to the hierarchical structure of the objects and the feature's new or removed objects are taken into account in the process.

Zoom out/1:1/in; clicking on these icons changes the view of the relation graph by increasing it, shrinking it or changing it back to its original size.

Fit to Screen; clicking this icon adjusts the zoom level so that the entire relation graph is visible at the current tab size.

Save as image; clicking this icon opens a dialog box for selecting the name and save location for creating an image file in PNG format. The created picture file contains the entire relation graph at the selected zoom level.

Grouping size: How many objects are to be displayed at the same time using the "Show linked objects" context menu function or by double-clicking can be specified in this field.

▶/⧓ Follow mode; clicking this icon switches to follow mode. Two states can be configured:

- ▶ Clicking on an object in the feature's tree structure also displays the object in the workspace and selects it in relation graphs.
- ⧓ Clicking on an object in relation graphs also then displays the object in the workspace.

4.5.2 Display of relation graph

The relation graph displays objects from the stores and their connections to each other. These connections can be relationships between parent and child objects as well as references between different objects.

The object that the relation graph was retrieved for is used as the root node for the view and appears at the far left in the relation graph. When expanding outgoing connections for an object, the target nodes are arranged to the right of the object. Each link between two objects is represented by an arrow that points to the child or referenced objects.

Each object is shown as a rectangle and contains the following information:

- Object icon; the same icon that is also displayed in the tree structure of individual stores.
- Suitcase symbol (🧳); specifies whether the object is currently included in the feature. It is placed on the object icon for identification
- Display name; the language-dependent display name from the relevant stores is displayed. Alternatively, the developer can set another text as the display name.
- Preview symbol; if present, a preview of the object or included images is shown as a symbol.
- Exclamation mark (!/!); displayed if missing references exist for the object or a child object. The color rules match the tree view in the "Included objects" area of the feature combination (see section 4.3.2, page 27).

Edge lines are drawn between objects that are related to each other. These lines are shown differently depending on the missing or found status:

- Solid line; used for explicitly added objects (references).
- Dashed line; used for child elements that are child elements of an explicitly included object.
- Gray; used between objects if their connection is intact in the current feature combination (i.e. always child objects, referenced objects if the reference target is included in the feature).
- Red; used between objects if a missing required reference is present.
- Yellow; used between objects if a missing optional reference is present

4.5.3 Context menu on objects

A context menu can be called up on each object. The functions of the context menu are active or grayed out depending on the state of the object.

Add to feature: The selected object and all of its child elements are included in the feature.

Remove from feature: Explicitly included objects and their automatically included child objects are removed from the feature.

> *Objects that have been added to fulfill the dependencies of a removed object are not implicitly removed.*

Show element in workspace: Calling this function opens a tab with this object's forms in the SiteArchitect edit area for viewing. The object cannot be edited at this point; this is indicated by a clock symbol on the object icon.

Add all required edges: Calling this function integrates all of the required missing references of the selected object into the feature combination.

Add all optional edges: Calling this function integrates all of the optional missing references of the selected object into the feature combination.

Expand related objects (double-click): If there are connections to other objects that are not yet displayed, calling this function triggers the display of these objects. The maximum number of objects that are displayed there is specified on the icon bar under "Grouping size". Another group of objects can be displayed by calling the function (or double-clicking) again.

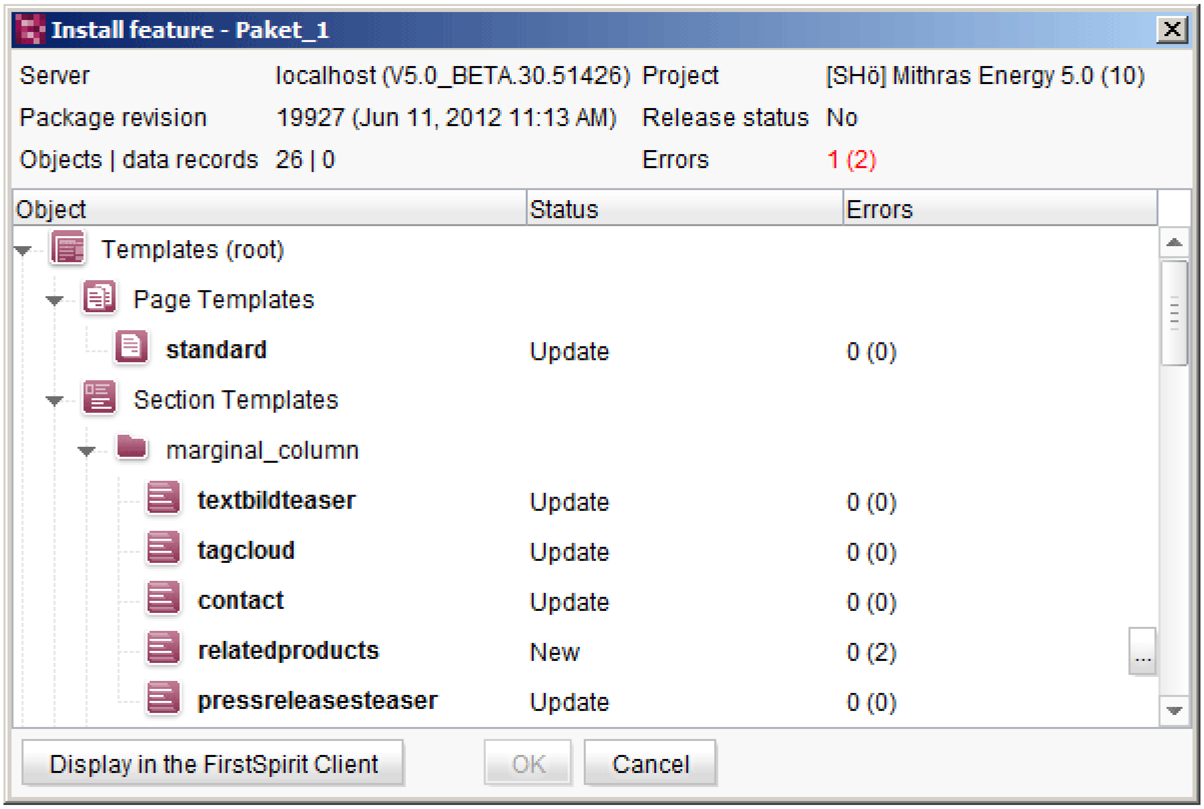
If not all linked objects were able to be displayed, this is indicated by an extra object with the label "Show next elements (X total)". Another group of objects can also be displayed by double-clicking this extra object.

Collapse related objects: All linked objects currently being displayed can be hidden by calling this function. All of a linked object's subordinate objects are also hidden in the process. Thus, if an object is hidden and then immediately displayed again, then its subordinate objects continue to be hidden

4.6 Updating a feature in a target project

A Content Transport feature can be installed in a target project in two ways. Either using the "Install feature" entry in the empty feature area or using the icon on the icon bar of the Content Transport area.

After selecting a feature zip file from the file system, a dialog box opens with an overview of the combination for the selected feature and its included objects.



Install feature - Paket_1

Server	localhost (V5.0_BETA.30.51426)	Project	[SHö] Mithras Energy 5.0 (10)
Package revision	19927 (Jun 11, 2012 11:13 AM)	Release status	No
Objects | data records	26 | 0	Errors	1 (2)

Object	Status	Errors
Templates (root)		
Page Templates		
standard	Update	0 (0)
Section Templates		
marginal_column		
textbildteaser	Update	0 (0)
tagcloud	Update	0 (0)
contact	Update	0 (0)
relatedproducts	New	0 (2)
pressreleasesteaser	Update	0 (0)

Figure 4-13: Target project - Installing updates

Feature overview

Server: Specifies the name of the server where the feature was created.

Project: Specifies the name of the source project on the server.

Package revision: The maximum revision for all of the objects included in the feature and the date and time of the maximum revision are displayed here.

Release status: Specifies whether the objects included in the feature are installed in a released version.

Objects | data records: Specifies the number of objects that are included in the feature.

Errors: Specifies the number of errors that are expected upon installing the feature in the target project. The number in front of the parenthesis specifies missing references in the feature; the errors are marked in red and the feature cannot be installed. The number in parentheses specifies the optional missing references; the errors are marked in yellow and it is possible to install the feature despite them.

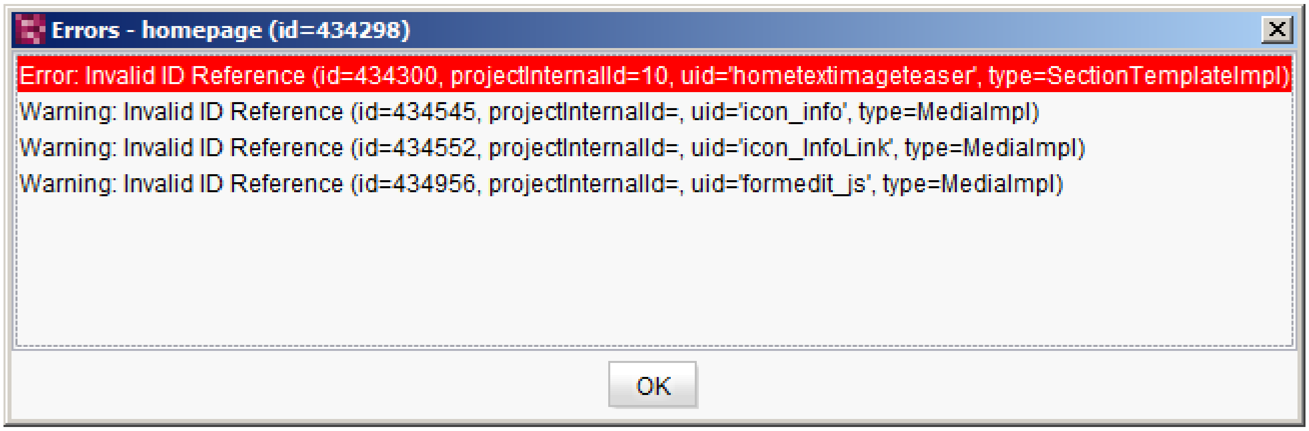
List of included objects

Object: The objects included in the feature are listed in their hierarchical structure sorted by stores. As in the SiteArchitect tree structure, the individual objects can be expanded or collapsed here.

Status: Each object can assume the "New" or "Update" state. For "Update", the object is already present in the target project and may have changed since the last feature installation. The object can be checked using the **Display in FirstSpirit Client** button.

Errors: The type and number of errors can be read for each object here. The number in front of the parentheses specifies the required missing references and the number in parentheses specifies the number of optional missing references for this object.

After each object with errors, a dialog with a detailed list of the errors can be called up via the ... icon.



Error: Invalid ID Reference (id=434300, projectInternalId=10, uid='hometextimageteaser', type=SectionTemplateImpl)
Warning: Invalid ID Reference (id=434545, projectInternalId=, uid='icon_info', type=MediaImpl)
Warning: Invalid ID Reference (id=434552, projectInternalId=, uid='icon_InfoLink', type=MediaImpl)
Warning: Invalid ID Reference (id=434956, projectInternalId=, uid='formedit_js', type=MediaImpl)

Figure 4-14: List of object-related errors

Errors occurring due to required missing references are marked in red in this dialog.

General warnings

General warning that do not refer to a specific object are displayed in the lower area of the dialog box. These warnings include mismatches in the project settings, such as for project languages or image resolutions.

The **Display in FirstSpirit Client** button is only active if an object that has the "Update" state is selected and could be overwritten by installing the feature. Clicking this button displays the selected object in the FirstSpirit Client edit area.

Clicking the **OK** button carries out the update. The button is only active if no required missing references exist for installing the feature. The user is prompted to select a database layer if one is not yet present in the target project; then the update is carried out.

4.7 Limitations and Notes

- Objects included via Content Transport do not have a feature relation or namespace extension and also cannot be protected from being overwritten. **However:** A distinction is also made between explicitly and implicitly added objects when installing a feature in a target project.
    - **Explicitly** added objects are always created in the target project. If an object already exists in a target project, then the changes are reset again as soon as another update is carried out.
    - **Implicitly** added objects are always created in a target project if they were not present in the project until now. If an object already exists in a target project, then the changes to the target project remain intact when an update is carried out next.

- **Preservation of a node structure** which is to be transported is dependent upon the node which is explicitly added to a feature. All child nodes of that parent node are considered to be explicitly added to the feature as well.

  When such a parent node is updated in a target project and the parent node's child element structure in the target project differs from that included in the feature, child elements which are present in the target project but not present in the feature will be handled differently depending on the following criteria:

  a) Child elements will be deleted if they are either objects which do not possess an UID (e.g. folders in the Templates store, sections) or are objects which can never exist directly in a folder (i.e., query elements and table templates).

  b) Other child elements will be moved into an automatically created folder "Lost & Found".

  c) If a child element, which would normally be deleted according to the above rules, is referenced by other elements (e.g. a section which is referenced by a section reference), this element will be preserved – i.e. neither deleted nor moved – in the target project although it is not explicitly contained in the feature.

  If such an object cannot be deleted because of inbound references, feature installation will abort with an error message ("Unable to install feature file: Error installing feature. Unable to delete element XYZ. The element is still referenced by the following elements: […]"). In order to install the feature, references from other elements to elements which should be deleted due to feature installation must be removed manually, or the referencing elements must be deleted manually.

- If **missing references** have already been found in a target project, then the feature combination can be applied regardless.
- If objects (such as templates) are in **edit mode** in the target project, the import process for these objects cannot be carried out. Therefore, it should be ensured that no objects are in edit mode!
- Different numbers of **languages** are possible between a source and target project using Content Transport. If a language channel does not available in the source project, it is not overwritten in the target project.

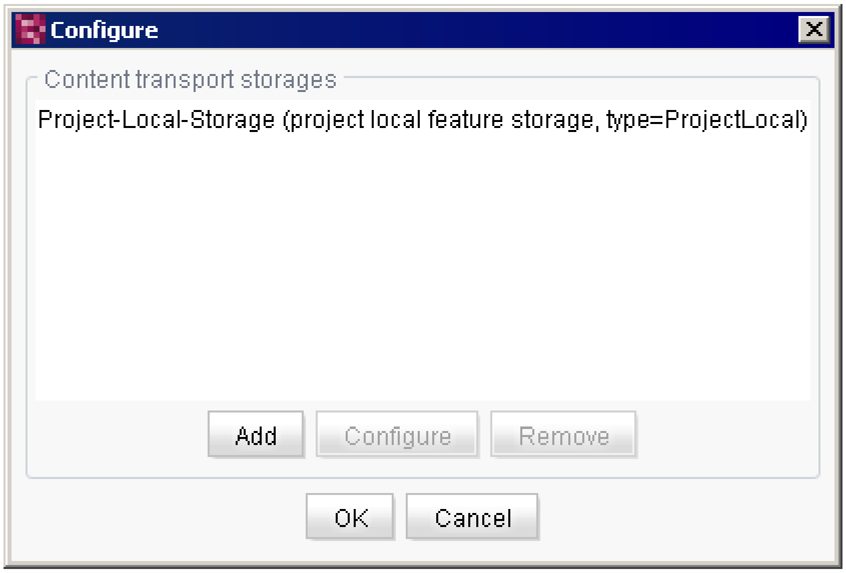
- Settings made in a source project when assigning **permissions** cannot be applied to a target project. Permissions have to be assigned in the target project manually. The permissions in the target project only have to be assigned once; all settings remain intact during future update processes.
- **Workflow states** are **not** transported.
- **Metadata** is transported but, depending on the configuration in the target project, can be incompatible with the selected metadata template.
- Settings made in the **ServerManager** cannot be applied to the target project by using Content Transport.
- In the case of the adoption behavior of **start node** information (start folders/start pages), a distinction is made between explicitly and implicitly added objects.
    - If in the case of the corresponding information carrier (parent node) it is an **explicitly** added object, the start node information will be set as it is in the source project.
    - If in the case of the corresponding information carrier (parent node) it is an **implicitly** added object, the start node information will not be transported and the corresponding start nodes in the target project will remain as they are.
    - An exception to this behavior would be in the case of objects that are directly underneath the root node. During the initial installation of a feature in the target project, the start node information is set automatically if:
      - no start node exists yet, or
      - it is a page reference (in the case of the transported node).

4.8 Configuring storage locations

Different locations can be configured for saving created ZIP files. In addition to the **local file system** and **network drives**, **external storage locations** can be used, including Internet-based locations such as Dropbox. To be able to use external storage media, the relevant modules need to be created. This type of module will then handle Internet service authentication, for instance.

Storage locations are configured in the ServerManager project properties under "Project components"/"FirstSpirit Content Transport Storage App". Each project must have its own storage locations configured. The "FirstSpirit Content Transport Storage App" project component is installed automatically during a new install or a server update to version 5.1 if the `license.PACKAGEPOOL` license key is present in the `fs-license.conf` license file with the value `1`. This system project component cannot be removed.

After clicking "Configure" or double-clicking on the "FirstSpirit Content Transport Storage App" project component, the following window appears:



Project-Local-Storage (project local feature storage, type=ProjectLocal)

Figure 4-15: Configuring storage locations for Content Transport content

A default storage location called "Project-Local-Storage" is offered on the local FirstSpirit server. Other directories can be configured as an alternative. As with external storage locations, the following "Add" and "Configure" functions are used for these:

Add: A dialog appears in which the user can select from the available storage locations. The "File system feature storage" entry can be used to specify a directory that can be accessed from the FirstSpirit server. Access to external storage locations must be modified in the FirstSpirit module. A reference name and a display name can be specified for the storage location in the following dialog:

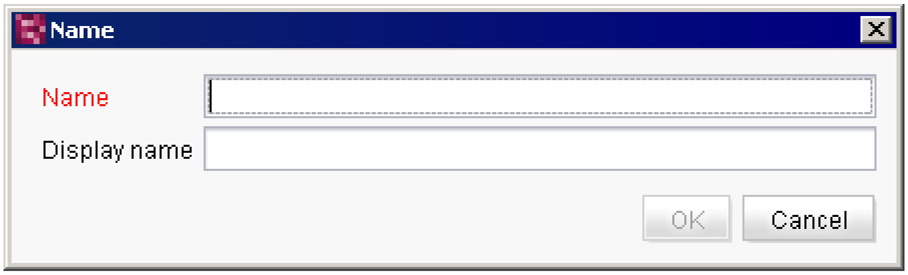
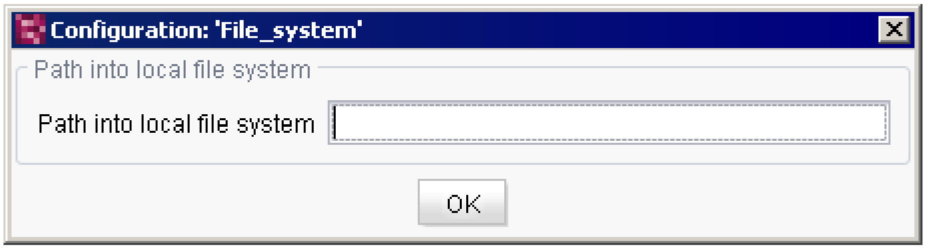


Figure 4-16: Specifying a name for the storage location

Clicking "OK" opens a dialog where the storage can be configured.

Configure: A dialog appears in which access to the storage location can be configured. If it is "File system feature storage", the path to the desired directory can be entered in the following dialog:



Configuration: 'File_system'

Path into local file system

Path into local file system

Figure 4-17: Specifying a path

Access to external storage locations must be provided by FirstSpirit modules. The external storage configuration dialog may therefore vary from implementation to implementation. Advanced configuration of the "Project-Local-Storage" default storage location is not possible.

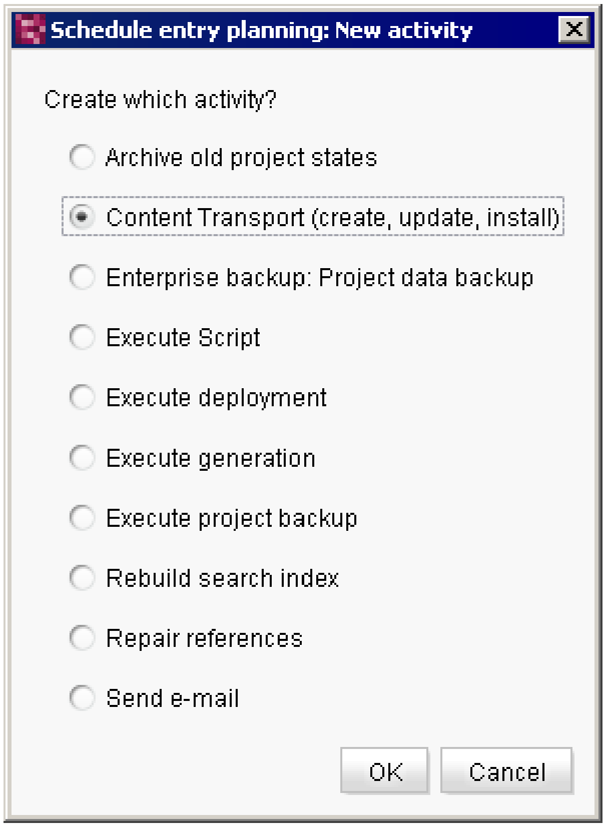
Remove: Removes access by the Content Transport function to the selected storage location.

The storage locations configured using this project component are then offered as selection options when generating and installing feature ZIP files in SiteArchitect, for example (see Figure 4-2).

4.9 Automatic creation, updating and installation of features

FirstSpirit schedule management can be used to set an automatic update of Content Transport content at predefined times. To do this, the feature ZIP files must be stored in an **external storage location** (refer to section 4.8, page 40 for more information).

For this purpose, a schedule with the "Content Transport (create, update, install)" action must be created in the FirstSpirit ServerManager project properties:



Schedule entry planning: New activity

Create which activity?

○ Archive old project states

● Content Transport (create, update, install)

○ Enterprise backup: Project data backup

○ Execute Script

○ Execute deployment

○ Execute generation

○ Execute project backup

○ Rebuild search index

○ Repair references

○ Send e-mail

Figure 4-18: New activity

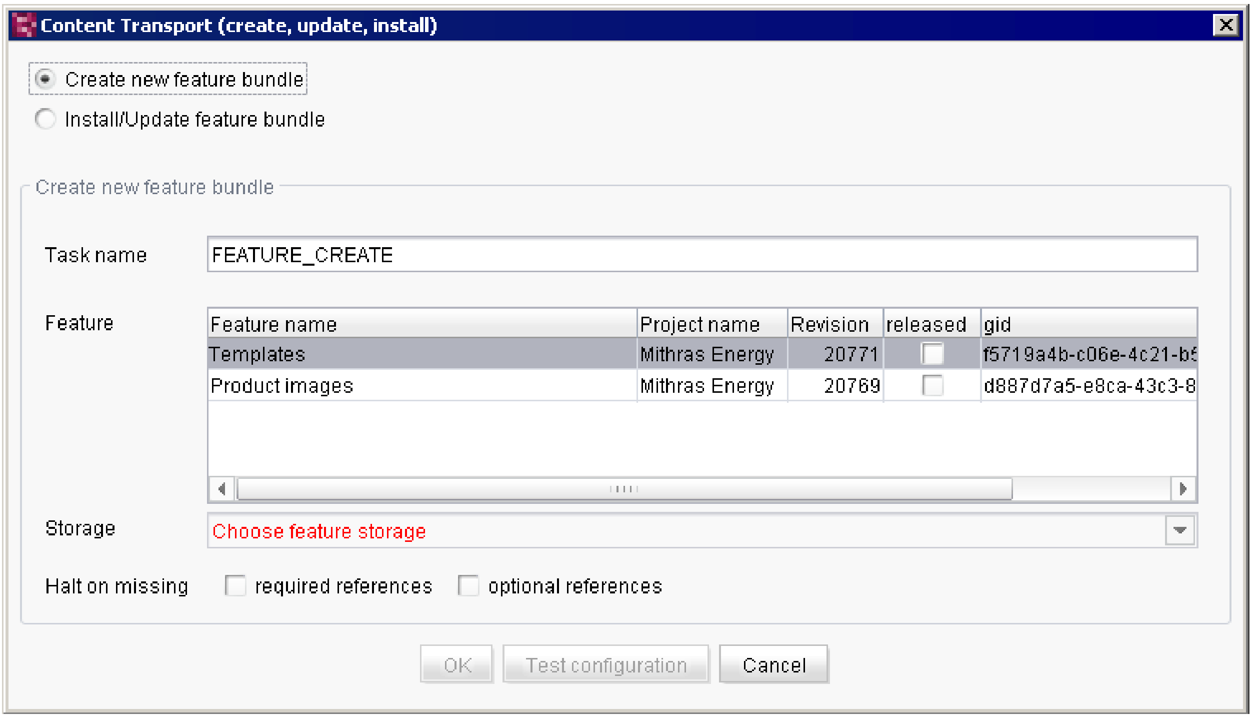
When using this activity, two options are available to choose from in the following dialog:

- **Create new feature bundle** (refer to section 4.9.1, page 44)
  This option allows for regular generation of a feature ZIP file with the current state of the feature's content, e.g. the current development state of templates, at predefined times. The feature ZIP file with the current state is then stored at the selected storage location ("push"). From there the content is then made available for import to other projects (on the same or other FirstSpirit servers).

- **Install/Update feature bundle** (refer to section 4.9.2, page 46)
  This option allows the user to regularly import features to the current project at predefined times ("pull"). If only the local file system is available as a storage location, the features can be exchanged between projects on a FirstSpirit server using the relevant schedule; if external storage locations are available (refer also to section 4.8, page 40 for more information), the exchange can take place across server boundaries.

In schedules where activities are already present, this option is no longer displayed.

4.9.1 Exporting existing feature bundles by schedule ("Create new feature bundle")



● Create new feature bundle
○ Install/Update feature bundle

Create new feature bundle

Task name: FEATURE_CREATE

Feature

Feature name	Project name	Revision	released	gid
Templates	Mithras Energy	20771	□	f5719a4b-c06e-4c21-b5
Product images	Mithras Energy	20769	□	d887d7a5-e8ca-43c3-8

Storage: Choose feature storage

Halt on missing □ required references □ optional references

Figure 4-19: Content Transport activity – Create new feature bundle

Name: A name for the activity must be specified here. This will be used on the "Actions" tab of the Content Transport schedule and distinguishes it from the schedule's other actions.

Feature: The features available on the current FirstSpirit server are listed here. To ensure that a desired feature appears here, it first needs to be saved in the FirstSpirit SiteArchitect on the server (refer to the "Save feature" function in chapter 4 for more information).

Storage: Here is where the user selects the desired location where the feature is to be stored.
A default storage location called "Project-Local-Storage" is offered on the local FirstSpirit server.
Additional storage locations can be configured in the project properties (see section 4.8, page 40).

At the time a feature is created, a check is made to determine if there are any required or optional references missing (see section 4.3.3, page 28 or section 4.3.4, page 29 for more information). After creating a feature that is selected in this activity, the content is usually edited further. In particular deleting or moving nodes can result in new **inconsistencies**. When a current feature version is created automatically, the references (dependencies) are therefore rechecked. The following two options can be used to handle possible inconsistencies:

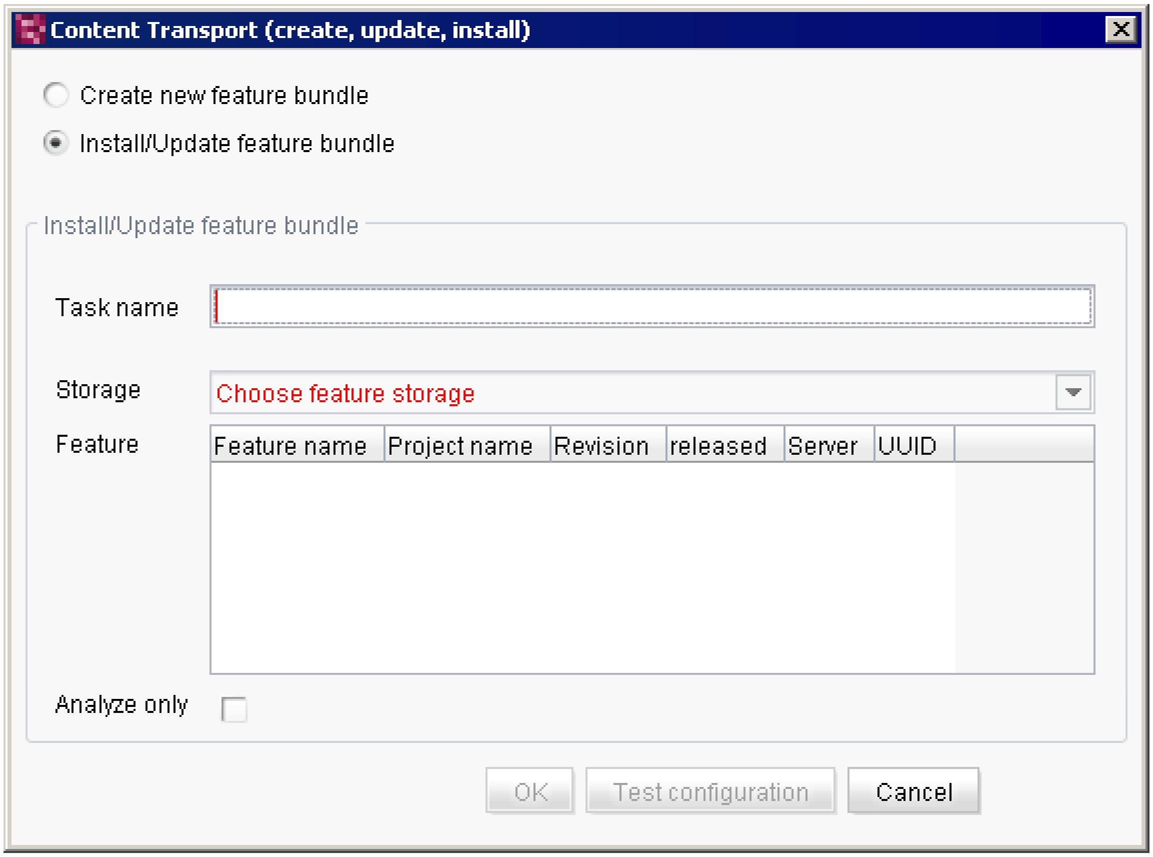
Halt on missing required references: If this option is active, generation of the selected feature is canceled if the feature is missing required references.

Halt on missing optional references: If this option is active, generation of the selected feature is canceled if the feature is missing optional references.

The specified configuration can be tested by clicking on the "Test configuration" button. Clicking "OK" saves the applied settings.

Execution time(s) and intervals for generating the feature are configured in the schedule on the "Properties" tab (for more information on schedules, refer to "Schedule entry planning" Chapter in the *FirstSpirit documentation for administrators*).

4.9.2 Importing feature bundles by schedule ("Install/Update feature bundle")



Content Transport (create, update, install)

○ Create new feature bundle

● Install/Update feature bundle

Install/Update feature bundle

Task name

Storage: Choose feature storage

Feature

Feature name	Project name	Revision	released	Server	UUID

Analyze only □

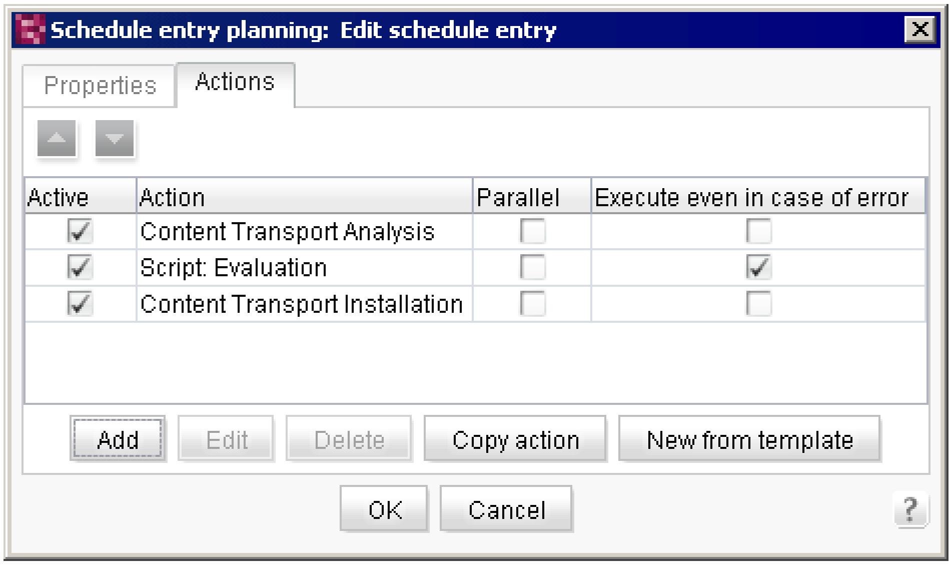
Figure 4-20: Content Transport activity – Install/Update feature bundle

Name: A name for the activity must be specified here. This will be used on the "Actions" tab of the Content Transport schedule and distinguishes it from the schedule's other actions.

Storage: The location where the feature was stored and will be imported is specified here.
A default storage location is offered on the local FirstSpirit server: "Project-Local-Storage".
Additional storage locations can be configured in the project properties (see Chapter 4.8, page 40).

Feature: If a storage location from the "Storage" drop-down list is selected, the features available at the selected storage location will be listed here. To ensure that a desired feature appears here, it must be stored at the selected storage location first (via the "Create new feature bundle" schedule (see section 4.9.1, 44) or via the "Save feature" SiteArchitect function).

Analyze only: When **manually** installing a feature in the target project, a check is made to determine if there are any required or optional references missing (refer to section 4.6, page 35 for more information). If there are required references that are missing, the feature will not be installed. If the installation of a feature is **automatic** due to the schedule described here, this option can be used to handle it. If this checkbox is active, it will only check for missing references. A script that follows can then evaluate the results of the analysis, for instance, and then halt or continue an installation depending on the results.



Active	Action	Parallel	Execute even in case of error
☑	Content Transport Analysis	□	□
☑	Script: Evaluation	□	☑
☑	Content Transport Installation	□	□

Figure 4-21: Example of a Content Transport schedule

If there are missing required references, the feature will **not** be installed.

The specified configuration can be tested by clicking on the "Test configuration" button. Clicking "OK" saves the applied settings.

Execution time(s) and intervals for generating the feature are configured in the schedule on the "Properties" tab (for more information on schedules, refer to "Schedule entry planning" Chapter in the *FirstSpirit documentation for administrators*).

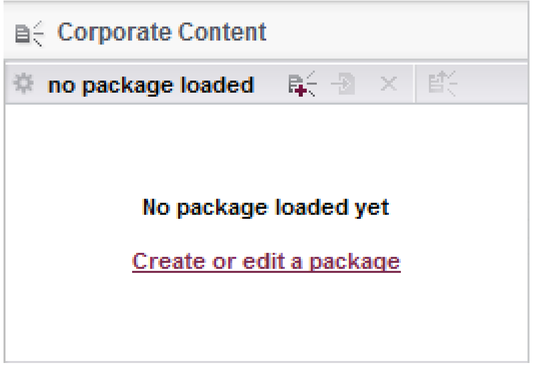
5 CorporateContent (package store)

The "CorporateContent" store area is used for creating new packages and for editing existing packages. It can be opened using the icon from the vertical icon bar in SiteArchitect.

In parallel to the package store known from previous versions, some functions can also be called up using the "CorporateContent" menu item on the SiteArchitect menu bar.

Creating and combining a package in a source project and creating and editing a subscription in a target project are described in the following sections:

▪ Creating or loading a package	Section 5.1, page 49
▪ Package combination	Section 5.3, page 66
▪ Flyout menu	Section 5.4, page 70
▪ Graphical representation of dependencies	Section 5.5, page 71
▪ Functions using the "CorporateContent" menu item	Section 5.6, page 75
▪ CorporateContent context menu in stores	Section 5.7, page 106
▪ Transferring existing projects into package master projects	Section 5.8, page 113
▪ CorporateContent for developers	Section 5.9, page 124
▪ Shared database access	Section 5.10, page 135



No package loaded yet

Create or edit a package

Figure 5-1: CorporateContent store area

There are entries for creating and editing ContentTransport features on the ContentTransport area's icon bar.

The name of the opened package is displayed after this icon. "No package loaded" is displayed if no package is loaded.

Create or load package; clicking this icon opens a dialog for creating a new package or loading a package that already exists. (For creating a package see section 5.1.1, page 49)

Create version; clicking this icon opens a dialog with an overview of all versions that have been created from the package. (For creating a package version see section 5.1.2, page 60)

Close package; clicking this icon closes the opened package – after affirming a confirmation prompt.

Publish; clicking this icon opens a dialog with all of the package versions available for publication

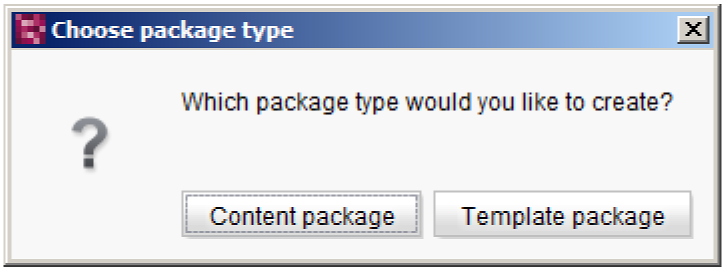
5.1 Creating or loading a package

Clicking the icon or the "Create or edit package" in the empty package area displays a prompt for whether a new package is to be created or an existing package is to be loaded.

5.1.1 Creating a new package

The process of creating a new package follows multiple steps explained below.

5.1.1.1 Selecting a package type



Choose package type

Which package type would you like to create?

Figure 5-2: Dialog for selecting a package type

The package type for the new package can be assigned here. The package type is also displayed in the "Edit package properties" dialog box but cannot be modified any further there.

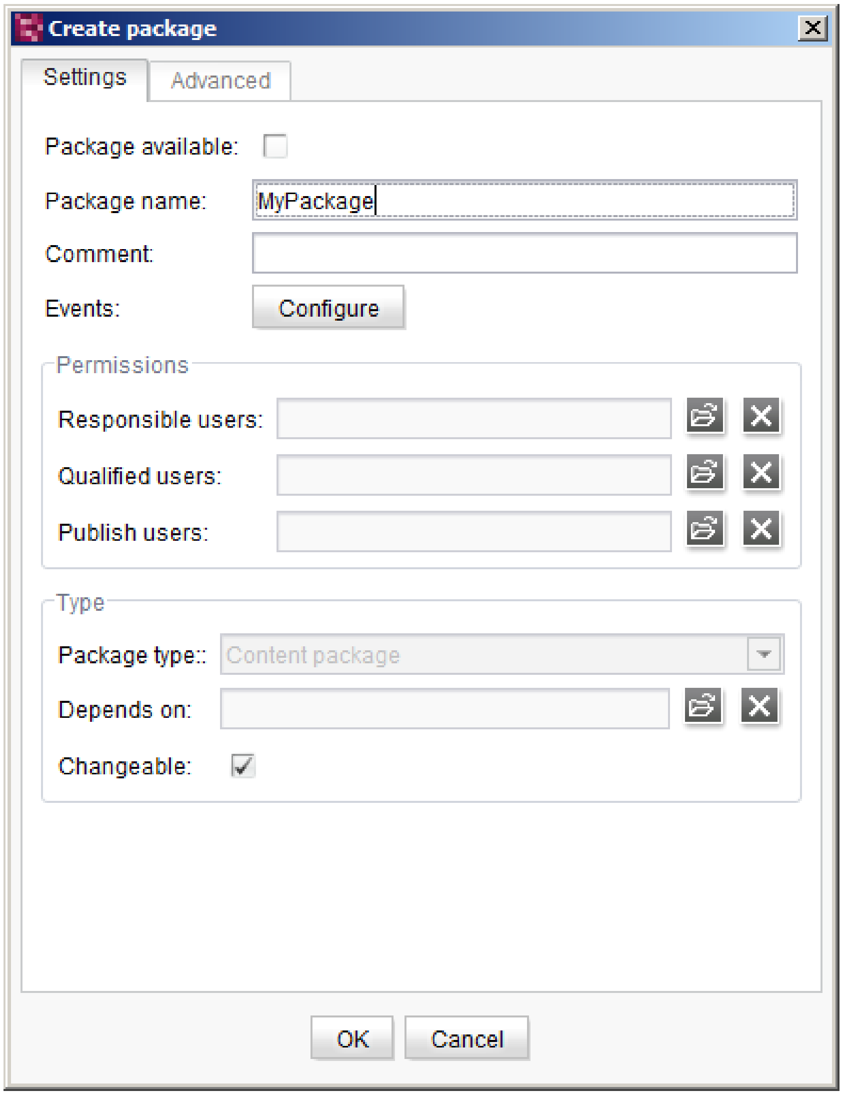
Content package: Clicking this button selects a content package as the package type. A content package is only allowed to contain objects from the page store, media store and site store and these are also the only stores displayed when selecting package contents.

Template package: Clicking this button selects a template package as the package type. A template package is allowed to contain objects from the template store, content store and media store and these are also the only stores displayed for selecting package contents. The media integrated here should be limited to *media referenced directly in the templates*. Other media objects should be integrated into a content package.

Depending on the selected type, the "Create package" dialog then opens. All of the initial settings for the package are set by the administrator of the master project there.

> ! *If objects from a database schema are to be integrated into the package, the database configuration has to be adjusted in the target project's project properties (see section 5.10, page 135). Otherwise a corresponding error message will be output when importing the package into the target project later on (see section 5.6.6, page 93).*

5.1.1.2 Creating a package – Settings tab



Package available: ☐

Package name: MyPackage

Comment:

Events: Configure

Permissions

Responsible users:

Qualified users:

Publish users:

Type

Package type:: Content package

Depends on:

Changeable: ☑

Figure 5-3: Create/edit package - Settings

Package available: The new package is made available to all target projects if this checkbox is **checked**. If the checkbox is **unchecked**, the package is not made available and cannot be selected for subscribing in target projects.

Package name: Unique name for the package; specified initially during creation and cannot be changed afterwards.

Comment: Optional comment for the package.

Events: Clicking the **Configure** button opens the "Configure events" dialog box (see section 5.1.1.2.1, page 53).

Permissions

Editing permissions for the package are configured in this area. A new package is initially created by the master project's administrator, who also assigns permissions for the package. Package properties can also be edited by all *authorized* persons once the permissions have been defined here.

Responsible: These are the persons responsible for the package in the master project. Responsible persons are informed via e-mail if a new package version is available or a new package version has been imported. Using the 📂 icon, another user can be added to the list of responsible persons and they can be removed from the list using the ☒ icon.

Authorized: These persons may edit the package properties (permissions, dependencies, etc.) and make content changes in the package such as adding events or deleting start nodes. Using the 📂 icon, another user can be added to the list of responsible persons and they can be removed from the list using the ☒ icon.

Publisher: These persons may publish packages and thus make them available for import into target projects. Using the 📂 icon, another user can be added to the list of responsible persons and they can be removed from the list using the ☒ icon.

Type

In this area, the selected package type can be read or a package dependency can be defined.

Package type: Specifies the package type that was selected in the "Select type" dialog box when creating the package (content or template package).

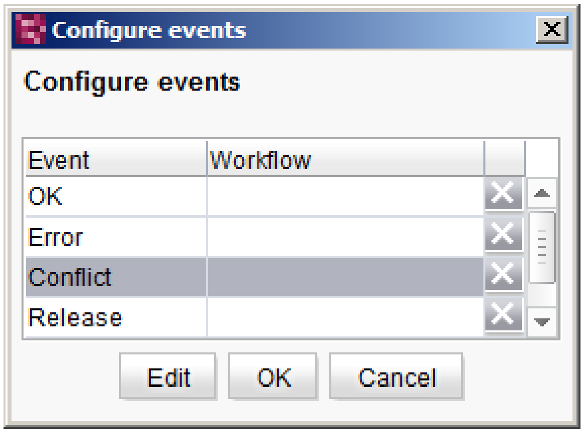
Dependent on: Only active for content packages. Manual dependencies on template packages are set here. If the content package is subscribed to, then the customer also has to subscribe to the specified, associated template package here. Template packages do not have any dependencies. Thus the field is disabled for the template package type. A content package dependency on an existing template package can be defined using the 📂 icon. The desired template package can either be selected from a list of all packages subscribed to in the same project (source project) or from another project. Clicking the ☒ icon removes the selected dependency again.

Modifiable: Write permission to the imported objects is granted for target projects if this checkbox is **checked**. If the checkbox is **unchecked**, the imported objects can be seen and used in target projects but they cannot be modified.

Since only one template package can be selected here, it is absolutely essential that all templates (page, section, link templates, etc.) used as a basis for pages and sections from the content package are included in this template package. Also refer to section 5.8.1.2, page 115 in this regard.

A content package can also have dependencies on other content packages. These content-related dependencies are not shown here! They are, however, visible in the version list for a package and in the detailed information for a package.

5.1.1.2.1 Configuring events for a package



Event	Workflow
OK	
Error	
Conflict	
Release	

Figure 5-4: Dialog - Configuring events

All of the events defined for the package are listed in the table and the scripts or workflows assigned to the event are shown. There are two types of events: Default events and what are known as package-specific events. **Default events** are provided by the system and handle the most common procedures when importing packages. Default events are:

- **OK**: The assigned workflow is carried out after the package version is imported successfully.

- **Error**: The assigned workflow is carried out if the package version is imported incorrectly.
- **Conflict**: The assigned workflow is started if there is a conflict situation after the package version is imported.
- **Release**: The assigned workflow is carried out after the package version is imported successfully, if no automatic release is configured in the subscription (see section 5.6.4.2, page 87). Thus, for example, all of the objects contained in the package can be released in the target project automatically.
- **Update**: The assigned workflow is carried out after the package version is imported successfully. The selected workflow is initiated for all of the nodes that have not been newly imported into the project but have been changed instead.

Clicking the **Edit** button opens a list with all known workflows from the source project. The desired workflow is selected from this list.

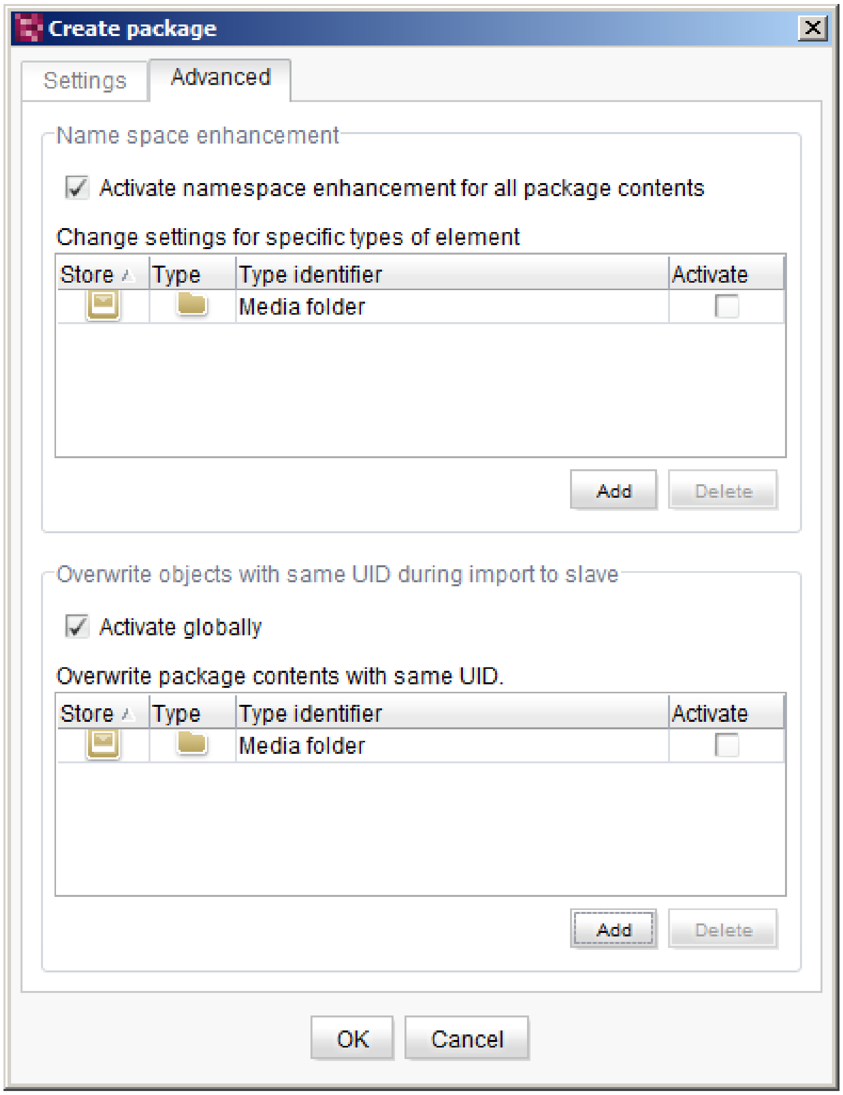
Clicking the **OK** button saves the changes and closes the dialog.

Clicking the **Cancel** button closes the dialog; any changes made are not applied.

All events configured in the package store are taken over in target projects with a subscription. However, the option of once again changing the event configuration for a package is present in the subscription store. The workflows that have been defined for the package in the source project can be changed again in the target projects (see section 5.6.4.4, page 90). These changes are not visible in the source project and are not applied to other target projects either.

The workflows from the source project can be assigned to a package. The workflows are not known in the target project during the first import. In this case, the required workflows have to be imported into the target project using a template package first. Only afterwards can events in additional packages be configured and implemented.

5.1.1.3 Creating a package - Advanced tab



Name space enhancement

☑ Activate namespace enhancement for all package contents

Change settings for specific types of element

Store	Type	Type identifier	Activate
		Media folder	□

Overwrite objects with same UID during import to slave

☑ Activate globally

Overwrite package contents with same UID.

Store	Type	Type identifier	Activate
		Media folder	□

Figure 5-5: Creating/editing a package - Advanced

Namespace extension:

In this area, namespace extensions for package contents can be activated or deactivated globally or for individual element types.

Enable namespace extension for all package contents: If this checkbox is **checked,** namespace extension is enabled for all package contents. This means that if an object is added to a package, the reference name is provided with a "@PackageName" extension. Afterwards, potential references to the added object have to be adapted in the project (see section 5.8.1.7, page 118 ff.).

If the checkbox is **unchecked**, namespace extension is disabled for all package contents. If an object is added to the package, the reference name remains unchanged (i.e. the reference name does not receive a "@PackageName" extension). Whether a master project's package contents overwrite the objects present in the target project or they are to be created under a different name in the target project can be defined using conflict handling for importing package contents into a target project (see section 5.8.1.7, page 118) in this case.

Change settings for specific element types: Enabling or disabling namespace extension is usually only desired for specific element types. Therefore, the global setting for package contents can be limited to specific element types.

Store: Representation of the store areas as an icon (the same as the tree display in FirstSpirit SiteArchitect). The column can be sorted.

Type: Representation of the element type as an icon (the same as the tree display in FirstSpirit SiteArchitect). The column can be sorted.

Type identifier: Name of the element type. The column can be sorted.

Enable: Namespace extension can be enabled or disabled for the respective element type by checking or unchecking this checkbox. If the checkbox is **checked**, namespace extension is enabled for the selected element type. If an object of the selected type (such as a format template) is added to a package, the reference name is provided with a "@PackageName" extension. Afterwards, potential references to the added object have to be adapted in the project (see section 5.8.1.7, page 118 ff.).
If the checkbox is **unchecked**, the namespace extension is disabled for the selected element type. If an object of the selected type (such as a format template) is added to a package, the reference name remains unchanged (i.e. the reference name does not receive a "@PackageName" extension). Whether a master project's package contents are to overwrite objects present in the target project or not can be defined using conflict handling for importing package contents into a target project in this case.

Clicking the **Add** button opens a dialog for selecting the desired element types that are to be added to the list (see section 5.1.1.3.2, page 59).

Clicking the **Delete** button can remove a selected element type back out of the list. The global settings for namespace extension again apply for this element type after it is removed.

Overwriting identically named objects during import into a target project

In this area, overwriting can be enabled in the target project for identically named objects or just for identically named objects of a specific type (e.g. format templates).

Enable globally: If this checkbox is **unchecked**, overwriting contents in a target object with identically named package contents is prevented (default setting). Conventional conflict handling that is also used when creating identically named objects in a project takes effect in this case: If a reference name (Uid) that has already been specified in a namespace is used, FirstSpirit automatically replaces the name with a unique name, usually by attaching a number. Thus, the package contents are created under another name in the target project in this case.
If the checkbox is **checked**, identically named contents in the target project are overwritten by package contents from the master project during importing. Thus, if the package contains a format template with the unique name "b", then if an identically named format template is imported into the target project it is overwritten by the identically named format template from the master project.

Overwriting package contents with the same Uid: In most cases, overwriting identically named contents in a target project is only desired for specific element types. Therefore, the global setting for package contents can be modified for specific element types.

Store: Representation of the store areas as an icon (the same as the tree display in FirstSpirit SiteArchitect). The column can be sorted.

Type: Representation of the element type as an icon (the same as the tree display in FirstSpirit SiteArchitect). The column can be sorted.

Type identifier: Name of the element type. The column can be sorted.

Enable: The default settings for import handling can be modified by checking or unchecking this checkbox. If the checkbox is **checked** (default setting), overwriting identically named objects in a target project is enabled for the selected element type. In this case, existing contents in a target project can be overwritten.
If the checkbox is **unchecked**, overwriting identically named objects in a target project is prevented for the selected element type. If an identically named object of the selected type (such as a format template) already exists in the target project, the object remains intact in the target project and the new package content is imported into the target project under a different name. Adjustments in the target project may be necessary in this case.

Clicking the **Add** button opens a dialog for selecting the desired element types that are to be added to the list (see section 5.1.1.3.2, page 59).

Clicking the **OK** button opens the new package for further editing in the CorporateContent store.

5.1.1.3.1 Namespace extension

Overlap between package contents is not permitted when creating packages; each project node is only allowed to belong to just one package. In order for the affiliation of an object to a package to be unique and as transparent as possible for the package developer, what is known as "namespace extension" has been introduced for package objects. When using it, an "@" and the package name are appended to the reference name of a package's objects ("ObjectName@PackageName").

> *The reference names with namespace extension can be displayed in the tree structure using the option "Show reference names in tree" in the "View" / "Preferred display language" menu.*

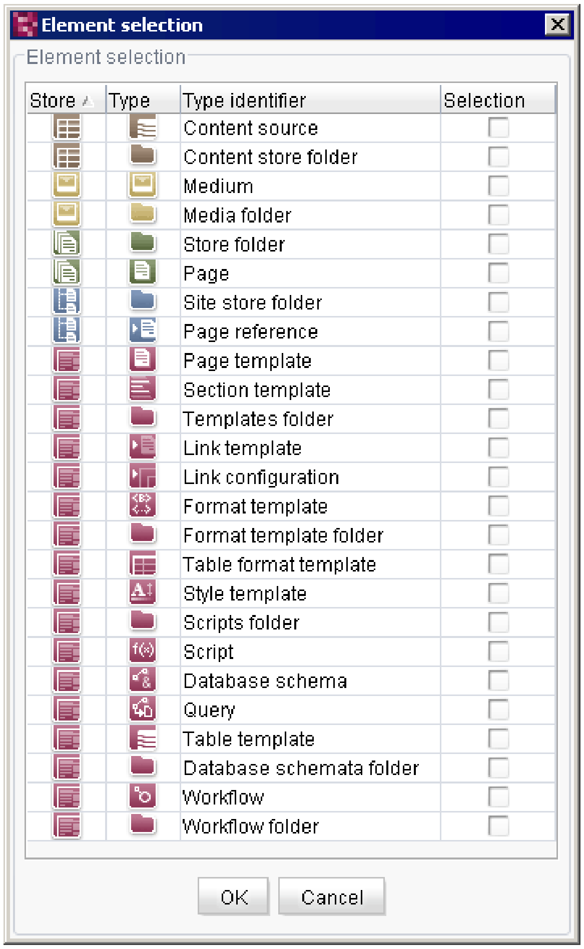
After being added to the package, all of the objects receive this namespace extension. Subsequently, all of the objects in the project that use the "old" reference name have to be changed; this means that the old reference name has to be replaced by a new reference name (with "@PackageName") everywhere. Sometimes these changes have to be made manually (see section 5.8.1.6, page 118 to section 5.8.1.8, page 120).

Namespace extension is problematic for package contents with identical reference names in the master and target project. This primarily affects standard format templates ("Bold", "Italics", etc.) that are present in every FirstSpirit project and combined into a folder under the "Format templates" node in the template store. They are used for formatting text and are used in the DOM editor and DOM table input components, for instance (also see *FirstSpirit Manual for Developers (Basics))*. Assignment to the corresponding buttons (e.g. "Bold") is lost in these input components by using namespace extension. Namespace extension can lead to errors in the master and target project in this case (see section 5.8.1.7).

The template developer can disable namespace extension for standard format templates as well as for other objects that have identical reference names in the master and target project.

5.1.1.3.2 Adding new element types

Clicking the **Add** button opens the "Element selection" dialog:



Store	Type	Type identifier	Selection
		Content source	☐
		Content store folder	☐
		Medium	☐
		Media folder	☐
		Store folder	☐
		Page	☐
		Site store folder	☐
		Page reference	☐
		Page template	☐
		Section template	☐
		Templates folder	☐
		Link template	☐
		Link configuration	☐
		Format template	☐
		Format template folder	☐
		Table format template	☐
		Style template	☐
		Scripts folder	☐
		Script	☐
		Database schema	☐
		Query	☐
		Table template	☐
		Database schemata folder	☐
		Workflow	☐
		Workflow folder	☐

Figure 5-6: Element selection for namespace extension

For a description of the Store, Type and Type identifier columns see section 5.1.1.3.1, page 58.

Selection: Checking this checkbox applies the selected elements to the list of selected element types. Only element types that are intended to be content for the package later on have to be selected in the process. For instance, no element types from the template store or content store (pink or brown icons) have to be selected in content packages.

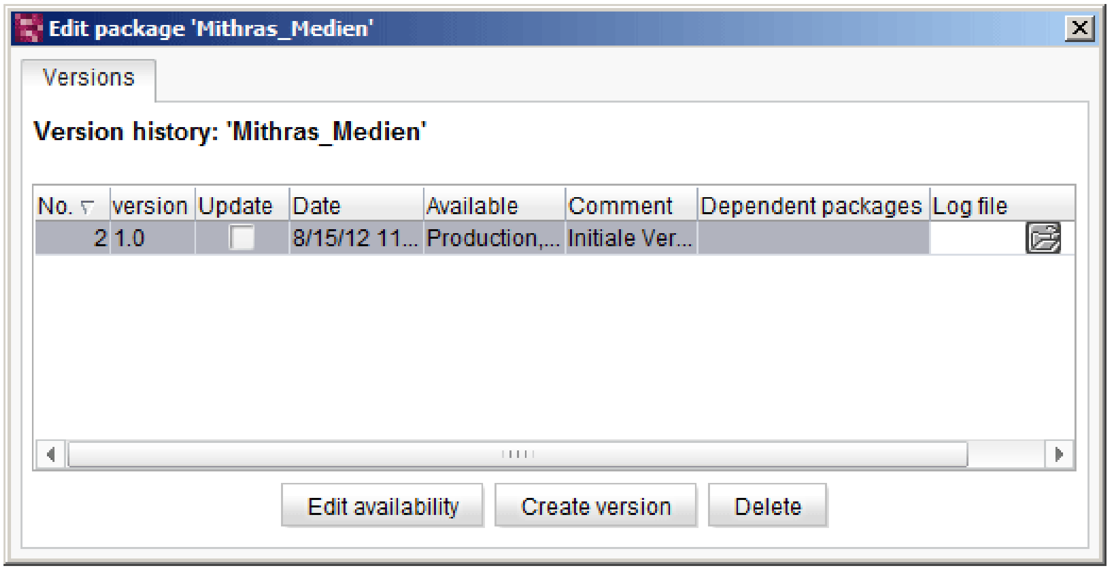
The default setting for transfer to the table is always the opposite of the global settings that have been defined via the "Enable namespace extension for all package contents" checkbox.

This means, if namespace extension for package contents is *disabled globally*, then namespace extension is *enabled directly* when transferring the selected element types.

If, on the other hand, namespace extension for package contents is *enabled globally*, then namespace extension is *disabled directly* when transferring the selected element types.

5.1.2 Creating a package version

Clicking the icon opens a dialog with an overview of all of the versions that have been created from the package.



Version history: 'Mithras_Medien'

No.	version	Update	Date	Available	Comment	Dependent packages	Log file
2	1.0	□	8/15/12 11...	Production,...	Initiale Ver...		

Figure 5-7: Editing package versions

No.: The unique version number that is assigned automatically when creating a new package version.

Version: The version designation specified by the package's creator.

Update: This check shows that the most recently released state of the integrated objects was used for the version.

Date: Date and time the package version was created.

Available: Shows the publication groups that the package version is available to.

Comment: Optional comment on the package version.

Dependent packages: Shows the dependent packages (templates and content packages, also see section 2.2.2, page 9) for the respective package version.

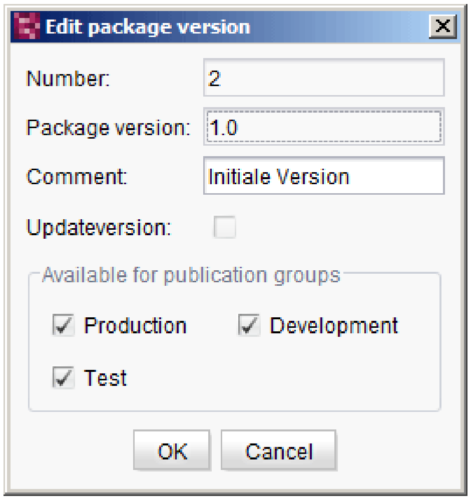
Log file: The appropriate log file can be displayed in a separate dialog here.

Using the **Edit availability** button or double-clicking the desired package version opens the "Edit package version" dialog box (see section 5.1.2.1, page 61).

Using the **Create version** button opens the "Create package version" dialog box (see section 5.1.2.2, page62).

The selected version is removed from the package using the **Delete** button.

5.1.2.1 Editing package availability



Edit package version

Number: 2

Package version: 1.0

Comment: Initiale Version

Updateversion: □

Available for publication groups

☑ Production ☑ Development

☑ Test

Figure 5-8: Edit package version dialog

Number: Unique version number. The field is inactive and cannot be edited.

Package version: Manually specified version designation when creating a new package version. The field is inactive and cannot be edited.

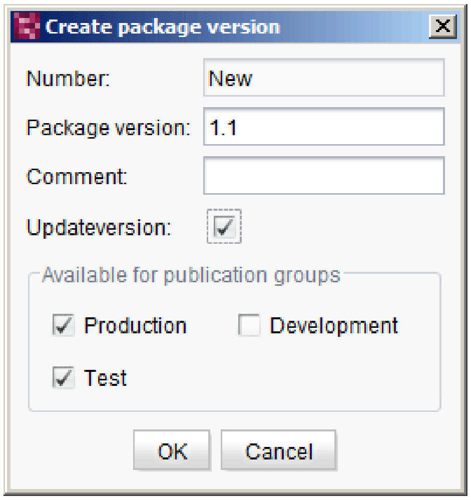
Comment: Optional comment. An existing comment can be modified or a new comment can be inserted at this point.

Update version: If this is checked in this field, then the most recently released state of the integrated objects is used for this version.

Available to publication groups: All of the available publication groups are shown as checkboxes here. The availability of the package version for the edited publication group changes by checking or unchecking a checkbox. If a checkbox is checked, the package version is available for import. If the checkbox is unchecked, the package version is not available to this publication group.

Clicking the **OK** button applies the changes to the existing package version.

5.1.2.2 Create a version



Create package version

Number: New

Package version: 1.1

Comment:

Updateversion: ☑

Available for publication groups

☑ Production □ Development

☑ Test

Figure 5-9: Create package version dialog

Number: In place of a unique version number, the entry "New" is shown here. The version number is specified by the system automatically when creating a new package version (field is inactive). Since, at this point, there still is not a new package version, a number cannot be displayed at this point.

Package version: Optionally, in addition to a version number assigned by the system, a "descriptive" (more meaningful) version number can be specified here.

Comment: Optional comment for the new package version.

Update version: The most recently released state of all integrated objects is used when creating the new version if this checkbox is checked. The current state of all integrated objects is used when creating a new version if this checkbox is unchecked.

If the package contains objects that

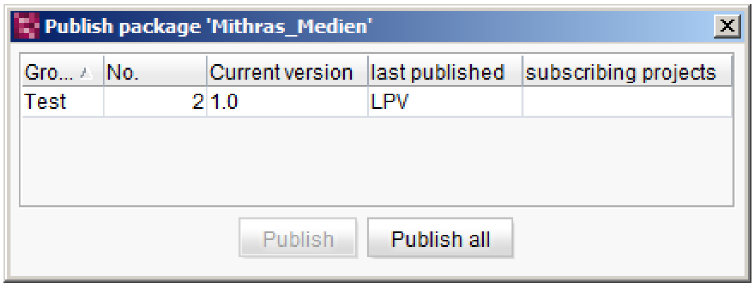
- *have never been released and the checkbox is selected or*
- *the objects have not currently been released and the checkbox is not selected,*

then an error message will appear to cancel the creation of a new version.

Available for publication groups: All available publication groups are shown as checkboxes here (see section 5.6.7, page 95). The availability of the package version for the respective publication group changes by checking or unchecking a checkbox. If the checkbox is **checked**, the package version is available for import to that publication group. If the checkbox is **unchecked**, the package version is not available to that publication group. A package version can be available to multiple publication groups; subscriptions, on the other hand, are always concluded for precisely one publication group (see section 5.6.4.2, page 87). If a package version is available to the "Test" and "Production" publication groups, a subscription for the "Test" publication group and a subscription for the "Production" publication group can access the package version.

5.1.3 Publishing a package

Clicking the icon opens a dialog where the respective most up-to-date package versions are listed in a tabular overview for known publication groups.



Publish package 'Mithras_Medien'

Gro...	No.	Current version	last published	subscribing projects
Test	2	1.0	LPV	

Figure 5-10: Publishing a package – Current version

Group: Publication group for which the package version has been marked as "available".

No: Unique package version number assigned automatically by the system.

Current version: Manually specified version designation.

Last published: Shows the last published version.

Subscribed projects: Shows all of the projects that have concluded a valid, active subscription for this package version and this publication group.

A package version goes through final publication using the buttons in the lower portion of the dialog box. Packages can only be published if:

- The person doing editing has publication permissions for the package.
- An active subscription exists for the package version and the publication group.

If the desired package version is marked in the table, it can be published by clicking the **Publish** button. Importing contents from the master project starts at this moment in all target projects that have concluded a valid, active subscription, with automatic updating, for this package version and the specified publication group.

The button is inactive and publication is not possible if one of the conditions listed above is not met.

Alternatively, *all* of the package versions shown in the window can be published together as well. The **Publish all** button is always active; however, only package versions that meet all of the conditions listed above are published.

Package dependencies should absolutely be determined before publication (see section 2.2.2, page 9). Dependencies on template packages are defined in the package properties. These dependencies are checked automatically. If the dependent template packages are not published or are published in incorrect order, publication is canceled and an error message is displayed.

Optional dependencies on other content packages are shown in the "Detail info" dialog box (see Figure 5-22: Detailed information on a **package**), which can be called up via the package overview. These dependencies are not checked automatically during publication. If the dependent content packages are not updated or are not updated in the correct order (1. importing the dependent content package, 2. importing the package containing the references to the dependent package), they can cause errors in the target project: For instance, if the referenced page and page reference are in different packages when publishing page references. If, in this example, the package with the page reference were published and then the package with the referenced page, this would cause an error in the target project. In order to resolve the error, the page reference would have to be locked for editing in the master project and then directly unlocked again. Then a new package version (from the package with the page reference) is generated and republished, in the correct order this time.

5.2 Adding objects to a package

5.2.1 Using the tree structure of the stores

Adding new objects for a CorporateContent package can be started directly using the tree structure in the corresponding store. There is a "*CorporateContent – Start adding to package"* entry in the context menu for this purpose.

Clicking this menu entry displays the selected object as a dependency chart in the AppCenter area of SiteArchitect and it can finally be added to a package using another context menu there (see section 5.7.1, page 106).

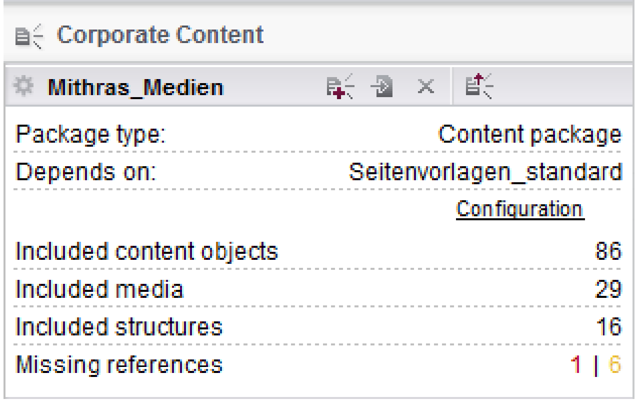
If an object in a store is copied to an area that is already part of a CorporateContent package, then a prompt appears asking whether this object is to be added to the corresponding package as well. Confirming this prompt adds the object to the package.

5.2.2 In the package combination

Additional objects can also be added to the package using the areas for necessary or optional dependencies. The checkbox in front of the respective object just has to be selected for the desired objects, then the objects are integrated into the package by clicking the **Add selected** button (see section 5.3.3 and 5.3.4 starting on page 68).

5.3 Package combination

5.3.1 Overview



Corporate Content

Mithras_Medien

Package type:	Content package
Depends on:	Seitenvorlagen_standard
	Configuration
Included content objects	86
Included media	29
Included structures	16
Missing references	1 | 6

Figure 5-11: Package – Overview

Package type: The package type configured when the package was created is specified here.

Dependent on: Manual dependencies on template packages are specified here. If the content package is subscribed to, then the associated template package specified here has to be subscribed to as well. Template packages do not have any dependencies. Thus the field is disabled for the template package type.

The package properties set when the package was created can be edited using the **Settings** button. A dialog for editing package properties opens (see section 5.1.1.2 and 5.1.1.3 starting on page 51).

Integrated content objects: How many objects are integrated into the package from the page store is specified here.

Integrated media: How many objects are integrated into the package from the media store is specified here.

Integrated structures: How many objects are integrated into the package from the site store is specified here.

Unfulfilled dependencies: How high the number of unfulfilled dependencies is in the entire package combination is specified here. The number of absolutely necessary objects is shown here in red; the number of optional objects is in yellow.

5.3.2 "Integrated objects" area

All objects integrated for the package are listed in this area. There is a distinction between explicitly and implicitly added objects. If an object is added explicitly, then all of the objects at a lower or higher level are implicitly added to the feature as well.

- Explicitly added objects are shown in normal text; this indicates that they can be removed from the list again by using the ✕ icon.
- Implicitly added objects are shown in text with less contrast and cannot be removed from the list.

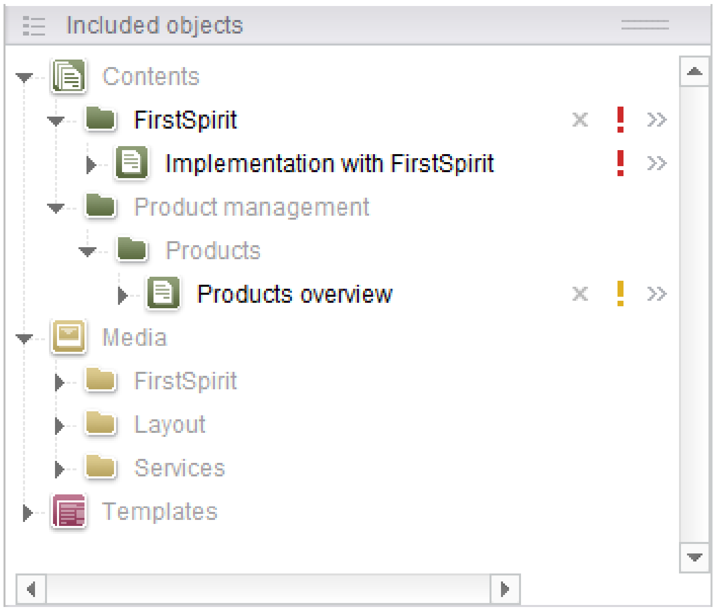


Figure 5-12: Package – Integrated objects

Clicking on an object in the list opens a tab with that object's forms in the SiteArchitect edit area for viewing.

⊰ Display dependency charts; clicking this icon opens a tab in the AppCenter area with a graphical representation of the hierarchical structure and the dependencies of the selected object (see section 5.5, page 71).

✕ Delete; this icon is only displayed if the associated object was explicitly integrated by the user. Clicking this icon removes the selected, explicitly added object from the list along with all implicitly integrated subobjects, after the user affirms a confirmation prompt. Higher level objects that are not used by other explicitly integrated objects are also removed.

! Unfulfilled optional dependencies; the yellow exclamation mark indicates that the respective object or a subordinate object has unfulfilled optional dependencies. The objects are listed in detail in the "Optional unfulfilled dependencies" area.

! Unfulfilled required dependencies; the red exclamation mark indicates that the respective object or a subordinate object has unfulfilled hard dependencies. The objects are listed in detail in the "Required unfulfilled dependencies" area.

» Object details; clicking this icon opens a flyout menu with object-specific information (see section 5.4, page 70). Clicking the icon again closes the flyout menu.

5.3.3 "Unfulfilled dependencies (own package)" area

All of the objects that have a dependency and belong to the same package type are shown in this area. If all of the dependencies are fulfilled then this area remains empty.

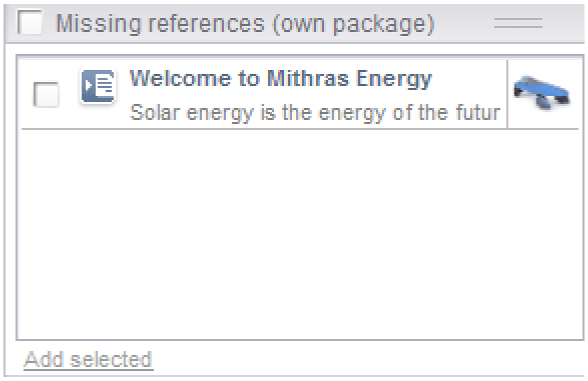

□ Missing references (own package)
□ Welcome to Mithras Energy
Solar energy is the energy of the futur
Add selected

Figure 5-13: Package – Unfulfilled dependencies (own package)

Dependent objects are shown in list form, each with a checkbox for selecting each individual object. Clicking on an object in the list opens a tab with that object's forms in the SiteArchitect edit area for viewing.

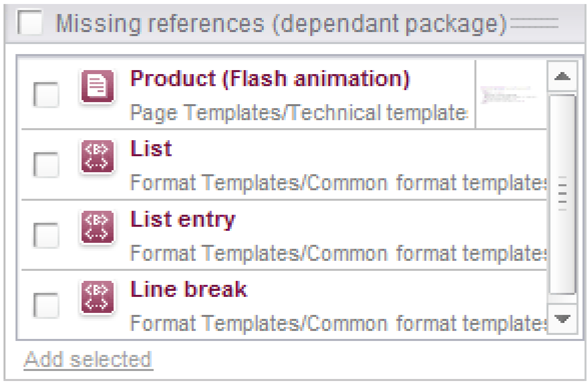
Unfulfilled dependencies (own package): If this checkbox on the top end of the area is selected, then the checkbox for selecting an object is selected for all of the objects in the list.

Clicking on the **Add selected** button integrates all of the objects selected in this area into the package combination.

Objects can be added only to a package with a suitable package type,

5.3.4 "Unfulfilled dependencies (foreign packages)" area

All of the objects that have a dependency but belong to a different package type are shown in this area. If all of the dependencies are fulfilled then this area remains empty.



□ Missing references (dependant package)

□	**Product (Flash animation)** Page Templates/Technical template
□	**List** Format Templates/Common format template:
□	**List entry** Format Templates/Common format template:
□	**Line break** Format Templates/Common format template:

Figure 5-14: Package – Optional unfulfilled dependencies

Dependent objects are shown in list form, each with a checkbox for selecting each individual object. Clicking on an object in the list opens a tab with that object's forms in the SiteArchitect edit area for viewing.

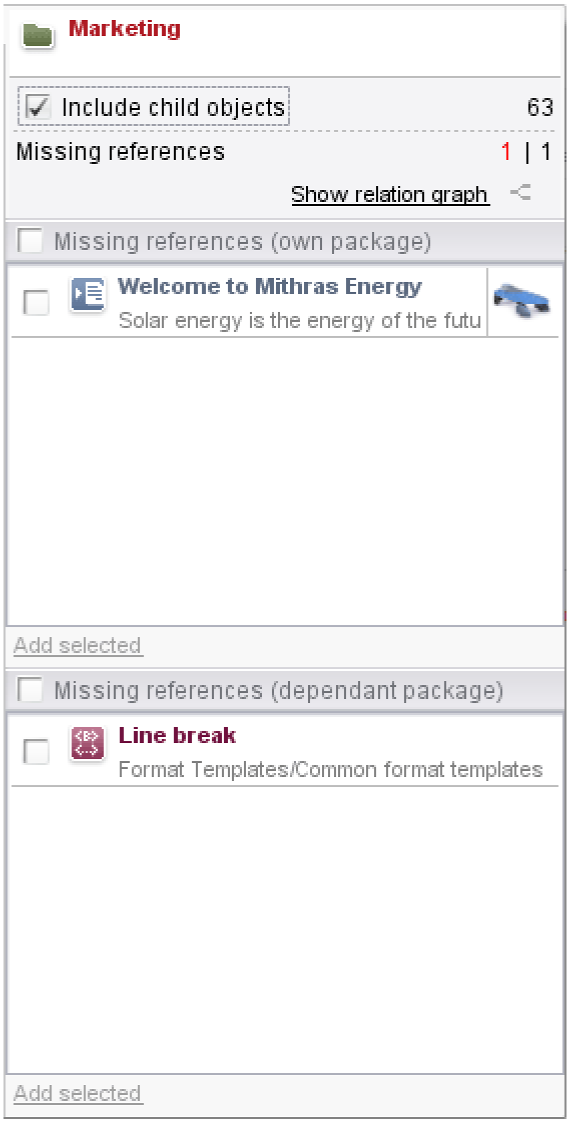
Unfulfilled dependencies (foreign packages): If this checkbox on the top end of the area is selected, then the checkbox for selecting an object is selected for all of the objects in the list.

Clicking on the **Add selected** button integrates all of the objects in the package combination selected in this area for which a dependency has been defined in the package settings (see section 5.1.1.2, page 51).

Objects can be added only to a package with a suitable package type,

5.4 Flyout menu

The flyout menu contains object-specific information that is shown in the same way as the information for the package combination.



Marketing

☑ Include child objects 63

Missing references 1 | 1

Show relation graph

□ Missing references (own package)

□ Welcome to Mithras Energy
Solar energy is the energy of the futu

Add selected

□ Missing references (dependant package)

□ Line break
Format Templates/Common format templates

Add selected

Figure 5-15: Flyout menu

The flyout menu always contains at least a tabular listing of object-specific data the same way as the feature overview.

- Icon and language-dependent display name for the displayed object
- "Include child objects" checkbox – If this checkbox is selected, then the unfulfilled dependencies for the displayed object are displayed along with all of the object's subobjects. If the checkbox is not selected, then only the unfulfilled dependencies of the displayed object are displayed.
- Number of subobjects
- Number of unfulfilled dependencies (own package and foreign package)
- A tab with a graphical representation of an object's dependencies can be displayed using the **Show dependencies** button in the AppCenter area (see section 5.5, page 71).

The Required and Optional unfulfilled dependencies areas are the same as the areas in the package combination with the same name (see section 5.3.3 and 5.3.4 starting from page 68).

5.5 Graphical representation of dependencies

The graphical representation is used to provide a flexible view of the hierarchical structure and dependencies of a package's embedded objects.

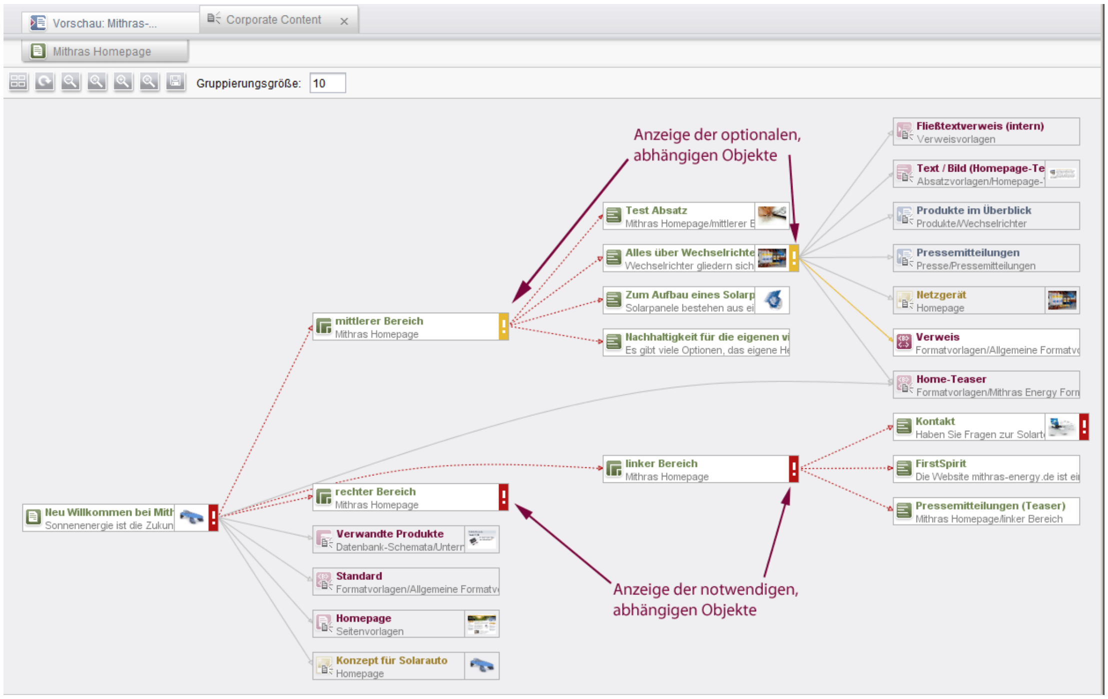

Vorschau: Mithras-...
Corporate Content
Mithras Homepage
Gruppierungsgröße: 10
Anzeige der optionalen, abhängigen Objekte
Fließtextverweis (intern)
Verweisvorlagen
Text / Bild (Homepage-Te
Absatzvorlagen/Homepage-
Test Absatz
Mithras Homepage/mittlerer E
Produkte im Überblick
Produkte/Wechselrichter
Alles über Wechselrichte
Wechselrichter gliedern sich
Pressemitteilungen
Presse/Pressemitteilungen
Zum Aufbau eines Solarp
Solarpanele bestehen aus ei
Netzgerät
Homepage
mittlerer Bereich
Mithras Homepage
Nachhaltigkeit für die eigenen vi
Es gibt viele Optionen, das eigene He
Verweis
Formatvorlagen/Allgemeine Formatvo
Home-Teaser
Formatvorlagen/Mithras Energy Forn
Kontakt
Haben Sie Fragen zur Solart
linker Bereich
Mithras Homepage
FirstSpirit
Die Website mithras-energy.de ist ei
rechter Bereich
Mithras Homepage
Neu Willkommen bei Mith
Sonnenenergie ist die Zukun
Pressemitteilungen (Teaser)
Mithras Homepage/linker Bereich
Verwandte Produkte
Datenbank-Schemata/Untern
Standard
Formatvorlagen/Allgemeine Formatv
Anzeige der notwendigen, abhängigen Objekte
Homepage
Seitenvorlagen
Konzept für Solarauto
Homepage

Figure 5-16: Show dependencies

The dependency chart is integrated as a reusable tab in the SiteArchitect AppCenter area.

5.5.1 Icon bar

Layout; clicking this icon automatically arranges the displayed objects in a uniform layout. In the process, layout changes made by the user are discarded without prompting.

Update; clicking this icon updates the information displays in the dependency chart. Changes to the hierarchical structure of the objects and the package's new or removed objects are taken into account in the process.

Zoom out/1:1/in; clicking on these icons changes the view of the dependency chart by increasing it, shrinking it or changing it back to its original size.

Fit to Screen; clicking this icon adjusts the zoom level so that the entire dependency chart is visible at the current tab size.

Save as image; clicking this icon opens a dialog box for selecting the name and save location for creating an image file in PNG format. The created picture file contains the entire dependency chart at the selected zoom level.

Grouping size: How many objects are to be displayed at the same time using the "Show linked objects" context menu function or by double-clicking can be specified in this field.

5.5.2 Display for dependency charts

The dependency chart displays objects from the stores and their connections to each other. These connections can be relationships between parent and child objects as well as references between different objects.

The object that the dependency chart was retrieved for is used as the root node for the view and appears at the far left in the dependency chart. When expanding outgoing connections for an object, the target nodes are arranged to the right of the object. Each link between two objects is represented by an arrow that points to the child or referenced objects.

Each object is shown as a rectangle and contains the following information:

- Object icon; the same icon that is also displayed in the tree structure of individual stores.
- Suitcase symbol (💼); specifies whether the object is currently integrated in the feature. It is placed on the object icon for identification
- Display name; the language-dependent display name from the relevant stores is displayed.
- Preview symbol; if present, a preview of the object or integrated images is shown as a

symbol.

- Exclamation mark (!/!); displayed if unfulfilled dependencies exist for the object or a subobject. The color rules match the tree view in the "Included objects" area of the feature combination (see section 4.3.2, page 27).

Edge lines are drawn between objects that are related to each other. These lines are shown differently depending on the fulfillment status:

- Solid line; used for explicitly added objects.
- Dashed line; used for implicitly added objects that are subelements of an explicitly integrated object.
- Gray; used between objects if their connection is intact in the current feature combination (i.e. always subordinate objects, referenced objects if the reference target is integrated in the feature).
- Red; used between objects if an unfulfilled required dependency is present.
- Yellow; used between objects if an unfulfilled optional dependency is present.

5.5.3 Context menu on objects

A context menu can be called up on each object. The functions of the context menu are active or grayed out depending on the state of the object.

Add to package "xyz": The selected object and all of its subelements are integrated into the package.

Show element in edit area: Calling this function opens a tab with this object's forms in the SiteArchitect edit area for viewing.

Show linked objects (double-click): If there are connections to other objects that are not yet displayed, calling this function triggers the display of these objects. The maximum number of objects that are displayed there is specified on the icon bar under "Grouping size". Another group of objects can be displayed by calling the function (or double-clicking) again.

If not all linked objects were able to be displayed, this is indicated by an extra object with the label "Show next elements (X total)". Another group of objects can also be displayed by double-clicking this extra object.

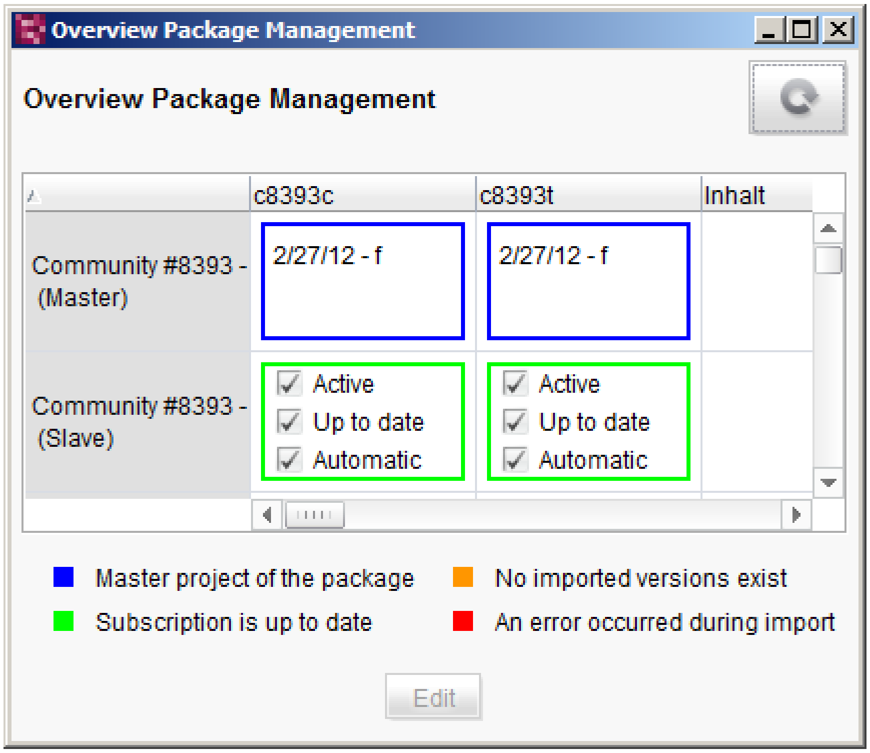
Hide linked objects: All linked objects currently being displayed can be hidden by calling this function. All of a linked object's subordinate objects are also hidden in the process. Thus, if an object is hidden and then immediately displayed again, then its subordinate objects continue to be hidden.

5.6 Functions via the "CorporateContent" menu item

Some of CorporateContent's functions can also be called up in the FirstSpirit menu bar under the "CorporateContent" menu item and its submenu items.

5.6.1 Overview menu item

The "Package store overview" dialog box opens via this menu function. The same information is shown here in the master project and target project(s).



Overview Package Management

Overview Package Management

	c8393c	c8393t	Inhalt
Community #8393 - (Master)	2/27/12 - f	2/27/12 - f	
Community #8393 - (Slave)	☑ Active ☑ Up to date ☑ Automatic	☑ Active ☑ Up to date ☑ Automatic	

🟦 Master project of the package

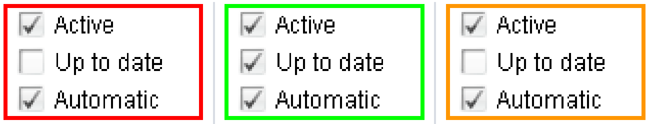
🟧 No imported versions exist

🟩 Subscription is up to date

🟥 An error occurred during import

Figure 5-17: Package store overview

The window shows the most important information about packages, projects and the current state of subscriptions in an overview window. All of the projects on the server are shown on the vertical axis and all packages that are known server-wide are shown on the horizontal axis here. The intersection between a package and a project displays quick information about the state of the subscription for the package in the corresponding project.



☑ Active
□ Up to date
☑ Automatic

☑ Active
☑ Up to date
☑ Automatic

☑ Active
□ Up to date
☑ Automatic

Figure 5-18: Quick information on a subscription in a target project

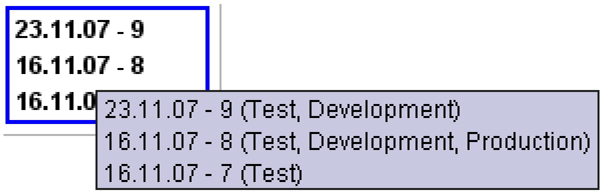
Subscriptions in **target projects** are shown with the following information:

Active – A checked checkbox indicates that the subscription to the package is active. The status can be modified directly in this overview by double-clicking the subscription or using the **Edit** button; this opens a dialog box with detailed information (see section 5.6.1.1, page 78). The detailed information could also be opened using the **Details** button in the "Edit subscriptions" dialog box (see Figure 5-33).

Current – A checked checkbox indicates that the current package version has already been imported into the target project. The checkbox is only checked for subscriptions marked in green; the checkbox is unchecked for subscriptions marked in orange or red (see below for subscription color coding).

Automatic – A checked checkbox indicates that the package contents are updated in the target project automatically as soon as a more up-to-date package version is made available (push process, also see section 2.2.5.1, page 14). The state can be changed using the *Update* parameter in Figure 5-29: Creating a subscription for package **'xyz'**.

Packages from **master projects** are displayed with the following information:



23.11.07 - 9
16.11.07 - 8
16.11.0

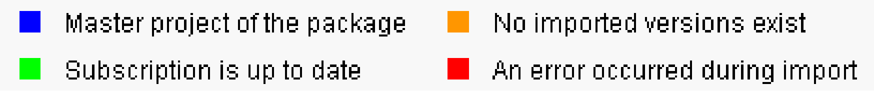
23.11.07 - 9 (Test, Development)
16.11.07 - 8 (Test, Development, Production)
16.11.07 - 7 (Test)

Figure 5-19: Quick information on a package in a master project

Packages from master projects show the last three package versions in the blue box. This allows the user to see at a glance how up-to-date the package version is in the respective project and which package version should be updated in the target projects. A tool tip with the associated publication groups is shown as additional information if the user hovers the mouse over the blue box.

If the intersections between package and project are empty, there is no subscription to the package in the target projects (*for information on creating a new subscription: see section 5.6.4 starting on page 86*). If only an empty blue box is shown, then no package version has been created for the package in the master project.

In order to show the state of a subscription in a clear and concise manner, color coding has been introduced in addition to quick information. This is shown as a colored box in the overview.



🟦	Master project of the package	🟧	No imported versions exist
🟩	Subscription is up to date	🟥	An error occurred during import

Figure 5-20: color coding for the state of a subscription

Blue box – Marks the master project for the respective package.

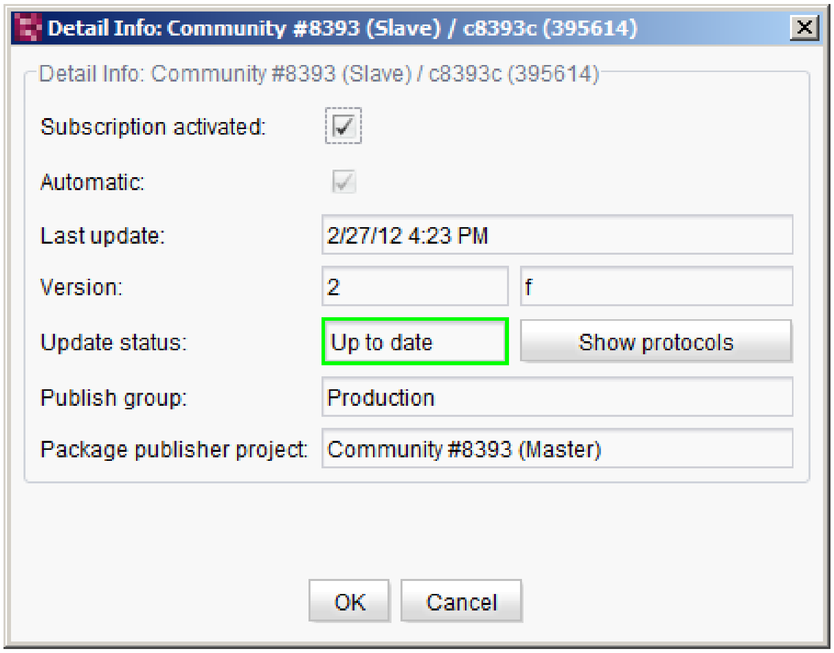
Green box – Means that the currently most up-to-date package version has already been imported into the target project successfully.

Red box – Marks an import into the target project that has an error. In this case, the log file from the detailed information (see section 5.6.1.1, page 78) should be called up (see section 5.6.1.3, page 81).

Orange box – Means that a more up-to-date package version has become available for import but the target project has not yet been updated (*for information on updating the subscription starting from a target project: see section 5.6.6, page 93, for information on updating the subscription starting from a master project: see section 5.6.3, page 85*).

Clicking the **Edit** button or double-clicking the desired package-project relationship opens the "Project/Package detailed information" dialog box. Information in addition to that from the overview can be viewed here for each intersection in the overview. A distinction is made here between *information on subscriptions* (green, orange and red boxes, see section 5.6.1.1, page 78) and *information on packages* (blue box, see section 5.6.1.2, page 80).

5.6.1.1 Detailed information on subscriptions



Detail Info: Community #8393 (Slave) / c8393c (395614)

Detail Info: Community #8393 (Slave) / c8393c (395614)

Subscription activated: ☑

Automatic: ☑

Last update: 2/27/12 4:23 PM

Version: 2 f

Update status: Up to date

Publish group: Production

Package publisher project: Community #8393 (Master)

Figure 5-21: Detailed information on a subscription

The name of the target project being subscribed to and the subscribed package with ID are shown on the window's title bar and as a header in the content area.

Subscription active: If this checkbox is checked, then the subscription is active for the package. This means that all new package versions are made available for importing in this project (also shown in the quick info). The checkbox is active and can be edited in this dialog (also see section 5.6.4.2, page 87).

Automatic: If this checkbox is checked, the target version is updated to a new package version automatically. The checkbox is disabled and is only used to provide information. The state can be modified in the subscription properties (see section 5.6.4.2, page 87).

Last update: Shows the date and time of the last update to the package in the target project.

Version: In the first field, shows the unique version number for the package version assigned by the system. In the second field, a version number manually specified by the master project's package developer is shown as well.

Update status: Shows the status of the update in the target projects. The information follows the color coding in the overview window. The three known states for a subscription are specified here:

- Current – The most up-to-date package version has been imported successfully.
- Outdated – A more up-to-date package version is available for import.
- Error – Incorrect import into the target project.

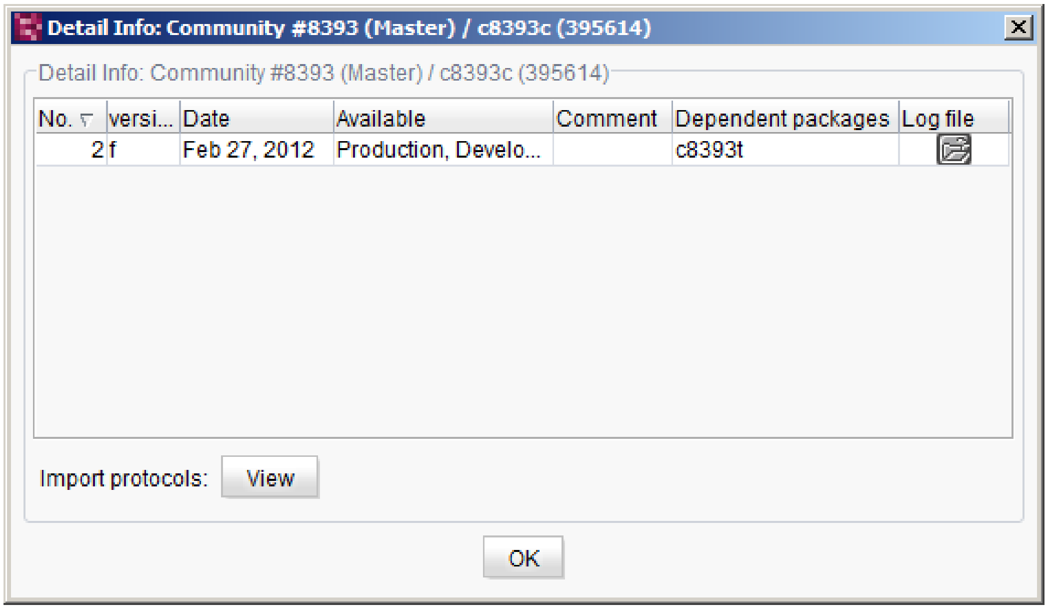
Clicking the **Show log** button opens the "Show log file" dialog box. The log file records the specific process while the packages are imported and, if an import has an error, is of particular interest for being able to evaluate the error that occurred. Additional information in section 5.6.1.3, page 81.

Publication group: Shows the publication group(s) for which the subscription has been concluded.

Package publication project: Shows the master project, i.e. the project where the package was created.

Clicking **OK** closes the "Detailed info" dialog box; any change to the "Subscription active" checkbox is applied to the subscription.

5.6.1.2 Detailed information on packages



Detail Info: Community #8393 (Master) / c8393c (395614)

Detail Info: Community #8393 (Master) / c8393c (395614)

No.	versi...	Date	Available	Comment	Dependent packages	Log file
2	f	Feb 27, 2012	Production, Develo...		c8393t	

Figure 5-22: Detailed information on a package

Additional information on a package can be called up in the blue box in the "Package store overview" dialog box by double-clicking.

Before updating subscriptions in a target project (see section 5.6.6, page 93), the detailed information about the package should be checked using this dialog in order to discover potential dependent content packages ("Dependent packages" column) that have to be imported ***before*** *the content package that contains references to dependent objects.*

The name of the master project being subscribed to and the subscribed package with ID are shown on the window's title bar and as a header in the content area.

The table shows the created package versions for the different publication groups (see section 2.2.3, page 11). The most up-to-date package version is shown at the top by default in the process.

No: Shows the unique version number assigned by the system.

Version: Shows the manually specified version name.

Date: Shows the date of version creation.

Available: Shows the publication group(s) that this package version is "available" for.

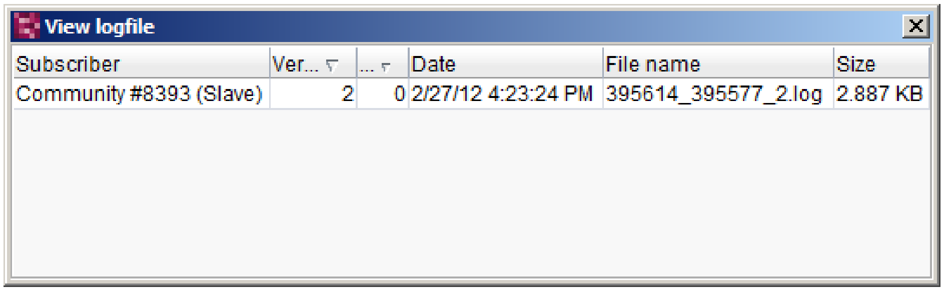
Comment: Optional comment on the package version.

Dependent packages: Shows the dependent packages (templates and content packages, also see section 2.2.2, page 9) for the respective package version.

Log file: The log file for package version creation can be viewed here.

Import log: The "Show log file" dialog box is opened using the **Display** button (see section 5.6.1.3, page 81).

5.6.1.3 Showing a log



Subscriber	Ver...	...	Date	File name	Size
Community #8393 (Slave)	2	0	2/27/12 4:23:24 PM	395614_395577_2.log	2.887 KB

Figure 5-23: Show log file

A log file is created each time a package version is imported into a target project. The log file records all of the information during the import process and is important for correcting any errors. A log file can be selected for each subscription and each imported package version using the "Show log file" dialog box. The table can be sorted by clicking the respective column.

Subscriber: Specifies the target project where the package was imported.

Version: Shows the version number assigned by the system.

No: Shows the number of attempts to import into a target project. The number is normally "0" if automatic importing is configured. However, if an error occurs when importing a package version, the import is initiated again and the number is increased by "1".

Date: Shows the date and time of the import.

File name: Shows the name of the log file. The name is a combination of:

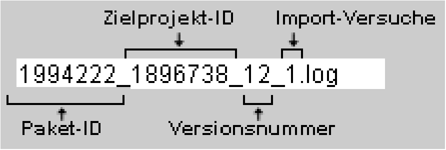
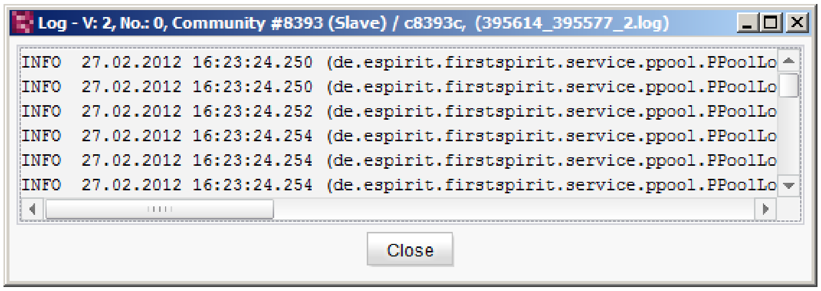


Figure 5-24: Log file name combination

Double-clicking an entry opens the associated log file:


Log - V: 2, No.: 0, Community #8393 (Slave) / c8393c, (395614_395577_2.log)

```
INFO  27.02.2012 16:23:24.250 (de.espirit.firstspirit.service.ppool.PPoolLo
INFO  27.02.2012 16:23:24.250 (de.espirit.firstspirit.service.ppool.PPoolLo
INFO  27.02.2012 16:23:24.252 (de.espirit.firstspirit.service.ppool.PPoolLo
INFO  27.02.2012 16:23:24.254 (de.espirit.firstspirit.service.ppool.PPoolLo
INFO  27.02.2012 16:23:24.254 (de.espirit.firstspirit.service.ppool.PPoolLo
INFO  27.02.2012 16:23:24.254 (de.espirit.firstspirit.service.ppool.PPoolLo
```

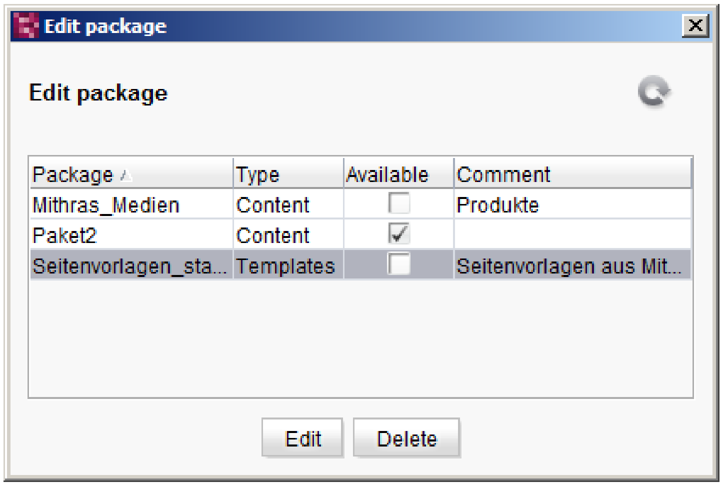
Figure 5-25: Log file

Log entries with the status `ERROR` are of particular interest. If there is an error during importing or updating, you may be able to find out here whether additional referenced objects from the master project are needed so that the import can then run successfully.

The log outputs can also be opened in an external editor. To do so, all of the outputs first have to be highlighted using the key combination **Ctrl+A** and then copied to the clipboard using **Ctrl+C**. Then the external editor is opened and the content in the clipboard is pasted into the editor using **Ctrl+V**. This process is particularly advantageous when analyzing larger files.

5.6.2 Package menu item - Edit packages

The "Edit package" dialog is opened using this menu function. All of the packages present in the source project are shown in this dialog box.



Edit package

Package	Type	Available	Comment
Mithras_Medien	Content	☐	Produkte
Paket2	Content	☑	
Seitenvorlagen_sta...	Templates	☐	Seitenvorlagen aus Mit...

Figure 5-26: Edit package – Package list

The table provides the following information on each package:

Package: Unique package name.

Type: Shows whether this is a content package or template packages.

Available: If this checkbox is **checked**, the package is available for target projects and can be subscribed to. A subscription can even be created if no package version exists for a package. If the checkbox is **unchecked**, the package is available to be subscribed to in the target projects.

Comment: Optional comment for the package.

Clicking the **Edit** button (or double-clicking the table row) opens the selected package for further editing in the CorporateContent store (see section 5.3, page 66).

Packages can be deleted from the table using the **Delete** button. In order to prevent a package from being deleted accidentally, a confirmation prompt is displayed before it is permanently deleted.

Deleting a package also removes all of the package versions! Therefore, it is not possible to directly delete packages where subscriptions have already been concluded. In this case, the following confirmation prompt is displayed first.

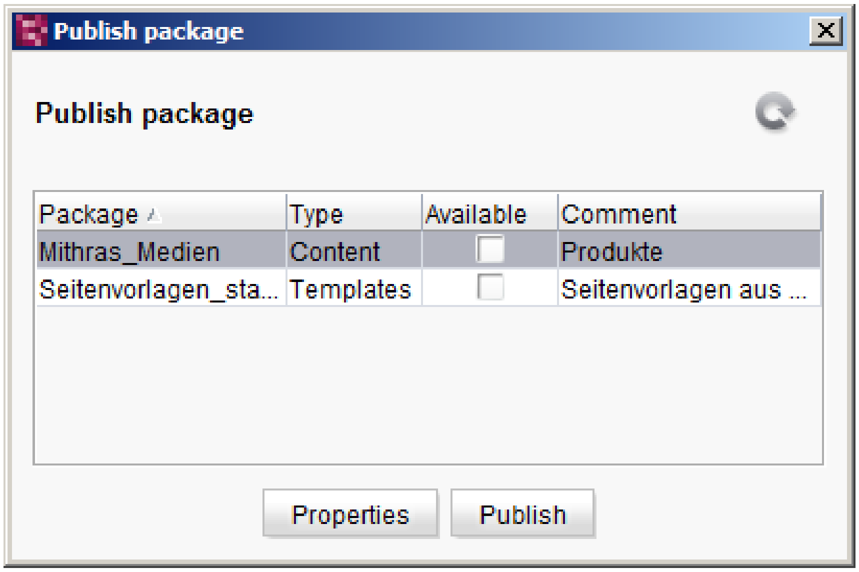
Affirming the confirmation prompt first deletes all existing subscriptions to the package and then the package itself.

The package cannot be deleted if a dependency on a template package is present. In this case, the link to the template package has to be removed in the content package's properties first. Only then can the content package be deleted.

> *If namespace extension has been enabled, the extended reference names ("ObjectName@PackageName") remain intact after a package is deleted; they are not reset to their original reference names.*

5.6.3 Package menu item - Publish packages

This menu function is used for updating package contents in target projects using what is know as the "push" process (also see section 2.2.5.1, page 14). The "Publish package" dialog box opens listing all of the existing packages in one table.



Package	Type	Available	Comment
Mithras_Medien	Content	□	Produkte
Seitenvorlagen_sta...	Templates	□	Seitenvorlagen aus ...

Figure 5-27: Publish package – Overview

Package: Unique package name.

Type: Shows whether this is a content package or template packages.

Available: If this checkbox is **checked**, the package is available for target projects and can be subscribed to. A subscription can even be created if no package version exists for a package. If the checkbox is **unchecked**, the package is available to be subscribed to in the target projects.

Comment: Optional comment for the package.

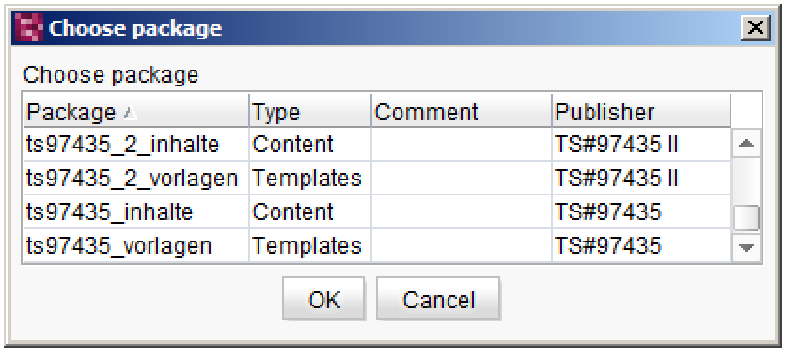
Clicking the **Properties** button opens a dialog with the package properties that were set when the package was created (see section 5.1.1.2 and 5.1.1.3 starting on page 51). The package properties cannot be modified at this point; they are only for informational purposes.

Clicking the **Publish** button opens a dialog where the most up-to-date package versions for each known publication group are listed in a tabular overview. This dialog can also be called up using the icon in the "CorporateContent" store (see 5.1.3, page 63).

5.6.4 Subscription menu item - Create subscription

A new subscription in a target project can be created using this menu function. Creating a new subscription takes a multi-step process; the steps are explained below. Only a target project's administrator can carry out the initial creation of a subscription.

5.6.4.1 Selecting a package



Package	Type	Comment	Publisher
ts97435_2_inhalte	Content		TS#97435 II
ts97435_2_vorlagen	Templates		TS#97435 II
ts97435_inhalte	Content		TS#97435
ts97435_vorlagen	Templates		TS#97435

Figure 5-28: Selecting a package

The "Create subscription" menu item opens the "Select a package" dialog box. All of the packages available on the server are shown in this dialog box. Only one package can ever be selected at a time. The table provides the following information on each package:

Package: Unique package name.

Type: Specifies whether this is a content package or a template package.

Comment: Optional comment for the package.

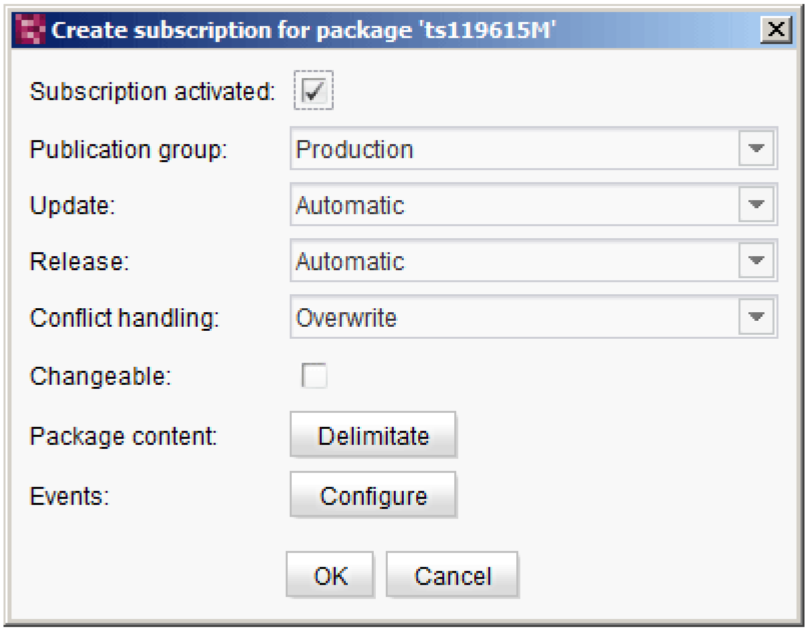
Publisher: Shows the name of the master project where this package was created.

Clicking **Cancel** closes the window. A dialog for editing the subscription opens (see section 5.6.5, page 91).

Clicking **OK** opens another dialog box for creating a subscription (section 5.6.4.2, page 87).

No new subscriptions can be created if no packages are available for subscribing. A dialog box appears with a corresponding error message.

5.6.4.2 Creating a subscription for a package



Create subscription for package 'ts119615M'

Subscription activated: ☑

Publication group: Production

Update: Automatic

Release: Automatic

Conflict handling: Overwrite

Changeable: ☐

Package content: Delimitate

Events: Configure

Figure 5-29: Creating a subscription for package 'xyz'

All of the settings for the subscription are set by the administrator of the target project in the "Create subscription for package 'xyz'" dialog box:

Subscription active: An update that can be initiated manually or automatically is provided for each new package version if this checkbox is **checked**. If the checkbox is **unchecked**, the package is not updated automatically in the target project. If a manual update is provided for the subscription, the administrator of the target project can update the subscription even if it is not "active" (see section 5.6.6, page 93).

> *A subscription can only be deleted starting from the source project. Therefore, in order to "cancel" a subscription, the "Subscription active" option should be unchecked here. In this case, the subscription can only still be updated manually; this prevents an update initiated from the master project.*

Publication group: A publication group can be selected for the subscription in the drop-down list (see section 2.2.4, page 13). All available publication groups are shown. If a publication group for which no package version has been made "available" is selected at this point, a subscription can, in fact, be created. However, an update (see section 5.6.6, page 93) only takes place if a package version for that publication group exists as well:

Update: The type of the update for the package in the target project can be selected in this drop-down list. If **automatic update** is set, importing is initiated from the master project and runs in the target project automatically. If, on the other hand, **manual update** is set, importing is initiated from the target project using the "Update subscription" menu item (see section 5.6.6, page 93). A manual update can also be carried out if the subscription is not "active".

Release: Release control for the package can be adjusted using this drop-down list. The release can be **automatic**, i.e. all included objects are released in the target project automatically after importing the package. However, the release can also be configured via a **workflow**. Both settings only apply if the target project is also working with releases (see section 2.2.5.3, page 16). If this is not the case, then the entries are simply ignored.

> *Different release states can occur when using releases in a target project if a package is imported again after the "Release" workflow is started. At this point, the newly imported object no longer corresponds to the initial released state.*

Conflict handling: This drop-down list controls the process in the event of a conflict when importing a package. These conflicts can only arise if the "Modifiable" checkbox is active (see below). This means that the package contents may be changed locally in the target project. A conflict situation could occur during the next update due to these local changes. The conflict is triggered only if the change status for an object is set to "Modified" or "Locked" (see section 5.7.4, page 109). The change status is configured manually using the context menu for the respective objects.

Depending on the change status that is set and the way conflict handling is configured, changes to objects are overwritten, copied or the update for the entire subscription is prevented.

- **Overwrite –** The local changes are overwritten by the new package contents.
- **Cancel** – The import is canceled.
- **Copy –** A copy of the node where the conflict occurred is created. An exception is made for nodes in the site store: Copies of the node are not created here, instead they are overwritten.

The exact results of conflict handling, depending on the change status that is set, are described in section 5.7.4 on page 109.

Modifiable: Write permission to the imported objects is granted for the target project if this checkbox is **checked**. If the checkbox is **unchecked**, the imported objects can be seen and used in the target project but they cannot be modified. An error message appears when trying to block the object in a target project. This setting also affects the order when importing objects into target projects (see section 5.6.8.3, page 101).

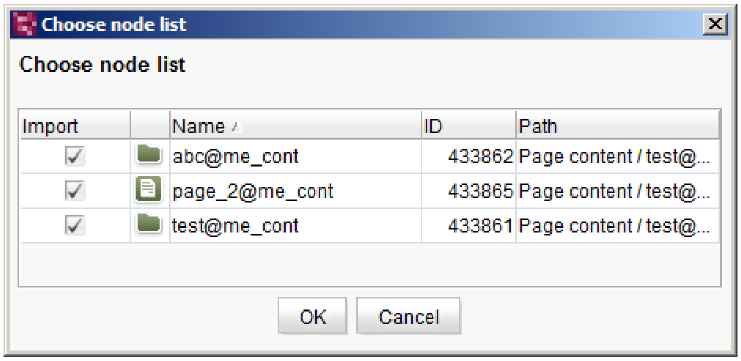
Package content: Using the **Limit** button opens the "Select node list" dialog box for limiting package contents during importing (see section 5.6.4.3, page 89).

Events: Using the **Configure** button opens the "Configure events" dialog box for editing or deleting events already present in the package (see section 5.6.4.4, page 90).

Clicking the **OK** button creates a new subscription.

5.6.4.3 Limiting package content in a subscription

Clicking the **Limit** button in the *package content* row opens the "Select node list" dialog box:



Choose node list

Choose node list

Import	Name	ID	Path
☑	abc@me_cont	433862	Page content / test@...
☑	page_2@me_cont	433865	Page content / test@...
☑	test@me_cont	433861	Page content / test@...

Figure 5-30: Selecting a node list

All of the objects included in a package version are listed in the dialog box.

Import: This checkbox is checked by default for each object. If specific objects are not intended to be imported into the target project, then the associated checkbox has to be unchecked. Pages from the page store can only ever be deactivated together with the child elements (sections) in this context.

Caution: If package contents are limited manually here, then the dependencies between package contents absolutely have to be taken into account (see section 2.2.2, page 9). If nodes that have to be included in the package are deleted manually here, then this will result in errors during importing.

Name: Shows the name of the object from the master project. Objects integrated into a package are provided with a namespace extension.

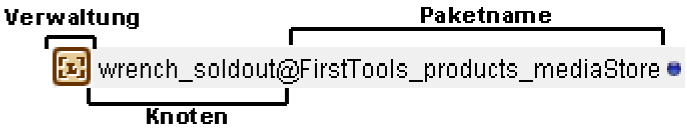


Figure 5-31: Namespace extension for package contents

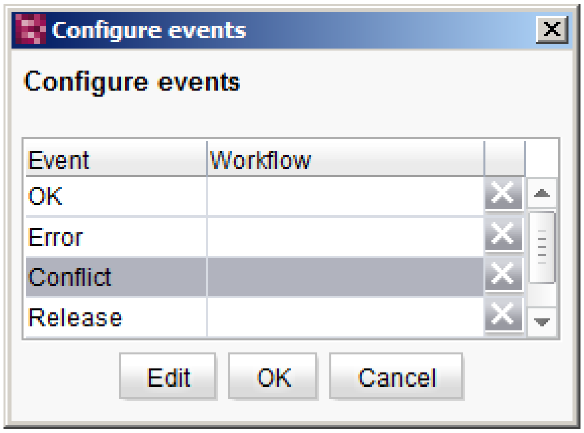
It is possible to deactivate namespace extension (see section 5.1.1.3.1, page 58). In this case, the objects are shown without the appended "@PackageName".

ID: Shows the object ID from the master project.

Path: Path to the object in the master project's project tree

5.6.4.4 Configuring events for a subscription

Clicking the **Configure** button in the *Events* row opens the "Configure events" dialog box.



Event	Workflow
OK	
Error	
Conflict	
Release	

Figure 5-32: Configuring events (in a target project)

Events, such as errors or releases, that have already been defined when creating a new package version in a source project can be assigned workflows here. The workflows can be deleted for the target project or be replaced by other workflows using this dialog. New events cannot be created.

☒ deletes an existing workflow from the event table.

Clicking the **Edit** button opens a dialog for selecting a new workflow.

Clicking the **OK** button saves the changes and closes the dialog.

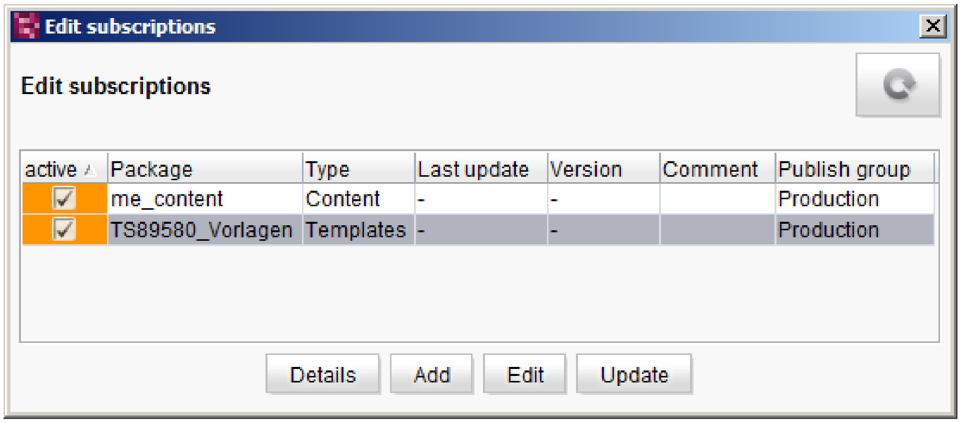
Clicking the **Cancel** button closes the dialog; any changes made are not applied.

5.6.4.5 Subscription is created

Once existing configurations have been made (as explained in sections 5.6.4.1 to 5.6.4.4), the subscription is initially shown in an overview (see Figure 5-33: Edit subscriptions) with orange highlighting (see section 5.6.6, page 93 for color coding subscriptions) and initially created using the **Update** button.

5.6.5 Subscription menu item - Edit subscription

The "Edit subscription" menu item opens the "Edit subscription" dialog box. A list of all packages subscribed to for the project is shown in this window.



active	Package	Type	Last update	Version	Comment	Publish group
☑	me_content	Content	-	-		Production
☑	TS89580_Vorlagen	Templates	-	-		Production

Figure 5-33: Edit subscriptions

Active: This checkbox is identical to the "Subscription active" box in the "Creating a subscription for **package** 'xyz'" dialog. If it is **checked**, then the subscription for the corresponding package is active and can be updated if a new package version is available (orange marking). If the checkbox is **unchecked**, then the subscription can no longer be updated (starting from the master project, see section 5.6.3, page 85). The status in the "Active" column can be changed in this view by clicking on the checkbox. Refer to section 5.6.6, page 93 and section 5.6.1, page 75 for subscription color coding.

Package: Unique package name.

Type: Specifies the type of the package (content package or template package – see section 2.2.1)

Last update: Date and time of the last subscription update in the target project using a new package version. If no entry has been made there, then an import has not yet taken place in the target project.

Version: Both the unique version number (assigned by the system) and the self-defined version number during package creation are shown here. The number assigned by the system is in parentheses here (e.g. "MyVersionA(No.12)"). If there is no entry present here, then there is not yet a package version for this package in the specified publication group.

Comment: Optional comment on the package version.

Publication group: Each subscription is concluded for precisely one publication group. Then a package version can be imported only if the subscription is active *and* the package has been marked as "Available" for the specified publication group.

Clicking the **Details** button opens the "Detailed info: project/package" dialog box. The window corresponds to the detailed information for the subscription from the "Package store overview" (see section 5.6.1.1, page 78). The window is used only for information; the displayed values cannot be modified.

Clicking the **Add** button opens the "Select package" dialog box; a new subscription is added to those already present in the list. The sequence is the same as for the "Create subscription" menu item (see section 5.6.4, page 86).

Clicking the **Edit** button opens the "Edit subscription for 'PackageName' package" dialog box. All of the settings for the subscription are defined in this dialog box (see section 5.6.4.2, page 87).

Clicking the **Update** button can be used to update a subscription directly from the "Edit subscription" menu item. The specific procedure for updating a subscription is described in section 5.6.6, page 93.

Subscriptions are also initially created using this button.

5.6.6 Subscription menu item - Update subscription

The "Update subscription" menu item is needed only for *manually updating* (see section 5.6.4.2, page 87) a subscription in a target project. However, all of the subscriptions can be updated this way depending on whether a manual or automatic update has been configured in the subscription or whether a subscription is marked as active or inactive. Thus, this function is used for updating using what is known as the pull process (also see section 2.2.5.1, page 14).

If a subscription is *active* and set to *automatic Update,*then the "Update subscriptions" button is normally not needed. In the event of an automatic update, the import is initiated from the master project by publishing a new package version (see section 5.6.3, page 85). However, if an error occurs during automatic updating in a target project, the update can be repeated easily using a manual update in the target project.

If a subscription is, in fact, set to *automatic update*, but was in the *inactive* state at the time of a new package publication, then the update is not carried out automatically. In this case, the subscription is marked as "Not up-to-date" and has to be updated manually.

If a subscription is set to *manual update*, the update always has to take place in the target project. The *active* or *inactive* status is not relevant for a manual update.

The "Update subscription" menu item opens the "Edit subscriptions" dialog box (see Figure 5-33). The window should already be familiar from the "Edit subscriptions" menu item (see section 5.6.5, page 91). Only the "Update" button is relevant for manually updating a subscription at this point.

Before updating, the package dependencies should be checked (see section 2.2.2, page 9). Dependencies on ***template packages*** *are defined in the package properties. These dependencies are checked automatically. If the dependent template packages are not updated or not updated in the right order, then the update is canceled and a corresponding error message is displayed.*

Optional dependencies on other **content packages** are shown in the "Detail info" dialog box, which can be called up via the package overview. These dependencies are not checked automatically during updating. If the dependent content packages are not updated or are not updated in the correct order (1. importing the dependent content package, 2. importing the package containing the references to the dependent package), they can cause errors in the target project: For instance, if the referenced page and page reference are in different packages when updating page references. If, in this example, the package with the page reference were updated and then the package with the referenced page, this would cause an error in the target project. In order to resolve the error, the page reference in the target project has to be deleted and then the package with the page reference has to be updated again.

> ❗ *If the subscription is for a template package that contains objects from a database schema, then the database configuration in the target project's project properties has to be adjusted before updating (see section 5.10, page 135). Otherwise a corresponding error is output.*

Clicking the **Update** button initiates a manual update of a subscription starting from the target project (see section 2.2.5.1, page 14). Since updating a subscription is a sensitive step, a confirmation prompt appears before updating.

Affirming this confirmation prompt starts the subscription update. An update only makes sense for subscriptions that do not have an up-to-date status. The update status can easily be seen using the color coding for the subscriptions in the "Edit subscriptions" dialog box (see Figure 5-33).

The default setup for an update is:

- 🟧 (orange) – The subscription is currently not up-to-date. A new package version is available for the subscribed package and the configured publication group that can be imported into the target project.

> ❗ *The orange coding (or green for "up-to-date", see below) only refers to contents; the package's properties (see section 5.1.1.2, page 51) or even the subscription's properties (see section 5.6.4.2, page 87) can be modified in relation to the last update even though a "green" status is displayed.*

The log for the package import into the target project can be displayed as needed after running an update. All possible errors are listed in detail here.

Then click on the ⟳ icon on the right upper edge of the window in the "Edit subscriptions" window. The new color coding for the subscription is displayed only after updating the view. The subscription is in either the "red" or "green" status after updating.

From 🟧 (orange) to 🟩 (green) = subscription was updated successfully.

From 🟧 (orange) to 🟥 (red) = an error occurred during the update. The subscription is not in an updated state. In this case, the log for the import should be displayed and evaluated.

A special case for an update occurs if the color coding is orange but no package version is available for importing yet. This error can occur if a subscription has already been created even though no package version is exists, likely because there are not any package versions but also because there are no available package versions for the subscribed publication group. An error message is displayed in this case:

> *If a new package version has been imported into the target project successfully, the editing environment for the target project still shows an old view of the project. Therefore, the view should be updated using F5 or the* 🔄 *button after each import. Only after this are all contents contained in the package shown with a corresponding symbol in the project tree. The symbol is only visible if the option "Show symbols (metadata, packages, permissions)" is active in the "View" menu item.*

For additional information on color coding subscriptions see section 5.6.1 starting from page 75.

5.6.7 Publication groups menu item

The "Publication groups" menu item simplifies the process of publishing and importing packages in complex operating environments for users (see section 2.2.4, page 13). For instance, by separating into three publication groups, Development, Production and Test, packages can be published in a test environment first and only then be implemented in a production environment as a tested, stable package version.

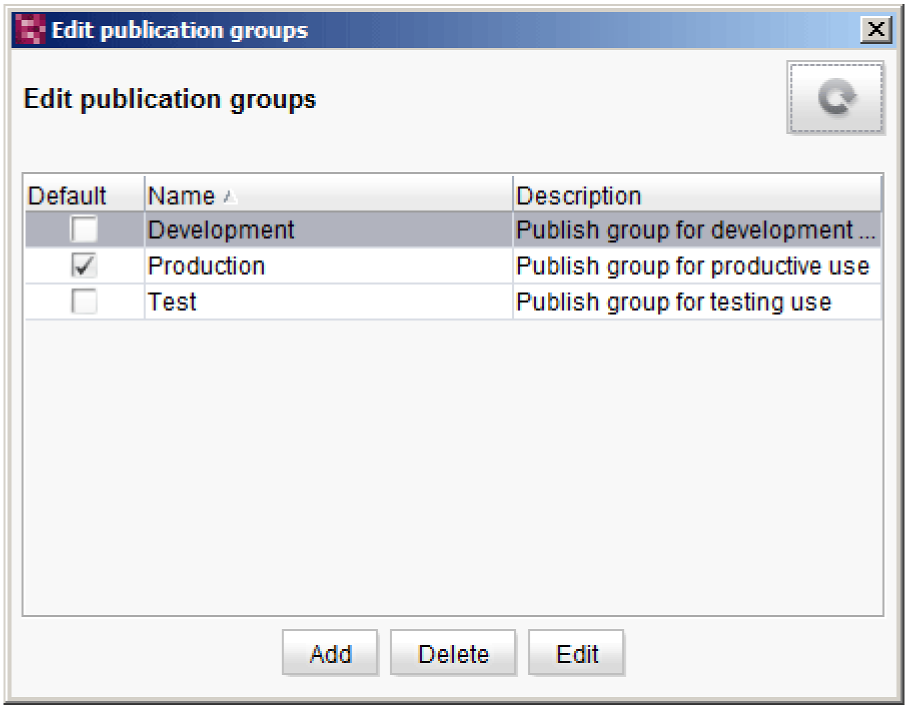
The publication groups are defined server-wide and thus are available in both master projects and target projects. Thus there are two different implementation areas for working with publication groups:

Publication groups in a source project: Which publication groups a package version is to be available for is assigned to each new package version during creation. (see section 5.1.2.1, page 61). The package versions can then be published for all of the publication groups or just for individual available ones. They are then ready for import into target projects.

Publication groups in a target project: Each subscription is concluded for exactly one publication group (see section 5.6.4.2, page 87). Thus, the most up-to-date package version available for this publication group is always imported into the target project. If, for instance, a subscription is concluded for the "Test" publication group, then only the most up-to-date package version made available to the "Test" publication group is imported.

5.6.7.1 Editing a publication group

The "Publication groups" menu item opens the "Edit publication groups" dialog box:



Edit publication groups

Default	Name	Description
☐	Development	Publish group for development ...
☑	Production	Publish group for productive use
☐	Test	Publish group for testing use

Figure 5-34: Editing publication groups - Overview

All publication groups present on the server are shown here in a table with the following information:

Default: The checked checkbox indicates the default (server-wide) publication group. This is selected by default when creating a subscription (section 5.6.4.2, page 87, "Publication group"). Exactly one publication group has to be defined as the default group at all times. This publication group cannot be deleted without first selecting a new publication group as the default group.

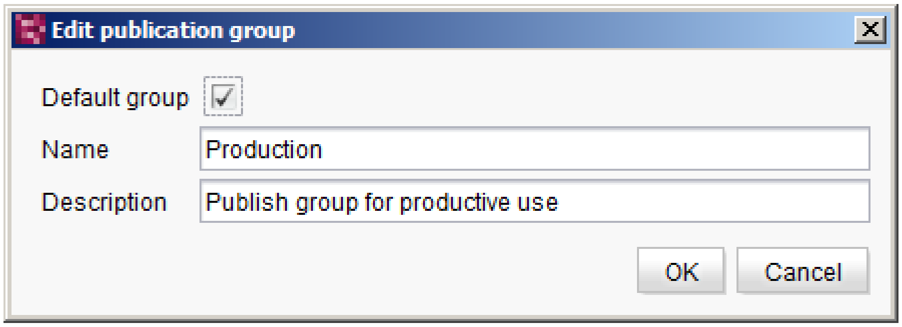
Name: Unique name for the publication group.

Description: Optional description of the publication group.

Clicking the **Add** button opens the "Create new publication group" dialog box. The rest of the process is described under the "Add publication group" menu item (see section 5.6.7.2, page 98).

Clicking the **Delete** button deletes a publication group. The rest of the process is described under the "Delete publication group" menu item (see section 5.6.7.3, page 99).

Clicking the **Edit** button opens the "Edit publication group" dialog box. The publication group highlighted in the table can be edited here.



Default group ☑
Name: Production
Description: Publish group for productive use

Figure 5-35: Editing a publication group

The information from the "Edit publication group" dialog box can be edited at this point for the selected publication group.

Default group: If this checkbox is checked, then this publication group is set as the default group. Exactly one publication group has to be defined as the default group at all times.

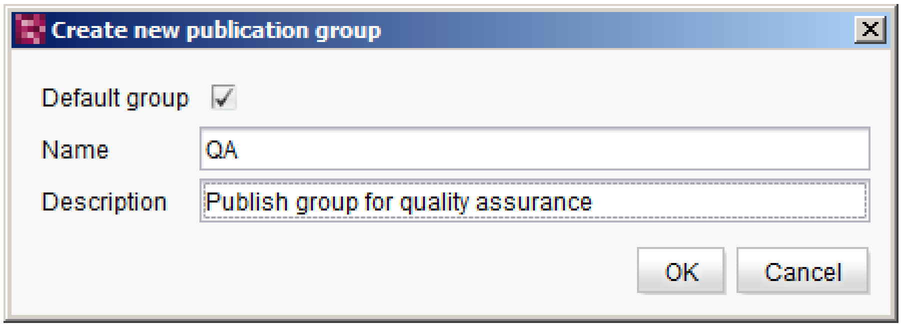
Name: A new name for the publication group can be specified in this field. Existing subscriptions under the publication group's old name remain intact and are now concluded for the new publication group name automatically. Thus, if needed, the publication group name only has to be changed here; manual adjustments at other points are not needed.

Description: A new optional description can be specified in this field.

Clicking **OK** confirms the changes and closes the window.

5.6.7.2 Adding a publication group

Clicking the **Add** button in the "Edit publication groups" opens the "Create new publication group" dialog box.



Create new publication group

Default group ☑

Name: QA

Description: Publish group for quality assurance

Figure 5-36: Create new publication group.

Default group: If this checkbox is checked, then this new publication group is created as the default group. The previous default group then loses this status since only one publication group at a time can be defined as the default group.

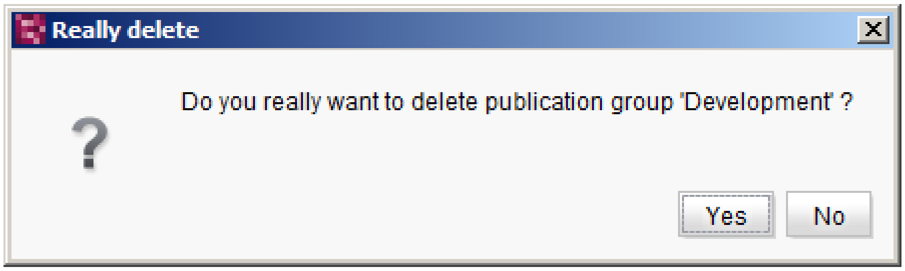
Name: Unique name for the new publication group. The new group cannot be added if the desired name is already assigned. The "Name" label is marked in red at this point in order to show the source of the error and, at the same time, the **OK** button is not active. Thus it is not possible to input two publication groups with the same name.

Description: Optional description of the new publication group.

Clicking the **OK** button creates the new publication group. It then appears in the "Edit publication groups" dialog box.

5.6.7.3 Deleting a publication group

A previously highlighted publication group can be deleted by clicking the **Delete** button in the "Edit publication groups" dialog box. In order to prevent a publication group from being deleted accidentally, a confirmation prompt is called up before it is permanently deleted.



Do you really want to delete publication group 'Development' ?

Figure 5-37: Deleting a publication group - Confirmation prompt

Clicking the **Yes** button deletes the publication group.

Clicking the **No** button cancels the dialog; the publication group is not deleted.

Publication groups can be deleted only if they are not used in any subscriptions.

A publication group that has been defined as the default group cannot be deleted. If a default group is to be deleted, a new publication group has to be defined as the default group in the "Edit publication group" dialog box first.

5.6.8 Combining package and target project contents

5.6.8.1 General

Contents can be transferred from a master project to multiple target projects using a package store. To do so, objects from a package are imported into the respective target project. In the target project, the package contents mix with the contents already present in the target project in the process. Thus, for instance, a page from the page store is maintained directly in the target project, however, another page is maintained in the master project and just imported into the target project. For most content, this combination of package and target project contents is possible without issue if certain rules are followed (e.g. dependencies). However, structures that normally cannot be created by themselves in a target project can also be combined using the package store, such as an individual section (see the following section of this document).

5.6.8.2 Combining sections

In the target projects, contents that are imported from a package can be supplemented by adding individual content, such as adding any number of sections to an imported page from a package. The "Modifiable" checkbox has to be checked in the subscription (see Figure 5-29) and in the package settings for this. This setting grants write permission to imported objects for the target project.

For instance, company-wide uniform terms and conditions pages can be distributed to individual subsidiaries using the package store. These pages and sections can then be supplemented within the subsidiaries by adding additional company-specific sections that are included in the target projects (subsidiaries) but not in the package. The general portion of the contents, the terms and conditions pages in this example, is maintained using the package store and updated; the specific sections are added in the target projects and maintained there as well. Sections inserted into the target projects remain intact when updating the subscription. If the order of the sections changes in the master project, then this can also affect the order of sections in the target project (see section 5.6.8.3, page 101).

Package contents can also certainly be reduced instead of being expanded. The package contents to be imported are simply restricted in the subscription for this purpose (see 5.6.4.3, page 89).

5.6.8.3 Order when importing objects into target projects

The order in which the objects in the master project are present in the object chain is taken into account as well when importing objects (such as sections) into target projects. This has to be adhered to to the greatest extent possible during the initial import into a target project as well as when importing modified objects, as well as in cases where object chains in a target project are expanded (see section 5.6.8.2, page 100).

Changes to imported objects in a target project can only be made if the "Modifiable" checkbox for subscriptions has been checked (see section 5.6.4.2, page 87). This setting also affects the order when importing objects:

If a subscription is marked as **"Not modifiable"**, the contents from the master project cannot be modified in the target project (no write permission in the target projects). Change authority is in the master project in this case. This setting affects the import order of objects in the target project. If the order – such as the sections of a page – in a master project changes, then this modified order is also applied in the target project when updating a subscription. This applies to both the first roll-out of contents into a target project as well as updating content that already exists into the target project.

If a subscription is marked as **"Modifiable"**, the contents from the master project can be modified in the target project. The content editors in the target project can add additional objects to imported package contents (such as a new section for an imported page) and modify the order of objects in the target project as well. These changes are not normally lost when updating a subscription again. Therefore, after initially importing package contents, the following applies:

- New objects added to already imported package contents in a target project (such as a new section to an imported page) remain intact when updating package contents in the target project.
- New objects added to existing package content in a master project (such as a new section added to a page that is already part of a package) are applied to the target project. Sorting into existing package contents occurs following specific rules in the process:

  The link to the previous object ("predecessor") has priority when sorting objects from the master project in the target project. This means that:

  - If a predecessor and a successor exist, then the new object is inserted after the predecessor.
  - If only a predecessor exists, it is inserted after the predecessor.
  - If only a successor exists, it is inserted before the successor.

Previously, objects were more closely linked with the subsequent object ("successor") if a successor was available.

The order defined for package contents in the target project (such as by the initial roll-out or resorting package contents in the target project) remains intact even if the package contents are resorted in the master project and are rolled out again.

Example 1 – Initial import of package contents into target projects:

A page with 3 sections (1, 2, 3) in the following order is contained in the master project:

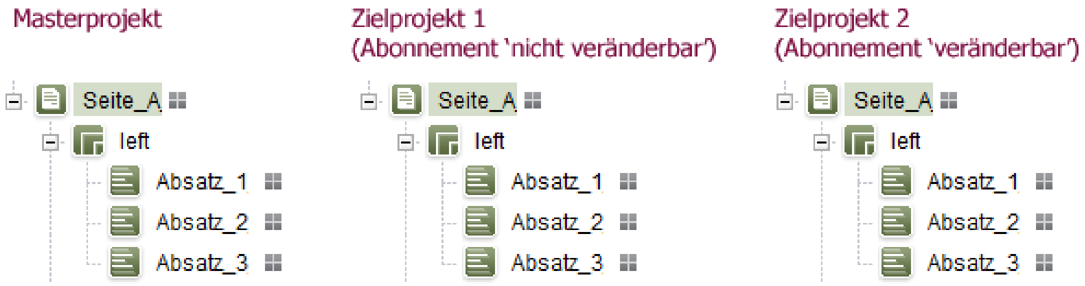


Figure 5-38: Example 1 - Page with three sections

If a package with these objects is rolled out to the target project, then the order of the sections is retained. This applies to the initial import into the target projects (regardless of whether the subscription is modifiable or not).

Example 2 – Updating package contents (without modification in the target project):

Now additional sections (a, b, c, d) are added to the page.

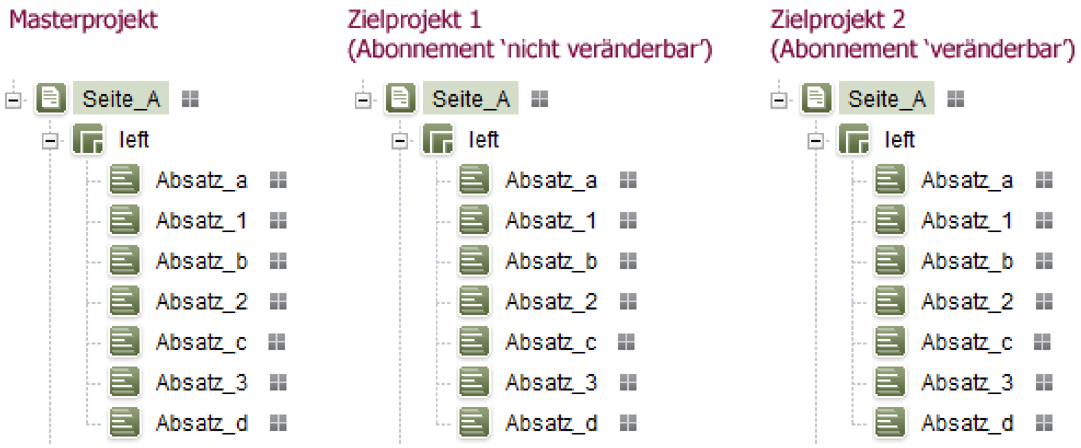


Figure 5-39: Example 2 - Inserting new sections

The order from the master project (a, 1, b, 2, c, 3, d) is applied to both target projects when updating. For target project 1, this is the standard behavior since the master project's order is retained. For target project 2, the behavior only takes effect because no changes to the package contents have taken place.

Example 3 – Updating package contents (in the event of a change in the target project):

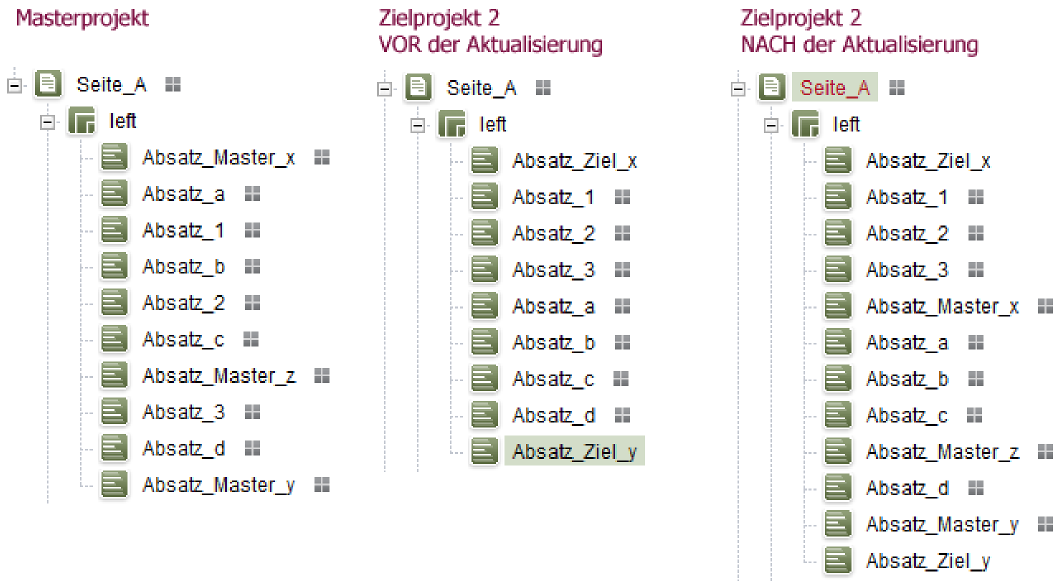


Figure 5-40: Example 3 - Combined sections in the target project

Three sections are added to the page in the master project (Absatz_Master_x through Absatz_Master_z). In contrast to the second example, this time changes are made to the page before the updated package is rolled out in the target project. These changes originate from the master project (only possible if the package contents in the subscription are marked as "Modifiable").

- The order of imported sections in the target project is changed by hand (1, 2, 3, a, b, c, d).
- Two new sections are inserted at the first and last positions (Ziel_x, Ziel_y)

After the renewed update of the package contents, the differences with example 2 are easy to see:

- The modified order of the sections (1, 2, 3, a, b, c, d) defined in the target project for the package contents remains intact even if the sections are arranged differently in the master project (a, 1, b, 2, c, 3, d)
- The new sections from the master project are inserted into the target project's existing contents based on the following rules:
  - Absatz_Master_x only has a successor (Absatz_a) and thus is inserted in the target project before the successor
  - Absatz_Master_y only has a predecessor (Absatz_d) and thus is inserted into the target project after the predecessor

  - Absatz_Master_z has a predecessor (Absatz_c) and a successor (Absatz_3) and is inserted in the target project after the predecessor since the predecessor is given priority.
- The new sections from the target project remain intact.

5.7 CorporateContent content menu in the stores

The "CorporateContent" context menu provides some functions for editing packages directly at objects in the project tree. It is called up by right-clicking directly on an object or a node from the project tree. The "CorporateContent" functionality is located under the corresponding menu entry in the context menu. The "CorporateContent" context menu is divided into five submenu items described in the following sections.

▪ Starting adding to a package	(section 5.7.1, page 106)
▪ Removing from a package	(section 5.7.2, page 107)
▪ Removing a package relationship	(section 5.7.3, page 108)
▪ Change status (unmodified/modified/locked)	(section 5.7.4, page 109)
▪ Newly integrating an original	(section 5.7.5, page 111)

If the submenu items are shown in a gray font instead of black then the specific function is not available on the highlighted object, such as on the store's roots.

5.7.1 Starting adding to a package (master project)

A node or object can be added directly to an existing package using the menu entry "*Start adding to package"*. This function is only available in master projects if the selected object is not already part of another package.

If no package is open when the menu entry is called up, then a package selection dialog for opening a specific package appears first. Only packages with the suitable package type are listed in this selection dialog. Then the selected object is displayed in the SiteArchitect AppCenter area as a dependency chart.

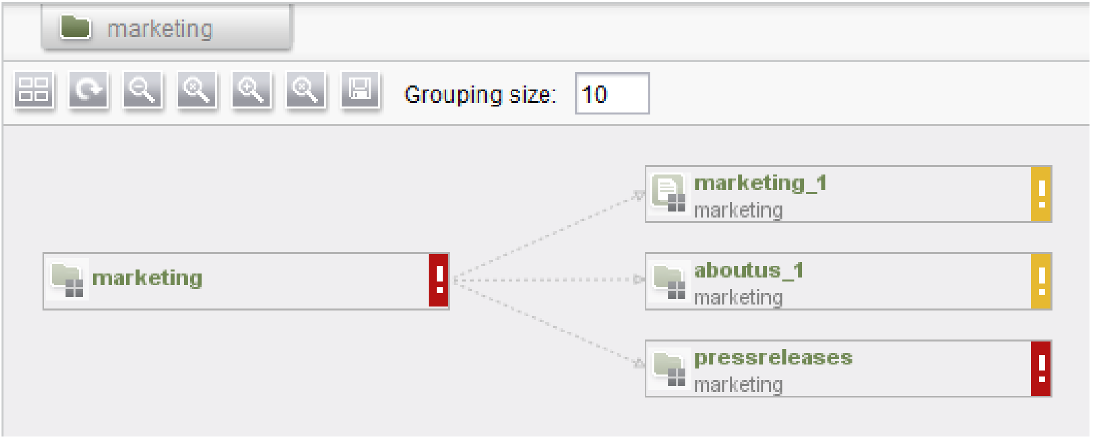


Figure 5-41: Adding an object - Dependency chart

Right-clicking the object in the dependency chart causes another context menu to appear. The selected object is added to the package using the entry *"Add to package 'xyz'"*.

> *Adding to a package is only possible if a package of suitable type is loaded. (Content package ↔template package)*

> *If the context menu is called up at a folder, then all of the lower level objects are added to the package. If the folder contains objects already integrated into another package then those objects are not added to the new package.*

5.7.2 Removing from a package (master project)

A node or object can be removed directly from a package using the "Remove from package" menu item. The "Remove from package" function is, of course, only available to objects that are already part of a package. Clicking the menu item opens a confirmation prompt.

Affirming the confirmation prompt with **Yes** removes the highlighted element from the package and closes the window. A package symbol is no longer displayed after the object name in the tree view and the highlighted object is no longer part of the package.

Clicking the **No** button cancels the process. The object is not removed from the project and the window is closed.

If an object is removed from a package, then the namespace extension remains, but the package symbol after the name indicating assignment to a package disappears. The object can now be added to a new package. In this case, the namespace extension also changes (a @ with the name of the new package is appended) and a symbol once again appears after the name in the project tree.

It is certainly possible to disable namespace extension (see section 5.1.1.3.1, page 58). In this case, the reference name remains unchanged when removing the object.

5.7.3 Removing a package relationship (target project)

While the first two context menu items are only relevant for source projects, i.e. for projects where packages are created, the third "Remove package relationship" menu item is used in target projects. The menu item can be carried out on all subscribed objects that have been imported into a target project from a package. These objects are shown in the target project with a blue dot and a package symbol after the name in the project tree.

Clicking the menu item removes the package relationship for an imported object. This means an object's relationship to a package is removed. This makes it possible to import objects from one package and to modify them in a target project even though write protection has been defined for the subscription.

If the *Overwrite identically named objects when importing in a target project* package property is enabled, then changes made are overwritten during the next update. If this option is disabled, then the object is created again in the target project as a copy during the next update. The modified object continues to remain intact.

Permission to remove a package relationship is only available to project administrators.

If a subscribed object's package relationship is removed, the package symbol after the object name in the project tree disappears. However, the namespace extension remains intact. As a result, no modifications have to be made in the referenced node. If namespace extension is disabled, then the reference name remains unchanged.

> *The "Remove package relationship" context menu function only removes the currently highlighted object – not any subordinate objects.*

5.7.4 Change status (target project)

The change status for a node or object can be set in this area. This option is only available in target projects and only for objects that are already part of a package. The options for the change status are only active if a package is marked as "Modifiable"; otherwise the corresponding options are grayed out.

The status values set in the target project are required for conflict handling when importing the package (see section 5.6.4.2, page 87, "Conflict handling" option). If, for example, changes are made to an imported page, a conflict can be triggered during the next subscription update by setting the "Modified" changes status. Conflict handling always depends on the status value set here:

- **Unmodified:** This status is set as the default for each object that is content for a package. The object is overwritten with the contents from the package the next time the package is updated. A conflict cannot occur with this setting.

- **Modified:** Setting this value triggers a conflict when the package is updated regardless of whether or not the package version has changed. Subsequent actions during conflict handling depend on the subscription's conflict settings (see section 5.6.4.2, page 87).
    - **Overwrite conflict handling:** The conflict is resolved by overwriting the object modified in the target project with the object from the package version imported during the update (this can be a new version with contents that have been modified in the master project or the same version that has already been imported). Changes made to the object in the target project are lost. After overwriting, the object in the target project matches the object from the master project.
    - **Cancel conflict handling:** The conflict is resolved by canceling the update to the subscription with an error message. No objects are updated.
    - **Copy conflict handling:** The conflict is resolved by creating a copy for the object being modified in the target project before the object is imported from the package version during updating. Numbering is appended to the object's reference name.

The original object from the target project remains intact; however, it is removed from the package link automatically (also refer to 5.7.3, page 108).

- **Locked:** The object is locked for updating with this setting; this means it is explicitly excluded from the subscription update. A copy of the object is created if a new package version is imported into the target project. The modified object remains intact in the target project, but it is removed from the package link automatically (also refer to 5.7.3, page 108). The new object is imported into the target project as a copy.

> ! *A change status can be set only if the package is marked as "Modifiable", otherwise the corresponding options are grayed out.*

The results of conflict handling and the change status in brief:

Change status	**Conflict handling**	Result
Unmodified	All	Only when modifying the package version: The object is updated; changes are lost.
Modified	Overwrite	When updating with a package version that is new or has been imported already: The object in the target project is updated with the contents from the master project; changes made to the object in the target project are lost.
Modified	Cancel	When updating using a package version that is new or has been imported already: The import is canceled; the object is not updated; changes from the target project remain intact.
Modified	Copy	When updating with a package version that is new or has been imported already: The object in the target project is removed from the package relationship and changes remain intact; a new object is created as a copy from the master project.

Locked	All	Only when changing the package version: The object in the target project is removed from the package relationship and changes remain intact; the new object is created as a copy from the master project.

The change status is required for such tasks as conflict handling when importing contents into different project languages (see section 5.9.2.1.2, page 126). If, for instance, a master project contains the project language English but the target project has the languages German and English, then the English contents are imported into the target project language only. The English contents then have to be translated into the German target project language. In this case, the change status for the translated pages should be set to "Modified" or "Locked". Otherwise the contents that have already been translated are overwritten again during the next subscription update.

If no change status is set for the object, then the changes are overwritten in the event of a subscription update.

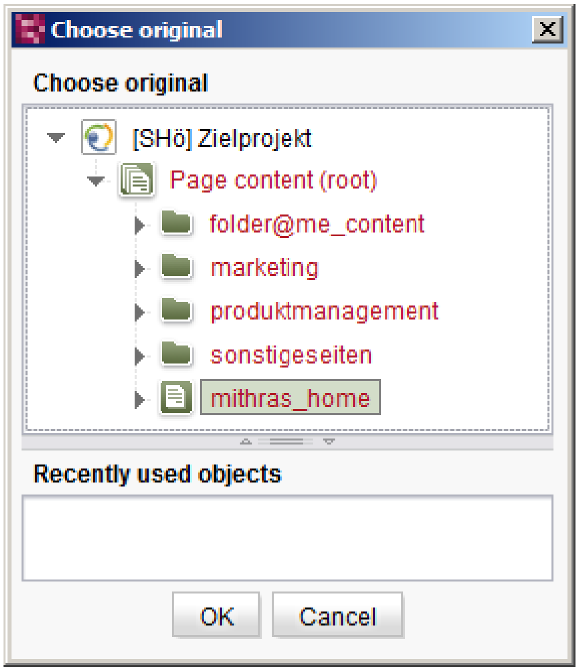
5.7.5 Reintegrating an original (target project)

In contrast to the "Start adding to package" (see section 5.7.1, page 106) and "Remove from package" (see section 5.7.2, page 107) context menu items, this function is available *only* in target projects that have subscriptions and only for objects that have a package link. The "Reintegrate original" function removes the object node where it was called from the package link and integrates a new object node into the package in its place. When doing so, the object should be an object that was part of this package previously but that does not currently have a package link.

The object node newly integrated into the package is selected from a list of all of the target project's objects. The combo box is limited by only displaying the store where the context menu was called (see Figure 5-42).

The "original" has to be compatible with the object node that is removed from the package link. This means, it has to be the same type of object node, such as a page from the page store based on identical templates.

The object node selection is not reviewed automatically, but rather is the responsibility of the editor. Removing or adding a package link is a sensitive action. If the wrong object is "reintegrated", like an object that was never a part of the respective package, this can lead to errors in the target project since the page and section templates do not fit the new object.


Choose original

Choose original

[SHö] Zielprojekt
Page content (root)
folder@me_content
marketing
produktmanagement
sonstigeseiten
mithras_home

Recently used objects

Figure 5-42: Selecting the original node to be reintegrated

A potential application area is integrating objects again, such as pages, after translation into a language not contained in the package. In order to protect this page from being overwritten again during translation, the change status for the section in the target project is set to "Modified" or "Locked" (see section 5.7.4, page 109). Then the configured "Copy" conflict handling in the subscription takes effect during a subscription update and creates a copy of the newly imported page. The changes in the translated page remain intact this way while the "old" page is removed from the package link. The page should be put back under package control after translation. The "Reintegrate object" function is required for this. The function is called on the currently imported page, i.e. the copy. The translated original page is then selected in the "Select original" list. The page is put back under package control after confirming the selection. The imported copy of the page loses the package link and can, if desired, be deleted from the target project (for translations, see section 5.9.2, page 124).

5.8 Transferring existing projects into package master projects

Generating a separate master project where package contents are managed exclusively is necessary in order to utilize the functionality of the package store. Each existing FirstSpirit project can take on the role of a master project and prepare package contents for import into other projects. Thus, a company subsidiary could take over maintaining the company display for its own web presence, thus becoming the master project for the corresponding package. Other subsidiaries would then subscribe to the company display package from this project. Afterwards, the master project continues to exist as a normal project.

Transferring an existing project to a package master project has to be planned carefully since temporary inconsistent intermittent states can occur due to the restructuring. It is also possible that references in the form and output tabs of templates and the like may have to be changed manually so that the master and target project work without error. This is because reference names can change due to package functionality. The process explained below (from section 5.8.1.1 through to and including section 5.8.1.9) should be followed precisely to avoid problems as part of the conversion.

5.8.1.1 Using the reference graph

As already explained in section 2.2.2, page 9, packages can only be imported successfully and used in a target project if they contain all required objects. In addition to the objects that the package developer explicitly adds to a package, there can also be dependent objects that are necessary for successfully working with the package in the target project. Content dependencies are automatically resolved in the so-called reference graph. This means that if objects from the page and site store are added to a package ("content package"), all dependent objects from the site store, page store and media store are taken over in the package ("implicit").

In contrast, dependencies to objects from the template store and the content store for this content package are not resolved automatically. Dependent objects from the template and content store, for example, a section template which is required for maintaining a section from the content package, have to be packed into their own package. The dependency between the content package and the template package is then defined in the content package. In order to be able to identify all dependences between contents package and template package, the reference graphs can also be used.

Reference graphs can be requested via the context menu **Tools / Show dependences**. The reference graphs for individual data records of the content store are queried via the context menu of the respective data record.

This function is only available to project administrators.

The tab in which open windows are located shows the dependencies of the object in the form of incoming and outgoing edges, for the current state (tab current state) and for the last state released (release state), as long as the project uses the release option:

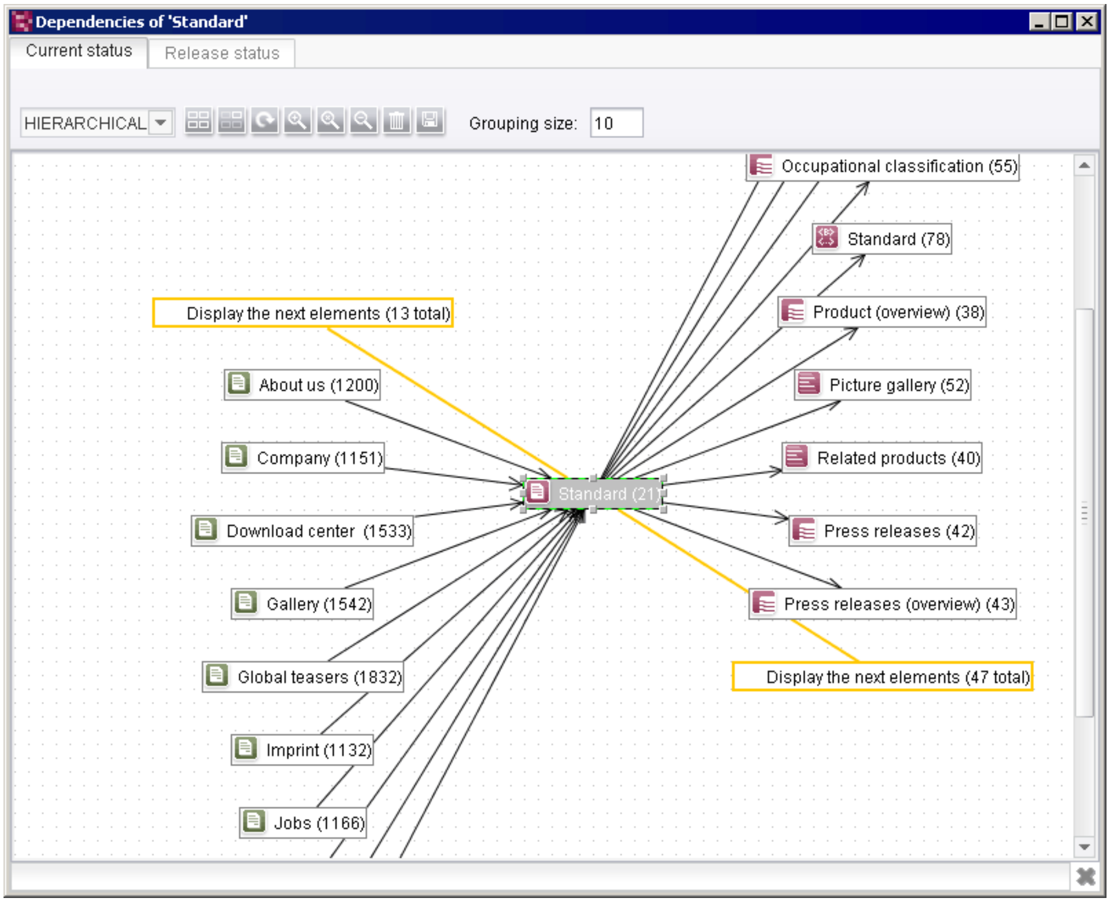


Figure 5-43: Display of dependencies via the reference graph

Each object for which a dependency exists is shown with ID and the object icon that belongs to it. By double-clicking on "Show the next element", additional dependent elements can be shown. By double-clicking on an element, the references to this object are also shown.

Additional information on reference graphs is in the FirstSpirit Manual for Editors.

5.8.1.2 Structuring the package contents

In order to simplify creation of a package, all content that is to later be integrated into one package is to be moved to a separate folder in the master project. This is not possible for all objects from the page, media, and template stores (not for objects from the Site Store). The folders later serve to aid in structuring the content in the target project. With the aid of the folders, it is more quickly visible what content was imported from a master project, and the content bundled in the folder is more clearly differentiated from the original content of the target project. All objects that are not bundled into packages in folders are added to the highest level in the respective store in the target project, and with that, the structure is lost. Likewise, structuring with folders is also beneficial for the clarity and transparency of the master project.

Alongside the explicitly added objects, there can also be objects implicitly added to the package if there are dependencies between objects (see section 2.2.2, page 9 and section 5.8.1.1, page 113). These implicitly added objects must be checked by the package creator and likewise stored in separate folders.

First, all templates needed must be stored in their own folders in the template store; this also applies to every subnode ("Pages", "Section templates", "Format templates", etc.). Media from the media store can be referenced within templates. These media objects which belong to a template, so-called technical media, can be integrated into a template package and contain, for example, JavaScript files (*.js), cascading style sheets (*.css) or graphic layout files (see section 2.2.1, page 9). In addition, in the media store, all technical media belonging to a template are collected in a folder. Non-technical media are integrated in content packages, and should, for example, also be stored in separate folders for this purpose.

> *Every object can only be contained in a maximum of one package.*

If, for example, technical media is needed in more than one package, then a second folder has to be created for this package, which contains a copy of this object, in the media store.

A requirement for a successful package creation is therefore always comprehensive project knowledge.

5.8.1.3 Limitation of image selection in templates

Due to the automatic display of dependencies within content packages, media files, for example, media data integrated via the input component DOM-Editor in a section, are implicitly added to the package as soon as the page is integrated with the corresponding section in the package. Under such circumstances, in this way, a very large number of implicitly referenced media is integrated into a package, which is available in the master project in different locations (for example, in different (sub) folders in the media store). This is both non-transparent and can lead to conflicts when importing the packages. A solution is offered by the limitation of the image selection option for the input component "DOM Editor" and "FS_Reference".

For the **"FS_Reference" input component**, `<FOLDER>` or `<SOURCES>` tags are used in the section or page template to set the limitation. Using the `<SOURCES>` tag makes it possible to limit the selection or display to defined folders (including subfolders). This involves a positive list; in other words, only the indicated folders are permitted. In order to allow a folder, a `FOLDER` tag is to be specified with the parameter `name` and a valid folder name.

If, alongside the image selection, the option to upload media is to be limited to a certain folder, then the `uploadfolder` attribute is also needed:

```
<FS_REFERENCE ...upload="yes" useLanguages="yes">

...

  <PROJECTS>
    <LOCAL name="." uploadFolder="test">
      <SOURCES>
        <FOLDER name="test" store="mediastore"/>
        <FOLDER name="test2" store="mediastore"/>
      </SOURCES>
    </LOCAL>
  </PROJECTS>
</FS_REFERENCE>
```

For the **"DOM Editor" input component**, the limitation is done via the link templates for internal links.

The image selection can in this way be limited to the folders which are also available in the package and were structured for a package (see section 5.8.1.2, page 115).

With this limitation, it should be noted that the folder names can be changed through the namespace extension, and in this way would have to be manually adapted later.

5.8.1.4 Limitation of template selection

Dependencies on templates are not resolved automatically. If input components which reference templates are used in a package, these dependencies have to be resolved manually.

The **input component "FS_LIST"** serves to integrate section templates within the Content Store or the Page Store. If this input component is to be used within the package store, the creator of the package has to take care to ensure that all referenced templates are integrated in the package and to create a dependent template package with the referenced section templates. Here as well, the selection of templates should be limited in order to increase transparency and to prevent user error.

For the "FS_LIST" input component, `<TEMPLATE>` or `<TEMPLATES>` tags are used in the section or page template to limit the template selection. The `<TEMPLATES>` tag makes it possible to limit the selection or the display to defined elements. This involves a positive list, in other words, only the indicated elements are permitted. In order to allow a template, a `TEMPLATE` tag is to be specified with the parameter `name` and a valid template reference name. In the package store context, the limitation has to be done via the reference names. In doing so, the unique name must be given for every section template in a separate `TEMPLATE` tag.

```
<FS_LIST name="st_downloadareas" ...>
    <DATASOURCE type="inline" maxEntries="6" useLanguages="no">
      ...
      <TEMPLATES source="sectiontemplates">
        <TEMPLATE uid="downloadcenterarea"/>
        <TEMPLATE uid="teaserlistelement"/>
      </TEMPLATES>
    </DATASOURCE>
    ...
  </FS_LIST>
```

As is made clear in the example, in this case also, the namespace extension, which is created by integration in a package, must be taken into account. These changes have to be carried out manually. With changes made frequently, a script can be created which automates this process.

5.8.1.5 Preventing language-dependent structures in templates

In general, multilingualism of templates is not supported by the package store. As long as the packages contain unified languages from the master project and the target projects subscribed to, such language-dependent structures present no problems. Multilingualism in templates always leads to problems if a language used in the target project does not appear in the master project, and thus was also not implemented in the templates. If in such a project environment templates have to be exchanged via the package store, then it must absolutely be ensured that no multilingualism is implemented in the templates. An exact explanation is given in section 5.9.2.3, page 128.

5.8.1.6 Automatic conversion in the Page Store

With the transfer of an existing project into a package master project, the reference names change through the namespace extension (as long as the namespace extension is not deactivated, see section 5.1.1.3.1, page 58). Reference names with namespace extension likewise have to be modified at all locations at which they are referenced in the project. In the Page Store, these references are automatically adapted to the package contents.

If, for example, a link to an object from the Site Store is stored within a page, then this reference:

```
<CMS_LINK language="DE" linktemplate="Interner_Link.standard"
sitestoreref="pageref:thisPage" text="Dieser Verweis" type="Interner Link"/>
```

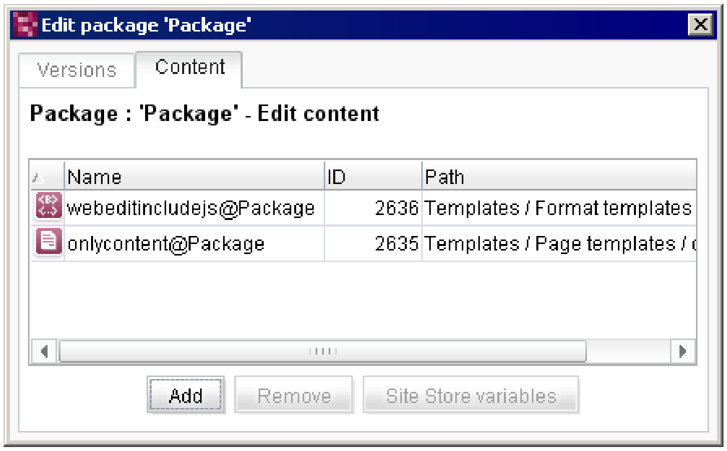
is automatically adapted to the namespace extension when adding the page "this page' to a content package:

```
<CMS_LINK language="DE" linktemplate="Interner_Link.standard"
sitestoreref="pageref:thisPage@package" text="Dieser Verweis" type="Interner
Link"/>
```

5.8.1.7 Manual conversion of templates

The behavior of the automatic conversion of references described in section 5.8.1.6 is not available in the template store. These must be adapted manually to the new package namespace extension.

If, for example, a page template (here: "onlycontent") is transferred into a package which references a link template (here: "WEBeditIncludeJS"), then the references within the template are automatically added to the template package:



Package : 'Package' - Edit content

Name	ID	Path
webeditincludejs@Package	2636	Templates / Format templates
onlycontent@Package	2635	Templates / Page templates / (

Figure 5-44: Package content when adding a page template with references

The references within the template are not adapted automatically. The page template "onlycontent" furthermore references:

```
$CMS_RENDER(template:"WEBeditIncludeJS")$
```

The references within templates have to therefore be adapted manually from the package developer:

```
$CMS_RENDER(template:"WEBeditIncludeJS@package")$
```

The adaptation has to be run for all applications of the link template in the master project. The applications in the project can best be determined via the reference graphs (see section 5.8.1.1, page 113). For the example mentioned, three references have to be manually edited later in three different page templates in the master project:

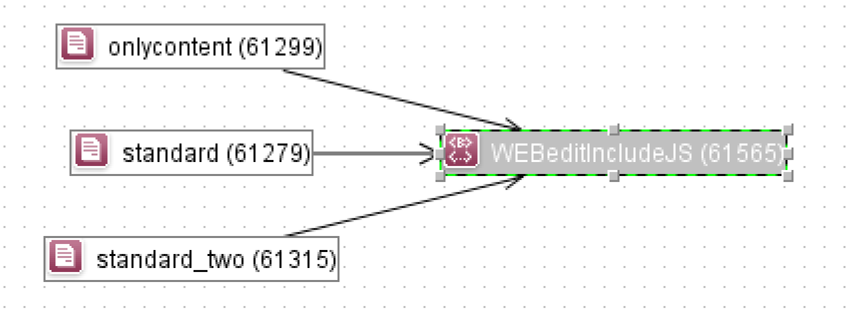


Figure 5-45: Dependencies of a format template

References in presentation channels: In templates, all references within the presentation channels that are indicated via the instruction $CMS_REF(...)$ or $CMS_RENDER(...)$ have to be edited manually. This affects the following object types:

- Media (media:...)
- Page references (pageref:...)
- Scripts (script:...)
- Templates (template:...)

Example:

```
src="$CMS_REF(media:"logo",abs:3)$"
```

has to be manually adapted after adding the medium "logo" to package "package".

```
src="$CMS_REF(media:"logo@package",abs:3)$"
```

References in the form area: Within the form area, references likewise have to be edited later. If, for example, a format template that is referenced within a DOM input component is added to a package, then the reference in the form area is manually adapted to the new reference name:

```
    <CMS_INPUT_DOM name="st_text" rows="8">
        <FORMATS>
          <TEMPLATE name="format@package"/>
        </FORMATS>
```

With standard format templates, the namespace extension has to be viewed critically. If references to standard format templates are changed, for example, „b@package“, these are also no longer recognized during an adaptation of the template `<TEMPLATE name="b@package"/>` within the input component. With that, for example, the assignment to the corresponding buttons in the DOM Editor (here: "Bold") is lost. There can be errors in the master and in the target project.

5.8.1.8 Manual conversion in the content store

As in the template store, references are not automatically converted in the content store. That means that, within the content store, all references to package contents have to be adapted manually.

If, for example, in an input component in the content store a link to an object from the site store is stored, then this reference:

```
<CMS_LINK language="DE" linktemplate="Interner_Link.standard"
sitestoreref="pageref:thisPage" text="Dieser Verweis" type="Interner Link"/>
```

is not automatically adapted to the namespace extension when adding the page "thisPage" to a content package. The namespace extensions have to be adapted manually by the template developer (compare to the example in section 5.8.1.6).

If the previous steps were carried out successfully, all requirements are fulfilled to transfer the existing project to a package master project. In the next step, the first package can be created in the new master project (section 5.1.1.1, page 49).

5.8.1.9 Checking the function in a test project

Creating and importing packages is a complex task. Before importing packages to a productive environment is used, the function is therefore to first be tested in a test project.

After the first package is created in the master project (section 5.1.1.1, page 49), the package properties were configured and the package content was added, and finally, a first package version was generated (section 5.1.2, page 60); then it must first be checked in the master project that the project still works correctly.

If implicit or explicit objects are added to a package, there is always extensive restructuring, for example through the name extension (section 5.1.1.3.1, page 58). If the name of a media object changes, the reference to the media file is also adapted in all pages, sections, templates, etc. For content packages, this restructuring is automatically adapted in the project via the reference graph (section 5.8.1.1, page 113). However, in individual cases, it can happen that references cannot automatically be resolved by the system or that the manual adaptation of the templates is faulty (see section 5.8.1.7, page 118). In this case, the master project no longer works as intended. If a media file is no longer referenced according to the namespace extension, then there will be errors when displaying the page.

If, after package creation, errors arise during generation in the master project, then the master project must first be repaired, for example, by changing the reference name. If the master project is functioning flawlessly, the package can first be imported into an "empty" target project. In the target project, as well, it is then checked whether the import was carried out properly and completely or whether possible required templates or referenced objects are missing in the package. If that is the case, these objects have to be added to the package and a new package version created and imported.

Only after this initial test is the master project to provide packages for the actual target project. Afterward, as well, there are to be comprehensive tests on every package version. These are not to be omitted (publication groups: section 2.2.4, page 13).

5.8.2 For similar projects

If multiple projects share the same content, the structure of a preconfigured project for the roll-out makes sense (here: division across multiple, similar target projects). In this roll-out project, a standard project structure and all required subscriptions can be centrally configured once. In connection with this, the project can be exported and is available as the base project for all target projects (for example, subsidiaries of a company). When importing the project, all required subscriptions in the project are created directly with it. In the case that all company subsidiaries would like to maintain their own website, but want to use the templates to create the pages and the entire, company-wide, unified company profile via a centrally administered package content, the use of a roll-out project makes sense.

5.8.3 Import / export

Exporting and importing via the ServerManager is also possible for package master projects and subscribed projects. These functions however have effects on the existing package and subscription structure.

5.8.3.1 Package master projects

! *If a package master project is exported and then re-imported, all previous package information gets lost. After importing the project, the symbols behind the object names in the project tree continue to be displayed and the namespace extensions are preserved (as long as these were not deactivated). However, the package information (as shown in Figure 5-26: Edit package – Package list) is no longer present. No more packages are displayed in the package overview. Thus the project is no longer a master project.*

Therefore, the existing, original master project should by no means be deleted. If it is, both the package information and the subscriptions in the target projects will be lost.

The only option for recovering the package information and thus the master project is a file system backup .

If subscriptions to contents of other projects in the package master project exist, these remain even after the import, but have to be manually updated (see the following section 5.8.3.2).

5.8.3.2 Subscribing projects

If a subscribing project is exported and then re-imported, the contents subscribed to from other projects are retained and continue to be displayed with a blue color coding behind the object name in the project tree. The subscriptions existing before the import are all set to the status "not up-to-date" and provided with an orange color coding, even if no version modification occurred in the master project before (see Figure 5-33: Edit subscriptions).

> ! *The subscriptions have to be updated manually after importing the target project.*

5.9 Corporate Content for developers

5.9.1 Individualization of the package contents in the target projects

Corporate Content can be used to import contents of a master project into different projects. In many cases, however, these contents are supposed to be displayed differently in the individual target projects. It is possible to make a direct intervention using the templates.

The layout in the target projects can be changed afterwards by directly modifying the templates. Here, the package contents must not be write-protected; that is, they are set as "modifiable "in the package version and in the subscription.

If templates from a template package in the target projects are modified, that can lead to problems. On the one hand, the project-specific modifications have to be redone after every update of the subscription; on the other hand, conflicts can arise when updating with a new package version, because innovations in the master project cannot be linked by force also with the modified states in the target project. One solution for these problems is an appropriate conflict resolution, which can be configured in the subscription (see section 5.6.4.2, page 87). Here, the "Copy" option has to be selected under "Conflict resolution"; this is used to create a copy of the modified template in the target project. This copy now has to be manually revised by the developer in the target project. Thus the modifications in the layout have to be done manually in the new template. If the old template is retained here, it can cause the project to stop functioning correctly.

Modifications to templates in the target projects should be carried out only in exceptional cases! The safe way to individualize contents in the target projects is to adapt them using structure variables.

5.9.2 Support for multiple languages

Since the implementation of FirstSpirit was designed very consistently for multilingual projects, these are also supported in the package store. However, the different languages do not have to be maintained in the master project; translations can also be made in the individual subsidiaries in the respective local language. In doing so, projects with a homogeneous language structure and projects with a heterogeneous language structure are differentiated.

5.9.2.1 Page contents

5.9.2.1.1 For projects with a homogeneous language structure

With a homogeneous language structure, the package supports the unifying quantities of all languages used in the projects subscribed to.

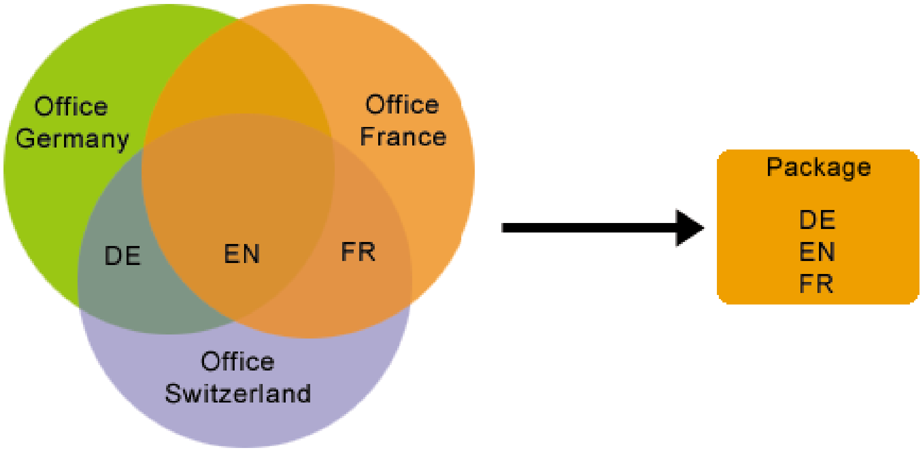


Figure 5-46: Packages with a homogeneous language structure

For example:

- German subsidiary: DE, EN
- French subsidiary: FR, EN
- Swiss subsidiary: DE, EN, FR

The package with a **homogeneous language structure** contains all three languages. Import into the target projects is thus uncomplicated, because each of the languages needed in the project is also contained in the package. If there are more languages in one package than are used in a target project, the extra languages are simply ignored in the target project. In the example above, the project of the Swiss subsidiary is the ideal candidate for the role of the master project.

5.9.2.1.2 For projects with a heterogeneous language structure

With a **heterogeneous language structure**, not all of the languages used in the projects subscribed to are also contained in the package.

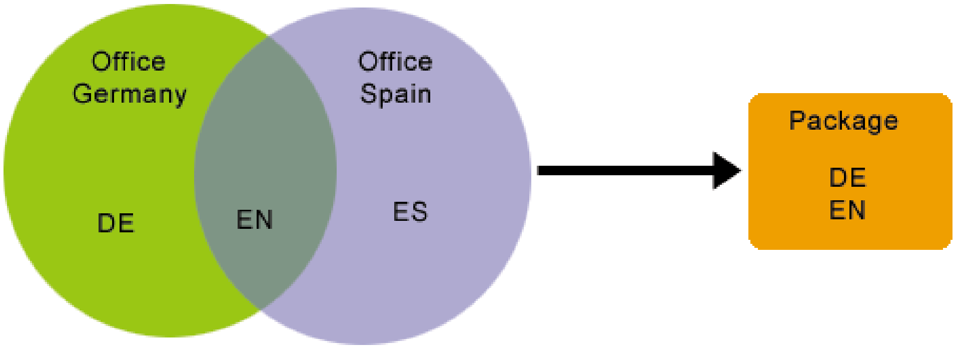


Figure 5-47: Packages with a heterogeneous language structure

For example:

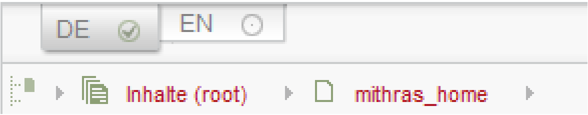
- German subsidiary: DE, EN
- Spanish subsidiary: ES, EN

Only the languages German (DE) and English (EN) are contained in the package. This means that when importing a package into the Spanish target project (ES), the Spanish-language content has to be specially translated. If this content is to be translated for the target project, that can be realized via a workflow which is started directly upon package import.

In addition, the following settings have to be configured:

1) First, the project settings in the target project with the untranslated language have to be configured. In **The ServerManager**, select the "Use master language" option for "**Language substitute**" in the project properties under the "Substitutions" item. The master language has to be a language contained in the package, such as English. If objects not existing in the proper language are imported into the project now, only the English language objects that will still have to be translated are imported.

2) The actual translation can be started after the initial import via a workflow. Using a "Translate new page" workflow, for example, the newly imported page can be sent to a translation office via XML export and then the translated result can be re-imported into the project. In this case, it is important whether or not the check marks on the language tabs (for the setting *"Page for this language completely translated*") are activated on pages of the Page Store.



- For all new pages that are initially imported into the project, the language option must be *deactivated*! The workflow should configure this setting for all new pages *before the import*. If the translation has been done, then the language option is *reactivated* for all new pages.
- For untranslated changes to a page already in the target project, the language option has to be *activated*. If the language option is deactivated, the contents are overwritten again in the next import.

5.9.2.2 Menu structures

All menu structures in a package, thus menu levels and page references, are taken over into the target projects from a package. There is a critical difference here between projects with a homogeneous language structure and projects with a heterogeneous language structure.

5.9.2.2.1 For projects with a homogeneous language structure

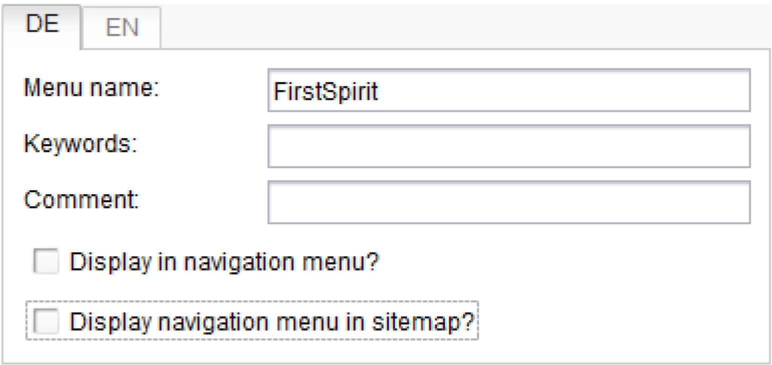
For projects with a homogeneous language structure, all menu structures contained in the package, including the language-dependent labels, are taken over for each language. If a menu level from the Site Store is integrated into a content package, the page references below the package and the accompanying pages from the Page Store are also added to the package. If the accompanying pages are moved from the Page Store into folders, only the referenced pages are taken over into the package with them, not the higher-level folders.

If there are more languages in one package than are used in a target project, the extra languages are simply ignored in the target project.

5.9.2.2.2 For projects with a heterogeneous language structure

For projects with a heterogeneous language structure, the same problems arise for menu structures as for page contents (see section 5.9.2.1.2, page 126). The target project supports languages for which there are menu structures in the package, but the respective menu labels are not translated.

In this case, a language substitution setting does not affect the master language available in the package. When importing menu structures of a package (only EN) into a target project (EN and DE), *no substitution* of the German menu names takes place. For the languages not included in the package (here, DE), the menu names are preassigned by the display name from the master language of the target project (here, EN). For projects with a heterogeneous language structure, the menu structures are not permitted to be displayed in either the navigation menu or the navigation overview for all languages that are in the target project, but not incorporated in the package. In the Site Store, therefore, the "Displays" setting at the folder level must be deactivated. This setting is automatically configured when importing the menu structures into a target project for every structure folder contained in the package for the unsupported language.



Menu name: FirstSpirit

Keywords:

Comment:

□ Display in navigation menu?

□ Display navigation menu in sitemap?

Figure 5-48: Display options at the folder level of the Site Store

After translating the labels, the boxes have to be manually checked again in order to make the navigations visible.

5.9.2.3 Templates

In general, Corporate Content does not cover multiple languages for templates. If templates are supposed to be exchanged via Corporate Content, it is imperative to ensure that the templates are not multilingual. Multiple languages always lead to problems if a language used in the project was not implemented in the templates, thus in target projects without a heterogeneous language structure.

5.9.2.3.1 Via common database access

One option for centrally maintaining language-dependent return values is to use a translation table in the Content Store.

Unlike the usual procedure for maintaining multilingual contents in the Content Store, here all languages are maintained via their own input fields. To do so, a column must be created for each individual target project language in the content source schema of the master project, and an input component has to be assigned to this column. The label of the individual input components is provided for only in the master language, in most cases "English". Now the language-dependent return values can be maintained centrally in the master project. Via a common database access, all target projects can gain (read) access to these language-dependent contents (see section 5.10, page 135).

If we proceed from the previous example of the combo box, the master project initially has two input components of the "text" type for the languages DE and EN. In the table, for each language included in the target projects, the language-dependent display value, such as "red", is assigned to a language-independent return value, such as "1". Now the language-independent return value "1" is all that is stored in the template. Then the language-dependent assignment is made for each language based on the translation table in the Content Store:

Example: Return value in the template "1" and key "DE" = Return value "red"

	DE	EN
1	Rot	Red
2	Blau	Blue

If a new language is added, for example, by a new Spanish subsidiary, the table schema in the master project has to be expanded by one column for ES and another input component of the "text" type has to be added. Then the table looks like this:

	DE	EN	ES
1	Rot	Red	ZERO
2	Blau	Blue	ZERO

Now the language-dependent return values for ES can be added in the master project. No more changes have to be carried out in the templates.

	DE	EN	ES
1	Rot	Red	Rojo
2	Blau	Blue	Azure

The master language of the target project must be in the package.

For all target projects, a common database layer has to be specified in the project settings in the ServerManager. For the database layer, the boxes for "No schema sync" and "Write-protected" must also be checked (cf. section 5.10.1).

Again, substitution of the label is possible only via a template change.

5.9.2.3.2 Local differences in the same language

In principle, it can also happen that conflicts arise when importing templates if the same language is used in both the package and the target project. While different countries can use a common language, for example English, there is nevertheless a number of aspects that can differ in the countries. A prominent example is local formatting differences, such as different **date** or **currency formats** in countries that otherwise have the same language.

Example:

- Date Germany: Tuesday 14.08.2001 16:47:48
- Date Switzerland: Tuesday 2001-08-14 16:47:58

When importing a package from a German master project into a "Swiss" target project, only the same language "DE" is recognized. Country-specific formats, however, are not taken into account here.

These problems can be circumvented by introducing a "new" language that takes such local differences into account; in the example, the new language "CH" would be introduced in the target project.

5.9.3 Using workflows and events

Within Corporate Content, "standard events" can be assigned to workflows. The assigned workflows are then carried out when the event occurs during or after the updating of a subscription in the target project (see section 5.6.4.4, page 90).

One way to apply this is to use a workflow to release all objects imported via a subscription. Since a workflow can always be started for one object only instead of for several objects simultaneously, a script is required to determine all nodes affected (see section 5.9.3.1, page 131).

So that both the workflow and the script can be carried out in the target project, both the workflow and the script also have to be in the target project.

5.9.3.1 Determining the nodes affected

Within a script in a workflow, one is in the WorkflowScriptContext.

First, the current session is required. This is obtained with

```
m_session = context.getSession();
```

Then get the ImportInfo object from the session:

```
m_importInfo = m_session.get("importInfo");
```

Finally, the UserService is required and the ImportInfo object is initialized:

```
m_userService = context.getUserService();
m_importInfo.setUserService(m_userService);
```

With the initialized ImportInfo object, now it is possible to determine the number of

- new nodes `(getNewNodeCount())`,
- modified nodes `(getUpdatedNodeCount())`,
- deleted nodes `(getRemovedNodeCount())`, and
- the nodes, at which a conflict arose `(getConflictNodeCount())`

The number determined is required in order to iterate across all nodes using a loop and to return nodes in index-related form.

```
NewNode = m_importInfo.getNewNode(index);
```

If, for example, the script is supposed to return the first new node, the call looks like this:

```
firstNewNode = m_importInfo.getNewNode(0);
```

Other operations can be carried out at the nodes determined using Access API.

Please refer to the API documentation for the complete syntax of ImportInfo.

After all operations have been carried out, the workflow has to be advanced by the script. To do so, use the method `doTransition`:

```
context.doTransition(NAME OF THE TRANSITION);
```

5.9.3.2 Exemplary workflow for the release

An exemplary workflow for the release of imported objects can be seen in the release workflow.

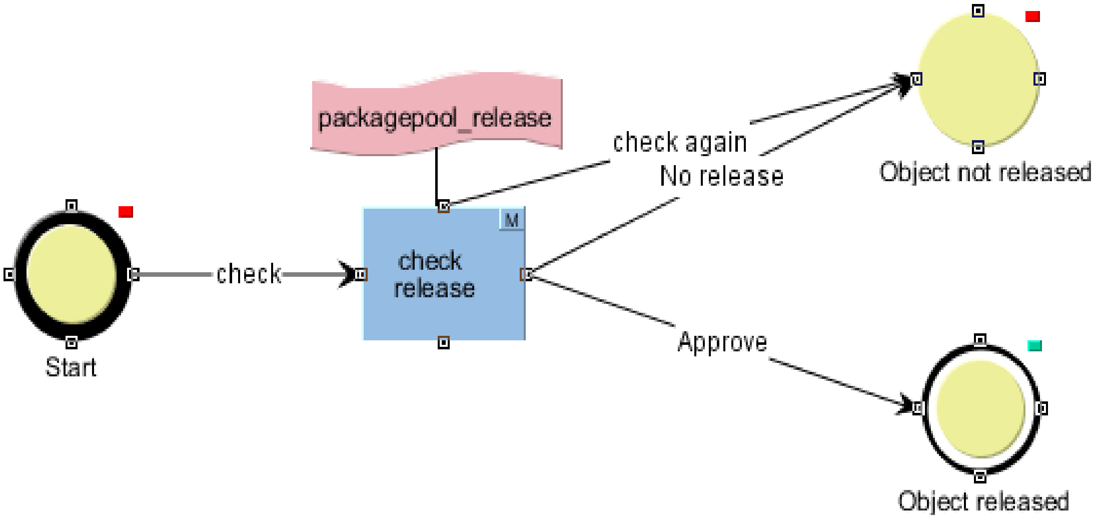
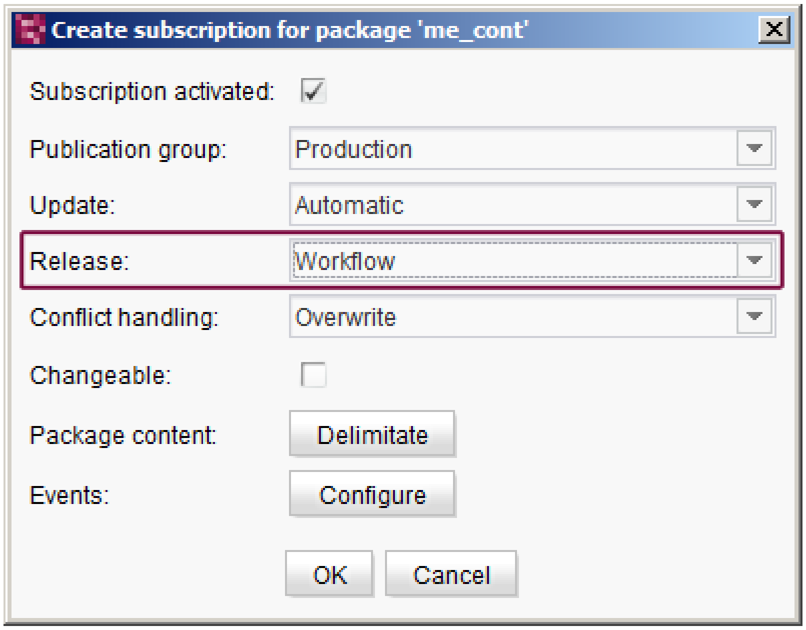


Figure 5-49: Release workflow

In order to use the release via a workflow in the release target project, the release has to be set in the subscription via a workflow (see also section 5.6.4.2, page 87):



Create subscription for package 'me_cont'

Subscription activated: ☑

Publication group: Production

Update: Automatic

Release: Workflow

Conflict handling: Overwrite

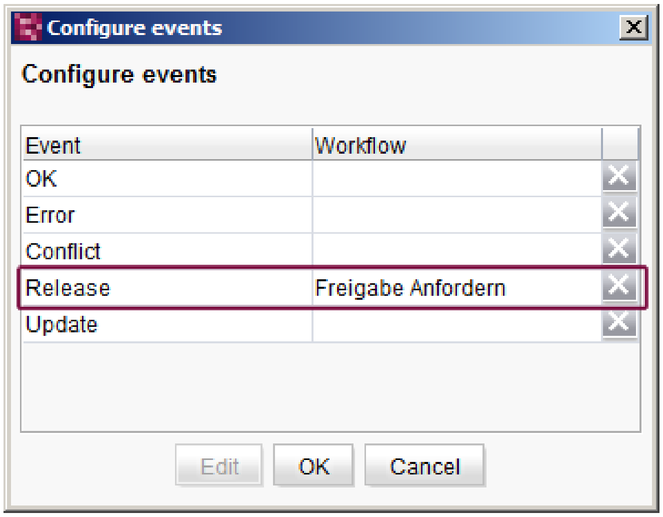
Changeable: □

Package content: Delimitate

Events: Configure

Figure 5-50: Setting the release in the subscription

In addition, under events, you have to use the **Configure** button to specify the "Release" event of the workflow specified under Figure 5-49 (see section 5.6.4.4, page 90):



Configure events

Configure events

Event	Workflow
OK	
Error	
Conflict	
Release	Freigabe Anfordern
Update	

Figure 5-51: Configuring events

If, in the release target project, the release is configured via a workflow, this is started as a context-free workflow as soon as there are new or updated nodes in the project. This means that the release does not take place in context-related form on an object in the project tree, but in context-free form via the task list.

If the editor advances the workflow with "reviewing", the script "packagePoolRelease" determines the number of the new or modified nodes in the target project. If there is at least one new or modified node, a list dialog opens in which the modified nodes are shown.

If the **release** is **granted** by confirming this dialog, all listed objects are released at once. (By double-clicking a node, the associated object can be displayed beforehand.)

If the **release** is **not granted** in this dialog, the listed objects are not released. However, they can be reviewed "again" in the task list (see Figure 5-49: Release workflow).

If the newly imported nodes have been released, an update of the stores should be carried out, after which the new or modified nodes are then shown as "released" (black font).

5.10 Shared database access

FirstSpirit has powerful mechanisms for connecting databases (see FirstSpirit Manual for Administrators). Within the editing environment, the connected databases are identified as content sources. The data records stored in the content sources can be integrated seamlessly into the web pages and (via the Page and Site Store) edited seamlessly in FirstSpirit without leaving the editing environment.

The tables that are displayed within the Content Store are only views of the database. To do so, it is first necessary to create a database schema in the FirstSpirit template store (new or generated from an existing database). Using a graphic editor, the project administrator can create the required tables in FirstSpirit SiteArchitect and place them in relationship to each other (or take them over from a connected database). For each table modeled within the schema, a table template can be generated (below the schema node). These table templates contain definitions of which input elements the editor can use later to enter the data into the corresponding tables or which input elements the editor can use to take over data of a reference table. In addition, the assignment of contents maintained via an input component to a database table in the physical database can be established using the "Mapping" tab.

Depending on the project administrator's settings for the configured database, the changes to a schema in SiteArchitect, such as adding a table to the physical database, can be applied ("Sync") or prevented ("No sync"). The contents maintained by the editors within the Content Store can also be written back to the database; alternatively, they can also not be written (write-protected). This is likewise dependent on the configuration.

For more information, see FirstSpirit Manual for Developers (Basics).

The following contents can be integrated into a **Template package** and distributed to other FirstSpirit projects via the package store:

- FirstSpirit database schemata
- FirstSpirit table templates
- FirstSpirit database queries

The following content can be integrated into a **Content package** and distributed to other FirstSpirit projects via the package store:

- Views of the database (nodes of the Content Store)
- Pages or page references that have a reference to a content source of the Content Store

The following applies here:

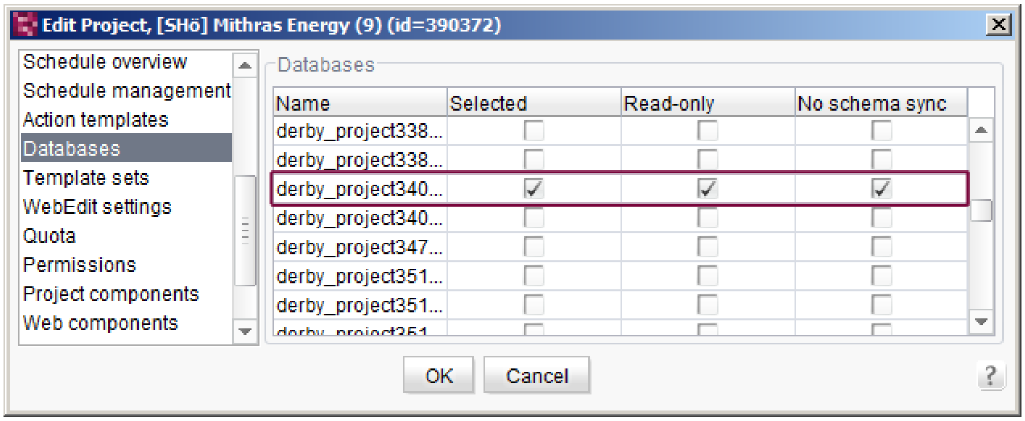
Shared access to the database (read-only): To exchange content via the package store, shared access must be configured in the project settings (ServerManager) of *all participating projects (master project and target projects).*

The package store supports distribution of database views (nodes of the Content Store) into multiple target projects for shared, read-only access to the corresponding database content. This means that during configuration of the corresponding database layer, the "Write-protected" and "No schema sync" boxes must be checked for the target projects. The configuration for joint use is described in the following sections (see section 5.10.1 starting on page 137 ff.).

Consider dependencies: If database views (nodes of the Content Store) are to be distributed via the package store to multiple target projects, it is first necessary to ensure that dependent objects, such as the corresponding database schemata, table templates and queries from the master project, are also part of the package (or a dependent package). Here, the sequence of adding can also be critical. If these dependencies are not considered, mistakes can appear when packing or importing a package. Example: A section template that contains a content list (FirstSpirit input component for selection and output of data records) is added to a template package. If the corresponding database schema was not added to the package earlier, an error occurs.

These dependencies cannot be removed automatically, as they can be for the content packages (see section 2.2.2), as this would have very far-reaching effects. Continuing with the example above, when adding the section template, for example, the schema and all table templates and table queries below it would become part of the package. However, this behavior is usually not desirable. Therefore, the package developer should give some thought in advance to making the package structure as effective as possible.

5.10.1 Configuring the target projects (read-only DB access)



Databases

Name	Selected	Read-only	No schema sync
derby_project338...	□	□	□
derby_project338...	□	□	□
derby_project340...	☑	☑	☑
derby_project340...	□	□	□
derby_project347...	□	□	□
derby_project351...	□	□	□
derby_project351...	□	□	□
derby_project351	□	□	□

Figure 5-52: Configuring a database layer in the target projects

First, the database layer of the master project (indicated by the red frame in the illustration) must be enabled under the "Databases" item. To do so, check the corresponding box in the "Selected" column.

For this database layer, the boxes for "No schema sync" and "Write-protected" must also be checked.

"No schema sync" defines that when importing a template package, the database tables are not recreated in the database.

Checking the box for "Write-protected" prevents shared write access from the target projects to the database. Read-only access to the database contents is then possible in all target projects (views of the database); however, changes to the database contents cannot be initiated from target projects.

For more detailed information about the "Multilingualism" use case with regard to shared database access via the package store, refer to section 5.9.2.3.1, page 129.

! *Changes to the database schema always have to be made in the master project, as the option "No schema sync" is not enabled here, and the changes have to be distributed to the target projects from there.*

! *Incompatible schema changes in the master project lead to problems in target projects, even if no subscription update has yet taken place!*

! *The master and target project should always use the same database schema.*

! *The following applies for multilingual projects: When transferring a schema to a package, the language structures of both the master project and the target project must be taken into account in the master project (see section 5.10.5.1, page 140 and section 5.10.5.2, page 141).*

5.10.2 For existing databases

If shared database access is to be implemented for projects with an already existing database and/or existing data records, some adjustments are necessary.

Assume for instance that a data record that references an object from the media store by name exists in the database. When importing a data record, the "test" media file, which is not yet part of a package, would be selected and referenced by "media:test" in the data record. If shared database access for multiple projects is to be implemented at this point, all of the referenced objects have to be available in one package as well. As soon as the "test" media file is added to a package, the name changes to "test@PackageName" (if namespace extension has not been disabled, see section 5.1.1.3.1, page 58). However, the existing reference in the data record continues to reference "media:test" with the result that the media file for this data record can no longer be found. In order for the media file to be displayed again when displaying the data record, the reference has to be configured to the new name ("media:test@PackageName") manually or automatically using a script.

All objects referenced in an existing database have to be available in the target projects. Therefore, it is recommended that all objects be made available to target projects using a package when sharing a database. Then the references are subsequently adjusted in the database. In this case, the **media selection limitation** explained previously in section 5.8.1.3, page 116 should be implemented for all of the templates used in the Content Store. These media may only be selected from defined package directories (see 5.8.1.1, page 113) since this is the only way to ensure that the required media are available in all projects.

If new objects, such as media, are to be inserted, then these are to be imported in the master project and made available to the target projects by creating a new package version. This can be achieved using automatic updating via publishing (section 5.1.2, page 60 and section 5.6.3, page 85) or by updating in the target project manually (section 5.6.6, page 93).

In addition, the mapping of languages for the master project and target projects should be taken into account when transferring a schema for a multilingual project into a package in the master project (see 5.10.5, page 140).

5.10.3 New databases

In contrast to using an existing database, referential integrity does not come into play for a new database since the database does not yet contain any data.

The mapping of the master project's languages has to be taken into account when transferring a multilingual project's schema to a package (see 5.10.5, page 140). Even for new databases, the **media selection limitation** explained previously in section 5.8.1.3, page 116 is recommended for all of the templates used in the Content Store. These media may only be selected from defined package directories (see 5.8.1.1, page 113) since this is the only way to ensure that the required media are available in all projects.

5.10.4 "contentSelect" function

The "contentSelect" function requires special attention for projects with shared database access. Adjustments in the <CMS_PARAM> tags within a function have to be made manually. This applies to all of the master project's templates, i.e. including templates that are not integrated into a package. The reason for this is the namespace extension for the jointly used database schema. If namespace extension has not been disabled, then the schema name changes if the schema is distributed to target projects using the package store.

```
<CMS_PARAM name="schema" value="News"/>
```

becomes:

```
<CMS_PARAM name="schema" value="News@MyPaket"/>
```

All of the master project's templates that use the "contentSelect" function have to be adapted manually in this case. Even templates that are not used in a package have to access the `News@MyPackage` schema immediately.

Advantage: If the templates have been adjusted in the master project once, then the target projects do not need any further changes. They take on the already updated templates via the package store.

```
<CMS_FUNCTION name="contentSelect" resultname="fr_sc_news">
          <CMS_PARAM name="schema" value="News"/>
          <QUERY entityType="News">
          <ORDER>
            <ORDERCRITERIA attribute="Datum" descending="1"/>
          </ORDER>
          </QUERY>
</CMS_FUNCTION>
```

becomes:

```
<CMS_FUNCTION name="contentSelect" resultname="fr_sc_news">
          <CMS_PARAM name="schema" value="News@MyPackage"/>
          <QUERY entityType="News.Overview@MyPackage">
          <ORDER>
             <ORDERCRITERIA attribute="Datum" descending="1"/>
          </ORDER>
          </QUERY>
</CMS_FUNCTION>
```

5.10.5 Language-dependent content

The data from individual input components visible in the Content Store is stored in a database table when using shared database access. Since the schema should not be modified in the target project, the languages for an input component have to be defined in the master project.

You can choose between two different processes for mapping languages:

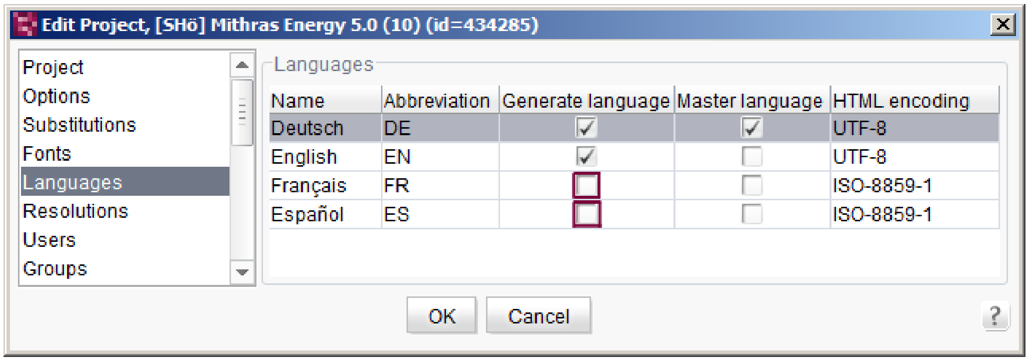
1. Implicit modeling of language-dependency
2. Explicit modeling of language-dependency

5.10.5.1 Implicit modeling of language-dependency

For implicit modeling of language-dependency, all of the languages for the target projects have to be added to the master project languages. This set union of all project languages is then taken into account when creating a database schema.

The languages are added to the "Project properties" under the "Languages" item in the ServerManager. The "Generate language" option should be disabled to prevent languages added via target projects from also being used when generating the master project.

Then a column for each language has to be created in the database schema and the columns are referenced in the mappings for the table template.



Name	Abbreviation	Generate language	Master language	HTML encoding
Deutsch	DE	☑	☑	UTF-8
English	EN	☑	☐	UTF-8
Français	FR	☐	☐	ISO-8859-1
Español	ES	☐	☐	ISO-8859-1

Figure 5-53: Implicit modeling of language dependency

Example:
If a master project has German and English, the first target project has Spanish and English and the second target project has French and English, then:

1. The languages Spanish and French have to be added to the master project's properties,
2. Columns for German, English, French and Spanish have to be created in the database schema for the input component and they have to be referenced in the mappings for the input component.

5.10.5.2 Explicit modeling of language dependency

In contrast to implicit modeling, languages are not mapped using a project property for explicit modeling; this is done using a database schema instead. This means, a column is created in the database schema for each language's input component. Then an input component has to be defined on the table template's form tab for each column and referenced in the mappings.

5.10.6 Different database layers in the master and target project

Direct access to a database via target project is rarely desired in a production environment. If the target projects are not intended to access a database directly but to access a copy of that database instead, then the master project's database layer is managed by FirstSpirit in most cases and the copy for the target projects is managed by a database administrator via an export. In this case, the master project works on a database layer managed by FirstSpirit and the target projects are managed on a layer that has to be updated manually by the database administrator. This results in the states between the master and target project often being asynchronous and results in errors in target projects.

> ! *During an initial import, the option "No schema sync" under the "Databases" item has to be disabled in the project settings (ServerManager). This option must then be re-enabled after the initial import (see section 5.10, page 135).*

If the target projects are to work off of a copy of the original database, the database schema should be duplicated in the master project. A separate database layer is assigned for target projects for this database schema. If just the duplicated database schema is deployed now, then master and target projects always work on one state.

In order to circumvent this problem, the original database schema should be duplicated in the master project and another layer should be assigned to the duplicate. Only the duplicate schema is then made available to the target projects during deployment.

Advantage: The database is managed using FirstSpirit.

6 Legal notices

The "CorporateContent" and "ContentTransport" functions are products of e-Spirit AG, Dortmund, Germany.

Only the license agreed upon with e-Spirit AG is valid with respect to the user for using the functions.

Details regarding potential third-party software products in use not created by e-Spirit AG, their separate licenses and, if applicable, their update information can be found on the start page of every FirstSpirit server in the "Legal notices" area.