



Infopark CMS Fiona

Administration / Layout

Infopark CMS Fiona

Administration / Layout

While every precaution has been taken in the preparation of all our technical documents, we make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein. All trademarks and copyrights referred to in this document are the property of their respective owners. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without our prior consent.

Contents

1	Introductory Comments	11
2	Introduction	12
3	Concepts	13
3.1	User Administration	13
3.1.1	Global Permissions	13
3.1.2	File-Specific Permissions	14
3.2	Files and Formats	15
3.2.1	File Types and Export	15
3.2.2	Fields, Field Sets and Field Presetting	16
3.2.3	Workflows	17
3.2.4	Checks and Functions	17
3.3	Jobs	18
3.4	Channels	19
4	Managing Users and User Groups	20
4.1	The Users Section	21
4.1.1	Creating a User	22
4.1.2	Editing the User's Personal Preferences	25
4.1.3	Changing User Fields and Deleting Users	26
4.2	The Groups Section	26
4.2.1	Creating Groups	27
4.2.2	Changing Group Properties and Deleting Groups	29
4.3	The User Fields Section	30
4.3.1	Creating User Fields	31
4.3.2	Editing or Deleting User Fields	32
5	Configuring the Content Manager	34
5.1	The File Formats Section	34
5.1.1	Defining File Formats	36
5.1.2	Editing Field Properties	39
5.1.3	Creation Constraints	41
5.1.4	Version Settings	43
5.1.5	Checks and Functions	44
5.1.6	Configuring Field Sets	47

5.1.7 Editing and Deleting File Formats	50
5.2 The Fields Section	51
5.2.1 Defining Fields	53
5.2.2 Editing Field Properties	56
5.2.3 General Properties	56
5.2.4 Special Properties	56
5.2.5 Functions	57
5.2.6 Input Fields	58
5.2.7 Deleting Fields	59
5.3 The Workflows Section	60
5.3.1 Defining Workflows	61
5.3.2 General Properties	61
5.3.3 Edit Steps	62
5.3.4 Sign Steps	63
5.3.5 Editing or Deleting Workflows	64
5.4 The Jobs Section	65
5.4.1 Defining Jobs	67
5.4.2 General Properties	67
5.4.3 Execution	68
5.4.4 Script	69
5.4.5 Executing and Canceling Jobs	69
5.4.6 Editing or Deleting Jobs	70
5.4.7 Editing the Execution Schedule of Jobs	71
5.4.8 Viewing the Job Log and Job Output	72
5.5 The Channels Section	73
5.5.1 Defining Channels	74
5.5.2 Editing or Deleting Channels	75
6 Layouts	77
6.1 The Function of Layout Files	77
6.2 Designing a Layout Concept	78
6.3 Editing Layouts	79
6.4 Layout Structure	79
6.4.1 XML and XHTML Marking	79
6.4.2 HTML Header	80

6.4.3	Style sheets	80
6.4.4	HTML Main Content and other Field Values	81
6.4.5	Automatic Navigation Entries	83
6.4.6	Micronavigation	83
7	Automatically Generated Tables of Contents	85
7.1	Demands on Start Pages	85
7.2	Components of Automatically Generated Tables of Contents	85
7.2.1	Structure of an Automatically Generated Table of Contents	85
7.2.2	NPSOBJ Parameters for Automatically Generated Tables of Contents	86
7.2.3	Setting the Number of Entries in Tables of Contents	86
7.2.4	Using other Fields	87
7.2.5	Entry Sorting	87
7.3	Controlling Entries	88
7.3.1	Querying File Types	88
7.3.2	Querying Field Values	89
7.3.3	Combined Comparisons	90
7.4	Examples of Automatically Generated Tables of Contents	91
7.4.1	Formatting As a Series of Paragraphs	91
7.4.2	Including Further Field Values	91
7.4.3	Formatting As a Two-Column Table	91
7.4.4	Separation by File Type	92
7.4.5	Linking Sibling Files	92
7.5	Selecting Tables of Contents Using Field Values	93
8	Collections of Links	94
8.1	NPSOBJ Tags for Generating Collections of Links	94
8.2	Content and Formatting of Link Lists	94
8.2.1	Evaluate links to CMS files	95
8.2.2	Selecting and sorting the elements in collections of links	95
9	Creating RSS Feeds	97
9.1	The Concept of RSS Feed Creation	97
9.2	Using the npsobj newslst Instruction	97
9.3	Examples for Using News Lists	98
9.3.1	How to create a list of 10 latest news articles	98
9.3.2	How to create an RSS feed for politics news	99

9.3.3	Sending a Newsletter	100
10	Export Variables	101
10.1	Storing and Recalling Texts	101
10.2	Storing and Recalling Context Lists	102
10.3	Local Export Variables	103
11	Further NPSOBJ Instructions	105
11.1	Using the Main Content of a File	105
11.2	Execute System Commands	105
11.3	Generating Absolute Paths in Scripts	106
12	Examples of Layouts	107
12.1	Nested Layouts	107
12.1.1	Creating a Sitemap As a Table of Contents	107
12.1.2	Sitemap with Selected Subfiles	108
12.1.3	Infinite Recursion	108
12.1.4	Define Table Layouts	108
12.2	Frame Layouts	109
12.2.1	Automatically Create Framesets with the NPSOBJ Frame Command	110
12.2.2	Implement Frames with Folder Hierarchies	112
13	The NPSOBJ Syntax	114
13.1	Conventions	114
13.2	Evaluation in a Context	115
13.2.1	What Is a Context?	115
13.2.2	The Structure of Names	115
13.2.3	Names in File Contexts	117
13.2.4	Names in Link Contexts	119
13.2.5	Names Whose Value Is a File	119
13.2.6	Names in All Contexts	119
13.2.7	@ References	119
13.2.8	Formatting Values with Tcl Procedures	121
13.3	npsobj break	124
13.3.1	Syntax	124
13.3.2	Task	124
13.3.3	Example	124

13.4	npsobj condition	124
13.4.1	Syntax	124
13.4.2	Task	125
13.5	npsobj context	127
13.5.1	Syntax	127
13.5.2	Task	127
13.5.3	Example	127
13.6	npsobj frame	127
13.6.1	Syntax	127
13.6.2	Task	128
13.6.3	Example	128
13.6.4	Using target="_top" in the Preview	128
13.7	npsobj includetext	129
13.7.1	Syntax	129
13.7.2	Task	129
13.7.3	Example	129
13.8	npsobj insertvalue anchor	129
13.8.1	Syntax	129
13.8.2	Task	130
13.8.3	Example	130
13.9	npsobj insertvalue dynamiclink	130
13.9.1	Syntax	130
13.9.2	Task	130
13.9.3	Example	131
13.10	npsobj insertvalue image	131
13.10.1	Syntax	131
13.10.2	Task	132
13.10.3	Example	132
13.11	npsobj insertvalue link	132
13.11.1	Syntax	132
13.11.2	Task	132
13.11.3	Example	133
13.12	npsobj insertvalue meta	133
13.12.1	Syntax	133
13.12.2	Task	133

13.12.3	Example	133
13.13	npsobj insertvalue systemexecute	133
13.13.1	Syntax	133
13.13.2	Task	134
13.13.3	Example	134
13.14	npsobj insertvalue template	134
13.14.1	Syntax	134
13.14.2	Task	135
13.14.3	Examples	135
13.15	npsobj insertvalue var	136
13.15.1	Syntax	136
13.15.2	Task	136
13.15.3	Example	136
13.16	npsobj list	137
13.16.1	Syntax	137
13.16.2	Task	137
13.16.3	Example	139
13.17	npsobj micronavigation	139
13.17.1	Syntax	139
13.17.2	Task	139
13.18	npsobj modifyvar append	140
13.18.1	Syntax	140
13.18.2	Task	140
13.18.3	Example	140
13.19	npsobj modifyvar clear	140
13.19.1	Syntax	140
13.19.2	Task	141
13.19.3	Example	141
13.20	npsobj modifyvar range	141
13.20.1	Syntax	141
13.20.2	Task	141
13.20.3	Example	141
13.21	npsobj modifyvar set	141
13.21.1	Syntax	141
13.21.2	Task	142

13.21.3	Example	143
13.22	npsobj modifyvar sort	143
13.22.1	Syntax	143
13.22.2	Task	143
13.22.3	Example	144
13.23	npsobj newslst all	144
13.23.1	Syntax	144
13.23.2	Task	144
13.23.3	Example	144
13.24	npsobj newslst selected	144
13.24.1	Syntax	144
13.24.2	Task	145
13.24.3	Example	145
13.25	npsobj switch	145
13.25.1	Syntax	145
13.25.2	Task	145
13.25.3	Example	146
13.26	npsobj table	146
13.26.1	Syntax	146
13.26.2	Aufgabe	147
13.26.3	Example	147
14	Providing Additional Means for Editing Content in the Preview	148
14.1	Displaying Hints in the Preview	148
14.2	Offering Actions	149
14.2.1	Syntax	149
14.2.2	Parameterizing Wizards	150
14.2.3	Dynamically Positioned Editing Elements	150
14.3	Controlling Automatic Display of Editing Elements	153



1 Introductory Comments

This manual is designed for administrators who maintain users and user groups in the Content Management Server and who would also like to set up file formats, fields, workflows and jobs. The manual also addresses designers who would like to create and maintain layouts for the Content Management Server.

As a reader of this manual, you should already have general computer knowledge as well as experience with the Internet and with WWW-browsers. As an administrator, you should have knowledge of the user and group concepts, and the editing concepts of Infopark CMS Fiona. As a designer, you need considerable basic knowledge of files and versions in the Content Management Server as well as a good knowledge of HTML.

The explanations and operation descriptions in this online manual refer to the English-language user interface with the standard defaults of the Content Management Server. The explanations are neutral with regard to the operating system and browser you are using. If you have any questions or problems, please consult the manual of the respective product.

The Content Navigator is only functional if Java and Javascript have been enabled in your browser. The preview function requires that your browser accepts cookies.

2

2 Introduction

Infopark CMS Fiona provides you with flexible, powerful software to create, maintain and publish extensive online systems (websites). The Content Management Server is the main component of this system. It administers your documents and gives you the possibility to design and edit them in a controlled way.

A website is a complex construction. It consists of many areas with different characteristics with regard to topic, layout and function. It is created, maintained and expanded jointly by editors, designers and administrators, perhaps even programmers. Your company's web presence should always be up-to-date and attractive and the corporate design should be consistent. The Content Management Server helps you to achieve this.

You can create users and user groups via the user interface of the Content Management Server. You can add users to groups and define their authorizations by granting them permissions. You control a group's access to each individual file via user groups. You can create workflows and connect each work step to a user group. You can then determine which groups may use which formats via global permissions.

Formats and fields give you the possibility to determine the properties of all files and to make them dependent on their position in the folder hierarchy. You can specify the workflows you have defined in formats in order to specify the stations the file has to come across in your company. Using jobs, you have a means at your disposal that you can use to define Tcl scripts via the HTML user interface and have them run at set times.

Although these means are powerful, they only become effective when all the responsibilities of those involved are gathered together in detail and the structure of your website is fixed. They then contribute to making your work more efficient.

3 Concepts

3.1 User Administration

The Content Management Server has a simple but flexible user administration.

User administration is logically strictly separate from the Content Management Server and is addressed by the latter via an interface. Because of this architecture, the Content Management Server can be combined with another user manager, such as e.g., one based on LDAP.

The Content Management Server administers every user by means of a login name, which "embodies" the user as a unique identifier. A password that allows the user to access the Content Management Server is assigned to every login name.

The user administration integrated into the Content Management Server is also responsible for user groups and the allocation of users to groups. It also administers the allocation of global permissions to users and groups.

Using global permissions, you determine the authority of individual users or members of a user group with regard to their administrative tasks. For the purposes of logical user management, the user groups should first be set up and then granted permissions. Users can then be placed in the groups to grant them the permissions of those groups. A user is automatically a member of a default group. He or she can also be a member of other user groups.

In the Content Management Server, user groups also fulfill an important task with regard to files, because they can be granted file-specific permissions. These permissions determine how the user groups are able to access files.

In addition to users and user groups, the Content Management Server also manages so-called user fields, which can be helpful in user administration. In these field values, you can store additional user-specific information such as the address or private telephone number of the respective user, thereby allowing you to have such information available as needed. All defined user fields are assigned to all logins.

3.1.1 Global Permissions

By granting users global permissions, the users are given authorizations which do not relate to working with individual files, but to administrative activities. A user who, for example, has user administration permission may create, edit or delete logins. Specifically, the following global permissions can be granted:

- **Administrator** (`permissionGlobalRoot`): Users to whom this permission has been granted possess all authorizations. An administrator is also referred to as a *superuser*. This permission is

always required if no other permission is sufficient for performing an action (such as creating or editing jobs).

- **Edit users and groups** (`permissionGlobalUserEdit`): Create, edit or delete login names and user groups; grant permissions
- **Edit User Fields** (`permissionGlobalUserAttributeEdit`): Create, edit or delete user fields
- **Edit global settings** (`permissionGlobalRTCEdit`): Create, edit or delete file formats, fields, channels, and workflows.
- **Perform an export** (`permissionGlobalExport`): Execute commands with which folders can be exported.
- **Creating mirror files** (`permissionGlobalMirrorHandling`): This permission is required for creating [mirror files](#). Additionally, the file-specific restrictions apply, meaning that the user requires the `permissionCreateChildren` permission for the folder concerned.

In addition to the permissions mentioned, additional global permissions, such as e. g. permissions for the use of formats, can be freely defined in the `globalPermissions` entry in the `userManagement.xml` configuration file of the Content Management Server: In each format definition, one of the predefined or additional permissions can be specified so that a user can only create files based on that format if he or she has this global permission.

3.1.2 File-Specific Permissions

In addition to global permissions, the Content Management Server also has file-specific permissions. These permissions can be granted for each file and determine the authorizations that the members of user groups have with regard to the file. For example, a user must have administration permission for a file to be able to delete it; to be able to change the field values of its draft version, he or she requires write permission.

File-specific permissions cannot be granted to individual users, but only to user groups. The following permissions can be granted for each file:

- **File Administration** (`permissionRoot`)
Users to whom this permission has been granted possess all authorizations. The administration permission is required to delete, rename or move files, as well as to be able to unrelease them. As the administrator of a file, you have permission to release it regardless of its workflow status and to sign it in the verification phase even if you are not a member of the user group designated to do so.
- **Read File** (`permissionRead`)
All access to the file that does not change it.
- **Edit Version** (`permissionWrite`)
Create a new draft version or edit a draft version (this also includes modifying field values).
- **Create Subfiles** (`permissionCreateChildren`)
This permission can only be granted for folders. This permission allows its possessor to create subfiles.
- **Read Live Version** (`permissionLiveServerRead`)
This permission can be used in conjunction with the Portal Manager. Among other features, this CMS component provides document-specific access permissions for the visitors of your website. The groups to whom this permission can be granted may be different from those for the other permissions, if this has been configured like this by the administrator. If a file's *Read Live Version* permission has been assigned at least one user group, only the respective group members are permitted to access the document concerned. Otherwise, all visitors may access it.

Whether a user can create a file in a folder depends both on his or her file-specific permissions in the folder and his or her global permissions. First, the user needs file creation permission in the folder, which he or she automatically has if he or she has administration permission in the folder. If a particular global permission is required to use the format upon which the new file is to be based, then the user also requires this permission. If he or she has global administration permission, he or she may use any format.

If a user is able to create a new file, the same access permissions apply for this file as for the folder containing it. The file inherits the permissions of its folder. If these permissions do not include read and write permissions, the Content Management Server grants the missing file-specific permissions to the user's default group.

Permissions are not inherited retroactively. If, for example, a user is granted the administration permission for the folder containing a file, the permissions the user has with regard to this file remain unaffected by this measure.

3.2 Files and Formats

The folder hierarchy from your website consists of many files. These files are created in the CMS based on formats. The file properties are configured using formats and in doing so the purpose and type of the files are determined. A format is to a certain extent a set of rules that the Content Management Server applies when a user creates a new file.

3.2.1 File Types and Export

Whether and how a file is exported is already determined by its type, which is part of the file format used for the file. When files of the types *Folder*, *Document*, *Image* or *Resource* are exported, a corresponding output file is created for each file. Files of the *Layout* type are not exported.

Files of the *Image* or *Resource* type are not exported using layouts. Instead, they are exported as they are, unmodified. Image files hold the images that were imported into the CMS. Resources can have any content and are mostly offered to the visitor of your website for download (e.g. PDF files).

The Difference between Folders and Documents

In contrast to documents, folders can have subfiles and thus help to structure your website. A folder on your website can function as a start page with references to, for example, subordinate pages. However, with regard to their contents, there is no difference between files of these two types.

The output file of a folder and a document normally corresponds to a web page. However, when exporting files of these types, the CMS does not simply create a file containing the main content of the released version of the CMS file. Instead, it uses a layout file for the export, the base layout.

Function of the Base Layout

The base layout is a special exemplar of the *Layout* file type. Usually, it consists of HTML text and export instructions. When exporting a folder or a document, the Content Management Server processes the base layout like a program to create the corresponding output file. The HTML text contained in the layout file is written to the output file unchanged. Using a special tag, the NPSOBJ instruction, you can instruct the Content Management Server to export field values such as meta tags or the main content of a released version, for example. If the appropriate instructions are missing in the base layout, the file is exported incompletely.

If several base layouts exist (e.g. one for the pure text, one for the complete current layout, and one for the relaunch) every user can specify his individual base layout in his personal preferences. This makes it possible to simultaneously work with several layouts of the same website. In the Content Management Server default installation, there is only one base layout file which is simply named `mastertemplate`. Base layout files are often contained in the base folder.

Layouts Make Use of Other Layouts

Typically, the base layout file references further layouts. When exporting a file, the Content Management Server first searches for a referenced layout in the folder containing the file. If the layout does not exist, it searches for it upwards in the folder hierarchy, towards the base folder. This mechanism makes it possible to use local layouts, which are only effective in subsections of your website.

Layout files are the ideal location for layout and navigation elements that are to enclose the main content. As a result, they are a means by which you can efficiently create a website with a homogenous layout. The procedure with which the CMS, i.e. the Content Management Server and the [Template Engine](#), searches for layout files during the export gives you additional support in creating website areas with individual layouts.

3.2.2 Fields, Field Sets and Field Presetting

Files always consist of a main content and a set of fields. There are many predefined fields in the CMS, e.g. for the name of the file, the title of its content, etc. Furthermore, additional fields can be defined by a CMS administrator.

Using fields, you can store information in the Content Management Server relating to a file or version, or provide the version of a file with additional customer-specific information. Every field has a name and a title, a type and a value. The Content Management Server differentiates between file fields and version fields.

File Fields

File fields are predefined by the Content Management Server and are created for each file independently of its type when the file is created. The values of these fields are absolutely necessary for maintaining the files and defining their relationship to each other, and for controlling the export.

The fields *ID* and *Path* are automatically preset with values by the system. The ID is the unique identification number of a file in the Content Management Server. It cannot be changed. In contrast, the value of the *Path* field changes when the file is moved within the folder hierarchy. In this case, the field value is adjusted by the Content Management Server.

Every file has a *Name* field. Its value can be set by the user who creates the file. The name of the file is used to clearly identify it within a folder. If no name is given, the Content Management Server gives the file a predefined name; in the default installation, this is *new*. If additional names are created automatically, the Content Management Server ensures their uniqueness by appending a counter to the predefined name.

In addition to *ID*, *Path*, and *Name*, files have (among others) the file fields *File Type*, *Workflow*, *Format*, *Maintainer* and *Exclude from Export*.

The file type is derived from the file format and cannot be changed. The value of the *Workflow* field is also derived from the format; it can, however, be subsequently changed by a file administrator. The *Format* field contains the name of the format of a file. Its value can be changed. However, this is only

necessary in exceptional cases and should be avoided (the file can become invalid). The value of the file field *Exclude from Export* can also be changed.

Version Fields

In addition to the file fields in the Content Management Server, version fields also exist which are not created for a file itself but for the draft version of a file. A version field can be assigned a different value in each version of a file. Some version fields are predefined, while others are custom fields.

As with file IDs, versions also have *Version IDs* to clearly identify them. As with some file fields, you can preset the predefined version fields *Title*, *File Extension*, *Valid from* and *Valid until* with values in the format of the file to which the version belongs.

With additional (i.e. custom) version fields there are more field types available (see also [Type](#)). If you create, for example, a field of the *Selection* type, the editor of a version can select a value from several field values, namely those that were preset by the CMS administrator.

Custom version fields in the Content Management Server should be conceived of as field definitions, which you can reference in formats. In doing this, you determine which fields the draft versions of the files – based on the corresponding format – are to have. When you first create a draft version, it is given the fields that are referenced in the format on which the corresponding file is based. In accordance with the format of their file, versions can have different custom fields and any number of them.

The version fields of a format can be grouped together in field sets in the format definition. The field sets are displayed on an file's details view as sections. The values of the fields can thus be edited in a comfortable and structured way.

Furthermore, in formats, you can determine which version fields are to be obligatory fields, i.e. which fields of a draft version must contain a value so that the respective file can be released. Additionally, all fields can be optionally preset with values so that the fields of the draft version of new files based on this format contain preset values.

3.2.3 Workflows

In many cases, files in the Content Management Server run through a workflow before they are exported. Using workflows, you can determine which user groups are responsible for editing and verifying a file. Files can run through both minimum and substantial workflows.

In file formats, administrators can preset the workflow field with the name of a workflow. As the draft version of a file is created, the Content Management Server determines the workflow to be used for the file by first reading out the value from the format of the file. The workflow with this name is then stored in the draft version of the file.

A new workflow begins whenever a draft version is created, a version is rejected, or a released version is unreleased. Since the workflow definition is stored in the version, an active workflow cannot be restarted or modified by assigning the name of a different workflow definition to the *Workflow* field or by modifying the workflow definition itself. You can do this, however the changes only become effective when a new draft version of the file concerned is created.

3.2.4 Checks and Functions

The purpose of checks and functions is to ensure that your individual criteria regarding form and content of your CMS files are met. The checks and functions are Tcl routines that are processed

whenever particular application events occur in the Content Management Server, for example, when an editor changes the content of a file. These Tcl routines are part of file format and field definitions. They can be entered directly in the configuration user interface of the editorial system, meaning that no shell access to the system is required.

Checks

Checks are Tcl routines that are executed in addition to the checks the system performs. They give you the possibility to attach your own additional conditions to the completion of an operation. The release of a file, for example, is only possible if the file is complete. One built-in criterion for completeness is that all internal links point to existing CMS files. By writing another checking routine, you can extend the list of completeness criteria so that a file can only be released if the system's criteria plus your own ones are met. There are two types of checks that can be defined for each file format:

- Using the [Creation Check for Files in the Folder](#) you can, for example, check during file creation in a folder whether the subfile is based on a format valid for subfiles.
- Using a [Completion check](#) you can check versions to see if they are complete according to your criteria. The Tcl code for the creation check for subfiles or the completion check can be defined in file formats.

Functions

Functions are also custom Tcl routines. The system calls them after an editor has changed the contents of a field but before the modified content is stored in the draft version. Therefore, functions are suitable for analyzing content and, if necessary, for changing it.

- The [Value Assignment Function](#) of a field is always called when the value of a field is changed. You can use this feature to check whether a field has an acceptable value and to change the value if necessary.
- You can use the [Value display function](#) for fields in order to calculate the field value to be displayed in the Content Navigator. The Tcl code of a value assignment function or a value display function is part of a field definition.
- You can specify a [Version assignment function](#) or a workflow modification function in the file format definition. The version assignment function is used to verify the consistency of the field values of a version and is called by the Content Management Server before it sets the field values of a version.
- Finally, you can use the [Workflow modification function](#) to modify the workflow of a draft version before the version is created.

3.3 Jobs

Jobs enable you to have custom Tcl scripts executed once or repeatedly. Jobs are available in both the Content Manager and the Template Engine. With the latter, however, jobs can only be configured via the [Tcl interface](#).

There are system jobs and user jobs. System jobs are predefined routines for the purpose of, for example, transferring update records to the Template Engine in the context of the incremental live server export. They can neither be deleted nor modified. User jobs, on the other hand, can be created, modified, and deleted as desired. Both job categories share the same name space, i. e. two jobs cannot have the same name even if they belong to different categories.

The jobs in a category are placed in the execution queue at execution time unless they have already been added to it. This prevents a more frequently executed job from being placed in the queue several times if this job or another one is currently running.

All jobs in a category are processed sequentially since they share the same execution queue. This has the advantage that several jobs can be triggered at the same time and will nevertheless be processed consecutively (initiation and execution are asynchronous). As a result, performance losses caused by simultaneous execution of many time-intensive Tcl scripts can be avoided.

The execution of jobs is logged. The maximum number of log entries the system creates per job is determined by the value of the `jobMaxLogLength` system configuration entry.

How jobs can be defined in the HTML user interface is described in section [The Jobs Section](#).

3.4 Channels

Channels are helpful when content needs to be categorized by topic. Especially on portal sites, channels let the visitors access their favourite topics more quickly.

The Portal Manager and the Content Management Server support channels. In the Content Manager, [channels can be defined](#) to which the editors can then assign files with the help of the built-in *channels* version field. In the context of the incremental export, the files and the channel definitions are transmitted to the live server.

On the live server, news feeds can be automatically generated. These feeds can then be delivered by the Portal Manager, for example. Whether a file is to be regarded as a news article can be controlled by an option in the file's format, i. e. in the editorial system.

For the purpose of personalizing web pages, the live server read permission can be granted to user groups for each file, requiring users to log-in if they wish to read an article.

4

4 Managing Users and User Groups

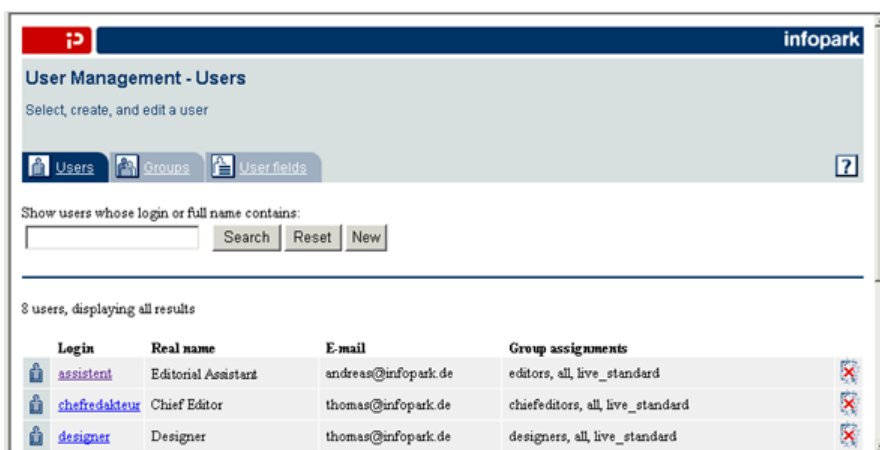


The Content Management Server administers logins (login names) with their associated passwords, user fields and user groups. Using it, you can grant global permissions to users and user groups and file-specific permissions to groups. The way in which user administration of the Content Management Server is organized is explained in section [User Administration](#).

As a Content Management Server user who is able to create, delete or change logins or user groups, you must have the global permission *Edit users and groups*. To be able to define user fields, the permission *Edit user fields* is required.

Please note that several administrators can edit and also save configuration values (like the properties of a user group) in parallel. In this process the values saved immediately before are overwritten.

To create, edit or delete users, groups or user fields, select *Extras > User management* from the menu of the Content Navigator. The *User Management - Users* page is displayed with the three sections *Users*, *Groups* and *User Fields*. You can move to a section by clicking the corresponding tab:



In all sections of the user administration there are search functions you can use to display users, user groups or user fields in the main area of each section.

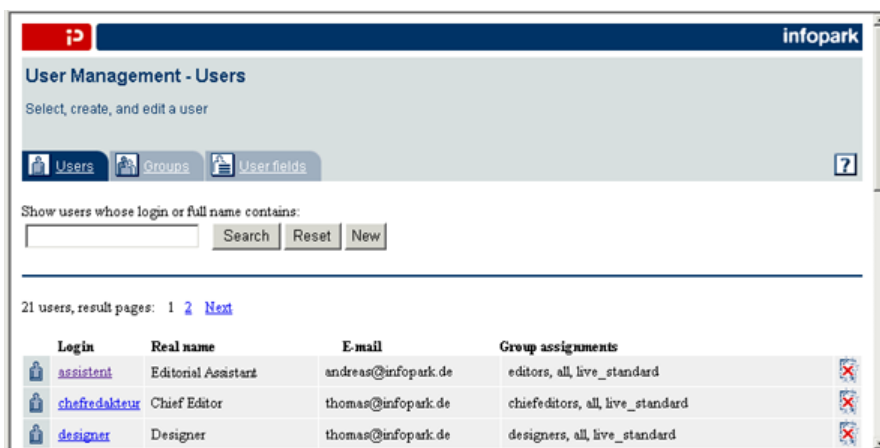
If you would like to have a list of users, groups and user fields displayed whose name or full name contains a search term, enter the search term in the entry field of the appropriate section and click on the *Search* button. A list of the corresponding users, groups or user fields is then displayed in the main area of the section. You can change the search term by entering another search term in the entry field of the appropriate section and clicking on the search button a second time.

You can also use the search function to display all the users, groups or user fields registered in the Content Management Server. To do this, leave the entry field empty and click on the *Search* button.

The number of users, groups or user fields displayed in each list in the main area of a section depends on your personal preferences. If the list contains more entries than you have set as default, a sequence of numbers in ascending order will appear above and below the list. Each number refers to an additional results page which you can display by clicking on the number. You can also move to the next or the previous results page using *Previous* or *Next* respectively.

4.1 The Users Section

In order to create a user, to allocate him or her to groups, to grant him or her permissions and assign values to user fields, click on the *Users* tab on the *User Management* page. The Content Management Server displays the *Users* section:



This page is also displayed if you click the *Browse* or *Add* button in a dialog in order to select a user. In this case, the *New* button is not present and a *Cancel* button is provided in the upper area of the page allowing you to cancel the selection action as required.

Please use the search function to display users. How the search function is used is described in section [User Management](#).

A list of those users whose login or full name contains your search term is displayed as a search result in the main area of the section. Each list entry consists of a user's login, the full name and the e-mail address. The list also shows which groups he or she is a member of.

The number of list entries on each result page depends on your personal preferences. If the list contains more entries than can be displayed on a result page, you can move to the other result pages by clicking on *Previous* or *Next* or one of the numbers.

If this page was displayed because you clicked the *Browse* or *Add* button in a dialog, you can now select a user by simply clicking his or her login.

If you click on the login of one of the users listed, the Content Manager displays all the available information concerning this user:

User Management - Users - View
The properties of the user "Redakteur"

Users Groups User fields

New Edit Delete

Change password Edit preferences Specify local applications Specify views and layout

Login: redakteur
Real name: Redakteur
E-mail: redakteur@mail-example.net password demo
Default group: editors
Group assignments: editors, all, live_standard
Owner: root
Locked: No
Global permissions:

On the user's view page you can see the predefined fields, group assignments, global permissions, and the custom fields (if some exist).

Starting on the view page, you can also change the user's personal preferences such as his password, his basic settings, his local applications, and Content Navigator settings by clicking the corresponding buttons. The tasks of the buttons *New*, *Edit*, and *Delete* are described in the following sections.

4.1.1 Creating a User

To create a new user, click on either the view page of a user or on the button *New* in the *Users* section. The form *User Management - User - New* is opened:

User Management - User - New
Create a new user

General properties

Login:
The login cannot be changed after creation!

Real name:

E-mail:

Owner:
admins or
Browse

☐ Locked

Group assignments

The form is divided into the areas *General Properties*, *Group Assignments*, *Global Permissions* and *User Fields*, which are described in the following.

General Properties

You determine the general properties of the new user in the uppermost area of the form:

The screenshot shows a web form titled "General properties". It contains the following elements:

- Login:** A text input field with a warning note below it: "The login cannot be changed after creation!"
- Real name:** A text input field.
- E-mail:** A text input field.
- Owner:** A text input field followed by a "Browse" button.
- Locked:** A checkbox labeled "Locked".

Login

Enter the future login of the user in the *Login* field. Login names and the names of the user groups belong to the same name space; a name that is already being used as a group name cannot therefore be used later as a login name (and vice versa).

Real Name

Enter the full (real) name of the user in this field.

E-Mail

Here you can enter the E-mail address of the user.

Owner

When you create a new user, you become his or her administrator. This means that only you can edit the user data, e.g. set the values of the user fields or block the user's access to the Content Management Server. Furthermore, you can place the user in user groups that you have created yourself.

Both individual users and user groups can be the owner of a user. You can change the owner of the user by entering the login or the group name in the *Owner* field. Alternatively, you can click on the *Browse* button to switch to the [selection page for users](#) and [groups](#). You can limit the search from the start by entering a part of a user's name in the entry field and afterwards clicking on the *Search* button. On the selection page you can start a new search at any time by entering the search term in the entry field and clicking on *Search*. Then select a user or a group from the list which appears by clicking on its login or name, respectively. The new owner is then placed in the entry field. You can also cancel the search by clicking the *Cancel* button.

Please note that the new administrator (owner) - whether an individual user or a user group - needs the User Administration global permission.

Locked

It is possible to deny a user access to the Content Management Server if this should be necessary. In this case enable the *Locked* option.

Group Assignments

In this area of the form, you can allocate the user to user groups:

The screenshot shows a web form titled "Group assignments". It has two main sections. The first section is labeled "Default group:" and contains a text input field followed by a "Browse" button. The second section is labeled "Groups the user is a member of" and contains a list box (which is currently empty), an "Add" button, and a "Remove" button.

Default Group

Determine the default group of the new user here by entering the name of the group in the *Default Group* field.

Alternatively, you can enter a search term and click on the *Search* button to search for the desired group. Afterwards, a list of those user groups whose name or full name contains the search term is displayed on the selection page for groups. If you click on the *Search* button without entering something in the entry field then the [selection page for groups](#) is opened in which you can use the search function to find the desired group. Click then on the name of the group in the corresponding list entry to move it to the *Default Group* field. If you would like to cancel the process, click on the *Cancel* button.

After you have confirmed the settings by clicking on *OK* the user's default group is automatically added to the list of groups the user is a member of (in case it is not already contained in the list).

The Content Management Server grants a user file-specific permissions via the default group when he or she has created a file but does not have read or write permission. This is the case when, due to his or her group assignment, the user is allowed to create files in the folder, but is not a member of a group with read or write permissions (the new file inherits the permissions of the folder containing it).

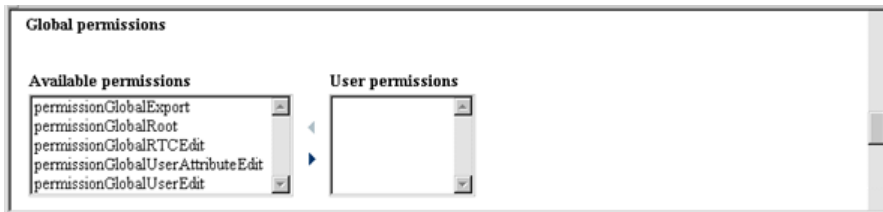
Assigning Users to Groups

A new user is initially not a member of any group. In order to assign a new user to user groups, click on the *Add* button next to the list *Groups the user is a member of*. This opens a search page on which you can search for the desired group and select it by first clicking on the *Search* button and then selecting the group from the search results by clicking on its name. The group is then added to the list *Groups the user is a member of*. Repeat this process for all the groups to which the user is to be assigned.

In order to remove a user from a group, mark the groups in the list and click on the *Remove* button. Subsequently, the marked group no longer appears in the list. However, the Content Management Server does not allow you to remove the user from the user group determined as his or her default group.

Global Permissions

In this area of the form, you assign users global permissions:



You can grant the new user [global permissions](#) by selecting them from the *Available Permissions* list and clicking on the arrow pointing to the right. All permissions which have been assigned to the user are displayed in the *User Permissions* list.

In order to revoke the user's permissions, select the corresponding permissions in the *User Permissions* list and click on the arrow pointing to the left. The permissions removed from the list are moved back to the *Available Permissions* list.

In addition to the permissions listed in the *Global Permissions* area of the form, you can define other permissions in the Content Management Server configuration file *cm.xml*. These permissions also appear in the *Available Permissions* selection field. These permissions play a special role in [file formats](#): One global permission can be determined in each file format. A user must have this global permission to be able to create files based on this format.

If you grant permissions to individual users, please note that this will make user administration more difficult. We recommend you grant users permissions via user groups.

User Fields

In this area of the form, you can assign values to the user fields:

If necessary, additional fields can be defined in the [User Fields](#) section.

Confirm the entries for the new user with **OK**. The Content Management Server then displays the view page of the new user.

4.1.2 Editing the User's Personal Preferences

If necessary, you can also configure the user's [personal preferences](#), e.g. his or her password. However, this is only possible when you are the user's administrator or a superuser.

In order to edit a user's personal preferences, first find the desired user in the *User* section using the search function. Then click on the login in the corresponding list entry to switch to the view page of the user. Here, you can edit the user's password, her basic personal preferences, her local applications, and her Content Navigator settings by clicking the corresponding buttons.

4.1.3 Changing User Fields and Deleting Users

In order to change user data, first find the desired user in the *Users* section using the search function. If you then click on the login in the list entry of the user, the Content Management Server displays the view page of the desired user. Here you click on the *Edit* button to open the *User Management - User - Edit* form:

In this form, you obtain all the available information concerning this user. The form contains the same data as the form for [creating a user](#). The login cannot however be changed.

Click *OK* to accept the changes for the user. The CMS then re-displays the view page of the user.

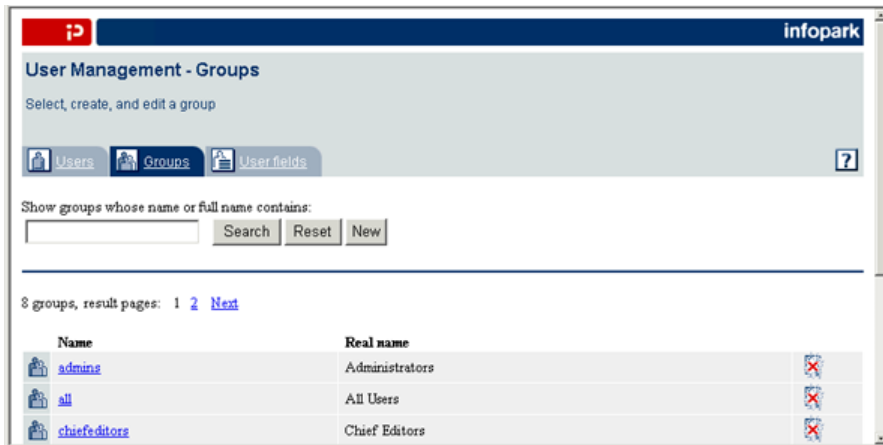
If you would like to permanently remove a user from the Content Management Server, localize him or her using the search function in the *Users* section and then click on the delete button to the right of the list entry. The user can also be deleted via his or her view page. To do this, click on the *Login* of the desired user in the search result and then on the *Delete* button on the view page. The CMS asks you to confirm the deletion:

If you click on *OK*, the user is deleted. The deletion cannot be reversed.

4.2 The Groups Section

The [user management](#) of Infopark CMS Fiona allows you to create user groups, grant them permissions and put users into groups. Users can be members of several groups.

In order to create, edit or delete user groups, switch to the *Groups* section by clicking on the corresponding tab.



This page is also displayed if you click the *Browse* or *Add* button in a dialog in order to select a user group. In this case, the *New* button is not present and a *Cancel* button is provided in the upper area of the page allowing you to cancel the selection action if required.

You can use the search function of this section to have groups displayed. Proceed as described in section [User Management](#).

A list of those groups whose name or real name contains your search text is displayed as a search result in the main area of the section. The list entries each contain the name and the real name of a group.

The number of groups displayed in each list in the main area of the section depends on your personal preferences. If the list contains more entries than you have set as default, a sequence of numbers in ascending order will appear above and below the list. Each number refers to an additional results page which you can display by clicking on the number. You can also use *Previous* and *Next* for browsing.

If this page was displayed because you clicked the *Browse* or *Add* button in a dialog, you can now select a user group by simply clicking its name.

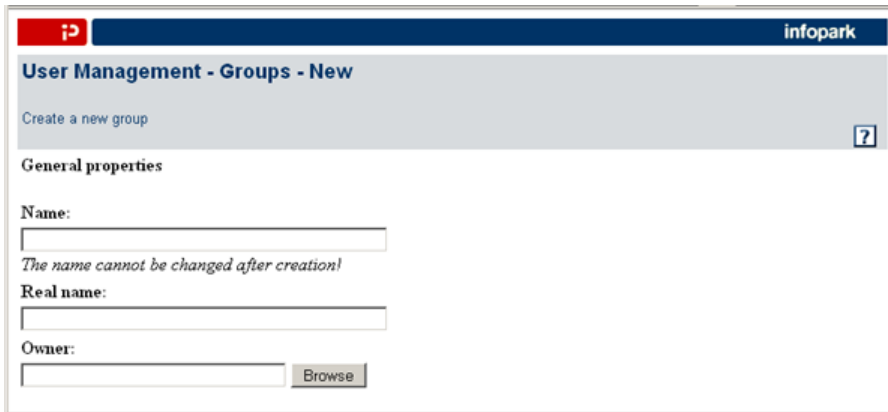
If you click on the name of a group listed, the Content Manager displays all the available information concerning this group



The name and the real name as well as the group members, the owner and the global permissions granted can be seen one below the other on the view page of the group.

4.2.1 Creating Groups

To create a user group, click on the *New* button in the *Groups* section or on a group's view page. The form *User Management - Groups - New* is displayed:



The form consists of several areas which are described in the following.

General Properties

In this area of the form, you determine the general properties of the user group:



Name

Please enter a unique identification for the new group in the *Name* field. The name of a group cannot be subsequently changed.

Please note that group names and the login of the users belong to the same name space. A name being used as the name of a group cannot therefore be used later as a user login (and vice versa).

Real Name

Please enter a meaningful description of the group in the *Real Name* field.

Owner

As with users, groups also have an owner who is allowed to administer the group. Initially, you (as the person who created the group) are the owner of the group. You can, however, enter a user or a user group as the future administrator of the group in the Owner field. To do this, enter the login or group name of the new owner in the entry field. You can limit the search from the start by entering a part of a user's name in the entry field and afterwards clicking on the *Search* button. On the [selection page for users](#) or [Groups](#) you can start a new search at any time to find the desired owner. Then click on the login or the name in the corresponding list entry to place the new owner in the entry field.

If you would like to specify an owner, you should check whether the new owner has the global permission to administer users and groups, and if necessary, grant him or her this permission.

Group Members

In this area of the form, you can determine which users are to be members of the new group:

A new user group initially has no members. You can assign a user to the new group by clicking on the *Add* button. This opens the user selection page on which you can use the search function to find the desired user. If you click on a user's login in the search result, the Content Manager moves the login to the *Users in Group* list. Repeat the process for all the users to be placed into the new group.

In order to remove users from the group, select them from the *Users in Group* list and click on the *Remove* button. Subsequently, the selected user no longer appears in the list.

Global Permissions

You assign global permissions to the group in this lowermost area of the form:

You can grant a group global permissions by selecting the corresponding permissions from the *Available Permissions* list and clicking on the arrow pointing to the right. The permissions assigned to the group are displayed in the *Group Permissions* list.

In order to revoke the group's permissions, select the corresponding permissions in the *Group Permissions* list and click on the arrow pointing to the left. The permissions removed from the list are moved back to the *Available Permissions* list.

In certain circumstances, other global permissions which you can select are also shown in the *Available Permissions* list. How additional permissions can be used in the CMS is described in section [Global Permissions](#).

Confirm the entries for the new group with *OK*. The Content Manager then displays the view page of the new group.

4.2.2 Changing Group Properties and Deleting Groups

If you would like to edit group properties, please first find the desired group in the *Groups* section using the search function. If you then click on the name in the list entry of the group, the CMS displays the view page of the desired group. Here, you click on the *Edit* button to open the *User Management - Groups - Edit* form:

In this form, you can edit all the properties of the desired group (except the group name). The form corresponds to the form used to [create a group](#).

Click on *OK* to accept the changes for the group. The Content Manager then displays the view page of the group.

In order to delete a group, first find the respective user group in the *Groups* section using the search function and click on the delete button to the right of the list entries. The group can also be deleted via its view page. To do this, click on the name of the desired group in the search result and then on the *Delete* button on the view page. The Content Management Server asks you to confirm the deletion:

If you click on *OK*, the group is permanently deleted as long as it is not the default group of a user or several users. In this case, you can only delete the group after you have assigned another default group to the user or users.

4.3 The User Fields Section

User fields are used to supplement the user data stored by the Content Management Server with additional information such as a telephone number or the name of the department in which the user is employed.

If you would like to create, edit or delete user fields, click on the *User Fields* tab on the *User Management* page. The Content Management Server displays the *User Fields* section:



You can use the search function in the *User Fields* section to have user fields displayed. How it functions is described in section [User Management](#).

A list corresponding to your search criteria is displayed in the main area of the section. This list contains, according to the search text, all the user fields or only those whose name contains the search text. The list entries each contain the name and the type of user field.

The number of user fields displayed in the list in the main area of the section depends on your personal preferences. You can move forward and backward by clicking on *next* or *previous* as long as the list contains more entries than you have set as default. Additionally, you can move to other results pages by clicking on one of the numbers in the ascending sequence above or below the list.

If you click on the name of a user field listed, the Content Manager displays all the available information concerning this user field:



The name and type are displayed one below the other on the view page of the user field. Additionally, the possible values of the user field are displayed for user fields of the types *Selection* or *Mutliple Selection*.

4.3.1 Creating User Fields

To create a user field, click on either the view page of a user field or in the *User Fields* section on the button *New*. The form *User Management - User Fields - New* is opened:

Name

Enter the name of the user field in the *Name* entry field. Do not use `login` as a name.

Type

Select the field type from the popup menu *Type*. The available field types are *Selection*, *Date*, *Multiple Selection*, and *String*.

Click *OK* to create the field. The Content Manager then displays the view page of the user field.

In order to determine the allowed values of a field of *Selection* or *Multiple Selection* type, open the *User Management - User Fields - Edit* by clicking on the *Edit* button:

Enter all the values the field can have in the multiple-line *Values* entry field and click *OK* to complete your input. Please note that you can only enter one value in each line of the input field.

A user field of the type *Selection* or *Multiple Selection* is now completely defined and the Content Management Server re-displays the view page of the new user field.

4.3.2 Editing or Deleting User Fields

You can subsequently change the available field values for a user field of types *Selection* or *Multiple selection*. To do this, first find the desired field in the *User Fields* section using the search function and click on its name in the corresponding list entry. The Content Management Server then displays the view page of the user field. Click on the *Edit* button to open *User Management - User Fields - Edit* :

User Management - User fields - Edit

Edit user field "Department" ?

Name: Department
Type: Selection
Values
One value per line

Development
 Management
 Marketing
 Sales
 Services

Ok Cancel

Here you can edit the possible values of the field by entering the new values one after the other in the *Values* entry field. If necessary, you can of course delete values as well.

Click on *OK* to accept the edited field values and to return to the view page of the user field.

If you would like to delete a user field, use the search function in order to display it in the list of search results in the main area of the *User Fields* section. Click then on the delete button to the right of the list entry or switch to the view page of the desired user field by clicking on the name, and click there on the *Delete* button. The Content Management Server asks you to confirm the deletion:

User Management - User fields - Delete

Delete user field "Abteilung" ?

Do you really want to delete this user field?

Name: Abteilung
Type: Selection

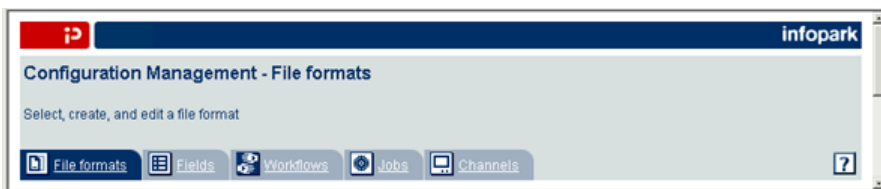
Ok Cancel

The user field is permanently deleted when you click on *OK*.

5

5 Configuring the Content Manager

In order to create file formats, fields, workflows, jobs, or channels, select *Extras > System configuration* from the Content Navigator menu. The *Configuration Management* page with the sections *File Formats*, *Attributes*, *Workflows*, *Jobs*, and *Channels* is opened. You can switch to a section by clicking on the corresponding tab.



File formats, fields, workflows, and channels are part of the so called runtime configuration. The configuration values can only be edited by a user if he or she has the permission *Edit global settings*. Jobs, however, can only be created, edited, or deleted by users with global administration permission.

5.1 The File Formats Section

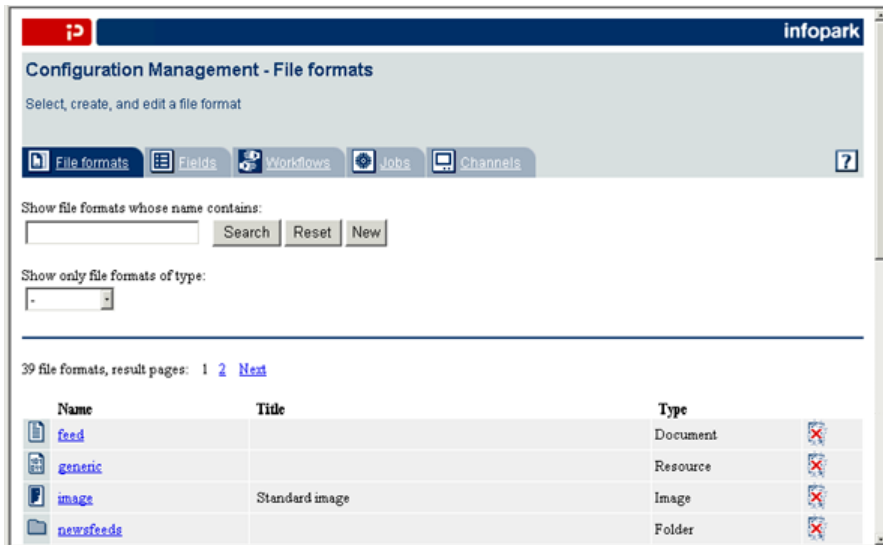
All files in the Content Management Server are created on the basis of file formats. A file format defines, among other things, the following properties of the files which are based on them.

- File type and file name extension;
- custom fields and field sets, in which fields can be grouped together;
- a global creation permission: only users who have this permission may create files based on this file format;
- valid formats for files contained in folders;
- the name of a main content layout - a layout file which is always processed when a main content of a file based on the file format is exported or displayed as a preview;
- several functions: a function is Tcl code, which is executed by the Content Management Server when an application-specific event occurs, e.g. when a version has been edited.

To a certain extent, therefore, file formats define regulations which have to be observed by the Content Management Server when you create a new file, or modify or export the version of a file.

In order to be able to define file formats, you must have the permission to edit global settings.

If you would like to create, edit or delete a file format, switch to the *File Formats* section by clicking on the *File Formats* tab on the *Configuration Management* page.



This page is also displayed if you click the *Browse* or *Add* button in a dialog in order to select a file format. In this case, the *New* button is not present and a *Cancel* button is provided in the upper area of the page allowing you to cancel the selection action if required.

In order to display file formats in the main area of the section, please use the search function.

If you would like to only display those file formats whose names contain a particular search text, enter the search text in the *Show file formats whose name contains* field and click on the *Search* button. A list of file formats whose names contain the search text is then displayed in the main area of the section. In order to change the search text, enter the new search text in the *Show file formats whose name contains* field and click again on the *Search* button. If you would like to have all available file formats displayed, leave the entry field empty and click on the *Search* button.

You can also refine the search by entering the file type as an additional search criterion. To do this, select one of the possible file types from the *Show only file formats of type* popup menu.

The list entries of the search result each contain the name, the title and the file type of a file format.

The number of file formats displayed in the list in the main area of the section depends on your personal preferences. If the list contains more entries than you have set as default, a sequence of numbers in ascending order will appear above and below the list. Each number refers to an additional results page which you can display by clicking on the number. You can also move forwards and backwards (*using next or previous*).

If this page was displayed because you clicked the *Browse* or *Add* button in a dialog, you can now select a file format by simply clicking its name.

If you click on the name of a file format listed, the Content Manager displays all the available information concerning this format on the view page of the file format:

5.1.1 Defining File Formats

To create a new file format, click on the button *New* either in the *File Formats* section or on the view page of the file format. The form *Configuration Management - File Formats - Create* is displayed:

Here you can determine the properties of the new file format. Whether the files of this format are to be marked as new on the live server by default when they are committed or released can only be set by editing the file format (see also [Editing and Deleting File Formats](#)) after it has been created.

The form consists of several areas which are described in the following.

General Properties

You determine the general properties of the file format in the uppermost section of the form.

Active

Using this option, you determine whether the user is allowed to create files based on this format. Use this option when a file format is to be available and disable it when the file format is not to be used, but may not be deleted because files exist that are based on this format.

Name

Enter the name of the new file format in this entry field. The name cannot be subsequently changed.

Title

Enter here the title of the file format in the language in which you configured your personal preferences. You can also enter the title of the file format in other languages by switching to the *Configuration Management - File Formats - Multilingual Titles* form via the *Other Languages* button.

Enter the title in the respective language in the corresponding entry field and confirm your input with **OK**.

If no title is available in the language the user configured in his or her personal preferences, the Content Management Server displays the name of the file format. The languages (and therefore the determinable language-specific titles) are dependent on the installation.

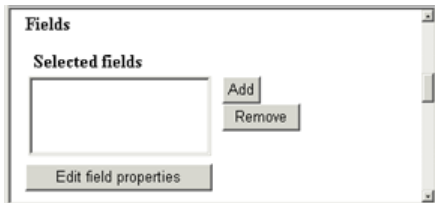
Type

In this section of the form, you determine the type of files created on the basis of this file format.

Determine the file type by selecting the corresponding option. The selected file type determines the type of all files based on this format and cannot be subsequently changed.

Fields

In this section of the form, you can define the fields of the new file format and input field properties, i. e., for example, preset fields with values.



Fields

Here you select the [custom version fields](#) assigned to the draft versions of all files based on this file format.

A new file format initially has no custom version fields assigned to it. You can assign fields to a file format by clicking on the *Add* button to find the field on the [field selection page](#) with the aid of the search function. The Content Manager moves the name of the selected field to the list of fields when you click on the name of the field in the corresponding list item of the search result. Repeat the procedure until you have moved all the desired custom fields to the *Selected Fields* list.

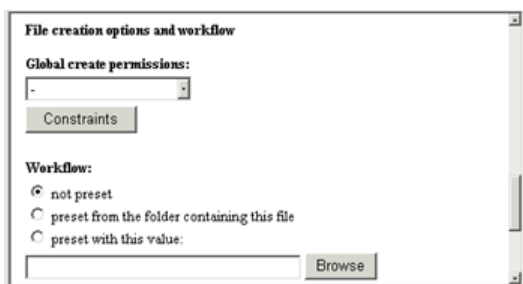
If you would like to remove fields from the *Selected Fields* list, select them in the list and then click on the *Remove* button. The selected custom fields are then removed from the *Selected Fields* list.

Edit Field Properties

You can preset the fields you have selected with values so that upon creation of a file based on this format the fields of its draft version are already preset with these values. Additionally, you can set a main content layout for the new file format. To do this, click on the [Edit Field Properties](#) button.

File Creation Options and Workflow

In this area of the form you can bind the use of the file format to a global permission. Furthermore, for file formats of the *Folder* type you can define valid formats for subfiles and the creation check for subfiles as well as set a workflow for the subsequent files of this format:



Global Create Permissions

Here, by selecting the corresponding option, you can determine which global permission a user needs to be able to create files based on this file format. If no global creation permission is set, all users may use this file format, provided that in the folder containing the file they possess the file-specific permission to create subfiles.

Constraints

If you click on the [Constraints](#) button, a form opens in which you can set valid subfile formats - exclusively for file formats using the file type *Folder* - and enter the Tcl code as the creation check for subfiles.

Workflow

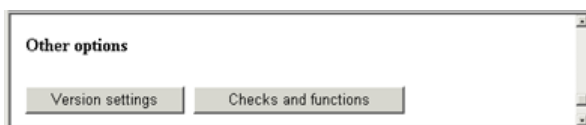
Here you can assign a workflow to a file format. If you select the *not preset* option in the selection field, the files based on this format have no workflow. By selecting the *preset from the folder containing the file* option, you determine that files which are created on the basis of this file format are to run through the same workflow as the folder containing them.

If you would like to assign your own workflow to the file format, select the *preset with this value* option. Enter the name of the workflow in the entry field. Alternatively, you can click on the *Browse* button to switch to the selection page for workflows. You can limit the search from the start by entering a part of a workflow's name in the entry field and afterwards clicking on the *Search* button. On the [selection page](#) you can start a new search at any time to find the desired workflow. Then click on the name of the workflow in the corresponding list item to move it to the input field.

You can create and edit workflows in the [Workflows](#) section of the *Configuration* area.

Other Options

Other options for file formats are grouped together in this area of the form. Here you can set several checks and functions. Furthermore, you can determine the valid file name extensions as well as the available editors for the main content of files of this format:



Version settings

Here you can define the valid file name extensions for files of the format concerned. You also have the possibility here to determine the editors to be available as main content editors on the Content Navigator page of a file of this format. To do this, click on the [Version Settings](#) button.

Checks and Functions

If you would like to define checks and functions for the file format, click on the corresponding button to switch to the *Configuration Management - File Formats - Checks and Functions* form.

Confirm your input for the new file format with *OK* to move to the view page of the new file format.

5.1.2 Editing Field Properties

There are two possibilities to open the form to edit the field properties of a file format: If you are about to create a file format, click on the *Input Field Properties* button in the *Fields* area in the corresponding form. However, if you would like to preset the fields of an existing file format, first find the desired file format in the *File Formats* section using the search function. Then click on the file format name in the list entry to switch to the view page of the file format. To be able to open the editing form for file formats, click on the *Edit* button. Then switch to the *Fields* area of the form and click on the *Edit Field Properties* button to move to the corresponding form:

Configuration Management - File Formats - Field Properties
Edit field properties of "Article"

Name: Article
Title:

Standard fields

Title

☐ mandatory
☒ not preset
☐ preset from the folder containing this file
☐ preset with this value:

Exclude from export

☒ not preset
☐ preset from the folder containing this file
☐ preset with this value:
☐

File extension

Mandatory Fields

By activating the *mandatory* option, you can define for each field whether it has to be set to a valid value so that a file can be committed or released. If the editor of the files based on this file format have to, e.g. assign a value to the *Title* field, then select the *mandatory* option for the *Title* field.

Fields of the types *String*, *Text* or *HTML* are considered valid when the user has entered any string as a value. Date fields are considered valid when the numbers entered can be interpreted by the Content Management Server and are within permitted ranges. For the fields of the *Selection* or *Multiple Selection* types, at least one of the predefined values must have been selected so that they are considered as set.

To make a version's completeness dependent on other criteria, you can use the file format property *Completion Check*.

Presetting Options

The values of fields in a file format definition can be preset. In doing so, the fields already have preset values when a file based on this format is created.

All custom fields as well as the built-in CMS fields in the table below can be preset. Please note that *Exclude from export as well as the sort criteria, the sort mode, and direction can only be preset in version 6.7.1 or later. The workflow cannot be preset on this page but on the superordinate page.*

Field with presetting option	Folder	Document	Image	Resource	Layout
Title	•	•	•	•	•

Exclude from export	•	•	•	•	
File extension	•	•	•	•	
Main content	•	•	•	•	•
Sort criteria, mode, direction	•				
Channels	•	•	•	•	
Workflow	•	•	•	•	•

By means of a selection field, you can also specify whether and in what manner a field is preset with a value. If you select the *not preset* option in the selection field, the field in the draft versions of new files based on this format has no value. The field is therefore not preset.

By selecting the *Preset from the folder containing the file* option for a field, you define that the fields are to be preset with the field values from the folder in which the files concerned are contained (the fields inherit the values from this folder).

If you would like to preset fields with constant values, mark the *preset with this value* option in the selection field. Here, enter or select which values the fields are to have when the draft version of a file based on this file format is created. Depending on the [field type](#), different input elements are available for selecting or entering the value. These input elements do not necessarily correspond to the input fields you have set for the respective fields and which users can use to edit the field values.

Special Main Content Options

The option *automatically computed using this main content layout* is additionally available for the field *Main Content*. If you select this option, enter the name of the main content layout in the entry field. A main content layout ("body template") is a layout that is used whenever the main content of a version is requested (for example, in an NPSOBJ instruction during the export or in the preview). Main Content [layouts](#) are used to unify the look and feel of documents. A main content layout cannot be specified for file formats that define layouts.

Alternatively, you can also preset the main content of the version of a file using the *preset with this value* option. This option is only displayed if you have specified a file name extension. You can select the file name extension from the [valid extensions](#) you have defined for the file formats.

Use this feature, for example, to give new folders or documents a constantly identical main content. The editor of a file can of course change the main content or the value of each field at any time as long as no main content layout is specified in the file format.

Click on *OK* to accept the field presets.

5.1.3 Creation Constraints

Creation Constraints are preset file formats for files in a folder as well as a skript which is executed whenever a file is created in a folder. These constraints can only be defined for folders. You might, for example, want to define that in folders based on the *Press* format only files based on the *Press release* and *Short news* formats may be created. By means of the creation check for files in the folder you can additionally make sure that a file is only created if particular conditions are met, for example, if the maximum number of files in the folder has not been reached yet.

There are two ways for opening the form in which the creation constraints can be defined. If you are about to create a file format, click on the *Creation Constraints* button in the *File Creation Options and Workflow* area in the corresponding form. However, if you would like to define or modify the constraints in an existing file format, localize the file format first in the *File Formats* section using the search function. Click on the file format name in the list item to open the view page of the file format and there on the *Edit* button to switch to the editing form for file formats. Here, click on *Creation Constraints* button in the *File Creation Options and Workflow* area of the form to open the corresponding form.

Valid File Formats for Files in the Folder

For a file format that defines files of type *Folder*, you can determine which file formats the subfiles of this folder may be based on. You can determine several valid subfile formats. To do this, you can click on the *Add* button. This opens the [selection page for file formats](#) in which you can use the search function to find the desired file format. The Content Manager moves the name of the desired file format to the *Valid formats for subfiles* list when you click on the name of the file format in the corresponding list entry. Repeat the process for all the file formats to be moved to the list.

If you would like to remove a file format from the list, mark the desired format and click on the *Remove* button.

If no file formats are specified, subfiles may be based on any format.

Creation Check for Files in the Folder

If you define a file format of the *Folder* type, you can enter Tcl code in the *Creation Function for Subfiles* field. This code is then processed by the Content Management Server every time a subfile is to be added to a folder. The Content Management Server only creates the file when the code has set the `result` variable to a value not equal to 0 (zero). If the code is called by the Content Management Server, the `objId` variable is set to the file ID of folder containing the file, and the name of the file format on which the new file is based is stored in the `subObjClassName` variable. `result` has the preset value 1. Write operations are not permitted in this and other functions.

Please confirm the changes by clicking the *OK* button.

5.1.4 Version Settings

The version settings include the name extensions the exported files are allowed to have as well as the editors available for editing the main content of the draft version.

There are two possibilities to open the form in which these settings can be modified: If you are about to create a file format, click on the *Edit and Export* button in the *Additional Options* area in the corresponding form. However, if you would like to edit the *Edit and Export* options in an existing file format, find the file format first in the *File Formats* section using the search function. Then click on the name of the file format in the corresponding list entry to switch to the view page of the file format. Here, click on the *Edit* button to switch to the edit form for file formats. In the *Additional Options* area of the form, click on the *Edit and Export* button to open the corresponding form.

Valid File Extensions

In the *Valid File Extensions* list, you can select the name extensions that output files are allowed to have when they are created during export. At the same time, the name extensions you selected determine the permitted name extensions of the files that can be uploaded as a main content.

If you create a new file format, the file name extensions that are preset as valid for the file type you have selected are displayed in this form. If you create a file format and after confirming your entry for the valid file name extensions you select a different file type, the Content Management Server will not allow you to create the file format.

In the standard installation, the common file name extensions are predefined. System administrators can configure the file name extensions available using the `mimeType`s entry in the [content](#) section of the system configuration.

To set the valid file name extension of the file format, select the desired extensions in the *Unselected File Name Extensions* list and click on the arrow pointing to the right. The selected file name extensions are moved to the *Selected File Name Extensions* list. If you would like to remove extensions

from the list, mark them and click on the arrow pointing to the left to move them back to the *Unselected File Name Extensions* list.

If you do not specify any name extensions, all extensions which the file type supports are valid when a file is imported. For example, the *image* file type supports the name extensions *gif*, *png*, *jpg* and *jpeg*.

Available Editors

Several main content editors are available in the Content Navigator: an *Internal Editor*, a *Local Application*, an *HTML Editor* and *TinyMCE* (up to version 6.7.3 *Microsoft HTML Editor* is available instead of *TinyMCE*). Which main content editor can be selected in the file format definition depends on the file type of the file format. The main content of a draft version of types *Folder*, *Document* or *Layout*, can be edited using all the available editors. Files of the *Image* type can only be edited using a local application. Local applications and the internal editor are available for *Ressources*.

Available Editors

You can offer the editorial employees five editors in any combination for files with this format. Please note that binary data can only be edited using a local application. The editors are:

- **Internal Editor**
The internal editor is a multi-line input field with which you can edit the HTML text of the main content.
- **HTML Editor**
Select this editor to start the optionally integrated HTML editor from the Content Navigator. This editor is a Java applet to edit the main content in HTML text mode or WYSIWYG mode.
- **TinyMCE** (CMS Fiona 6.8.0 and later)
A JavaScript-based WYSIWYG editor for the main content.
- **Microsoft HTML Editor** (up to CMS Fiona 6.7.3)
Using this editor, the editorial staff can edit the main content of files with Microsoft's Internet Explorer 6. In editing mode the Internet Explorer displays most HTML text as it would be displayed after the export.
- **Local Application**
This editor is a Java applet which gives the user the possibility to edit the main content using any application installed on the local computer. Each user can set the path to the local application to be used for each file name extension in his or her personal preferences. If the editor is selected from the Content Navigator, the Java applet starts and opens the application associated with the file name extension.

Click on *OK* to accept the input.

5.1.5 Checks and Functions

Checks and functions allow you to hook into particular actions performed with versions. This is done by means of Tcl code.

There are three checks and functions that can be specified in file formats: the completion check, the content assignment function, and the or workflow modification function. Two ways to open the form in which they can be edited exist: If you are about to create a file format, click on the *Checks and Functions* button in the *Other Options* section in the corresponding form. However, if you would like to modify them in an existing file format, localize the file format first in the *File Formats* section using the search function. Click on the file format name in the list entry to switch to the view page of the file format and there on the *Edit* button to switch to the editing form for file formats. Here, click on

the *Checks and Functions* button in the *Other Options* section of the form to open the corresponding form:

The screenshot shows a web interface titled 'Configuration Management - File Formats - Functions' with the 'infopark' logo in the top right. Below the title bar, there is a link 'Edit functions' and a help icon. The form contains the following fields:

- Name:** newsfeeds
- Title:** Newsfeed folder
- Version assignment function:** A text input field with a dropdown arrow.
- Workflow modification function:** A text input field with a dropdown arrow.
- Completion check:** A text input field with a dropdown arrow.

Version assignment function

The Content Management Server calls up the version assignment function before it sets a version's field values. This occurs after you have edited the fields in a field set or the main content of a content. The function is called after the [value assignment functions](#) of the individual fields have been executed. Your Tcl code only has access to the Content Management Server data in read-only mode. The function code can use the following variables:

- **modifiedAttributes:** The list of version fields and their values that are to be changed by the write operation, encoded as name-value pairs. The blob of the content is not contained in this list.
- **contentId:** The ID of the content whose fields are to be changed.
- **inBlobFile:** The name of the file in which the function finds the blob in case it has been modified. If the blob has not been changed, this variable is undefined.
- **outBlobFile:** The name of the file to which the function must write the blob if it is changed. This file does not exist when the function code is entered. If the function creates it, the Content Management Server sets the blob to its contents.
- **result:** On successful execution, this variable should be set to 1, and otherwise to 0 by the function to signal the execution status to the Content Manager.
- **messages:** If **result** is 0, the Content Manager interprets the value of **messages** as an error message to be output.

You can use your code to check the consistency of the field values and, if necessary, change the *modifiedAttributes* list. Please note that the main content of a version is referenced separately via the *inBlobFile* and *outBlobFile* variables. This is shown in the following example for a main content containing text:

```
if {[info exists inBlobFile]} {
    set fh [open $inBlobFile r]
    fconfigure $fh -encoding utf-8
    set blob [read $fh]
    close $fh
}
```

```

### Code modifying the blob and other fields goes here.
### Now, let's store the blob in the provided file:

set fh [open $outBlobFile w]
fconfigure $fh -encoding utf-8
puts -nonewline $fh $blob
close $fh
set result 1
}

```

If your code has set `result` to 1, the Content Management Server interprets the `modifiedAttributes` as field name-value pairs and stores the field values. Input values which are saved in an internal format (such as date values) are converted in this process. If, on the other hand, you set `result` to 0 (zero), the Content Management Server does not save the field values. In this case, the Content Management Server assumes that an error has occurred and that your code has stored one or more error messages in the `messages` list. The Content Management Server then displays these error messages.

Workflow modification function

You can use the code entered here to influence the workflow of a draft version before the content is created. It is called up when a file is created or if one of the workflow actions *Edit*, *Reject* or *Unrelease* is performed. Before the Content Management Server calls the code, it sets the `contentId` variable to the ID of the draft version concerned and the `objId` variable to the ID of the file to which the content belongs. Furthermore, the workflow parameters are stored in the `workflow` variable. These parameters result from the workflow assigned to the file format of the file. The parameters are stored in the `workflow` variable as a Tcl List. The elements of this list are:

- `editGroups editGroups`
editGroups is the list of user groups that are to edit the content.
- `signatureDefs signatureDefs`
signatureDefs is the list of signature definitions. Each definition is a list made up of two elements, the first element being the name of a signature field and the second element the name of the user group whose members are permitted to provide the signature.
- `allowsMultipleSignatures multiSig`
multiSig contains 0 if it is not permissible for the same user group to sign a content more than once. Otherwise, the parameter contains the value 1.

Using your Tcl code, you can modify the workflow definition by saving it in the `workflow` variable. The following example illustrates how different workflows can be used depending on the path:

```

set path [obj withId $objId get path]
switch -regexp $path {
/intranet {set workflow {allowsMultipleSignatures 1 editGroups Intranet_Editors
signatureDefs {{Sig_Chiefs1 Chiefs1}}}}
/internet {set workflow {allowsMultipleSignatures 1 editGroups Internet_Editors
signatureDefs {{Sig_Chiefs2 Chiefs2}}}}
}

```

Completion check

Here you can enter Tcl code which the Content Management Server will execute after a draft version was modified. Among other things, the values of fields can be checked for consistency here.

The Content Manager Server only calls the code if all other completion checks have been successfully concluded or if the reasons for incompleteness need to be determined. The Content Management

Server does not allow any write operations (such as changing a field value). However, variables can be set.

The following global variable is set by the Content Management Server prior to executing the code:

- `contentId`: the ID of the version to be checked.

The result is returned by means of the following (global) variables:

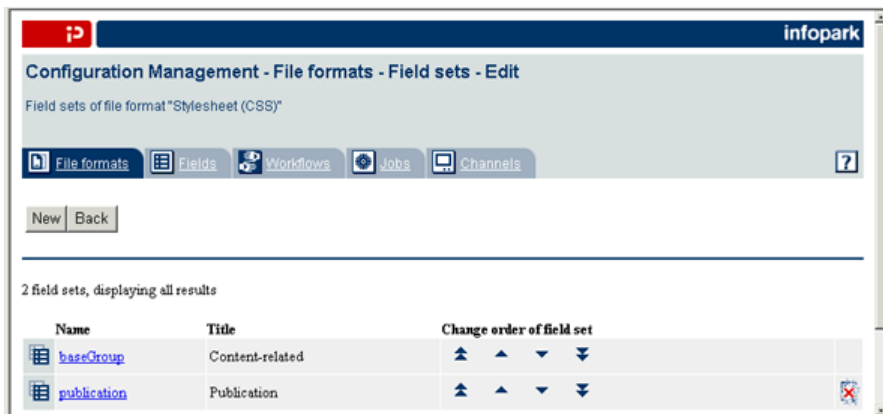
- `result`: 0 (zero) indicates that the version is incomplete. Other return values will be interpreted as *complete*.
- `messages`: A list of messages indicating the reasons for incompleteness (string array)

After you have determined the checks and functions of the file format, click on *OK* to save the changes.

5.1.6 Configuring Field Sets

The fields you have added to a file format can be grouped together in field sets. You can group the fields in such a way that editors can enter the field values in sensible combinations when they edit a file's content. The field sets are displayed in the details section of the Content Navigator. What is displayed can be [configured](#) by an administrator.

In order to create a field set or to change the order of the field sets, click on the *Show Field Sets* button in the display of the file format properties. Content Management Server now lists the field sets defined in the file format:

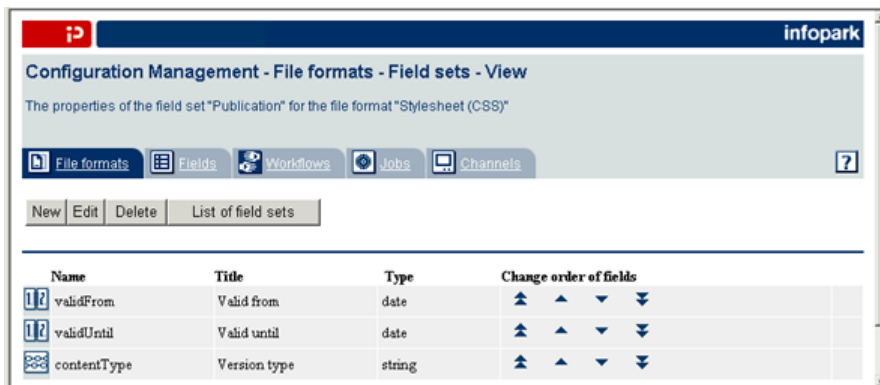


When a new file format is created, only the base group, which contains all the fields, is present. If you create other field sets, you can always assign them the fields currently available in the base group. A field will be automatically removed from the base group when it is assigned to a different group. The base group therefore always contains the fields that have not been assigned to any other group. The base group can neither be deleted nor is it possible to manually remove fields from this group.

After a new file has been created using the Content Navigator, the fields contained in the base group are offered to the editors for editing. Remove fields from this group or add fields to it in order to have the editors edit the most important fields immediately after file creation.

To change the order of the field sets, click on the corresponding image in the *Change Order of Field Sets* column. You can move a field set to the beginning or the end of the field set list using the double arrows pointing upwards or downwards respectively. Click on the single arrow pointing upwards or downwards to move the corresponding field set one position up or down respectively.

Click on the name of a field set to display the view page of this field set:



The fields assigned to this group are displayed here. The order in which the fields are displayed defines the order in which they will be listed in the field set when you open the section of this field set in the Content Navigator.

You can change the order of the field by clicking on the corresponding image in the *Change Order of Fields* column. You can move a field to the beginning or the end of the field set list using the double arrows pointing upwards or downwards respectively. Click on the single arrow pointing upwards or downwards to move the corresponding field one position up or down respectively.

Creating Field Sets

To create a new field set, click on the button *New* in the field set list or on the field set's view page. The Content Management Server displays the *Configuration Management - File Formats - Field Sets - Create* form:

Name:

The name cannot be changed after creation!

Title:

Fields:

Available fields	Fields in this group
Body	
Channels	
Version type	
Title	
Valid from	

Enter the name of the new group in the *Name* field. You can also determine the title of the group in the language configured in your personal preferences, or optionally in the languages for which your Content Management Server has been set up.

To do this, click on the *Other Languages* button to open the *Configuration Management - Field Sets - Multilingual Titles* form:

Enter here the multiple-language titles of the field sets and confirm your input with **OK**. The title the Content Management Server displays in the HTML user interface depends on the language the user has configured in his or her personal preferences. If no title is available in the language the user has configured in his or her personal preferences, the Content Management Server displays the name of the field set instead.

Selection of Fields

Initially, a new field set has no fields assigned to it. Select the desired field from the *Fields in base group* list and click on the arrow pointing to the right.

The selected fields are moved to the *Fields in this group* list. If you would like to remove fields from the list, mark the respective field and click on the arrow pointing to the left to move it back to the *Fields in base group* list.

Click on **OK** to create the field set. The Content Manager then displays the view page of the new field set.

Editing and Deleting Field Sets

In order to edit or delete a field set, first localize the file format to which the field belongs using the search function in the *File Formats* section and then click on the name of the desired file format in the search entry to switch to the file format properties view page. If you click here on the *Show Field Sets* button, the Content Manager displays the list of configured field sets. Switch to the view page of the desired field set by clicking on the field set name in the corresponding list entry of the field set page.

To be able to edit the properties of a field set, click on the *Edit* button. The Content Manager now shows the *Configuration Management - Field Sets - Edit* form:

In this form, you can edit all the properties of the desired field set (except the group name). The form contains the same data as the form for creating a new field set.

Click on *OK* to accept the changes. The Content Management Server then re-displays the view page of the field set.

You can delete a field set by clicking on the delete button to the right of the list entry of the desired group in the field set list or by switching to the view page of the field set and clicking on the *Delete* button there. The delete procedure needs to be confirmed in a safety query:

The screenshot shows a web browser window with the title bar 'infopark'. The main content area has a header 'Configuration Management - File formats - Field sets - Delete'. Below the header, it says 'Delete field set "Publication"'. A red warning message asks 'Do you really want to delete this field set?'. Below this, the 'Name' is 'publication' and the 'Title' is 'Publication'. At the bottom, there are 'Ok' and 'Cancel' buttons.

If you confirm the safety query with *OK*, the field set will be permanently deleted. The fields assigned to it are added back to the base group.

5.1.7 Editing and Deleting File Formats

In order to edit or delete a file format, first switch to the *File Formats* section in the *Configuration* area. Find the file format using the search function and click on the name of the desired file format in the list entry to open the view page of the file format. Click on the *Edit* button to open the *Configuration Management - File Formats - Edit* form:

The screenshot shows a web browser window with the title bar 'infopark'. The main content area has a header 'Configuration Management - File formats - Edit'. Below the header, it says 'Edit file format "Newsfeed-Ordner"'. The form is divided into sections. The 'General Properties' section includes a checked 'Active' checkbox, 'Name: newsfeeds', and a 'Title:' label with an empty text box and an 'Other languages' button. The 'Type: Folder' section includes an unchecked checkbox 'Mark as new on the live server'. At the bottom, there is a 'Fields' section.

In this form, you can edit the file format, i. e. edit the selection of the fields assigned to it, for example. The form contains almost the same data as the form for [creating a new file format](#). The name of a file format and the file type can, however, no longer be changed.

In addition to the options available when a file format is created, you can specify here for all files except layouts whether they are made available as news on the live server when they are committed or released. From these files, news feeds can be generated which then can be delivered by the Infopark Portal Manager, for example. This option should always be selected if documents are to be assigned to [channels](#).

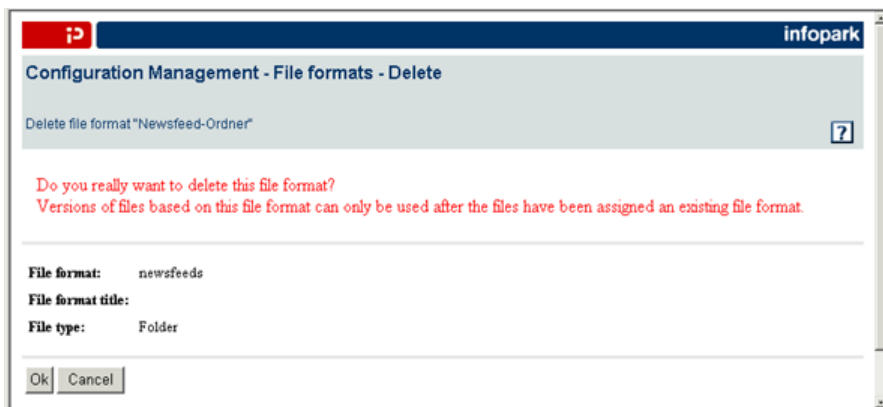
Please look up the meaning of the other input elements in the sections [Defining File Formats](#), [Edit Field Properties](#), [Creation Constraints](#) as well as sections [Checks and Functions](#) and [Version Settings](#).

Please note that changes to a file format affect all the files and their draft versions which are based on this format. If a mandatory field is added to the format for example, then all draft versions in which this field does not have a value are subsequently incomplete and cannot be committed or released.

Click on **OK** to accept the changes. The Content Management Server then re-displays the view page of the file format.

If you would like to delete a file format, localize the desired file format in the *File Formats* section using the search function. Click then on the delete button to the right of the list entry or switch to the view page of the file format by clicking on the name of the desired file format, and click on the *Delete* button there.

However, the Content Management Server does not allow you to delete a format that is still referenced by another format definition in which there are subfiles based on the format to be deleted. It is, however, possible to delete a format that files are based on. Since these files and their contents can only be re-edited after they have been assigned a different format, you have to confirm the deletion procedure in a safety query:

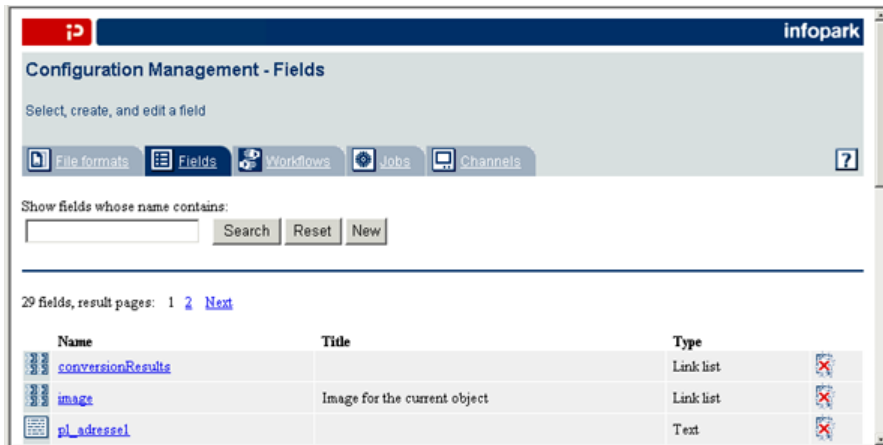


The file format is permanently deleted when you click on **OK**.

5.2 The Fields Section

Infopark CMS Fiona offers you to create custom fields, to add them to file formats and thereby make it possible to enrich your content. Every time you create a file, its draft version is equipped with the [fields](#) that are preset in the file format. You can preset the fields of a draft version with values. Later, on export, you can read out the field values from the Content Management Server and have them [inserted in the created web pages](#).

To define, edit or delete a field, switch to the *Fields* section by clicking on the *Fields* tab on the *Configuration* page. The Content Management Server displays the *Fields* section:



This page is also displayed if you click the *Browse* or *Add* button in a dialog in order to select a field. In this case, the *New* button is not present and a *Cancel* button is provided in the upper area of the page allowing you to cancel the selection action if required.

Please use the search function in the *Fields* section to have fields displayed.

If you would like to only display the fields whose names contain a particular search text, enter the search text in the *Show fields whose name contains* field and click on the *Search* button. A list of fields whose names contain the search text is then displayed in the main area of the section. The list entries each contain the name and the type of the field. In order to change the search text, enter the new search text in the *Show fields whose name contains* field and click again on the search button. If you would like to have all available fields displayed, leave the entry field empty and click on the *Search* button.

You can also refine the search by entering the field type as an additional search criterion. To do this, select one of the available field types from the *Show only fields of type* popup menu.

The number of fields displayed in the list depends on your personal preferences. If the list contains more entries than you have set you can browse the results using *next* and *previous*. Additionally, there is a sequence of numbers in ascending order above and below the list. Each number refers to an additional results page which you can display by clicking on the number.

If this page was displayed because you clicked the *Browse* or *Add* button in a dialog, you can now select a field by simply clicking its name.

If you click on the name of a field listed, the Content Manager displays all the available information concerning this field:



5.2.1 Defining Fields

You can create a new field by clicking on either the view page of a field or on the button *New* in the *Fields* section. The form *Configuration Management – Fields – Create* is opened:

Name










Enter here the name of the field. The name cannot be subsequently changed. Please note that field names must not begin with a digit and that there are names which are used internally by the *Content Manager* and therefore cannot be given to fields. The Content Management Server generates an error message if you try to create a field with one of these names. The following names are reserved:

anchors, archivedContentIds, blob, blobLength, body, bodyTemplateName, channels, children, codeForActionPreview, codeForPreview, codeForSourceView, codeForThumbnail, collection, contentIds, contentType, createNewsItem, displayTitle, editor, encodedBlob, encodedExportBlob, exportBlob, exportCharset, exportFiles, exportMimeType, externalAttributes, externalAttrNames, externalAttrTypeDict, frameNames, freeLinks, getKeys, hasChildren, hasSuperLinks, height, id, isCommitted, isComplete, isEdited, isReleased, isRoot, lastChanged, linkListAttributes, mimeType, name, next, nextEditGroup, nextSignGroup, noPermissionLiveServerRead, objClass, objectId, objectsToRoot, objType, parent, path, permissionCreateChildren, permissionLiveServerRead, permissionRead, permissionRoot, permissionWrite, prefixPath, previous, reasonsForIncompleteState, releasedVersions, setKeys, signatureAttrNames, sortValue, subLinks, superLinks, superObjects, suppressExport, textLinks, thumbnail, title, toclist, validControlActionKeys, validCreateObjClasses, validFrom, validObjClasses, validPermissions, validSortKeys, validSortOrders, validSortTypes, validUntil, version, visibleExportTemplates, visibleName, visiblePath, width, workflow, workflowComment, workflowName, xmlBlob.

Type

Select the field type from the *Type* popup menu. The type determines which values can be assigned to a field. It cannot be changed after field creation. In the Content Management Server the following field types are available:

Field Type	Description
------------	-------------

	Date	Fields of type <i>Date</i> record date values. The Content Management Server saves all date entries in so-called canonic form. In this form, a date is represented as a 14-digit string made up of - starting from the left - the year, month, day, hours, minutes and seconds. With the exception of the four-digit year, all components are two-digit. For example, the date <i>8.18.2001</i> is saved as <i>20010818000000</i> .
	String	Fields of type <i>String</i> can contain any text. The Content Management Server converts all whitespace characters (such as tabs and line breaks) to spaces and removes leading and trailing spaces when it saves strings. The length of a string is limited to 250 characters. No coding translation is applied to strings.
	Text	The value of a field of type <i>Text</i> can contain text of any length. If the value is read out during a preview or an export, all HTML-specific characters are translated so that they remain readable. For example, the pointed bracket <i><</i> becomes <i>&lt;</i> .
	HTML	The values of fields of type <i>HTML</i> can contain HTML texts of any length. In contrast to the <i>Text</i> type, the Content Management Server does not translate the <i>HTML</i> field value to conform to HTML. This means that the characters are used as they were entered. Furthermore, all links contained in the field values are transferred into the Content Management Server's link management.
	Markdown	Like HTML, however additionally supports the Markdown syntax in the field value. This field type is available from Version 6.6 of the CMS. In the preview and during the export, Markdown text is only evaluated if the Rails Connector is used.
	Selection	In a field of type <i>Selection</i> , the possible values that the Content Management Server user can assign to the version field are predetermined from the start.
	Multiple Selection	The possible values are similarly predetermined for enumeration fields with multiple selection. When the Content Management Server user determines the field value, however, he can select not only one, but several of the predefined values.
	Link List	Fields of the <i>Link List</i> type contain free links, i.e. links which associate a content with any file or any external document. You can store any number of free links in a <i>Link List</i> type field.
	Signature	Signature fields are special fields whose values are set by the Content Management Server when the user signs a file. The value of a signature field is made up of two parts, the point in time at which the signature was provided, and the login name of the user who provided it. You require signature fields to be able to define workflows in which files must be signed.

Title

Enter here a meaningful title in the language which you configured in your personal preferences. As a rule, it makes it easier to understand the meaning of a field value.

If your Content Management Server is set up for multiple language use, you can switch to the *Configuration Management – Fields – Edit – Multilingual Titles* form via the *Other Languages* button and here enter titles in each language in the corresponding fields.

Please confirm your input with **OK**. The title the Content Management Server displays in the HTML user interface depends on the language the user has configured in his or her personal preferences.

Description

Use the input field *Description* to give other users additional information about the field and its function. The description is available as help text, when the field's value is edited via the Content Navigator. If your Content Management Server is set up for multiple language use, you can switch to the *Configuration Management – Fields – Multilingual Descriptions* form by clicking on the *Other Languages* button and then enter descriptions here in the respective foreign language:

Please also confirm your input with **OK**.

Searchable on Content Manager

Select this option to index the values of this field in the optional Infopark Search Server and to be able to search for them in the Content Management Server during an extended search.

Searchable on Live System

Select this option to index the values of this field in the Infopark Search Server and to be able search for them on the Live Server during a search.

Click then on **OK** to save the field definition. The Content Manager then displays the view page of the new field.

5.2.2 Editing Field Properties

To edit the properties of a field – for example its description or its input field type – first find the field using the search function in the *Attributes* section. Switch to the view page of the field properties by clicking on the name of the field in the list entries. Click on the *Edit* button on the view page of the field. The Content Management Server displays the *Configuration Management – Fields – Edit* form:

Configuration Management - Fields - Edit
 Edit field "Newsletter recipients" ?

Name: ip_newsletterRecipients
Type: Multiple selection

Title:
 Other languages

Description:
 Other languages

☒ Searchable in Content Manager
☐ Searchable in live system

Values:
 One value per line

Functions:
Functions

Input fields:
 Multiple selection field Details

Ok Cancel

You can change the field definition at any time (provided you have the permission to change global settings). It is therefore possible, for example, to shorten or supplement the list of enumeration values for the *Selection* or *Multiple Selection* field type. However, such changes can result in draft versions becoming incomplete because the value of one of their fields is no longer in the list of permitted values.

5.2.3 General Properties

The uppermost area of the form contains the same data as the form for creating a [new field](#). However, the name and type of the field cannot be changed.

5.2.4 Special Properties

The variable properties of a field depend on its type. The most important field properties are:

- **Selection/multiple selection: Values**

You can enter the possible values for fields of the types *Selection* or *Multiple Selection*. To do this, enter all the values the field can have in the multiple-line *Values* input field. Please note that only one value per line can be entered.

- **HTML: Wanted tags**

For fields of the *HTML* type, you can define here the tags the user is permitted use. Specify the tags (without the enclosing angle brackets) in the *Wanted tags* multi-line input field. Enter only one tag per line.

- **Linklist:**

Minimum and maximum number of links (from version 6.7.0)

For fields of the *Linklist* type, you can enter here the minimum and maximum number of links the field accepts. The CMS checks whether the number of links contained in the field is in the specified range. If this is not the case, the corresponding file cannot be released.

From version 6.7.2, link lists may be empty without causing an error if the demanded minimum number of links is not equal to zero. To force a link list to be non-empty, make it mandatory in the file format.

5.2.5 Functions

The behavior of fields can be extended by means of custom functions. If you would like to define one of the functions for fields, click on the Functions button to open the *Configuration Management – Fields – Edit – Functions* form:

Value Display Function

The value display function is a Tcl routine used to calculate the value of a field to be displayed in the HTML user interface. The function is called when the value of the field is set. Two values are passed to the function:

- **value:** The field value. The Tcl routine can read out this value and reset it if necessary
- **contentId:** The ID of the version containing the field value. Via the version parameter `objectId` (`content withId contentId get objectId`) you can, for example, read out the file format belonging to the file in order to display different values depending on the file format.

The Tcl routine can return error messages by means of the `error` command. If no Tcl code is given for the value display function for a field, the original value of the field is displayed. The routine only has access to the Content Manager data in read-only mode. In particular, the field value is not changed by setting the `value` parameter.

Value Assignment Function

The value assignment function is a Tcl routine which the Content Management Server executes each time the value of the field is changed. Use this feature for example to check whether a field has a valid value and to change the value if necessary. To the Tcl routine two arguments are passed:

- **value:** The field value. The Tcl routine can read out this value and reset it if necessary
- **contentId:** The ID of the version containing the field value. Via the version parameter `objectId` (`content withId contentId get objectId`) you can, if required, read out field values of the file to which the version belongs.

The Tcl routine can return error messages with the `error` command. The routine does not have write access to the data in the Content Management Server.

5.2.6 Input Fields

For all fields except those of type *Signature* (and *Link list* up to version 6.0.x), you can define the type of input field the user may use to set the field value. To do this, select the type of input field from the popup menu.

The input field types the Content Management Server offers in the *Input Fields* popup menu depend on the type of the field you have defined. The following types are defined in the Content Management Server:

Input Field Type	Description
Single Line Text Input Field	This type is available for fields of types <i>String</i> , <i>Text</i> , <i>HTML</i> and <i>Date</i> .
Multiple Line Text Field	This type is available for fields of the <i>String</i> , <i>Text</i> and <i>HTML</i> type.
Password Field	This type is only available for strings. Characters entered into this type of field are displayed as another, always identical character. The field is used for password entry.
Popup Menu	This popup menu is displayed when the user can determine the value of a field of the <i>Selection</i> type.
Radio Button	For fields of the <i>Selection</i> type, the Content Management Server displays a list of possible values. A radio button appears in front of each list element with which the element can be selected. Only one element in the list can be selected.
Multi-Selection Box	For fields of the <i>Multiple Selection</i> type, the Content Management Server displays a list of permitted values, from which the editors can choose as many as desired.
Checkboxes	For fields of the <i>Multiple Selection</i> type, the Content Management Server displays a checkbox in front of each permitted value which the editors can activate. You can activate as many checkboxes as desired to select the corresponding values.
HTML Editor	Users can comfortably edit the values of HTML fields using the HTML editor available as an option for the Content Manager. To allow you to edit field values, an <i>Edit</i> button appears on the HTML user interface next to the field value.

Microsoft Editor	As with the HTML editor, users can comfortably edit the values of HTML fields with this editor. The Microsoft editor makes use of the editing mode of Microsoft's Internet Explorer 6. To allow users to edit individual field values, an <i>Edit</i> button is provided on the NPS Navigator page.
Local Application	This editor serves to edit the values of HTML fields using a local application installed on the client. This application can be specified in the user's personal preferences. The application for the <i>html</i> file name extension is used.
Read-only	Select this type when the user is not to be able to enter the field value in the HTML user interface. The <i>Read-only</i> type is available for all field types.
Link List Field (configurable from version 6.5.0)	Fields of the <i>link list</i> type can be edited using the dialog Edit Link Lists . From version 6.7.0, a starting folder for selecting the link destination can be specified. Furthermore, you can restrict the files to be offered for selection by specifying their file formats. Enter the names of the formats as a space-delimited list.
Wizard	Fields of all types except <i>signature</i> can be edited using a wizard. Linklists can be edited by means of a wizard in version 6.5 or later. To the wizard specified the standard arguments plus the additional arguments <code>wizard.attributeName</code> (name of the field to edit) and <code>wizard.contentId</code> (ID of the draft version) are passed. The wizard itself needs to store the field values that have been modified.

To determine the details for the input field selected, click on the *Details* button. A form appears in which you can determine the details for the respective input field – e.g., the width of text fields:

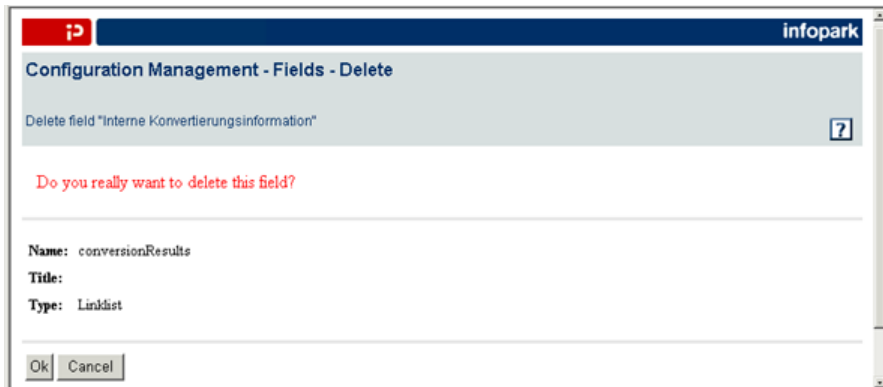
The screenshot shows a web-based configuration window titled "Configuration Management - Fields - Edit - Input Field". It has a header bar with an "infopark" logo. Below the header, there's a sub-header "Edit input field" with a help icon. The main area contains the following fields: "Name: ip_newsletterRecipients", "Title: Newsletter recipients", and "Number of rows:" followed by an empty text input box. Below these is a checkbox labeled "empty value allowed" which is checked. At the bottom left are "Ok" and "Cancel" buttons.

Enter the desired details of the input field and click on *OK* to return to the *Configuration Management – Fields – Edit* form.

Click *OK* to save the changed field properties. The Content Management Server then re-displays the field's view page.

5.2.7 Deleting Fields

If you would like to delete a field, first find it in the *Fields* section using the search function. Click then on the delete button to the right of the corresponding list entry or the name of the field to switch to the view page of the field, and click on the *Delete* button there. The Content Management Server asks you to confirm the deletion:



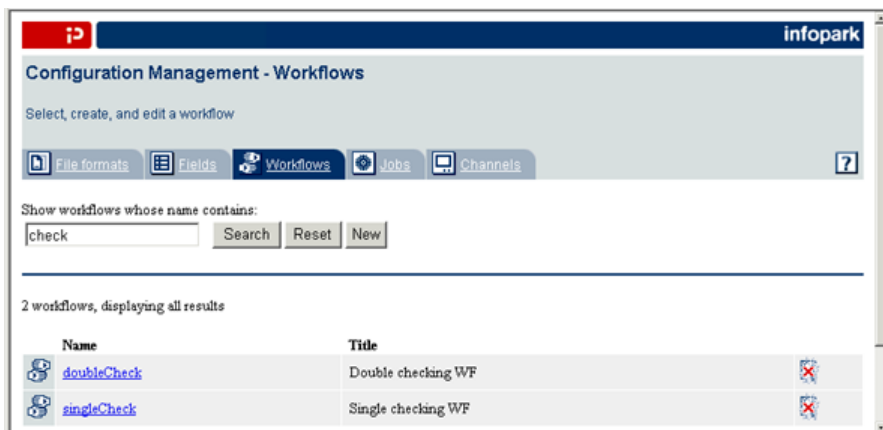
The field is permanently deleted when you confirm the safety query with **OK**.

The Content Manager does not however allow you to delete a field still used in file formats. In this case, first remove the field from the respective file formats and then delete it.

5.3 The Workflows Section

Workflows determine the editing and verification steps a file is subject to on its way to being released. The Content Management Server automatically assigns the workflow defined in the file format upon which the file is based to every draft version of a newly created file (see Workflows).

If you would like to create, edit or delete a workflow, switch to the *Workflows* section by clicking on the *Workflow* tab in the *Configuration* area.



This page is also displayed if you click the *Browse* or *Add* button in a dialog in order to select a workflow. In this case, the *New* button is not present and a *Cancel* button is provided in the upper area of the page allowing you to cancel the selection action if required.

Use the search function of the *Workflows* section to display the workflows. If you would like to only display the workflows whose names contain a particular search text, enter the search text in the *Show workflows whose name contains* field and click on the *Search* button. A list of workflows whose names contain the search text is then displayed in the main area of the section. If you would like to search for workflows which contain a different search text, enter this search text in the *Show workflows whose name contains* field and click again on the search button. If you would like to have all available workflows displayed, leave the entry field empty and click on the *Search* button.

The number of workflows contained in the list depends on your personal preferences. There might be a sequence of numbers in ascending order above and below the list. Each number refers to an

additional results page which you can display by clicking on the number. You can also switch forwards and backwards (using *next* and *previous*).

If this page was displayed because you clicked the *Browse* or *Add* button in a dialog, you can now select a workflow by simply clicking its name.

If you click on the name of a workflow listed, the Content Manager displays all the available information concerning this workflow on the its view page:

Configuration Management - Workflows - View

Properties of workflow "Double checking WF"

File formats Fields **Workflows** Jobs Channels

New Edit Delete

Name: doubleCheck
Title: Double checking WF

Workflow is enabled: Yes

Edit steps: Step	Group
1	Redakteure

Sign steps: Step	Group	Field
1	Designer	Signature
2	Chefredakteure	Signature 2

Multiple signatures allowed: No

5.3.1 Defining Workflows

You can create a new workflow by clicking on the view page of a workflow or on the button *New* in the *Workflows* section. The form *Configuration Management – Workflows – Create* is opened:

Configuration Management - Workflows - Create

Create a new workflow

General Properties

☒ Workflow is enabled

Name:

The name cannot be changed after creation!

Title:
 Other languages

Edit steps:

The form is divided into the areas *General Properties*, *Edit Steps* and *Sign Steps*, which are described in the following.

5.3.2 General Properties

You determine the general properties of the workflow in the first section of the form:

☒ Workflow is enabled

Name:

The name cannot be changed after creation!

Title:

Active

This option allows you to determine whether a workflow should be usable. If the workflow is to be usable, select this option. If you currently do not require this workflow definition but are not sure that it will not be used later, deactivate the workflow.

Name

Enter the name of the new workflow in this entry field. The name cannot be subsequently changed.

Title

Enter here the title of the workflow in the language in which you configured your personal preferences. If your Content Management Server is set up for multiple language use, you can switch to *Configuration Management – Workflows – Edit – Multilingual Titles* form by clicking on the *Other Languages* button:

infopark

Configuration Management - Workflows - Edit - Multilingual Titles

Define title in other languages

German title:

English title:

Spanish title:

French title:

Italian title:

Enter here the title in the respective foreign language in the corresponding field. Please confirm your input with *OK*. The title the Content Management Server displays in the HTML user interface depends on the language the user has configured in his or her personal preferences. If there is no title in the language the user has set, the name of the workflow is displayed as the workflow title.

5.3.3 Edit Steps

In the second area of the form you can determine which user groups take part in editing files:

Edit steps:

Step	Group	
1	NewsEditors (news editors)	Browse
2	NewsReaders (news readers)	Browse
3	NewsMasters (head of news department)	Browse

Insert step: after 3 Remove step: last

Sign steps: before 1, after 1, after 2, after 3

Here you can define which user groups are to edit a file which runs through this workflow. To do this, click on the *Sign Steps* button. If you have defined an edit step, you can insert a workflow step at any position using the *Insert Step* button and the popup menu to the right. Enter the name of the group of editors in the entry field. Alternatively, you can click on the *Browse* button to switch to the selection page for groups. You can limit the search from the start by entering a part of a group's name in the entry field and afterwards clicking on the *Search* button. On the [selection page](#) you can start a new search at any time by entering the search term in the entry field and clicking on *Search*. Then click on the name of the desired user group in the corresponding list entry to move the group name to the entry field. Repeat the procedure until you have all the edit steps necessary for your workflow.

If you have defined too many edit steps by mistake or you would like to remove a group from the edit workflow, you can remove the excessive steps by selecting the edit step to be deleted from the popup menu to the right of the *Remove Step* button and then clicking on this button.

5.3.4 Sign Steps

In the third area of the form you can determine which user groups take part in verifying files:

Sign steps:

Step	Group	Attribute
1		

Insert step: after 1 Remove step: last

☐ Multiple signatures allowed

You determine the name of the user group whose members provide the signature, i. e. those who may sign a content the same way as described above. The Content Management Server saves the signature in a signature field. Enter the name of the field in the entry field for each verification step. Alternatively, you can click on the *Browse* button to switch to the selection page for fields. You can limit the search from the start by entering a part of a field's name in the entry field and afterwards clicking on the *Search* button. On the [selection page](#) you can start a new search at any time to find the desired field. You move the name of the field to the entry field by clicking on the field name in the corresponding list entry.

Whenever a user signs a file with this workflow, the Content Management Server saves the date, time and login name of the user signing the file in the signature field. Signature fields are created in the same manner as other [custom content fields](#).

Multiple Signatures Allowed

Enable this option if, for this workflow, employees of the groups entitled to provide signatures for a content may sign more than once for this workflow. If, e.g. a verifier is a member of two of the

workflow's verification groups, he or she may provide signatures in both verification steps when this options is enabled. File administrators are always permitted to sign the contents of their files.

Click on *OK* to conclude workflow definition. The Content Manager then displays the view page of the new workflow.

5.3.5 Editing or Deleting Workflows

If you would like to edit the properties of a workflow, first find the workflow using the search function in the *Workflows* section and click on the name of the workflow in the corresponding list entry to switch to the view page of the workflow. Click on the *Edit* button. The Content Management Server displays the *Configuration Management – Workflows – Edit* form:

The screenshot shows a web browser window with the infopark logo. The page title is "Configuration Management - Workflows - Edit". Below the title bar, it says "Edit workflow \"Double checking WF\"". There is a help icon (?) on the right. The form is divided into sections:

- General Properties:**
 - ☒ Workflow is enabled
 - Name: doubleCheck
 - Title: Double checking WF (text input field)
 - Other languages (button)
- Edit steps:**
 - Step Group:

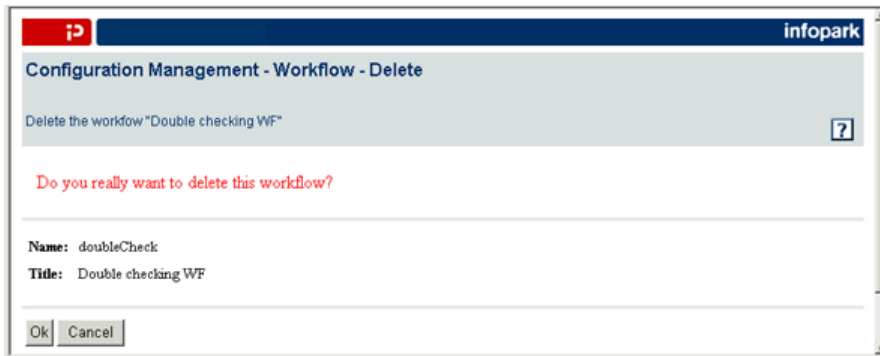
1	editors	Browse
---	---------	--------
 - Insert step after 1 Remove step last
- Sign steps:**

Here you can make changes to the workflow properties. The form contains the same data and options as the form for creating a new workflow. The name of the workflow cannot however be changed. Please look up the meaning of the input elements in the sections *General Preferences*, *Edit Steps*, and *Verification Steps* under [Defining Workflows](#).

Click on *OK* to accept the changes to the workflow properties. The Content Management Server then displays the view page of the workflow.

If you have changed a workflow, the changes will first become effective when a new draft version is created. This is always the case when a new file is created, a released content is edited, or a committed content is rejected.

If you would like to delete a workflow, locate it in the *Workflows* section using the search function. Click then on the delete button to the right of the corresponding list entry or the name of the workflow to switch to its view page, and click on the *Delete* button there. The delete procedure has to be confirmed in a safety query:



The workflow is permanently deleted when you click on **OK**.

The Content Manager does not however allow you to delete a workflow still used in a file format. In this case, assign a different workflow to the corresponding file format and then delete the workflow as described above.

5.4 The Jobs Section

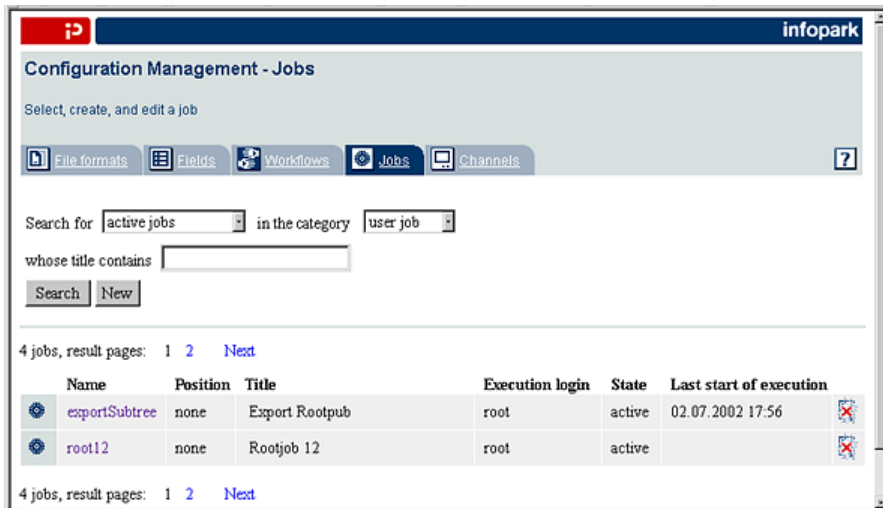
There are predefined and user-defined (custom) [jobs](#) available. You can define, change, or delete user-defined jobs just as you like. Predefined jobs, however, can not be created or deleted, and it is also not possible to modify their code. Of course the execution schedule of all jobs can be set as desired. The following predefined jobs exist:

Job name	Task
systemPublish	Transfers file update information to the Template Engine and initiates on the live server the export and the publishing process.
systemTransferUpdates	Transfers file update information to the Template Engine .
systemSendReminderNotifications	Sends e-mails for due reminders .

In the *Jobs* configuration section you can create and edit jobs if you have the global administration permission. If required, you can execute jobs, i. e. place them in the execution queue. You can also cancel jobs, i.e. remove them from the queue. It is however not possible to cancel a job whose Tcl script is currently being run. In this case, the job is at position 0 in the queue and cannot be removed from it.

Please note that jobs are executed in server mode only (i. e. not when the Content Manager is run using the -single command line parameter).

If you would like to create, edit, delete, execute or cancel a job, or view its log, switch to the *Jobs* section by clicking on the *Jobs* tab in the *Configuration* area.



Use the search function of the *Jobs* section to display the jobs. If you would like to display jobs whose titles contain a particular search text, first select one of the job criteria (*all jobs*, *waiting jobs*, *active jobs*, or *jobs executed under my login*) as well as the job category from the popup menus. Then enter the search text in the entry field and click on the *Search* button. The list of jobs matching the search criteria is then displayed in the main area of the section. You can start a new search by entering a new text in the entry field and clicking on the *Search* button again. If you would like to have all jobs available in a category displayed, select the *All Jobs* criteria, leave the entry field empty, and click on the *Search* button.

The list entries of the search result each contain the name of the job, its position in the queue and its title, the login under which the job may be executed as well as the status of the job and the date it was last executed.

The number of jobs displayed in the list depends on your personal preferences. If the list contains more entries than you have set as default, a sequence of numbers in ascending order will appear above and below the list. Each number refers to an additional results page which you can display by clicking on the number. You can also move forwards and backwards (using *next* and *previous*).

If you click on the name of a job listed, the Content Manager displays all the available information concerning this job on its view page:

Configuration Management - Jobs - View

The properties of the job "Partially export the folder hierarchy"

File formats Fields Workflows **Jobs** Channels

New Edit Delete Start job Show log entries Show last output

Name: exportSubtree
 Title: Partially export the folder hierarchy
 Category: user job
 Comment:
 Active: Yes
 Execution permission:
 Execution login: root
 Last execution start: 14.12.2004 13:32 (server time: 14.12.2004 13:32)
 Last execution end: 14.12.2004 13:32 (server time: 14.12.2004 13:32)
 Last execution result:
 Queue position: none
 Next execution start:
 Schedule: No schedule set.
 Script: obj withPath base/intranet exportSubtree filePrefix /tap

5.4.1 Defining Jobs

If you are a superuser, you can create a new job by clicking on the button *New* on the view page of a job or in the *Jobs* section. The Content Management Server displays the *Configuration Management – Jobs – Create* form:

Configuration Management - Jobs - Create

Create a new job

Jobs

General Properties

Name:

The name cannot be changed after creation!

☒ Active

Title:

Comment:

Execution

The form is divided into the areas *General Properties*, *Execution* and *Script*, which are described in the following.

5.4.2 General Properties

You determine the general properties of the job in the first section of the form:

Name

Enter here the name of the Job. In order to prevent conflicts with system job names, the name must neither start with `system` nor with an underscore character. Furthermore, the usual rules for names apply (no special characters, only the characters 0 to 9, A to Z, a to z, and the underscore are allowed).

Active

Using this option, you can determine whether a job is to be usable, i. e. can be placed in the queue. If you do not want to use the job, deselect this option to deactivate the job.

Title

Enter the title of the new job in this entry field. Please note that the title of a job can have a maximum length of 250 characters including all character transformations (from, for example, ANSI to UTF-8).

Comment

Enter an as meaningful description of the new job as possible in this entry field. The description of a job also can have a maximum length of 250 characters including all character transformations (from, for example, ANSI to UTF-8).

5.4.3 Execution

In the second area of the form, you can determine under which login the job script will be executed and which global permission is required to be able to execute or cancel the job. Furthermore, here you can define the job's execution schedule:

Execution Login

Enter the login name of the user under whose login the job script is to be executed in this field. If you leave the entry field empty, the Content Management Server uses as the execution login the login name of the current user when the job is saved.

If you do not know the login name of the user under whose name the job script is to be executed, click on the *Browse* button. A selection page for users is displayed. You can limit the search from the start by entering a part of a user's name in the entry field and afterwards clicking on the *Search* button. On the [selection page](#) you can start a new search at any time to find the desired user. When you click on the login of the desired user in the corresponding list entry, the login name is moved to the *Login, under which the script will be executed* field.

Execution Permission

From the *Execution Permission* popup menu, select the global permission the user needs to start the job, i.e. to place it in the queue, or to cancel the job. If you do not enter a global permission, the job can be executed by all users.

Schedule

A job's [schedule](#) determines when the job is executed. New jobs initially do not have a schedule. However, you can create one by clicking the *Create schedule* button.

5.4.4 Script

In the third area of the form, you can determine which Tcl script is to be run when the job is executed. Enter the Tcl code in the entry field:

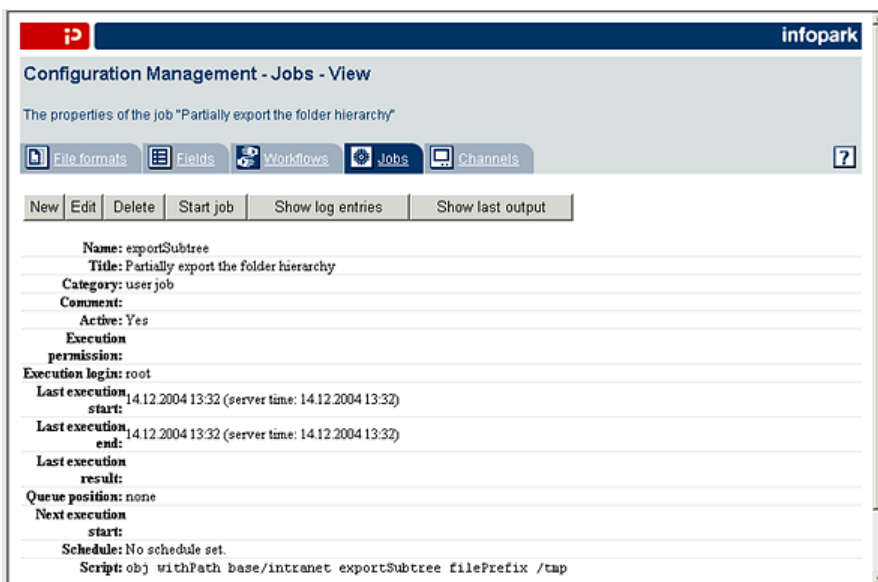


Please note that the execution result can be set by assigning a value to the `result` variable. If you want the Tcl script to output data, please do not write to the standard output device. Instead, call the `writeToJobLog` procedure and pass to it the string to be output as the only argument.

Click on *OK* to conclude the job definition. The Content Management Server then displays the view page of the new job.

5.4.5 Executing and Canceling Jobs

If you would like to execute or cancel a job, first find the desired job in the *Jobs* section using the search function and click on the name of the job in the corresponding list entry. The Content Management Server displays the definition of the job in the *Configuration Management – Jobs – View* page:



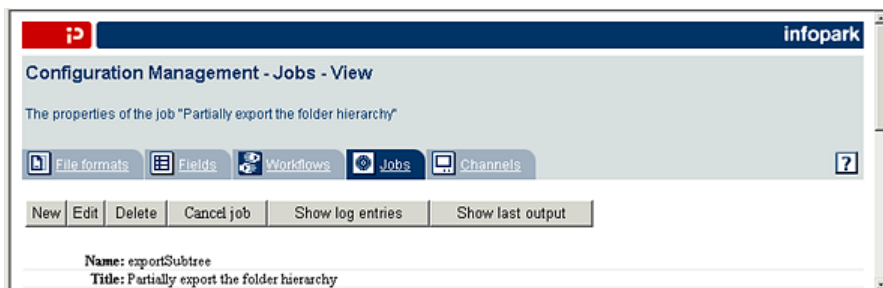
To execute a job, click on the *Start Job* button. This button is only displayed when you possess the necessary permission to execute the job, or when no execution permission has been specified for this job.

The Content Manager now asks you to confirm the execution of the job in a safety query:



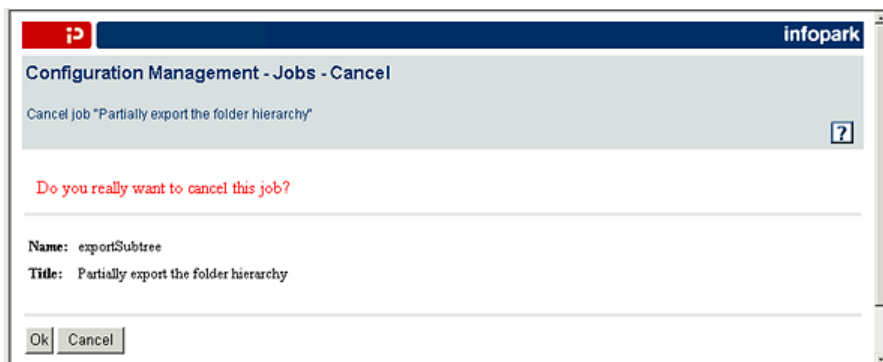
Confirm the safety query with *OK*, when you want to start the job. If not, click on the *Cancel* button to return to the job definition.

If there is a job in the execution queue, the *Cancel Job* button appears on the view page of the job. You can click on it to remove the job from the queue:



This button is only displayed when you possess the necessary permission to execute the job, or when no execution permission has been specified for this job.

The Content Manager asks you to confirm the cancellation of the job in a safety query:



If you would like to cancel the job, confirm the safety query with *OK*. If not, click on the *Cancel* button to return to the job definition.

5.4.6 Editing or Deleting Jobs

To edit the properties of a job, first find the desired job in the *Jobs* section using the search function and then click the name of the job in the corresponding list entry. On the view page of the job which then appears, click the *Edit* button to open the *Configuration Management – Jobs – Edit* form (you need to be a superuser to be able to do this):

Configuration Management - Jobs - Edit

Edit Job "Partially export the folder hierarchy" ?

General Properties

Name:
exportSubtree

☒ **Active**

Title:
Partially export the folder hierarchy

Comment:

Execution

Please look up the meaning of the input elements in the sections *General Preferences*, *Execution* and *Script* under [Defining Jobs](#).

When you have edited a job definition, the changes become effective only the next time the job is executed.

Click on *OK* to accept the changes to the job properties. The Content Management Server then displays the view page of the job.

To delete a job, first find the respective job in the *Jobs* section using the search function and click on the delete button to the right of the list entry. The job can also be deleted on its view page. To do this, click on the name of the desired job in the search result and then on the *Delete* button on its view page. The Content Management Server asks you to confirm the deletion:

Configuration Management - Jobs - Delete

Delete Job "Partially export the folder hierarchy" ?

Do you really want to delete this job?

Name: exportSubtree

Title: Partially export the folder hierarchy

The job is permanently deleted when you click on *OK*. The Content Management Server does not however allow you to delete a job which is in the queue, i.e. is currently being executed. In this case, first cancel the job and then delete it as described above.

5.4.7 Editing the Execution Schedule of Jobs

When defining or editing a job, as was described in section [Defining Jobs](#), you also have the possibility to create or edit its execution schedule. For this purpose, click the *Create schedule* button or the *Edit* button next to a schedule entry, respectively. The following form is then displayed:

Configuration Management - Jobs - Edit - Schedule

Edit the schedule of the job "Partially export the folder hierarchy"

These settings refer to the server time zone: CEST
The current time is: 14.12.2004 17:32 (server time: 14.12.2004 17:32)

Minute	Hour	Weekday	Day	Month	Year
0	0	Monday	1	January	2004
1	1	Tuesday	2	February	2005
2	2	Wednesday	3	March	2006
3	3	Thursday	4	April	2007
4	4	Friday	5	May	2008
5	5	Saturday	6	June	2009
6	6	Sunday	7	July	2010
7	7		8	August	2011
8	8		9	September	2012
9	9		10	October	2013
10	10		11	November	
11	11		12	December	
12	12		13		
13	13		14		
14	14		15		

Ok Cancel

An execution schedule can contain several entries, each of which can be edited on the page shown above. All entries refer to the server's time with its specific timezone, which is displayed at the top of the page.

In the main area of the form, you can define single or recurring executions by means of the *Minute*, *Hour*, *Day*, *Weekday*, *Month*, and *Year* lists. A single execution is specified by selecting exactly one value from each of the lists except *Weekday*.

Recurring executions can be defined by selecting several values or no value at all from a list. If no value has been selected, all the values in the list concerned implicitly count as selected. If, for example, neither days nor weekdays are specified for a job execution time, the job will be executed daily. Thus, an entry with no selection from any list has the effect that the job is executed every minute until the entry is deleted or modified. If days as well as weekdays are selected, the job is executed whenever a selected day or weekday matches. The system does not prevent you from defining jobs with execution times that never occur (June 31, for example).

Click *OK* to finally save your definition, or *Cancel* to return to the job editing page without modifying the schedule. On the editing page, the schedule entries are displayed as follows:

Schedule:

Minute	Hour	Weekday	Day	Month	Year		
0	0	all	all	all	all	Edit	Delete
30	1, 10	all	all	all	all	Edit	Delete

Add entry

Every line represents an entry and is provided with an *Edit* and a *Delete* button. If an entry has been accidentally deleted, please cancel the job editing and edit it again.

Please also take account of the information in section [Jobs](#).

5.4.8 Viewing the Job Log and Job Output

Each time a job is executed, the Content Manager (and also the Template Engine) creates a log entry for this job. In this entry, the start time and the execution result are recorded. The maximum number of log entries per job is determined by the `jobMaxLogLength` system configuration entry. If necessary, older entries are automatically deleted.

You can open the log of a job by clicking the *Show log entries* button on its view page (see [The Jobs Section](#)). The list of log entries is then displayed:



The log provides information about the time at which a job was executed and what the result of the respective execution was. The execution result can be set in the job script by assigning a value to the `result` variable.

Please note that an error is generated if a job script writes data to the standard output device. In the script, you can call the `writeToJobLog` procedure with the `message` argument to output text intended to be displayed on the output page.

In order to view the output of a job, please click the *Show output* button next to the entry concerned. The job output page is then displayed:



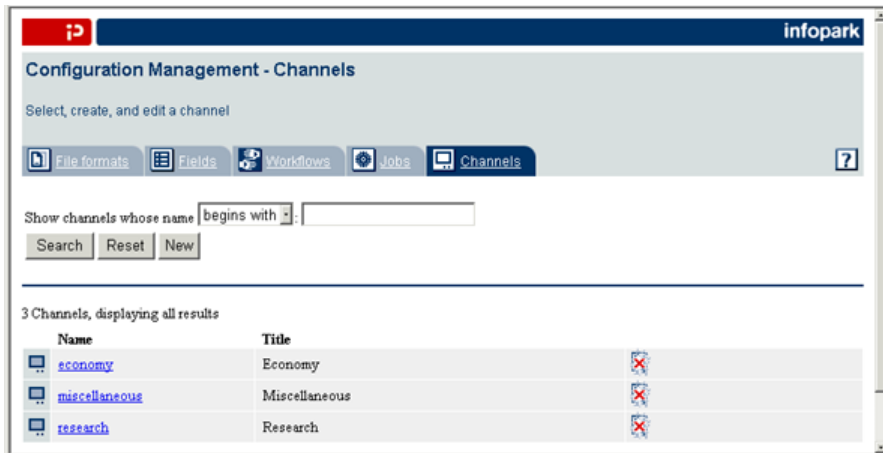
The job output page can also be opened from the view page of a job by clicking the *Show last output* button. In this case the output page always provides the current output of the job and is reloaded regularly. Additionally, a *Refresh* button (located next to the *OK* button) is available. You can use it to manually reload the page and thus have the current output displayed.

Click *OK* to return to the list of log entries. Here you can also click on *OK* in order to return to the view page of the job.

5.5 The Channels Section

In the *Channels* section you can define channels to which the editors can then assign files (news articles, for example). By using the Portal Manager, channels and the messages assigned to them can be displayed on the live server on a per-user basis.

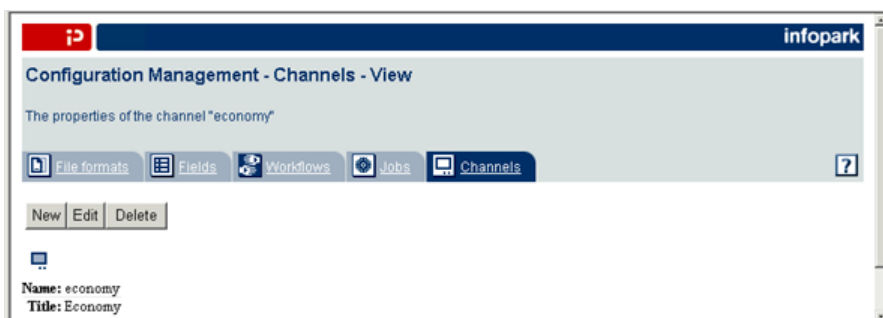
In order to create, edit, or delete a channel, please switch to the *Channels* section by clicking on the *Channels* tab in the *Configuration* area.



Please use the search function of the *Channels* section to display the channels that have already been created. Please enter a part of the name of the channel you are searching for in the entry field and click the *Search* button. The list of channels whose name contain the search text is then displayed in the main area of the section. You can start a new search at any time by entering a new text in the entry field and clicking on the *Search* button again. If you would like to have all channels listed, leave the entry field empty, and click on the *Search* button.

The list entries of the search result each contain the name and the title of the channel. The number of channels displayed in the list depends on your personal preferences. If the list contains more entries than you have set as default, a sequence of numbers in ascending order will appear above and below the list. Each number refers to an additional results page which you can display by clicking on the number. You can also move forwards and backwards using *next* and *previous*.

If you click on the name of a channel listed, the Content Manager displays the available information concerning this channel on its view page:



5.5.1 Defining Channels

You can create a new channel by clicking on the button *New* on the view page of a channel or in the *Channels* section. The Content Management Server displays the *Configuration Management – Channels – Create* form:

Using this form the following channel properties can be defined.

Name

Enter here the name of the channel. By means of periods in the name, channel hierarchies can be formed. For example, the channel *weather.europe.france* is contained in the channel *weather.europe* which in turn is contained in the channel *weather*.

Title

Please enter in this entry field the title of the new channel in the language you configured in your personal preferences. Please note that the title of a channel can have a maximum length of 250 characters including all character transformations (from, for example, ANSI to UTF-8).

If your Content Management Server is set up for multiple language use, you can switch to *Configuration Management – Channels – Edit – Multilingual Titles* form by clicking on the *Other Languages* button:

Enter here the titles in the respective foreign languages in the corresponding fields. Please confirm your input with *OK*. The title the Content Management Server displays in the HTML user interface depends on the language the user has configured in his or her personal preferences. If there is no title in the language the user has set, the name of the channel is displayed as the channel title.

Click on *OK* to conclude the channel definition. The Content Management Server then displays the view page of the channel.

5.5.2 Editing or Deleting Channels

To edit the properties of a channel, first find the desired channel in the *Channels* section using the search function and then click on the name of the channel in the corresponding list entry. On the view

page of the channel which appears, click on the *Edit* button to open the *Configuration Management – Channels – Edit* form:

Please look up the meaning of the input elements in the section [Defining Channels](#).

Click on *OK* to accept the changes to the channel properties. The Content Management Server then displays the view page of the channel.

Please note that files can not be committed or released if a channel to which the files are assigned is deleted.

To delete a channel, first find it in the *Channels* section using the search function and click on the delete button to the right of the list entry. The channel can also be deleted on its view page. To do this, click on the name of the desired channel in the search result and then on the *Delete* button on its view page. The Content Management Server asks you to confirm the deletion:

The channel is permanently deleted when you click on *OK*. This causes files, which have not been released, and that are assigned to this channel to become incomplete.

6

6 Layouts

6.1 The Function of Layout Files

Layouts serve to efficiently realize a website with a homogenous design. Frequently used elements such as logos and navigation elements need only be set up once in a layout and can then be used for all files in the folder hierarchy located at or below the level of the layout in the hierarchy.

For exporting every file of the folder or document type (to whose file extension the `text/html` MIME type has been assigned) the Content Manager and the Template Engine use a layout file as the base layout. Also, when exporting a file only its released content is considered. Which layout file is used depends on the `defaultTemplate` user preference in the Content Manager, while in the Template Engine the `defaultTemplate` system configuration entry determines the name of the base layout file. Being able to configure the base layout file to be used for exports makes it possible to maintain several layouts at the same time. A layout can be activated spontaneously and as desired by making the relevant layout file the base layout prior to the export.

The base layout file is processed like a program. The resulting website is created by processing this program for every released file exactly once. Every time the program is processed, particular instructions in the base layout relate to the file currently being exported. Of course you can use not only these instructions, but also standard HTML-tags in layouts.

You can use instructions in a layout such as the base layout to access the field values of the file currently being exported, in order to create `meta` tags, links and other HTML elements from them. Furthermore, instructions are available with which you can have tables of contents (link lists), a micronavigation, sitemaps or other link structures automatically generated.

The instructions are introduced with the keyword `NPSOBJ`, which is used in the same way as an HTML tag. When processing the base layout file the Content Management Server and the Template Engine evaluate the `NPSOBJ` tag to a string, which is then written into the output file in place of the entire `NPSOBJ` tag. For example, the following instruction is replaced with the value of the custom field named `abstract`. This value is thus inserted into the generated web page:

```
<NPSOBJ insertvalue="var" name="abstract">
```

Fundamentally, the base layout is simply a special layout file which can function as initial layout upon export because of the instructions it contains. Besides the base layout you can use as many other layouts as you wish to give your website or subpages a homogenous layout.

Using an `NPSOBJ` instruction it is possible within a layout to use other layouts or the same layout. For example, from a base layout you can call respective layouts for creating the navigation elements at the

head and foot of a page, keeping the base layout as general as possible and thus making it universally applicable. All the layouts used during an export, are called a layout set.

The way in which your layouts work can be checked at any time with the [preview function](#).

Main Content layouts

A main content layout is a layout file referenced in a file format (see [Defining File Formats](#)). Upon export and when a preview page is generated it formats the main content of every file based on this format. Main Content layouts serve to define and to consequently standardize the appearance of documents. Therefore, a main content can be modified neither with an editor in the HTML user interface nor with Tcl commands if a main content layout has been specified in the file format of the relevant file. In this case, it is also not possible to import the main content from a file.

Main Content layouts are especially suitable for generating uniform documents like data sheets or product specifications. Such documents are primarily made up of field values that can be arranged and styled according to your wishes by means of a main content layout.

The main content layout used for exporting released contents or displaying them in the preview is stored directly in the contents as they are released. It is therefore necessary to release the files concerned again after a (different) main content layout has been assigned to the file format or the main content layout has been removed from it. This also applies when the main content layout itself has been modified. Draft versions, however, are affected immediately after changes have been made to the main content layout.

Please note, that when main content layouts are used on an incrementally exported live server, all files directly or indirectly referencing a main content layout must be located in the same hierarchy branch as the layout and on the same or a subordinate hierarchy level. Otherwise, the referencing files are not re-exported when the layout is updated.

6.2 Designing a Layout Concept

Before you create layouts in the editorial system of Infopark CMS Fiona, you should always develop a concept in which the structure of the website, its layout, and any possible further uses of the content are determined. In doing this the following aspects should be given particular consideration:

- Which users are responsible for creating and caring for the layouts?
- Are several layout sets required because the online publication is to be generated individually for different target groups?
- Should the layout be defined globally or do individual areas of the folder hierarchy require individual layouts?
- In which places in the layout must further layout files be provided for changing elements of the page (title images, navigation entries, automatically inserted field values, among other things)?
- In which places in the layout should automatically generated navigation entries appear?
- What information does the editorial staff require in order for the mechanisms provided for in the layouts to work (e.g. mandatory field values)?

You should place as much emphasis as possible on the flexibility of the layout concept to minimize maintenance expenditures and the costs for subsequent adjustments. This means in particular that layouts should be used consistently: a layout file should be provided in every area of the layout in which future changes are to be expected.

If already in the base layout the main content of a file is inserted using

```
<npsobj name="body" insertvalue="var"/>
```

then in the subfolder it is no longer possible to design this part of the page using local layouts. Therefore, one initially calls only a layout as a wildcard for the main content, for example in the form

```
<npsobj name="localtemplate" insertvalue="template"/>
```

and in the same hierarchy level creates a layout with the name *localtemplate* and the content

```
<npsobj name="body" insertvalue="var"/>
```

In this way further formatting can be defined as required for each folder in a dedicated *localtemplate*.

6.3 Editing Layouts

Before you edit an already released layout you should create a copy of the layout that you can revert to if the edited layout does not function according to your wishes after release. This security precaution is unnecessary if archiving is switched on in your Content Management Server.

You can create and edit layouts in the same manner as folders or documents. Links to other files in layouts are also not handled differently than links in folders or documents. Unlike files of these two types, however, the CMS always uses the entire main content of a layout file and not just the HTML code in the `body` element.

The Content Management Server and the Template Engine support so called empty-element tags in accordance with the XML specification. Such tags are empty elements at the same time. They are tags ending with a slash and the end-of-tag-character:

```
<empty-element-tag />
```

They are equivalent to elements with an opening and a closing tag and no content, i.e.:

```
<element-tag></element-tag>
```

Empty-element tags are preserved upon export. This means that the Content Manager and the Template Engine do not convert them to elements with an opening and a closing tag.

6.4 Layout Structure

6.4.1 XML and XHTML Marking

When content is to be exported as XML or XHTML the document must first be marked accordingly. For this purpose, a corresponding XML marker such as

```
<?xml version="1.0"?>
```

at the beginning of a line or XHTML marker such as

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

must be exported first. Both tags must be placed at the beginning of the exported document. They are not recognized when they are contained in processing instructions (<? . . . >) or are part of an SGML comment (<!-- . . . -->).

The markers not only serve the client (mostly the browser) to determine the type of the document but also influence the export of so called empty element tags. These tags are exported as such when at least one of the following two conditions is met:

- The empty element tag is contained in the content part (in a field) to be exported.
- The empty element tag is generated by an NPSOBJ instruction (e.g. <npsobj insertvalue="image" name="..." /> and an XML or XHTML marker was exported before.

In all other cases no empty element tags but single tags without closing slash are exported.

6.4.2 HTML Header

In the HTML header of a layout the desired `meta` elements are inserted into the web page to be generated. Especially the file name extension and the character set are important here. Usually, also the exported file's title is translated into the HTML title of the web page and further version fields into the corresponding `meta` elements:

```
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  <title><npsobj name="title" insertvalue="var"/></title>
  <npsobj name="field-name" insertvalue="meta" />
</head>
```

For each field to be translated into a `meta` element a dedicated NPSOBJ instruction corresponding to the above example must be included in the layout. Please note in the above example that the name of the field becomes the name of the `meta` element. To be able to freely determine the name of the `meta` element (independently from the field names) use an NPSOBJ tag corresponding to the following example:

```
<npsobj insertvalue="meta" name="meta-name" content="field-name" />
```

To be able to use your layouts universally, you should read out the fields intended as `meta` elements in a central layout. You can include this layout in the base layout or another layout with the following instruction:

```
<npsobj name="template-name" insertvalue="template" />
```

6.4.3 Style sheets

If you wish to include style sheets in your website, please use the form <link rel="stylesheet" type="text/css" href="styles.css">. This causes the links used to be added to the link management. Furthermore, it is important to specify the `title` tag attribute and use the same title for

all the style sheets in a set. This makes it possible for the browser to offer a set of styles under a unique title. Example:

```
<link rel="stylesheet" type="text/css" href="typo.css" media="screen" title="Standard" />
<link rel="stylesheet" type="text/css" href="navi.css" media="screen" title="Standard" />
```

6.4.4 HTML Main Content and other Field Values

You can access the main content of the exported file exactly as you access every other field. Field values are read out with an NPSOBJ tag in accordance with the following example:

```
<npsobj name="field-name" insertvalue="var"/>
```

In this it is unimportant whether you are accessing file fields (like `name`), built-in version fields (like `lastChanged` or `title`) or custom version fields.

main content (usually in a layout referenced by the base layout) into the resulting output file using the following instruction:

```
<body>
  <npsobj name="body" insertvalue="var"/>
</body>
```

Please note that it is not possible to set the value of an HTML tag attribute using an NPSOBJ instruction, for example in the following manner:

```
<!-- This is illegal -->
<A href="/" title="<npsobj insertvalue="var" name="myField"/>">
Click here</A>
```

For this purpose, please use an [@ reference](#) or a formatter that generates the complete HTML element (see [npsobj insertvalue var](#)).

Please also note that NPSOBJ instructions are only evaluated if they are contained in the main content of layout files, documents and folders and if, additionally, the MIME type of these files is `text/html`.

Furthermore, the exported pages (i. e. the layouts as well as the fields included into them) which are delivered via the Portal Manager must not contain Velocity code since the Portal Manager evaluates this code which might cause the page to be corrupted or not displayed at all. If you wish to insert Velocity code into a page, replace all `#` characters with `#` and all `$` characters with `$`.

The following table contains a selection of the file and content fields available. The full extent of the language is listed in [the syntax reference](#). Please note that the spelling of a field name in an NPSOBJ tag must correspond exactly to the actual spelling of the field name – upper and lower case is also observed.

Field name	Field value
body	The main content of the released version of the exported file.
contentType	The file name extension.

id	The file ID.
name	The file name.
path	The file's current path.
prefixPath	If the current file is a folder, then the path followed by the character "/" is returned, otherwise the path is returned.
title	The current file's title.
visibleName	If the current file is a document, the file name is returned followed by the file name extension. If it is a folder, only the file name is returned.
visiblePath	For documents this is the complete path including the file name with which the file is exported. For folders it is the complete path including the file name, followed by "/index", followed by the folder's file name extension.
lastChanged	The date of the last change to the current file. Changes to the file's content are hereby not considered. On export of a released file the last change is the release.
validFrom	The date from which the file's content is valid. This field is ignored for layouts. The value of validUntil must be empty or greater than validFrom.
validUntil	The date until which the content of the file is valid. If this value is null the document's validity is unlimited. This field is ignored for layouts.

The format in which date values are returned on export and in the preview is normally determined by the first value in the [export.validDateOutputFormats](#) system configuration entry. However, you can influence date formatting with the additional tag attribute `format="formatName"` in the NPSOBJ tag by specifying as `formatName` one of the other names defined in the system configuration entry mentioned above.

Das Format, in dem Datumswerte beim Export und in der Vorschau geliefert werden, wird normalerweise durch den ersten Wert im Systemkonfigurationseintrag [export.validDateOutputFormats](#) bestimmt. Mit dem zusätzlichen Tag-Attribut `format="formatName"` im NPSOBJ-Tag können Sie jedoch die Formatierung des Datums beeinflussen, indem Sie als `formatName` einen der anderen Formatnamen angeben, die im genannten Systemkonfigurationseintrag definiert sind.

For *Multiple selection* type fields whose value is made up from several elements, a string as separator must be specified as `separator="string"` in the NPSOBJ tag. Infopark CMS Fiona uses the separator to separate the individual elements of the field value from each other. You may specify an empty string as the separator.

The following example shows how the title of a document (built-in content field `title`) and a keyword list (custom content field `keywords`) are used as HTML `meta` elements, and how the date of the last change (fixed content field `lastChanged`) and the value of the custom field `author` are automatically inserted at the end of the HTML page created.

```
<head>
  <title><npsobj name="title" insertvalue="var"/></title>
  <npsobj name="keywords" insertvalue="meta" separator=","/>
</head>
<body>
  <npsobj name="body" insertvalue="var"/>
  <hr>
  <font size="-2">
    Last changed on
    <npsobj name="lastChanged" insertvalue="var"/>.
    Author: <npsobj name="author" insertvalue="var"/>
```

```
</font>
</body>
```

HTML text can also be generated only under particular conditions, for example only on export of folders or dependent on the value of a field.

6.4.5 Automatic Navigation Entries

Using NPSOBJ tags in layouts you can automatically have links to other exported websites created:

Link to the previous page

```
<npsobj name="previous" insertvalue="anchor">Previous</npsobj>
```

Link to the next page

```
<npsobj name="next" insertvalue="anchor">Next</npsobj>
```

Link to the parent page

```
<npsobj name="parent" insertvalue="anchor">Up</npsobj>
```

The previous or next file is determined by NPS using the sort key used in the folder containing the file (see [Entry Sorting](#)). The parent page corresponds to the folder containing the exported file in your folder hierarchy for which the navigation element is created.

Instead of the navigation texts *Previous*, *Next* and *Up* used in the examples above, you can also use images or fields of the target file by replacing the text with a reference to an image or with an NPSOBJ tag that reads out the field. For example use the title of the link-target file as a reference text by using NPSOBJ tags in the following way:

```
<npsobj name="previous" insertvalue="anchor">
  <npsobj name="previous.title" insertvalue="var"/>
</npsobj>
```

6.4.6 Micronavigation

A micronavigation is a path as HTML text in which every element except for the last is linked with the corresponding web page. Using an NPSOBJ tag you can have micronavigations automatically created, whereby you can determine the number of created elements.

```
<npsobj micronavigation="field-name" levels="3">delimiter</npsobj>
```

When the CMS processes the `micronavigation` instruction it forms the path from the base folder to the folder containing the current file, creating an `a-href` reference for every element. The link to the corresponding folder-index page is created as `href`, and the value of the field `field-name` is used in the context of the respective target folder as the link's reference text.

The individual elements separated by the entered string `delimiter` are output one after the other. You can use any separation string you wish, which means that even HTML code such as line breaks, links to images, etc. are allowed.

Using the tag attribute `levels` you can set the number of elements to be included in the micronavigation. If `levels` is a positive number, the base folder will be the micronavigation's first element. A negative value indicates that only the last elements in the path are to be used. This means that in a micronavigation constructed with `levels="1"` only the base folder will be included whereas with `levels="-1"` the micronavigation will only contain the folder containing the current file. `levels="0"` has the same effect as not specifying levels at all which means that the complete path is to be included in the micronavigation.

The following example creates a micronavigation from the titles of the parent folders and limits it to three elements. The entries are separated from one another with a slash:

```
<npsobj micronavigation="title" levels="3"> / </npsobj>
```

This NPSOBJ tag creates the following micronavigation on the web pages exported from the folder sub-hierarchy */de/company/press/archive*:

Displayed file	Created micronavigation
<i>/de</i>	Title of the base folder
<i>/de/company</i>	Title of the base folder / <i>German</i>
<i>/de/company/press</i>	Title of the base folder / <i>German</i> / <i>The company</i>
<i>/de/company/press/archive</i>	Title of the base folder / <i>German</i> / <i>The company</i>

All entries in the generated micronavigation are linked to the corresponding folder pages.

If you wish the last parent folder in the path to be displayed in the micronavigation, then the NPSOBJ attribute `levels` must be assigned a negative value:

```
<npsobj micronavigation="title" levels="-2"> / </npsobj>
```

On the web pages exported from the folder sub-hierarchy */de/company/press/archive*, this NPSOBJ tag creates the following micronavigation (a micronavigation can be inserted anywhere you like within the `body` element of a layout):

Displayed file	Created micronavigation
<i>/de</i>	Title of the base folder
<i>/de/company</i>	Title of the base folder / <i>German</i>
<i>/de/company/press</i>	<i>German</i> / <i>The company</i>
<i>/de/company/press/archive</i>	<i>The company</i> / <i>Press</i>

7

7 Automatically Generated Tables of Contents

7.1 Demands on Start Pages

The start pages in an online system play a special role in user prompting. The following demands should therefore be considered during conception and design of start pages:

- On every start page the visitor should get the most complete overview possible of the contents and structure of subordinate areas.
- Direct links to subdocuments with sufficient content descriptions should be offered to the visitor.
- The list of subdocuments must be up-to-date and complete.

With completely manually maintained tables of contents these criteria can usually only be fulfilled unsatisfactorily. Particularly in areas with frequently changing content (news, press releases, event calendars, trade-fair previews, among others) maintenance of the list of contents is work intensive and subject to error. Infopark CMS Fiona therefore makes NPSOBJ instructions available with which tables of contents can be automatically created. Their significant advantages:

- After initial creation, absolutely no maintenance is required for the table of contents.
- The entries of the table of contents automatically appear on the start page after a subfile is created and released. After a file is deleted it no longer appears in the overview.
- Specific information such as creation date, author, etc. can automatically be published by reading out the corresponding fields.
- If the fields *Valid from* and *Valid to* are given values, the file only appears in the table of contents during the publication time period.

7.2 Components of Automatically Generated Tables of Contents

7.2.1 Structure of an Automatically Generated Table of Contents

An automatically generated table of contents is created through a combination of standard HTML tags (for formatting the list) and NPSOBJ instructions:

```
<ul>
  <npsobj list="toclist">
    <li>
      <npsobj name="self" insertvalue="anchor">
        <npsobj name="title" insertvalue="var"/>
      </npsobj>
    </li>
  </npsobj>
</ul>
```

```

    </li>
  </npsobj>
</ul>

```

This code has the following effects: The entire table of contents is defined as an unordered list:

```

<ul>
  ...
</ul>

```

In the exported folder all released subfolders, documents and ressources are to be considered. All entries are placed on the page, summarized as an automatically generated table of contents:

```

<npsobj list="toclist">
  ...
</npsobj>

```

Every entry is formatted as an item of a list:

```

<li>
  ...
</li>

```

For every entry a link to the respective file itself is generated:

```

<npsobj name="self" insertvalue="anchor">
  ...
</npsobj>

```

As linked texts the titles of the files are used:

```

<npsobj name="title" insertvalue="var"/>

```

In the preview you can check at any time the way an automatically generated table of contents looks. Please note that with the `toclist` keyword only released and valid documents, folders and ressources appear in tables of contents. To include images as well, use `children`

7.2.2 NPSOBJ Parameters for Automatically Generated Tables of Contents

7.2.3 Setting the Number of Entries in Tables of Contents

You can optionally determine the file with which an automatically generated table of contents begins and how many files it contains by using parameters in the NPSOBJ tag. In this way you can, for example, restrict a list of current press releases to a maximum number.

If you wish the overview to begin at the fifth file, and restrict the list to three entries, use the following code:

```

<npsobj list="toclist" start="5" length="3">
  ...

```

```
</npsobj>
```

For a negative `start` parameter, the start position is determined beginning from the end. `start="-10"` and `length="3"` thus has the effect that from the last ten files, the first three are included in the list. Instead of `length`, you can also use `end` to specify the index of the last element.

7.2.4 Using other Fields

You are not restricted to using the titles of subfiles in the entries of automatically generated table of contents. You can enter any [predefined values](#) you like and include custom fields in your table of contents. For example you can have information about the name and type of the document, the author (custom field) and the creation date displayed right on the index page:

```
<ul>
  <npsobj list="toclist">
    <li>
      <npsobj insertvalue="var" name="name"/>.
      <npsobj insertvalue="var" name="contentType"/><br>
      <npsobj insertvalue="var" name="Author"/><br>
      <npsobj insertvalue="var" name="lastChanged"
        format="europeanDate"/><br><br>
    </li>
  </npsobj>
</ul>
```

Instead of `europeanDate` you can enter any date format you wish (see [validDateTimeOutputFormats](#)). As examples of further fields, you have `id` (file ID), `objClass` (file format), `objType` (file type), `title`, `validFrom`, `validUntil` and `version` available. Please note that Infopark CMS Fiona observes upper and lower case differences for field names.

If required you can utilize custom Tcl procedures to format field values retrieved with `insertvalue-var` instructions. This is especially helpful if you want to include scripts in your web pages. For this purpose, use the `formatter` tag attribute in the following way:

```
<npsobj insertvalue="var" name="name" formatter="procAlias" />
```

As the value of `formatter` the alias name of a Tcl procedure must be specified. The name of this procedure must have been assigned to the alias name in the system configuration entry `tclFormatterCommands`. Upon export and when preview pages are generated the `NPSOBJ` instruction is replaced with the return value of the procedure. You can learn more about the `formatter` tag attribute in section [npsobj insertvalue var](#).

7.2.5 Entry Sorting

The entries in automatically generated tables of contents are normally sorted in ascending order by file name. For each folder, however, you can individually determine the way its subfiles are to be sorted.

These settings for sorting are only effective, however, if no sort key is given in the `NPSOBJ` instruction concerned. The sort keys are determined in `NPSOBJ` tags with the parameters `sortkey1`, `sortkey2`, and `sortkey3` (primary, secondary, and tertiary criteria) as well as the modifiers `sortmodifier1`, `sortmodifier2`, and `sortmodifier3` in accordance with the following example:

```
<npsobj list="toclist" sortkey1="field-name"
  sortmodifier1="alpha ascending">
```

```
...
</npsobj>
```

Instead of `field-name` enter the name of any file or content field. In the example above, the files will be sorted in ascending order (`alpha ascending` is the default and thus does not need to be specified). Use code according to the following example to have the files sorted numerically descending instead of alphabetically ascending:

```
<npsobj list="toclist" sortkey1="field-name"
  sortmodifier1="numeric descending">
...
</npsobj>
```

As shown in the examples above, the modifiers `alpha` and `numeric` can be combined with `ascending` and `descending`. If file sorting is determined neither in the NPSOBJ tag nor by the sort keys of the folder containing the files, then the Content Management Server tries to use the field specified by means of the `content.sortKey` system configuration entry. If no sort key can be determined there either, then the Content Management Server sorts the files by their names.

The sort order also affects [automatically generated navigation entries](#).

7.3 Controlling Entries

Normally all released subfolders, documents and ressources appear in a table of contents created with `list=toclist`. An overview created with `list=children` on the other hand contains all subfiles, i. e. also images and layouts.

Now and then one may wish, however, not to include all files in the overview, but only files of a particular type or with a particular field value. Perhaps you also would like files of a particular format to be displayed differently.

This can be done using conditional export instructions.

7.3.1 Querying File Types

The entries in an automatically generated table of contents can be restricted to a particular file type by using the following query (e.g. all ressources for download). It is also possible to exclude files of a particular type from the table of contents.

Use the following code to include only files of type `document` in the table of contents:

```
<npsobj name1="objType" value2="document" condition="isEqual">
...
</npsobj>
```

In this example `name1` specifies the name of the field whose value is to be compared with the value of the constant `value2`. In binary conditions the subscripts 1 and 2 are used to define the order of the operands. This is necessary because there are operations (`hasPrefix`, `hasSuffix` and `matches`) in which the order of the operands is significant. Since the order of tag attributes is meaningless in HTML, the tag attribute names must be used to indicate the order. This is the purpose of the subscripts.

Analogously to the example above you can use `value2="publication"` to make only folders, and `value2="generic"` to make only ressources appear in the table of contents. To allow only folders and documents to appear, exclude files of type *ressource*:

```
<npsobj name1="objType" value2="generic" condition="isNotEqual">
...
</npsobj>
```

Alternatively you can use the parameter `negate` to reverse a condition. The following code has the same result as the code in the previous example:

```
<npsobj name1="objType" value2="generic" condition="isEqual" negate>
...
</npsobj>
```

If you wish to compare two field values, please use `name1` and `name2` as NPSOBJ parameters.

7.3.2 Querying Field Values

Files can also be included in automatically generated tables of contents or formatted there, dependant on field values. The following code checks whether the custom field `ShowInTocList` has a value.

```
<npsobj name="ShowInTocList" condition="isNotEmpty">
...
</npsobj>
```

With `condition="isEmpty"` you check whether a field value is empty. However if you wish to compare the value of a field such as `title` with a string, for example, use code in accordance with the following example:

```
<npsobj name1="title" value2="A File Title" condition="isEqual">
...
</npsobj>
```

If the field value is exactly equivalent to the entered string, the CMS processes the code before `</npsobj>`. Conversely, you can use `condition="isNotEqual"` to test for non-equivalence.

You can test whether the value of a field begins or ends with a particular string with `hasPrefix` or `hasSuffix`. The following example checks whether the title of the file begins with the string *Datasheet*:

```
<npsobj name1="title" value2="Datasheet" condition="hasPrefix">
...
</npsobj>
```

Query date fields

If you query date fields in conditional NPSOBJ instructions, then you should consider that the CMS applies the comparison to the canonic form of the date. In this format a date is represented as a 14-place string, beginning from the left with the year, month, day, hour and seconds, whereby except for the four-place year all entries are two-place. For example the date Dec. 18, 2010 is saved as 20101218000000.

Use `hasPrefix` as a comparison operator to compare only the relevant characters of the saved date with a string. Please note that the parameters in the comparison operations `hasPrefix` and `hasSuffix` are not commutative: the respective index with which `name` and `value` are provided determines the position of the parameter. The index is mandatory.

In the following example the files are determined that were last changed on May 15, 2011:

```
<npsobj condition="hasPrefix" name1="lastChanged" value2="20110515"> ...
</npsobj>
```

All files that were last changed in May 2011 can be found with the following code.

```
<npsobj condition="hasPrefix" name1="lastChanged" value2="201105"> ...
</npsobj>
```

Please note that every date value in the CMS is related to the timezone GMT. Thus if you wish to check whether a file is valid based on its validity period, you must use the value of the time comparison translated into GMT as `value2`.

7.3.3 Combined Comparisons

If, depending on the value of a field, many different texts are to be output, it is convenient to use a `switch` instruction instead of many `condition` instructions. With `switch`, field values can be compared more efficiently and *if-then-else* structures can be used in layouts. This is shown in the following example:

```
<npsobj switch="objClass">
  <npsobj casecond="isEqual" value="newspub">
    <npsobj insertvalue="template" name="newsnav" />
  </npsobj>
  <npsobj casecond="isEqual" value="archivepub">
    <npsobj insertvalue="template" name="archivenav" />
  </npsobj>
  <npsobj casecond="isEqual" value="productpub">
    <npsobj insertvalue="template" name="productnav" />
  </npsobj>
  <npsobj casecond="default">
    <npsobj insertvalue="template" name="standardnav" />
  </npsobj>
</npsobj>
```

One after the other, this code compares the `objClass` field with the character strings `newspub`, `archivepub`, and `productpub`. If the value of `objClass` equals one of these character strings, a corresponding navigation layout is included, and the rest of the `switch` instruction is skipped. If `objClass` matches none of these values, then the `casecond-default` instruction is processed because this condition is always satisfied.

The `proceed` tag attribute in the `NPSOBJ casecond` tag with its permitted values `yes` and `no` lets you control whether the next `casecond` instruction in the `switch` instruction is to be processed even if the condition is satisfied. This is by default not the case, i. e. the value of `proceed` is `no`, if this tag attribute has not been specified. This has the effect that the rest of the `switch` instruction is ignored.

7.4 Examples of Automatically Generated Tables of Contents

7.4.1 Formatting As a Series of Paragraphs

An automatically generated table of contents does not have to be formatted as a list. The following HTML code creates a simple series of paragraphs:

```
<npsobj list="toclist">
  <p>
    <npsobj name="self" insertvalue="anchor">
      <npsobj name="title" insertvalue="var"/>
    </npsobj>
  </p>
</npsobj>
```

7.4.2 Including Further Field Values

In this example the creation date of the file and the value of a custom field `Abstract` are both read out, separated from the title by a line break, and formatted differently:

```
<npsobj list="toclist">
  <p>
    <!-- Insert creation date-->
    <font size="-1">
      Created on: <npsobj name="validFrom" insertvalue="var"/>
    </font>
    <br>

    <!-- Insert linked title-->
    <b><npsobj name="self" insertvalue="anchor">
      <npsobj name="title" insertvalue="var"/>
    </npsobj></b>
    <br>

    <!-- Insert Abstract-->
    <i>
      <npsobj name="Abstract" insertvalue="var"/>
    </i>
  </p>
</npsobj>
```

7.4.3 Formatting As a Two-Column Table

The following code creates a two-column table, in which the first column lists the file titles and the second column contains the values of the custom field `Author`:

```
<table>
  <npsobj list="toclist">
    <tr><td>
      <!-- Insert linked title-->
      <b><npsobj name="self" insertvalue="anchor">
        <npsobj name="title" insertvalue="var"/>
      </npsobj></b>
    </td>
    <td>
      <!-- Insert Author-->
      <npsobj name="Author" insertvalue="var"/>
    </td>
  </tr>
</npsobj>
</table>
```

```

    </td></tr>
  </npsobj>
</table>

```

Of course you can also format the table using further HTML elements.

7.4.4 Separation by File Type

An overview divided according to further documents and resources for download can best be created as two overviews, each containing only files of one type:

```

<p>Further documents:
<npsobj list="toclist">
  <npsobj name1="objType" value2="document" condition="isEqual">
    <npsobj name="self" insertvalue="anchor">
      <npsobj name="title" insertvalue="var"/>
    </npsobj>
  </npsobj>
</npsobj>

<p>Files to download:
<npsobj list="toclist">
  <npsobj name1="objType" value2="generic" condition="isEqual">
    <npsobj name="self" insertvalue="anchor">
      <npsobj name="title" insertvalue="var"/>
    </npsobj>
  </npsobj>
</npsobj>

```

7.4.5 Linking Sibling Files

Sometimes it is convenient to be able to access – apart from the content of a file – also the list of all other files contained in the same folder, i. e. the sibling files.

This can be achieved by means of the following NPSOBJ code that can be inserted into a layout file. The code first sets the current context to the folder containing the current file using `list="parent"`. In this context a `toclist` is generated that contains all the files in the folder including the current document. By means of a comparison the current document is skipped so that only the sibling files are left over. The comparison refers to the name of the originally exported file and the file currently processed in the `toclist`. The original file is addressed using `context.context`. Inside a list, `context` jumps one level upwards, i. e. to the folder from which the list was generated.

```

<!-- Set context to folder containing the current document -->
<npsobj list="parent">
  <ul>
    <!-- Create toclist here -->
    <npsobj list="toclist">
      <!-- Take only documents into account -->
      <npsobj name1="objClass" value2="doc" condition="isEqual">
        <!-- Skip current document using a name comparison. -->
        <npsobj name1="name" name2="context.context.name" condition="isNotEqual">
          <!-- If the name of the originally exported file is different from -->
          <!-- the name of the current toclist entry, a link is output. -->
          <li>
            <npsobj insertvalue="anchor" name="self">
              <npsobj name="title" insertvalue="var" />
            </npsobj>
          </li>
        </npsobj>
      </npsobj>
    </ul>
  </npsobj>

```

```

</npsobj>
</ul>
</npsobj>

```

7.5 Selecting Tables of Contents Using Field Values

In the interests of achieving the most unified design possible for the tables of contents used in your system and to simplify the work for editorial staff it is recommended that you write the code for various tables of contents in a layout in advance and that you select which table of contents to use in a folder by using a custom field. To do this proceed as follows:

1. Create a field for selecting the type of the table of contents to be inserted (see [The Fields Section](#)).
Properties of this field:
Type: Enumeration
Name e.g.: TocListType
Selection values, e.g.: list, series of paragraphs, table ...
2. In the topmost folder level create a layout with a name such as *list* that contains the code for all tables of contents. Every table of contents is enclosed by a condition that queries the value of the field TocListType. An example:

```

<npsobj condition="isEqual" name1="TocListType" value2="ParagraphOrder">
  <npsobj list="toclist">
    <p>
      <npsobj name="self" insertvalue="anchor">
        <npsobj name="title" insertvalue="var"/>
      </npsobj>
    </p>
  </npsobj>
</npsobj>

```

3. Supplement all file formats for which the file type *folder* has been set with the TocListType field and this predefined main content:
`<npsobj insertvalue="template" name="list"/>`

These measures will insert the list layout as the default content in every folder. The editors must then simply determine which of the predetermined formats and contents is to be used for the table of contents in the folder by selecting a value for the field TocListType. The predetermined main content of every folder can of course be changed as required.

This mechanism has the desirable side effect that all changes to the layout of tables of contents can be made globally by changing the code in the layout. This does away with time-consuming editing of every individual folder.

8

8 Collections of Links

8.1 NPSOBJ Tags for Generating Collections of Links

With the exception of layouts, you can add free links to the draft version of every file – normally, however, only folders and documents are equipped with related links. The target of a free link can be another file (internal link) as well as an external document.

Free links are assigned to content fields of the `linklist` type. Using an NPSOBJ instruction you can generate a list of all the free links assigned to a particular linklist field in order to create a styled collection of links.

A collection of links can be created in two ways, either as a list or as a table. With the following code you insert the links as a list:

```
<npsobj list="relatedLinks">
  ...
</npsobj>
```

The following code will result in a table:

```
<npsobj table="relatedLinks" columns="3" direction="vertical">
  ...
</npsobj>
```

Using the parameter `columns` you determine the number of columns, `direction` determines whether the cells are filled from left to right (`horizontal`) or from top to bottom (`vertical`). In cells that remain empty, the CMS inserts a non-breaking space (` `). For formatting the table you can specify further parameters such as `border`, `width`, `height` etc.

8.2 Content and Formatting of Link Lists

Link collections usually consist of linked texts. Within an NPSOBJ-list or NPSOBJ-table instruction you can create a hyperlink for each free link in the list by means of an NPSOBJ-insertvalue instruction. The link destination is available to you as the value of the `destinationUrl` parameter. As linked text you can use the link title (`displayTitle`) as shown in the following example:

```
<ul>
<npsobj list="relatedLinks">
  <li>
    <npsobj name="destinationUrl" insertvalue="anchor">
      <npsobj name="displayTitle" insertvalue="var"/>
```

```

    </npsobj>
  </li>
</npsobj>
</ul>

```

You can specify the title of a free link as you create or edit the link.

In the example shown above, the link collection is formatted as an unordered list. However, you can use any HTML code for styling your list.

8.2.1 Evaluate links to CMS files

In link collections it is not only possible to access the URL of the link destination. If the link destination is a file you have full access to this file and can use its title as linked text, for example. You can access a field of the destination file by using the `destination` link parameter followed by the name of the desired field. Separate the two elements by a period. In the following example the value of the custom field `abstract` is queried:

```
<npsobj name="destination.abstract" insertvalue="var"/>
```

In this way, for your link collections, you can use any desired piece of information available in the destination file. The following sample code generates a link collection consisting of the titles and the creation dates of the destination files. The titles are linked with the destination files:

```

<ul>
<npsobj list="relatedLinks">
  <li>
    <npsobj name="destinationUrl" insertvalue="anchor">
      <npsobj name="destination.title" insertvalue="var"/>
    </npsobj><br>
    <npsobj name="destination.lastChanged" insertvalue="var"/>
  </li>
</npsobj>
</ul>

```

8.2.2 Selecting and sorting the elements in collections of links

In link collections you can use the same mechanisms as in tables of contents to restrict the lists to particular entries (see section [Controlling Entries](#)). If, for example, only links to files located in or below the `/documents/public/` folder are to be included in the list you can achieve this with an `NPSOBJ-condition` instruction.

```

<ul>
<npsobj list="relatedLinks">
  <npsobj name1="destination.prefixPath" value2="/documents/public/"
    condition="hasPrefix">
    <li>
      <npsobj name="destinationUrl" insertvalue="anchor">
        <npsobj name="destination.title" insertvalue="var"/>
      </npsobj>
    </li>
  </npsobj>
</npsobj>
</ul>

```

It is also possible to sort the links in a link list according to particular field values of the target files by specifying the `sortkey1` attribute in the `NPSOBJ-list` or `NPSOBJ-table` tag in order to determine the primary sort criterion. For example, in order to sort the elements by the date they were last changed you can use the following code:

```
<npsobj list="relatedLinks" sortkey1="destination.lastChanged">
  ...
</npsobj>
```

Analogous to the first criterion, the second and the third criterion can be specified using `sortkey2` and `sortkey3`, respectively.

9

9 Creating RSS Feeds

9.1 The Concept of RSS Feed Creation

In Infopark CMS Fiona, RSS feeds can be generated by exporting news lists using a layout file that creates the desired RSS format. The news lists themselves come into existence by assigning CMS files to channels (using the `channels` version field) and then releasing the CMS files.

Using the NPSOBJ instruction `newslst`, all news files or only those that meet particular criteria can be queried (for example those that have been assigned to particular channels). Furthermore, the maximum number of news articles to be retrieved can be restricted.

In Infopark CMS Fiona, the following features serve to create news feeds:

- In the system settings, channels can be defined. Channels are used for structuring news articles by their topic, i. e. for categorizing or classifying them, just like a user-defined field product-group classifies a file with respect to the products described in the contents of the file. The channel settings can be set via the *Extras > System configuration* menu item.
- The versions of documents and folders have the built-in field `channels`. The `channels` field's type is `multi-selection`, meaning that any number of the channels that have been configured can be assigned to it. In this manner, content is assigned to the channels.
- File formats have the `canCreateNewsItem` field (Mark as news on the live server). If this field has been activated for a format, the files with this format are added to an internal news list when they are released, provided that the version's `channels` field is not empty (meaning that this field contains at least one channel).
- It is possible to create news lists from files assigned to any combination of channels. For this, an NPSOBJ instruction and a Tcl command are available. Analogously to the `toclist` instruction, the NPSOBJ instruction returns a context list, so that the version fields can be queried. The name of the corresponding Tcl command is [news](#).

9.2 Using the `npsobj newslst` Instruction

For the purpose of creating news lists in layouts, the NPSOBJ `newslst` instruction can be used in the following three ways:

- List of all news in all channels:

```
<npsobj newslst="all" length="20">
  Text evaluated for each news article
</npsobj>
```

- List of the news articles assigned to at least one of the channels whose names are values of a field contained in the current document:

```
<npsobj newslst="selected" name="Channel-Feld" length="20">
  Text evaluated for each news article assigned to at least one of
  the channels that have been specified indirectly.
</npsobj>
```

The field type must be one of `string`, `text`, `enum`, or `multienum`. With fields of the types `string` and `text` the values need to be separated by commas. With multi-selection fields the field values are used as channel names.

- The list of news to which channels specified directly are assigned:

```
<npsobj newslst="selected" value="ch1, ch2, ..." length="20">
  Text evaluated for each news assigned to one of the channels
</npsobj>
```

A news article is never listed more than once in the generated news list even if it is assigned to more than one of the channels specified directly or indirectly.

Generated news lists contain only news articles whose publication date is in the past at the time of the export. Of course, the news list internally contains all of the files to which a channel has been assigned and whose format marks the file as a news article. The publication date corresponds to the creation time of the news file that can be determined via the `valid from` field of the working version. A news article that has been released accidentally can therefore be removed from the generated news list by specifying a publication date that lies in the future. Internally, the news article still exists unless the corresponding file has been deleted or its `channels` field has been cleared. A file is also removed from the internal news list when the channels are deleted to which the news article is assigned. However, this does not change the value of the `channels` field of the versions. The news article is not again placed into the news list when a deleted channel to which the article is assigned is created again.

When the Template Engine is used, all the files containing an `NPSOBJ newslst` instruction are evaluated again when files are changed or the channel configuration is modified. This is done to keep the news lists up-to-date. (The files are given a `usesAll` [dependency](#).)

9.3 Examples for Using News Lists

9.3.1 How to create a list of 10 latest news articles

- Create the channel `sitenews`.
- Create the file format `newsitem` and activate its option *Mark as new on the live server* (`canCreateNewsItems`).
- Create the file `news1` anywhere in the folder hierarchy. Use `newsitem` as the format of the file.
- Add code like this to the layout of the starting page:

```

<ul>
  <npsobj newslst="all" length="10">
    <li>
      <npsobj insertvalue="anchor" destination="self">
        <npsobj insertvalue="var" name="title" />
      </npsobj>
    </li>
  </npsobj>
</ul>

```

9.3.2 How to create an RSS feed for politics news

- Create the channel `politics`.
- Create the file format `newsitem` and enable its option *Mark as news on the live server* (`canCreateNewsItems`).
- Define a field `description`. Create the file format `feed` and add the `description` field to it. Do not activate `canCreateNewsItems` for `feed`.
- Create the folder `rssfeeds`.
- In the folder create a base layout which outputs an RSS feed (see below).
- In the same folder create the file `politicsfeed` using the `feed` format and set its `description` to "Company policy news". Assign to it the `politics` channel.
- Set the main content of the base layout to the following:

```

<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE rss PUBLIC "-//Netscape Communications//DTD RSS 0.92//EN"
  "http://my.netscape.com/publish/formats/rss-0.92.dtd">
<rss version="0.92">
<channel>
  <description>
    <npsobj insertvalue="var" name="description" />
  </description>
  <language>de</language>
  <title><npsobj insertvalue="var" name="title" /></title>
  <link>http://www.mysite.com/</link>
  <copyright>Copyright 1994-2008 Infopark AG</copyright>
  <generator>Infopark CMS Fiona 6.7.0</generator>
  <ttl>60</ttl>
  <npsobj newslst="selected" name="channels" length="20">
    <item>
      <title><npsobj insertvalue="var" name="title"/></title>
      <link>http://www.infopark.de<npsobj insertvalue="var" name="visiblePath" /></link>
      <description>
        <npsobj insertvalue="var" name="description"/>
      </description>
    </item>
  </npsobj>
</channel>
</rss>

```

To output the RSS feed in a different version (for example 2.0) you can modify the layout file as desired.

To generate another feed for sports news using the layout file above:

- Create a channel named `sports`.
- In the `rssfeeds` folder create the file `politicsfeed` using the feed format and set the file's `description` field to "Latest sports news". As channels select `sports`.
- Now create files based on the `newsitem` format and set their `channels` field to `sports`. The files will be added to the Sports news feed.

9.3.3 Sending a Newsletter

- Create a folder named `newsletters`.
- Write a base layout that generates the contents of the newsletter by reading fields from the corresponding files, for example.
- Create a file format named *newsletter*.
- In the `newsletters` folder create the file `politics` based on the file format `newsletters`.
- Into the main content of the `politics` file enter the e-mail addresses. Only one address per line.
- Write a [wizard](#) that sends the `exportBlob` field of `politics` to each e-mail address in the main content of `politics`.

10

10 Export Variables

You can use export variables in layouts in order to temporarily store texts and context lists. In a layout the stored values can be recalled and also modified as often as desired. Therefore, even extensive texts to be exported more than once need not be included several times in the layout or stored externally in another layout. This makes your layouts clearer and easier to maintain.

Export variables are set and modified with the NPSOBJ-[modifyvar](#) instruction. They can be accessed like fields and specified in NPSOBJ-[list](#) and NPSOBJ-[table](#) instructions.

10.1 Storing and Recalling Texts

A piece of text can be stored in an export variable and recalled in a different place in a layout by means of instructions like in the following example:

```
<npsobj modifyvar="set" varname="myvar">
  Author of this document: <npsobj insertvalue="var" name="author" />
</npsobj>
...
<npsobj insertvalue="var" name="export.myvar" />
```

This code assigns to the variable `myvar` the character string *Author of this document:*, followed by the value of the field `author`. Subsequently, the value of this variable is output. You may also access the value of the variable using the following code:

```
<npsobj insertvalue="var" name="myvar" />
```

However, because the `export.` prefix has not been specified, an unwanted content [dependency](#) for the current file context is generated if the variable does not exist.

During the export of the respective file, `myvar` can be read any number of times, unless this variable is deleted with the following instruction:

```
<npsobj modifyvar="clear" varname="myvar" />
```

The variable can also be accessed in included layouts (`insertvalue="template"`). However, this may quickly lead to incomprehensible layouts because the value of the variable can not be spontaneously determined from the included layout. Therefore, variables should only be used in layouts in which a value is assigned to them.

If a variable is to be assigned the value of a field only, i. e. no literal text, then this can be done directly in the instruction's tag:

```
<npsobj modifyvar="set" varname="myvar" name="parent.title" />
```

The names of export variables have precedence over the names of fields. You can, for example, create a variable named `toclist` and assign it a character string. Afterwards, however, you will not be able to have a table of contents generated using `<npsobj list="toclist">`. By means of the prefix `self`, however, the content or file field of the same name can be referenced again (`"self.toclist"`).

10.2 Storing and Recalling Context Lists

A context list is a list of files or links. Such a list is normally created with an `NPSOBJ-list` instruction in which a field name such as `children` or `toclist` or the name of a linklist field is specified.

Such lists can be stored in export variables. This alone would not have any advantages over using a built-in context list like `toclist` in an `NPSOBJ-list` instruction. The context list stored in an export variable, however, can be extended, i. e. other context lists can be appended to it. Additionally, the list stored in the variable can be sorted or truncated. The code in the following example assumes that the current file is a folder containing subfolders. The subfiles of all subfolders are to be stored in the variable named `mylist` if their respective `showInList` custom field has the value `YES` :

```
<npsobj list="toclist">
  <npsobj condition="isEqual" name1="objType" value2="publication">
    <npsobj list="children">
      <npsobj condition="isEqual" name1="showInList" value2="YES">
        <npsobj modifyvar="append" varname="mylist" name="self" />
      </npsobj>
    </npsobj>
  </npsobj>
</npsobj>
```

In the inner list the `modifyvar append` instruction appends to `mylist` the file currently processed in the list. This file is referred to with `self`, yielding, like `parent`, `previous` and `next`, a context list with exactly one element. Instead of the inner list we could have used

```
<npsobj modifyvar="append" varname="mylist" name="children" />
```

if it had not been necessary to use a `condition` instruction in order to select the files by `showInList`.

The code also shows that a `modifyvar-append` instruction can even be used if the export variable does not yet exist. This means that it is not necessary to initialize variables.

An export variable can be assigned a context list using the following instruction:

```
<npsobj modifyvar="set" varname="mylist" name="toclist" />
```

Next to `toclist`, all names returning a single context or a context list can be specified. These are, among others: `self`, `parent`, `children`, the names of variables to which a context list has been assigned, and the names of linklist fields.

The context lists in export variables can be truncated by means of a `modifyvar-range` instruction. For example, the following code removes the first element (i. e. the first context) from the list `mylist`:

```
<npsobj modifyvar="range" varname="mylist" start="2" />
```

As with the [NPSOBJ list instruction](#), you can combine `start` with either `length` or `end` to specify the desired part of the list. While `length` is used to indicate the number of desired elements, `end` lets you specify the index of the last element. The index of the first element is 1.

For sorting the context list stored in an export variable, the `modifyvar-sort` instruction is available. It is described in section [npsobj modifyvar sort](#).

You can use the variables in which you have stored context lists in `NPSOBJ-list` and `NPSOBJ-table` instructions in order to generate a link list, for example:

```
<npsobj list="export.mylist">
  <npsobj insertvalue="anchor" name="self">
    <npsobj insertvalue="var" name="title" /><br>
  </npsobj>
</npsobj>
```

Context lists are not restricted to file contexts but can also contain free links. Furthermore, file and link contexts can be mixed, if required. The code that processes such a mixed list, however, would have to test the type of each context. For this purpose one could, for example, read out in a `switch` instruction the value of a field (such as `objType`) which is never empty for files and always empty for links:

```
<npsobj list="export.mymixedlist">
  <npsobj switch="objType">
    <npsobj casecond="isEmpty">
      Must be link context
    </npsobj>
    <npsobj casecond="default">
      Must be file context
    </npsobj>
  </npsobj>
</npsobj>
```

From version 6.5.0, it is possible to access context lists containing exactly one context as if they were this context, i.e. it is no longer necessary to iterate over the list using `npsobj list`. Example:

```
<npsobj modifyvar="set" varname="savedContext" name="self" />
...
<npsobj condition="isEmpty" name="export.savedContext">
  <npsobj insertvalue="var" name="savedContext.path" />
</npsobj>
```

10.3 Local Export Variables

From version 6.7.1, export variables can be defined locally in layouts. Therefore, the same variable name can be used in several [sublayouts](#) without producing name conflicts during the export. In conjunction with the possibility to pass arguments to sublayouts, which also become local variables, layouts can be modularized and therefore are much easier to maintain.

The scope of a local variable is restricted to the layout in which the variable is defined. It is neither visible in the calling parent layouts nor in sublayouts on deeper levels.

In the defining instruction, `modifyvar set`, a variable can be marked as local by specifying the `local` attribute and setting its value to `true`:

```
<npsobj modifyvar="set" varname="myVar" local="true">  
  Contents of the variable  
</npsobj>
```

An existing global variable or local variable with the same name on a higher layout level will always be hidden by a local variable created with `modifyvar set`.

Local variables work analogously when `modifyvar append` is used. However, an existing variable, be global or local, will be used for appending the specified context list to it. Thus, with `modifyvar append`, a variable marked as local will only become local if it needs to be created.

11

11 Further NPSOBJ Instructions

11.1 Using the Main Content of a File

You can insert the main content of another document or publication into layouts by using an NPSOBJ instruction. In your code replace `file-path` with the complete path to the file to be inserted:

```
<npsobj includetext="file-path"/>
```

The inserted file will be completely managed by CMS as a reference. A link will be created between the two files that will be managed bi-directionally and can be edited exactly like other internal links.

In the Contnet Navigator, this type of link is displayed in the *Content* section.

11.2 Execute System Commands

The following NPSOBJ instructions can be used to execute Tcl procedures in layouts (replace `procedureAlias` with the procedure's alias and `content` with HTML code, text or an NPSOBJ instruction):

```
<npsobj name="procedureAlias" insertvalue="systemExecute">  
  content  
</npsobj>
```

Two parameters will be passed to the procedure to be executed: the ID of the file or link in whose context the instruction is executed as the first parameter and the evaluated `content` of the element as the second.

After the CMS has executed the procedure the NPSOBJ instruction will be replaced by the procedure result.

To be able to execute Tcl procedures in this way an administrator must have registered the aliases in the CMS initialization file (in the dictionary `tclSystemExecuteCommands`).

Please note that the Tcl code called has no write access to the CMS data and that the procedures to be executed at export-time must also be available on the live server if the Template Engine is used.

The supplied `editLink` system command

The CMS includes the predefined system command `editLink`. Using this command, you can create links to the Content Navigator page of the exported file:

```
<NPSOBJ insertvalue="systemExecute" name="editLink">
  Edit this file
</NPSOBJ>
```

For the links generated on export to function properly, the value of the Content Manager's system configuration entry `guiUrl` must be set to the URL of the GUI's web application.

11.3 Generating Absolute Paths in Scripts

Using the values of the `fsPathPrefix` and `urlPrefix` variables in scripts, you can generate absolute paths and URLs that work in the preview as well as on the live server.

For this purpose, an NPSOBJ instruction that generates an assignment statement in the desired script language needs to be inserted at the beginning of the base layout. Example:

```
<npsobj insertvalue="var" name="urlPrefix" formatter="phpVardef" />
```

A path can then be output to the file which is generated during the export:

```
<npsobj insertvalue="var" name="visiblePath" formatter="phpVardef" />
<? $absUrlPath = $urlPrefix.$visiblePath; ?>
```

This way of calculating the value of `$absUrlPath` works on the live server as well as in the preview.

12

12 Examples of Layouts

12.1 Nested Layouts

12.1.1 Creating a Sitemap As a Table of Contents

A sitemap is understood to be a depiction of the hierarchy of a website or a part of this hierarchy, whereby ideally the elements of this depiction are linked with the corresponding web pages. You can use the mechanisms for automatically generating tables of contents to create a sitemap with Infopark CMS Fiona. In contrast to the [Examples of Automatically Generated Tables of Contents](#), however, for a sitemap the table of contents must be generated up to the last hierarchy level.

This can be achieved by generating the table of contents in a layout that always calls itself if it encounters a subfolder when creating the table of contents. The process of calling itself is called recursion.

In the following example the entries are indented by `ul` elements nested within each other. Additional means of entry identification (e.g. with a type-dependant icon for the displayed file) are left out.

```
<ul>
  <npsobj list="toclist">
    <npsobj name1="objType" value2="generic" condition="isNotEqual">
      <li>
        <npsobj name="self" insertvalue="anchor">
          <npsobj name="title" insertvalue="var"/>
        </npsobj>
        <npsobj name="toclist" condition="isNotEmpty">
          <npsobj name="sitemaptemplate" insertvalue="template"/>
        </npsobj>
      </li>
    </npsobj>
  </npsobj>
</ul>
```

To create a sitemap using the above sample code, first create a layout file in the base folder with the name `sitemaptemplate`. Then edit the main content of the draft version with the text editor and insert the above code. Release the layout.

Finally switch to the base folder, open the main content of its draft version with the text editor and insert the following line:

```
<npsobj name="sitemaptemplate" insertvalue="template"/>
```

If you now open the preview of the folder the sitemap will be displayed.

Please note that NPSOBJ instructions should normally not be put into the body of folders and documents but into layouts only. This is to ensure that the website remains serviceable.

12.1.2 Sitemap with Selected Subfiles

Tables of contents created as in the above example with `<npsobj list="toclist">` contain only files of types *document*, *ressource* and *folder* (see [Components of Automatically Generated Tables of Contents](#)). If the sitemap is also to list images, the field `children` must be read out instead.

In the following example a sitemap is created which depicts only images in a folder hierarchy:

```
<npsobj list="children">
  <npsobj name1="objType" value2="image" condition="isEqual">
    <p>
      <npsobj name="title" insertvalue="var"/>
      <br>
      <npsobj name="self" insertvalue="image"/>
    </p>
  </npsobj>
  <npsobj name1="objType" value2="publication" condition="isEqual">
    <npsobj name="sitemaptemplate" insertvalue="template"/>
  </npsobj>
</npsobj>
```

You can use the sitemap by inserting the code into a layout and calling the preview.

12.1.3 Infinite Recursion

The Content Manager and the Template Engine monitor the recursive use of layouts. In this context, "recursion" means that a layout includes itself:

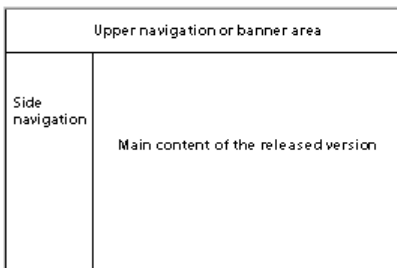
```
<!-- Layout file "myTemplate" -->
<npsobj insertvalue="template" name="myTemplate" />
<!-- ... -->
```

In such a construction, if no exit condition exists for this process or the exit condition is never satisfied, the recursion is infinite, meaning that the process of exporting the file concerned never ends. To ensure that this case never occurs, the Content Manager and the Template Engine terminate a recursion during the export and the preview if the number of inclusions exceeds the value specified in the [maxRecursionLevel](#) system configuration entry. Then, an error message is output.

In the case of an infinite recursion, check for example whether the main content of a file calls a layout that in turn inserts the main content of the file into the output file.

12.1.4 Define Table Layouts

Typically the pages of a website are given a navigation area and an area for the actual content. If you use tables (instead of Cascading Style Sheets, CSS) for this type of layout it is recommended that you place wildcards for navigation bars in the global layout and then for each folder create a layout which generates the navigation bar.



This type of layout can be realized with code in a layout in accordance with the following example:

```
<head>
  <title><npsobj name="title" insertvalue="var"/></title>
  <npsobj name="field-name" insertvalue="meta"/>
  ...
</head>

<body>
  <!-- Upper navigation-->
  <center>
    <a href="/german"></a>
    <a href="/english"></a>
    <a href="/email"></a>
  </center>

  <table>
    <tr><td>
      <!-- Wildcard for side navigation-->
      <npsobj name="nav_side" insertvalue="template"/>
    </td>

    <td>
      <!-- Insert main content-->
      <npsobj name="body" insertvalue="var"/>
    </td></tr>
  </table>
</body>
```

This layout references a further layout, `nav_side`, that is created in all subfolders that are to contain a side navigation, and that is provided with the respectively required links and images.

Layouts should always call further layouts if this generalizes the website structure and simplifies its creation.

12.2 Frame Layouts

With Infopark CMS Fiona you have the ability to create frame layouts in two ways. One way is to use the NPSOBJ instruction provided by NPS that can be used in layouts to automatically create framesets complete with their required content files.

Another way is to use the folder hierarchy to structure the frame hierarchy. This is done by creating a folder for the frameset and creating a subfolder for the table of contents frame. The main frames can be placed as documents in this subfolder.

The first method has the advantage that the editorial staff must not concern themselves about the underlying frame structure during document creation or editing: for every document, the CMS always creates a complete frameset during export and for the preview. The document is bound to its frameset, since every reference to a document exported in this way results in a reference to the

frameset in which it is to be displayed. It is thus possible to set a bookmark for the document without having to do without the surrounding frameset.

In the second method this is not the case, for the frameset is only created for every index page of a folder hierarchy, and not for the documents found in the sub-hierarchies themselves. If one follows a link to such a document then it will only be shown in its frame if the frameset was previously loaded, and the target frame predefined or given in the link tag. Since the document is not bound to its frameset no bookmark can be set for it.

You should use the first method mentioned for creating a webserver based on frames if you want to minimize the work involved in administering frame structure and if every document should always be displayed in its frameset during the preview and in the export results. The second method is useful if you place value on having a leaner web server. It does not make sense to combine the two methods.

12.2.1 Automatically Create Framesets with the NPSOBJ Frame Command

To create a frameset and the required frame content for an exported file you require a layout. In this layout, first the frameset and then every frame is defined using an NPSOBJ instruction in accordance with the following example:

```
<npsobj frame="frameName" defaulttarget="defaultTarget">
  frameContent
</npsobj>
```

During export and in the preview this type of NPSOBJ instruction has the effect of creating, besides the output file for the file, a further output file containing the evaluated content of the NPSOBJ-frame tag. The name of this frame content file is made up of the name of the file to be exported, a period, the `frameName` given in the NPSOBJ tag, and the file-name extension derived from the file name extension. Thus on exporting the document `sample`, besides the main file `sample.html` the file `sample.frameName.html` is generated.

In a layout for creating framesets and frames the main content of the file to be exported is inserted into the content of an NPSOBJ-frame tag, so that it exists as the frame content file. The generated main file, on the other hand, contains only the code enclosing the NPSOBJ-frame tag, thus primarily the code that defines the frameset.

The following example assumes that in the same folder the files `sampldoc` and the layout `frametemplate` are found. In `frametemplate` a two-part frameset and both of the frames associated with it are defined:

```
<html>
  <head>
    <title><npsobj insertvalue="var" name="title"/></title>
  </head>
  <frameset rows="100,*">
    <npsobj frame="banner">
      <html>
        <head>
          <title>Banner</title>
        </head>
        <body>
          
          <!-- Put navigation code here -->
        </body>
      </html>
    </npsobj>
```

```

<npsobj frame="content">
  <html>
    <head>
      <title><npsobj insertvalue="var" name="title"/></title>
    </head>
    <body>
      <npsobj insertvalue="var" name="body"/>
    </body>
  </html>
</npsobj>
</frameset>
</html>

```

The file `sampldoc` has the title `Sampldoc-Title` and contains the following code:

```

<h1>Sampldoc</h1>
<p>A sample document with an <a href="sample2.html">
internal link</a> and an <a href="http://www.infopark.de"
target="_blank">external link</a>.</p>

```

On exporting `sampldoc` using the layout `frametemplate` the three output files `sampldoc.html`, `sampldoc.banner.html` and `sampldoc.content.html` are created. If one follows a link to the document `sampldoc`, then the main file `sampldoc.html` is displayed.

`Sampldoc.html` essentially contains the `frameset` definition:

```

<html>
  <head>
    <title>Sampldoc Title</title>
  </head>
  <frameset rows="100,*">
    <frame name="banner" src="sampldoc.banner.html">
    <frame name="content" src="sampldoc.content.html">
  </frameset>
</html>

```

The file `sampldoc.banner.html`:

```

<html>
  <head>
    <title>Banner</title>
  </head>
  <body>
    
    <!-- Put navigation code here -->
  </body>
</html>

```

The file `sampldoc.content.html`:

```

<html>
  <head>
    <title>Sampldoc Title</title>
  </head>
  <body>
    <h1>Sampldoc</h1>
    <p>A sample document with an <a href="sample2.html" target="_top">
internal link</a> and an <a href="http://www.infopark.de"
target="_blank">external link</a>.</p>
  </body>
</html>

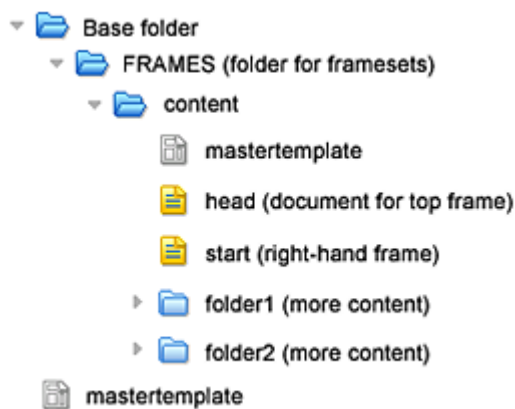
```

Please note that you can use the tag attribute `defaulttarget` in an `NPSOBJ-frame` instruction to determine the frame in which a linked document is to be displayed. The predefined value for `defaulttarget` is `_top`, you thus do not need to use this tag attribute if the links in a frame reference documents that were also exported as framesets. Other tag attributes that you use in `NPSOBJ-frame` tags are accepted without change in the resulting `frame` tag.

`NPSOBJ-frame` tags must not be nested, meaning that they must not contain further `NPSOBJ-frame` tags.

12.2.2 Implement Frames with Folder Hierarchies

The layout described in the section [Define Table Layouts](#) can also be realized with frames instead of with a table:



The frameset is first defined in the base layout (normally `mastertemplate`) in the base folder:

```
<html>
<head>
  <title>
    <npsobj name="title" insertvalue="var"/>
  </title>
</head>

<frameset marginheight="0" framespacing="0" border="0"
  scrolling="no" frameborder="0" rows="50, *">
  <frame src="/FRAMES/content/head.html" name="top">
    <frameset cols="100,450" framespacing="0" border="0"
      scrolling="no" frameborder="0" marginwidth="0">
      <frame src="/FRAMES/content/index.html" name="left">
      <frame src="/FRAMES/content/start.html" name="right">
    </frameset>
  </frameset>
</frameset>
<noframes>
  <p>Your browser does not support frames.</p>
</noframes>
</html>
```

The document `/FRAMES/content/head` contains the navigation bar for the upper frame, the document `/FRAMES/content/start` is defined as the start page for the right-hand frame.

The main content of the folder `/FRAMES/content` is displayed in the left-hand navigation frame. Here, for example, you can insert an automatically generated table of contents. In doing so you must consider that the links in the table of contents and the manually placed links must be provided with the appropriate targets:

```
<npsobj list="toclist">
  <npsobj target="right" name="self" insertvalue="anchor">
    <npsobj name="title" insertvalue="var"/>
  </npsobj>
</npsobj>
```

The layout `/FRAMES/content/mastertemplate` contains the layout definition for the bodies of the files displayed in the content frame:

```
<html>
  <head>
    <title>
      <npsobj name="title" insertvalue="var"/>
    </title>
  </head>

  <body>
    <npsobj name="body" insertvalue="var"/>
  </body>
</html>
```

All further files are similarly exported using this layout so that their respective bodies will be displayed in the same content frame if no other target is defined in the respective link.

Please note during conception of this type of frame layout with Infopark CMS Fiona, that framesets may not be defined within an HTML body, thus under no circumstances in a document or a folder and also not in a layout referenced in a folder or document. This would lead to the frameset definition also being used in the Content Navigator.

If you use several framesets you can place them individually in the base layouts of the respective subfolder. Alternatively several framesets can be defined in a global base layout whereby each of these framesets can be enclosed by a comparison query of a field value. This allows the editorial staff to set a frameset for their document by assigning an appropriate value to the field.

When you check the appearance of the website with the preview, the complete frameset must be loaded at least once into the preview window, since the preview for a subfile only displays the main content of the file without the enclosing frames.

13

13 The NPSOBJ Syntax

NPSOBJ instructions are a powerful instrument for creating HTML code. The instructions can be used in the main content of layout files, documents, and folders (however, in no other fields) to access files and their fields, and to insert the retrieved data (or HTML text generated from it) into the output file.

NPSOBJ instructions are evaluated whenever you open a preview using the Content Manager's user interface or trigger the export of a file. The Template Engine also processes NPSOBJ instructions. When dealing with NPSOBJ instructions and the export functionality of the Content Manager in this section, the Template Engine is always implicitly referenced as well.

When NPSOBJ instructions are evaluated, they are replaced with the result of the evaluation process. Initially, the file being exported serves as the data source for evaluation. However, NPSOBJ instructions for using any other file as a data source exist.

The Content Manager uses the base layout for exporting every file - with the exception of images and resources. As with other layouts, the base layout is processed exactly once from top to bottom. Every occurring NPSOBJ instruction is thereby replaced with HTML code or other instructions in accordance with the rules listed here.

In the following sections you will learn the syntactic and semantic rules the Content Manager uses during the export. The explanations describe the course of the export procedure and simultaneously specify the rules according to which the language element must be used to effect a successful export.

13.1 Conventions

For the formal representation of NPSOBJ elements we use a comprehensible notation. All syntax descriptions are printed in *Courier*. For symbols the standard font is used without any styles whereas literal characters are printed in **bold**. For variable strings (to be replaced with sensible values) *italics* are used. In the following example, the NPSOBJ-`includetext` instruction is defined:

```
npsobj_includetext ::=  
  <npsobj includetext = "internal_url" />
```

The symbols used are expanded except for terminal symbols like `string`. `string` denotes a character string that cannot be expanded any further.

Sometimes, symbols are combined with the `|` symbol (vertical bar). The `|` symbol denotes a disjunctive combination, which means that exactly one of the symbols combined must be used. Elements combined in this way may be grouped using round brackets.

Optional parts in the syntax descriptions are enclosed in square brackets. Parts that may be used once or several times are enclosed in braces.

13.2 Evaluation in a Context

NPSOBJ elements are used, for example, to read out the field values of a file and insert them in the exported HTML code. This section describes the evaluation mechanism for all NPSOBJ instructions.

13.2.1 What Is a Context?

As a CMS file is being exported, it serves as the data source for the layout with which the file is exported (e.g. the `mastertemplate`). This data source contains named values such as the title of the CMS file, accessible via `title`. Thus, the data source provides the context in which NPSOBJ instructions operate when they reference a name to retrieve its value:

```
<npsobj insertvalue="var" name="title" />
```

Here, the value of `title` in the current context is queried. If a context is queried for a value of a name unknown in this context, the empty value is returned.

The Content Management Server provides two context types, file and link contexts.

During the export of a file there are always at least two contexts, the initial and the current context. At the beginning of the export of a document or a folder, this CMS file provides both the initial and the current context. The initial context remains the same during the export of the CMS file. The current context, however, may change. The current context is changed by NPSOBJ instructions that do not yield a simple value such as `title` but a list of contexts. The following instructions return such a context list: `list`, `newslst`, `table`, and `context`.

By means of these instructions, NPSOBJ code can be executed for each context in the context list. An example:

```
<npsobj list="toclist">
  <npsobj insertvalue="anchor" name="self">
    <npsobj insertvalue="var" name="title" /><br />
  </npsobj>
</npsobj>
```

The 2nd-level NPSOBJ instruction, `insertvalue="anchor"`, is executed for each context contained in `toclist`. In each cycle of the `list` instruction, the context taken from the `toclist` becomes the current context. After all `toclist` contexts have been processed, i.e. after the `list` instruction has finished, the original context in which the `list` instruction was executed becomes the current context again.

Every query for a value is evaluated in the current context. The list of names available in the current context depends on the context type. With file contexts, the names available in them largely result from the file type and the file format.

13.2.2 The Structure of Names

To query a value in a [context](#) by means of an NPSOBJ instruction, one needs to specify the name of the value. In the simplest case this is a simple name (`simple_name`) such as `title` or `lastChanged`:

```
<npsobj insertvalue="var" name="title" />
```

In such a case the returned value is a string. For other names (such as `parent`), however, the returned value is a file. If one inserts this type of value directly into the output produced for the exported file, one gets a string representation of the file, which is generally not the desired result. It is therefore possible to ask this file in turn for a value. This can be done by appending the name of this value to the first name, separated by a dot. This way, several names can be combined to form a query that consists of more than one component:

```
<npsobj insertvalue="var" name="parent.title" /><br />
<npsobj insertvalue="var" name="parent.next.title" />
```

Formally, a name has the following structure:

```
composite_name ::= [ exporter_prefix_sequence ] [ simple_name_sequence ]
exporter_prefix_sequence ::=
    initial. | self. | export. | ( context. { context. } )
simple_name_sequence ::= [ simple_name . ] simple_name
```

The number of components in a composite name is in principle unlimited. By specifying a component, the current context for the additional component is temporarily shifted during the evaluation. Thus, with a composite name such as `parent.title`, the `parent` folder of the current file context is made the current context in which the value of `title` is determined. Finally, the previously current context becomes the current context again.

Prefix Sequences

By specifying a prefix sequence (`exporter_prefix_sequence`) the context or the name space of names can be changed during the export process:

Prefix	Effect
context	The context of the surrounding <code>list</code> , <code>newlist</code> , or <code>table</code> element is used. The prefix can be used several times to climb up several levels.
initial	The initial context is used.
export	The name space is altered temporarily: The name that follows this prefix is interpreted as the name of an export variable .
self	The current context is referenced.

A prefix is only effective while the value of the name component that follows it is being determined. Thus, only the first `insertvalue="var"` instruction in the example below refers to the context in which the `list` instruction is evaluated. The second `insertvalue="var"` instruction is evaluated in the context of the current `toclist` element instead:

```
<npsobj list="toclist">
  <b><npsobj insertvalue="var" name="context.title" /></b> -
  <npsobj insertvalue="var" name="title" /><br />
</npsobj>
```

13.2.3 Names in File Contexts

In the current context of a file (called the current file in the following) the names listed below are available. They return the value of the type specified here (*file*, *string*, etc.) unless an equally named export variable exists which returns a value of a different type (see also [npsobj_modifyvar_set](#) as well as [Export Variables](#)).

Names Whose Value Is a File

- `parent`: returns the folder containing the current file.
- `pub`: returns the folder containing the current file. This name is obsolete.
- `next`: returns, corresponding to the sort order to be used, the next file in the folder in which the current file is found.
- `previous`: returns, corresponding to the sort order to be used, the previous file in the folder in which the current file is found.
- `self`: returns the current file.
- `original`: returns the the original file if the current file is a mirror file, otherwise the current file is returned.
- `up`: returns the first file (on the way from the current file to the base folder), in which the value of the components following `up` is not a null value. `up` must not be the last component of a name.

Names Whose Value Is a Link

- `link.link_title`: returns the free link of the current file with the title `link_title`.

Names Whose Value Is a String

- `body`: returns the main content of the current file.
- `contentType`: returns the file name extension of the current file.
- `hasSuperLinks`: returns 1 if links point to the current file, otherwise 0 is returned.
- `id`: returns the ID of the current file.
- `isRoot`: returns 1 if the current file is the base folder, otherwise 0 is returned.
- `name`: returns the name of the current file.
- `objClass`: returns the file format name of the current file.
- `objType`: returns the file type name of the current file.
- `path`: returns the path of the current file.
- `permissions.permissionName`: returns the names of the user groups who have been granted the `permissionName` [file permission](#).
- `prefixPath`: If the current file is a folder, returns the path followed by a slash, otherwise returns the path.
- `title`: returns the title of the current file.
- `version`: returns the revision number of the current file.
- `visibleName`: If the current file is a document, returns the path followed by the file name extension. If it is a folder, returns only the file name.
- `visiblePath`: For documents the complete path including file name with which the file is exported; for folders the complete path including file name followed by the string `/index`, followed by the file name extension of the folder.

- All fields of the types `string`, `text`, `html` and `enum` defined in the current file's format. The `string` `.displayValue` may be attached to the names of content fields in order to determine their respective display value.
- All export variables to which a string has been assigned.

Names Whose Value Is a Date String

- `lastChanged`: returns the date of the last change to the current file. Changes in the draft version of the file are not considered in this. On exporting a released file the last change is the release.
- `validFrom`: returns the date from which the content of the file is valid. This value is always set. However, it is ignored for layouts.
- `validUntil`: returns the date until which the content of the file is valid. If this value is an empty value, the document has no limit on its validity. The value is ignored for layouts.
- all fields of the type `Date` defined in the file format of the current file.

Names Whose Value Is a List of Strings

- all fields of the type `multienum` defined in the file format of the current file.

Names Whose Value Is a List of Links

- `freeLinks`: returns the list of all free links in the file, i.e. the links contained in all `linklist` fields.
- `subLinks`: returns the list of all links in the file.
- `superLinks`: returns the list of links that point to the current file.
- `textLinks`: returns the list of links contained in the main content and in HTML fields of the file (except for inclusion and context links).
- all fields of the type `linklist` defined in the file format of the current file.
- all export variables containing a list of links.

Names Whose Value Is a List of Files

- `children`: returns the list of all subfiles (independently of their contents) if the document is a folder, otherwise an empty list.
- `objectsToRoot`: returns the list of all folders on the way from the base folder to the current file. The base folder is the first, the current file the last element in the list.
- `superObjects`: returns the list of files in which a link to the current file is set.
- `toclist`: returns the list of sorted subfiles that are not images or layouts.
- all export variables containing a list of files.

Names Whose Value Is a Context List

- all export variables containig a context list.

The names `toclist`, `previous` and `next` return their results depending on the predefined order of the files in a folder. This is also the case if the current context is created from a `list` or `table` element, in which the order of the files deviates from the default order because they have been sorted.

13.2.4 Names in Link Contexts

If the current context is a link then the following names are available:

13.2.5 Names Whose Value Is a File

- `destination`: the file that is the target of the link. For external links this is a null value.
- `source`: the file that is the starting point of the link.

Names Whose Value Is a String

- `destinationUrl`: A URL from the initial context to the link target. If the link does not point to an external target but to a file in the CMS, the path is relative.
- `displayTitle`: If the current link has a non-null `title`, this `title` is returned. Otherwise, the title of the destination file's released content is returned. For external links, always `title` is returned.
- `target`: The window (frame) in which the browser should open the link.
- `title`: If the current link has a `title`, returns `title`.
- all export variables to which a string has been assigned.

Names Whose Value Is a Context List

- all export variables containing a context list.

13.2.6 Names in All Contexts

The following names exist in all contexts:

Names Whose Value Is a String:

- `exportCharset`: returns the name of the export character set defined in the system configuration and looked up in the `exportCharsetMap` list.
- `exportMode`: returns the value `preview` if the name is evaluated while a preview page is computed. During the export, the value `normal` is returned. You can use this value to make the export of particular data dependent on the export mode.
- `fsPathPrefix`: returns the value of the system `export.absoluteFsPathPrefix` system configuration entry, i. e. the character strings with which file paths are prefixed if absolute paths are to be exported. See also section [Generating Absolute Paths in Scripts](#).
- `urlPrefix`: returns the value of the system `export.absoluteUrlPrefix` system configuration entry, i. e. the character strings with which the paths in URLs are prefixed if absolute paths are to be exported. See also section [Generating Absolute Paths in Scripts](#).

13.2.7 @ References

@ references enable you to use the values available in contexts as the values of HTML attributes. This can not be achieved by using NPSOBJ instructions in the following way:

```
<!-- DISALLOWED! -->
<a href="/public/file" title="<npsobj insertvalue="var" name="myField" />">...</a>
```

Instead, an @ reference can be used for this:

```
<!-- Up to version 6.7.0 -->
<a href="/public/file" title="@myField">...</a>

<!-- From version 6.7.1 -->
<a href="/public/file" title="@{myField}">...</a>
```

In @ references, all names are available that can also be used in NPSOBJ `insertvalue-var` instructions, i. e. the names of fields and export variables, for example.

For @ references the following rules and restrictions apply:

- @ references are only available in the attribute values of HTML and NPSOBJ tags;
- Up to version 6.7.0: Only tag attribute values beginning with the @ character and whose second character is an alphanumeric character (0 - 9, a - z, A - Z) or the underscore are recognized as @ references.

From version 6.7.1: @ references start with an @ character directly followed by the variable name enclosed in braces (see the example above). @ references can be used anywhere inside attribute values. Thus, static and dynamic text can be combined at will in these values.

- Up to version 6.7.0 only: If the first two characters of a tag attribute value are @ characters, they are converted to a single @ character which is not treated as the beginning of an @ reference.
- @ references neither work in processing instructions (PHP, JSP, etc.) nor in HTML comments.
- A substituted @ reference is always a character string or a context.
- If the value to which an @ reference refers is a list (fields of the *multiple selection* type or context lists), the | character is used to concatenate the individual list items.
- When content is imported into the CMS, @ references contained in URI attributes such as `href` are not taken over into link management.
- If a value cannot be computed, the @ reference is substituted with the empty string.
- Compound names such as `parent.visiblePath` are also supported.
- @ references are not suited for handling the values of multiple selection fields. For this, formatters should be used.
- If an @ reference refers to an export variable, the value of the export variable is not processed when the @ reference is evaluated since the export variable has already been processed at the time the export variable was set.
- @ references contained in link `title` or `target` attributes (i. e. in fields such as the main content) have priority over titles or targets that have been set through link management (i. e. via the Content Navigator, for example).
- @ references that have been entered via link management are not evaluated, meaning that they are exported as string constants. However, when content is imported, this string becomes an @ reference because it is then contained directly in the main content.

Examples

The line breaks and indents in the following examples have been inserted for clarity and need to be removed when this code is used. Otherwise the results produced (URLs) will be defective.

The following code generates part of a URL by combining a field value and static text:

```
<!-- Up to version 6.7.0 -->
```

```
<npsobj modifyvar="set" varname="urlpart">
  <npsobj insertvalue="var" name="varname" />?print=YES
</npsobj>
<a href="@urlpart">...</a>

<!-- From version 6.7.1 -->
<a href="@{varname}?print=YES">...</a>
```

The following code generates a URL which depends on the result of a conditional evaluation:

```
<npsobj modifyvar="set" varname="yesOrNo">
  <npsobj condition="isEqual" name1="z" value2="1">yesUrl</npsobj>
  <npsobj condition="isEqual" name1="z" value2="0">noURL</npsobj>
</npsobj>

<!-- Up to version 6.7.0 -->
<a href="@yesOrNo">...</a>

<!-- From version 6.7.1 -->
<a href="@{yesOrNo}">...</a>
```

13.2.8 Formatting Values with Tcl Procedures

When using one of the NPSOBJ instructions `insertvalue` [meta](#) and [var](#), you can have the CMS call a Tcl procedure during the export to modify the result of the NPSOBJ instruction before it is inserted into the output document. Such a Tcl procedure is called a formatter.

The formatter to be applied can be specified by means of a `formatter` tag in the NPSOBJ instruction, for example:

```
<npsobj insertvalue="var" name="title" formatter="formatter_procedure_alias" />
```

A formatter can also be specified in `insertvalue` [dynamiclink](#) instructions. However, link formatters are a special formatter type because the arguments passed to them are different from the ones passed to standard formatters (the latter are usually applied to field values).

The standard formatters are located in the directory `share/script/cm/serverCmds`. They must have been registered in the [export](#).`tclFormatterCommands` system configuration entry before they can be used.

The procedure whose alias name is `formatter_proc_alias` will be called when the NPSOBJ instruction is evaluated. The instruction will be replaced with the value returned by the procedure.

Definition of a Formatter Procedure

A standard formatter procedure can be defined as follows (from version 6.7.1, standard formatters are in the `formatter` name space):

```
namespace eval ::formatter {
  proc myFormatter {value type params} {
    return "To myFormatter the value $value, the type $type,\
      and the parameters $params were passed."
  }
}
```

The CMS passes the following arguments to the procedure, in the order given below:

- The first argument is the value of the field referenced by *name*; if the NPSOBJ instruction also contains the *format* or *separator* tag attribute then the formatted value will be passed to the formatter procedure.
- The second argument is the type of the value passed as the first argument. The following rules apply:
 - If the value results in a context list (this is the case for file and link lists such as *children* or *superlinks*, but also for *parent* etc.), the type is *list*.
 - If the value has been assigned using `npsobj modifyVar`, the type is *html*.
 - If the value results in a link instance, the type is *link*.
 - If the value results in a valid 14-character date string, the type is *date*.
 - In all other cases the type *string*.
- All other tag attributes and their values (including *name* but except *insertvalue*, *formatter*, *format* and *separator*) are passed as a list to the procedure as the third argument; the list contains the tag attribute names and the tag attribute values as name-value pairs.

Infopark CMS Fiona includes the formatter procedures described below. The names given are the alias names that can be specified in the NPSOBJ instruction as the value of the *formatter* attribute.

formatBlobLength

Task

Returns file sizes in a human-readable form. The result is a number as small as possible to which the appropriate unit of measurement (KByte, MByte etc.) has been appended.

Parameters (from version 6.7.1)

- *useBinaryPrefix*: A boolean value (*true* or *false*) that determines the prefix type of the measurement unit. If this value is *true*, the file size (Bytes) is divided by 1024, and binary prefixes (Ki, Mi, Gi, Ti) will be used. If the value is *false* or has not been specified, the divisor is 1000, and SI prefixes (K, M, G, T) are used.
- *maxSteps*: specifies the maximum number of divisions to be performed. If, for example, 2 is specified here, values greater than 1000 (or 1024, respectively) M(i)Bytes are still displayed as M(i)Bytes. The default value is 2.
- *precision*: Specifies the number of decimal places. The default value is 0.
- *separators*: Specifies the characters to be used as thousands separator and as the decimal point character. If only one separator is specified it serves as the decimal point character, and no thousands separator is used. The default value is *'.'*.

Examples

```
<npsobj ... formatter="formatBlobLength"/>
```

This NPSOBJ instruction generates the output 12 MByte from the determined value, 12345678.

The following example generates 12'056.3 KiByte from the input value, 12345678.

```
<npsobj ... formatter="formatBlobLength" useBinaryPrefix="true" maxSteps="1"
precision="1" separators="'. '"/>
```

phpVarDef

Task

This formatter generates a PHP variable definition with which a PHP variable is set to a particular value calculated during the export.

Parameters

- **name:** The name of the field whose value is to be determined. This name is also used as the variable name in the generated PHP assignment.

Example

```
<npsobj insertvalue="var" name="title" formatter="phpVarDef" />
```

By means of the instruction above the variable assignment `<?php $title = "Document Title"; ?>` is generated.

formatToRFC822

Task

This formats a date value in accordance with RFC-822. This standard format is used in RSS feeds, for example.

Parameters (from version 6.7.1)

- **withSeconds:** A boolean value (yes or no) that determines whether the seconds are output as well. The default is no. Since the standard demands that the seconds are present, 00 is output if this parameter is no.
- **withTwoDigitYear:** A boolean value (yes or no) that determines whether two digits are output for the year instead of four. The default setting is no.

Examples

```
<npsobj ... formatter="formatToRFC822"/>
```

This instruction generates a date in the form 15 Apr 2011 14:36:00 GMT.

The following instruction generates a date in the form 15 Apr 11 14:36:23 GMT.

```
<npsobj ... formatter="formatToRFC822" withSeconds="yes" withTwoDigitYear="yes"/>
```

dehtmlize

Task

Converts the HTML entities `<`, `>`, `"`, and `&` into their textual equivalents, `<`, `>`, `"`, and `&`.

Parameters

This formatter does not have any parameters.

Example

```
<npsobj ... formatter="dehtmlize"/>
```

The instruction above converts the input value `<div>Large & "small"</div>` to `<div>Large & "small"</div>`.

13.3 npsobj break

13.3.1 Syntax

```
npsobj_break ::=
  <npsobj break />
```

13.3.2 Task

Inside an NPSOBJ-list instruction, this element causes the rest of the NPSOBJ-list element to be ignored and no iterations for the remaining list elements to be made.

Outside an NPSOBJ-list element, the instruction causes the export of the file to be stopped.

13.3.3 Example

```
<npsobj list="toclist" sortKey1="attribute-name">
  <npsobj condition="isEqual" name1="name" value2="news">
    <npsobj break />
  </npsobj>
</npsobj>
```

13.4 npsobj condition

13.4.1 Syntax

```
npsobj_condition ::=
  ( <npsobj condition = "unary_condition" data [ negate = "" ]
    [ separator = "separator" ] [ format = "format_name" ] >
    content </npsobj> ) |
  ( <npsobj condition = "binary_condition" data1 data2
    [ negate = "" ] [ separator = "separator" ]
    [ format = "format_name" ] >
    content </npsobj> )

unary_condition ::= isEmpty | isEmpty | isNil | isNotNil

binary_condition ::= isEqual | isNotEqual | isGreaterThan |
  isLessThan | hasPrefix | hasSuffix | isCaseEqual | matches

data ::= ( value = "value" ) | ( name = "name" )

data1 ::= ( value1 = "value" ) | ( name1 = "name" )
```

```
data2::= ( value2 = "value" ) | ( name2 = "name" )
```

13.4.2 Task

By means of this instruction you can make the export of a piece of text depend on a condition. The condition to be applied is defined by specifying a condition name such as `isEqual` as the value of the `condition` tag attribute (see the complete list of condition names below). The operands of the condition are defined by means of further attributes.

At first, the condition is evaluated. If it is true the entire NPSOBJ element is replaced by *content*. If it is not true, the NPSOBJ element is replaced by the empty string.

If the parameter *negate* is specified, the value of the condition that results according to the following specification for the individual values for *condition* is negated. Only then is the decision made, whether the element is to be replaced by *content* or by the empty string.

The evaluation of *data*, *data1*, and *data2* follows the same scheme. It is defined for *data* here:

If `value="value"` is specified, then the value of *data* is the constant *value*. If *value* is not specified and `name="name"` is specified, then the value of *data* is the result of the evaluation of *name* as a string with the separator *separator* and the date format *format_name*. If (erroneously) neither the *value* attribute nor the *name* attribute are given, the value from *data* is the empty string. If (erroneously) `value="value"` and `name="name"` are specified, `name="name"` is ignored.

The separator must be specified if *name* has been specified and *name* references a list. This is the case with the values of *multiple selection type* fields.

Please note that in binary conditions *name* and *value* must be used with the subscripts 1 and 2. With the condition types `hasPrefix`, `hasSuffix`, `isGreaterThan`, `isLessThan`, and `matches` the subscript determines whether the operand is on the left or right side of the operator.

isEmpty, isNil

These conditions are true if the value of *data* evaluates to an empty string or list, or if it does not exist. The two conditions are isomorphic. `isNil` is available for compatibility reasons.

isNotEmpty, isNotNil

These conditions are true if the value of *data* exists and evaluates neither to an empty string nor to an empty list. The two conditions are isomorphic. `isNotNil` is available for compatibility reasons.

isEqual

The condition is only true if the value of *data1* is identical character by character with the value of *data2*.

isNotEqual

The condition is only false if the value of *data1* is identical character by character with the value of *data2*.

isGreaterThan

The condition is only true if the value of *data1* is greater than the value of *data2*. The values are compared character by character from left to right. Date values are converted to their 14-character ISO

representation prior to the comparison and lists are converted to strings. By negating the condition, a less-than-or-equal comparison can be performed.

isLessThan

Analogously to `isGreaterThan`, the condition is only true if the value of `data1` is less than the value of `data2`. By negating the condition, a greater-than-or-equal comparison can be performed.

hasPrefix

The condition is only true if the beginning of `data1` is identical character by character with `data2`.

hasSuffix

The condition is only true if the length `n` of `data2` is smaller than that of `data1` and the last `n` characters of `data1` are identical character by character with `data2`.

isCaseEqual

The condition is only true if the value of `data1` is identical character by character with the value of `data2`, whereby upper and lower case differences are not observed.

matches

The condition is only true if the value of `data1` matches the regular expression `data2`. If `data2` is not a valid regular expression, then the condition is false. If `data2` is a null string, the condition is true.

Regular expressions must obey the extended POSIX syntax. Please note that the functions actually available depend on the POSIX ERE implementation in the operating system used. In the following list the special symbols and their meaning is explained:

- [] In square brackets a range of symbols is specified in which the examined symbol must be found in order for the regular expression to match. If the examined character must not be found in the range of symbols, then the symbol `^` must be added as a prefix to the range. The minus sign can be used as a literal if it occurs as the first or last symbol. `[=-]` matches the characters `=` and `-`. Examples: `[abc]`, `[a-z]`, `[^0-9]`
- * The `*` after a component of a regular expression means: can occur any number of times (or not at all). Example: `1[a-z]*[0-9]*`
- .
- ^ The symbol `^` at the beginning of an expression requires the comparing string to match the regular expression from the beginning. Within square brackets the symbol means that the comparing string must not be contained in the specified range.
- \$ The symbol `$` at the end of an expression requires the comparing string to match the regular expression from the end. `\n` corresponds to a new line.
- + The plus sign after a regular expression means *once or more*. For example `[0-9]+` has the same meaning as `[0-9][0-9]*`.
- () Brackets are used for grouping. An operator such as `*` or `+` can be used on an individual character or on a regular expression enclosed in brackets. Example: `(a*(cb+)*)`.
- { } Curly braces are used for quantification. Placed after an element, `{n}`, `{n,m}`, or `{n,}` requires that the element occurs *n* times, *n* to *m* times, or at least *n* times, respectively. Example: `[0-9]{14}`
- | The vertical bar means *or*. Example: `(abc|def)`.

All of the symbols listed above must be preceded with a backslash character in order for them to be used as literals.

Example

In the following example in a link context a link will only be generated if the URL contains the string ://. This is only true, if the link is an external link.

```
<npsobj name1="destinationUrl" value2="://" condition="matches">
  <npsobj insertvalue="anchor" name="destinationUrl">
    <npsobj insertvalue="var" name="displayTitle"/>
  </npsobj>
</npsobj>
```

13.5 npsobj context

13.5.1 Syntax

```
npsobj_context ::=
  <npsobj context = "path">
    content
  </npsobj>
```

13.5.2 Task

This instruction causes *content* to be evaluated in the context of *path*. As *path* the path to a file is specified. This path is added to the link management.

13.5.3 Example

The following code first switches to the CMS file with the path `/newspub/news/news1`, which causes this file to become the data source for the subsequent instructions. Then, a link to this file is created, using its `displayTitle` as the linked text.

```
<npsobj context="/newspub/news/news1">
  <npsobj insertvalue="anchor" name="destinationUrl">
    <npsobj insertvalue="var" name="displayTitle"/>
  </npsobj>
</npsobj>
```

13.6 npsobj frame

13.6.1 Syntax

```
npsobj_frame ::=
  <npsobj frame = "name">
```

```
[ defaulttarget = "target" ]
{ other_attribute } >
content
</npsobj>

other_attribute ::= string = "string"
```

13.6.2 Task

This instruction can only be used in a layout (see also [Automatically Create Framesets with the NPSOBJ Frame Command](#)). Besides the output file that the CMS creates when exporting a file, a further output file is generated on evaluating this instruction. This file contains the names *objName.name.contentType*, whereby *objName* is the name of the exported file, *name* the name specified in the *frame* attribute of the tag and *contentType* the file name extension of the exported file. The output file contains the evaluated *content* of the instruction.

The instruction is replaced with a *frame* tag that at least contains the tag attributes *name* and *src*. The *name* attribute is set to *name* and the *src* attribute to the names of the output files created through the NPSOBJ-frame instruction. All other attributes (*other_attribute*) specified in the NPSOBJ-frame instruction are taken unchanged into the *frame* tag.

If the instruction contains the tag attribute *defaulttarget*="target", then in the evaluated *content* all link tags that support the attribute *target* are provided with this attribute, whereby its value is set to *target*. A *target* attribute already existing in the link tag, however, will not be changed. If no *defaulttarget* is given in the NPSOBJ instruction, then *_top* will be used as the value for *defaulttarget*.

NPSOBJ-frame instructions may not be nested.

13.6.3 Example

Please see section [Automatically Create Framesets with the NPSOBJ Frame Command](#) for examples.

13.6.4 Using target="_top" in the Preview

In a link, the targets *_top* and *_parent* cause the link destination to be displayed in the parent frame of the current browser window. Since the Content Navigator from version 6.5 uses frames for the preview, following a link in the preview window may cause the menu or toolbar to become inaccessible.

This behavior can be avoided by adapting the content. Please add the following code to the layout which generates the HTML head and body sections:

```
<head>
...
<npsobj condition="isEqual" name1="exportMode" value2="preview">
  <npsobj modifyvar="set" varname="adjustTargetsInPreview">adjustTargetsInPreview()</npsobj>
  <script type="text/javascript">
    function <npsobj insertvalue="var" name="adjustTargetsInPreview" />
    {
      var previewFrameName = "nps_browser_frame";
      var elements = document.getElementsByTagName('a');
      for (var i = 0; i < elements.length; i++) {
        var target = elements[i].target;
        if ("_top"==target || ("_parent"==target && previewFrameName == self.name)) {
```

```

        target = previewFrameName;
    }
    elements[i].target = target;
}
}
</script>
</npsobj>
...
</head>

<body onload="@{adjustTargetsInPreview}">
...
</body>

```

This script modifies link target values so that they never point outside the preview area.

13.7 npsobj includetext

13.7.1 Syntax

```

npsobj_includetext ::=
    <npsobj includetext = "internal_url" />

```

13.7.2 Task

During evaluation this instruction is replaced by the evaluated main content of the file that is referenced by *internal_url*. If this file does not exist the element is replaced with the empty string. The file must be a document or a folder.

Internally, this instruction is handled as a link of the type *inclusion link*. Thus, *internal_url* is subject to the [link management](#) and is automatically adapted when the link destination or the file containing this instruction is moved.

13.7.3 Example

```

<npsobj includetext="/docs/technical/index.html" />

```

13.8 npsobj insertvalue anchor

13.8.1 Syntax

```

npsobj_insertvalue_anchor ::=
    <npsobj insertvalue = "anchor"
        name = "name"
        { other_attribute } >
        content
    </npsobj>

other_attribute ::= string = "string"

```

13.8.2 Task

This instruction creates an `a` element. The evaluated `content` of the instruction becomes the content of the `a` element. The generated `a` tag contains the following attributes:

- An `href` attribute is generated. For determining its value, a URL for `name` is determined. `name` must yield a file context.
- All attributes of the NPSOBJ tag except `insertvalue` and `name` are added to the resulting `a` tag after the `@` references contained in their values, if any, have been resolved.

To create a link to the current file, an NPSOBJ `insertvalue=anchor` instruction with `name="self"` can be used.

13.8.3 Example

The code below creates a link to the folder containing the current file (i.e. to the `parent` folder). The title of this folder is used as the linked text.

```
<npsobj insertvalue="anchor" name="parent">
  <npsobj insertvalue="var" name="parent.title"/>
</npsobj>
```

13.9 npsobj insertvalue dynamiclink

13.9.1 Syntax

```
npsobj_insertvalue_dynamiclink ::=

  <npsobj insertvalue = "dynamiclink"
    destination = "destination"
    [ formatter = "formatter_procedure_alias" ]
    {other_attribute} />

other_attribute ::= string = "string"
```

13.9.2 Task

The purpose of the element is to generate `include` instructions in the syntax of any scripting language (PHP, JSP, SSI) and have the reference maintained by the automatic [link management](#). Due to its impact on performance, the element should only be used for the purpose for which it was designed – the server-side inclusion of files – and not for calculating relative paths. For the latter a formatter in an `insertvalue var` instruction or a `systemexecute` instruction is sufficient. Dynamic links only work in the [dynamic preview](#).

The type of the generated link is `dynamic`.

The attributes have the following meaning:

- If `formatter` has been specified, the Tcl procedure with the alias `formatter_procedure_alias` defined in the `export.tclDynamicLinkFormatterCommands` system configuration entry is executed. The return value of this procedure is inserted into the exported page in place of the complete NPSOBJ instruction. If `formatter` has not been specified, `destination` is inserted in place of the instruction.
- Using `destination`, the path or the URL to be processed by the formatting procedure is specified. `destination` can be a relative or an absolute path or an URL. Paths are transferred to the link management as internal, URLs as external links.
- Additional attributes can be specified as `other_attribute`.

Since the referenced file can be included by any file in the live server directory hierarchy and might itself contain links, the Content Manager and the Template Engine can be configured to generate absolute internal paths and URLs (inside the file concerned) using the `export.exportAbsolutePaths` system configuration entry. If this entry is set to `YES`, the prefixes for absolute paths and URLs are used. These prefixes can also be configured by means of the `export` entry.

The following arguments are passed to the formatting procedure in the given order:

- The URL of the destination, i. e. of the CMS file path, including the URL prefix, if applicable;
- the file system path of the destination, including the file system prefix, if applicable;
- the ID of the exported file;
- a list containing all `other_attribute` attributes and their values as name-value pairs.

13.9.3 Example

The following instruction creates a PHP-include instruction in the exported file. This inclusion references the file with the path `../some/path`. A link to the referenced file is created and taken into the link management. If – in the case of references to CMS files – the referenced file is moved, the link and the NPSOBJ instruction are automatically adapted.

```
<npsobj insertvalue="dynamicLink" destination="../some/path" formatter="phpInclude" />
```

The instruction returns the following PHP instruction:

```
<? include("/from/root/to/some/path"); ?>
```

13.10 npsobj insertvalue image

13.10.1 Syntax

```
npsobj_insertvalue_image ::=
  <npsobj insertvalue = "image" name = "name"
    { other_attribute } >
  </npsobj>

other_attribute ::= string = "string"
```

13.10.2 Task

This instruction creates an `img` tag including the attributes given below. The instruction cannot be used to generate image links with external URLs. For this, manually add an `img` tag to your layout.

- If `name` has been specified, a URL for `name` is queried. If this URL is not empty, a `src` attribute with this URL as its value is inserted. The value of `name` must yield a file context.
- All attributes of the NPSOBJ tag except `insertvalue` and `name` are added to the resulting `img` tag after the `@` references contained in their values, if any, have been resolved.

13.10.3 Example

In the example below, the context is switched to an image file first. This context is then referenced by means of `self` in the `insertvalue=image` instruction.

```
<npsobj context="/resources/banners/large">
  <npsobj insertvalue="image" name="self" height="50%" width="50%" />
</npsobj>
```

13.11 npsobj insertvalue link

13.11.1 Syntax

```
npsobj_insertvalue_link ::=
  <npsobj insertvalue="link" name = "name" />
```

13.11.2 Task

`link` elements can be placed into the `head` element of a website. They enable the browser (provided it supports this feature) to display additional navigation elements for accessing other pages relevant for the website structure (e.g. the parent document or the search page).

This NPSOBJ instruction Inserts three `link` elements into the output file. The first `link` tag contains the following attributes:

- `rel` with the value `up`
- `href` with the value that is returned by a query for the URL of `parent` in the context of `name`.

The second `link` tag contains the following attributes:

- `rel` with the value `next`
- `href` with the value that is returned by a query for the URL of `next` in the context of `name`.

The third `link` tag contains the following attributes:

- `rel` with the value `previous`
- `href` with the value that is returned by a query for the URL of `previous` in the context of `name`.

13.11.3 Example

```
<npsobj insertvalue="link" name="self"/>
```

13.12 npsobj insertvalue meta

13.12.1 Syntax

```
npsobj_insertvalue_meta ::=
  <npsobj insertvalue = "meta"
    [ content = "name" ]
    ( name = "varname" ) | ( http-equiv = "varname" )
    [ format = "format_name" ]
    [ formatter = "formatter_name" ]
    [ separator = "separator" ] >
  </npsobj>
```

13.12.2 Task

The element is replaced with a `meta` tag with two attributes:

- The attribute `name` or `http-equiv` with the value `varname`.
- The attribute `content` with a value described in the following.

The value of the `content` attribute is initially determined exactly as described in [npsobj_insertvalue_var](#); however the value `name` of the attribute `content` is used for evaluation, and not the value `varname` of the attribute `name` or `http-equiv`. Finally, reserved HTML characters (e.g. angle brackets) contained in the resulting string are converted to their HTML equivalents.

If `name` references a list then a separator must be specified. However, `separator` may be the empty string.

A `formatter` can only be specified from version 6.7.0. Its function is to [format the return value of the NPSOBJ instruction](#).

13.12.3 Example

```
<npsobj insertvalue="meta" name="keywords" content="keywordList" separator="" />
<npsobj modifyvar="set" varname="metaValue">text/html; charset="@exportCharset" </npsobj>
<npsobj insertvalue="meta" http-equiv="Content-Type" content="metaValue" />
```

13.13 npsobj insertvalue systemexecute

13.13.1 Syntax

```
npsobj_insertvalue_systemexecute ::=

<npsobj insertvalue = "systemexecute" name = "name" >
  content
</npsobj>
```

13.13.2 Task

Using this instruction, a computed value can be inserted into an output file created during the export. The instruction is often used to reformat field values of the file being exported (e.g. date values), or to compute a value from several field values.

For this the Tcl procedure referenced by *name* is called. To the procedure two arguments are passed, the ID of the file or link concerned, and the evaluated *content*. The NPSOBJ instruction is then replaced with the return value of the procedure.

name is taken to be the alias of a procedure name that has been specified in the [export.tclSystemExecuteCommands](#) system configuration entry. If the alias is not present in `tclSystemExecuteCommands`, no Tcl command is executed. Instead, the NPSOBJ element is replaced by the string `Error: Execution of '<name>' not allowed.`

Please note that the procedures to be executed at export-time must also be available on the live server if the Template Engine is used.

13.13.3 Example

The following code calls the Tcl function whose alias is `command_alias`. The ID of the file being exported and the value of the custom field `attr_name` are passed to the function as arguments. The code is then replaced with the function's return value.

```
<npsobj insertvalue="systemexecute" name="command_alias">
  <npsobj insertvalue="var" name="attr_name"/>
</npsobj>
```

13.14 npsobj insertvalue template

13.14.1 Syntax

```
npsobj_insertvalue_template ::=

<npsobj insertvalue = "template"
  ( name = "name" ) | ( key = "key" )
  { other_attribute } >
  content
</npsobj>

other_attribute ::= string = "string"
```

13.14.2 Task

The instruction replaces the element by an evaluated layout. If the `name` attribute is present, its value is used as the layout name, otherwise the layout name is looked up in the field named `key` in the current context.

The search for a layout file named as determined by the rules mentioned above starts in the current context. (When a file is exported, the current context initially corresponds to this file. However, in the course of the export, context changes may be caused by other export instructions.) If the current context is a folder, the search starts in this folder. Otherwise it starts in the folder containing the CMS file that provides the current context. If the current context is a link, the search starts at the file or folder containing the link.

If the layout file does not exist at the location determined as described above, the search will be continued further and further in the parent folders until the layout is found or the base folder is reached. If even the base folder does not contain the layout file, a corresponding error message is used as the result.

Up to version 6.7.0, additional attributes next to `name` and `key` as well as the `content` of this element are ignored.

Extensions from version 6.7.1

In addition to `name` and `key`, other attributes can be specified in the NPSOBJ tag for passing arguments to the layout file being included. The value of such an argument can only be text, not a context list. If desired, the value of an argument can also be computed using an @-reference.

The arguments are treated just like export variables are, however, they are only visible inside the layout which is called, meaning that they are local variables. To avoid name conflicts with global variables created using `modifyvar set` or `modifyvar append`, the names of the local variables should be unique. A local variable can be modified by means of `modifyvar set` or `modifyvar append` and deleted using `modifyvar clear`. If the local variable specified in one of these instructions does not exist, the instructions are applied to the global variable with the same name, if it exists.

In addition to the arguments, the `content` of the element is passed to the layout file being included using the `npsobjContent` variable. NPSOBJ instructions contained in the `content` are evaluated in the respective local context, i.e. when `npsobjContent` is read, not when the `content` is written to `npsobjContent`.

In the included layout file, the arguments and `npsobjContent` can be accessed like any other fields or export variables, for example by means of an `insertvalue-var` instruction.

13.14.3 Examples

The following code checks whether the file being exported or one of its parent folders contains the field `listTemplateName`. If so, the value of this field is interpreted as the name of a layout which is then used.

```
<npsobj condition="isNotEmpty" name="up.listTemplateName">
  <npsobj insertvalue="template" key="up.listTemplateName" />
</npsobj>
```

The following example demonstrates how an argument (as the `showbody` attribute) and a text (as the element body) can be passed to the included layout file which then outputs both. These techniques for passing values to layout files are available from version 6.7.1.

```
<!-- Instructions in the calling layout file -->
<npsobj insertvalue="template" name="TemplateName" showbody="1">
  <h1>Section Title</h1>
  <p><strong><npsobj insertvalue="var" name="title"/></strong></p>
</npsobj>

<!-- Contents of the TemplateName layout -->
<npsobj insertvalue="var" name="npsobjContent"/>
<npsobj condition="isEqual" name1="showbody" value2="1">
  <p><npsobj insertvalue="var" name="body"/></p>
</npsobj>
```

When the layout to be included is called, the following output is generated (the colors are just for the purpose of illustration):

```
<h1>Section Title</h1>
<p><strong>Title of the exported document</strong></p>
<p>This is the body of the exported document.</p>
```

13.15 npsobj insertvalue var

13.15.1 Syntax

```
npsobj_insertvalue_var ::=

<npsobj insertvalue = "var" name = "name"
  [ format = "format_name" ]
  [ formatter = "formatter_procedure_alias" ]
  [ separator = "separator" ] />
```

13.15.2 Task

The NPSOBJ instruction is replaced with a field value or the value of an export variable. The value is determined by making a query in the current context for the string value of *name* with the format *format_name* and the separator *separator*. If specified, the *separator* is used to separate the elements of a *Multiple selection* type field from each other.

The format *format_name* will only be used if the value of *name* has the *date* type. It refers to one of the named date and time formats defined in the [validDateTimeOutputFormats](#) system configuration entry.

By using the [formatter](#) tag attribute, the value of *name* can be formatted using a Tcl procedure.

13.15.3 Example

Output the title of the exported file:

```
<npsobj insertvalue="var" name="title" />
```

The following instruction generates a variable definition in the language PHP. For this to work, `phpVardef` must be the alias of a procedure that generates the corresponding PHP code.

```
<npsobj insertvalue="var" name="title" formatter="phpVardef" />
```

The `phpVardef` procedure is already included as a server command in the `serverCmds/formatter.tcl` file. When the procedure is called by the NPSOBJ instruction shown above the following output will be generated:

```
<?php $title = "Title of the current file"; ?>
```

13.16 npsobj list

13.16.1 Syntax

```
npsobj_list ::=

<npsobj list = "name"
  [ sortkey1 = "sortKey1"
    [ sortmodifier1 = sortModifier ]
    [ sortkey2 = "sortKey2"
      [ sortmodifier2 = sortModifier ]
      [ sortkey3 = "sortKey3"
        [ sortmodifier3 = sortModifier ] ] ] ]
  [ start = "startIndex" ]
  [ ( length = "length" ) | ( end = "end" ) ]
  [ reverse = "" ] >
content
</npsobj>

sortModifier ::=
"alpha" |
"numeric" |
"ascending" |
"descending" |
"alpha ascending" |
"alpha descending" |
"numeric ascending" |
"numeric descending"
```

13.16.2 Task

The instruction is frequently used to automatically generate tables of contents. It creates a list of file or link contexts by evaluating *name*. *name* stands for an identifier that yields a context list, for example `toclist` or `children`. Also, the name of a linklist field or an export variable to which a context list has been assigned can be specified.

For each list element, *content* is evaluated once (however, see [npsobj break](#)). For the evaluation of *content*, the current context is temporarily set to the respective list element. Therefore, all names (such as field names) refer to the current list element as long as the context in the *content* part is not changed.

Sorting elements

The context list can be sorted by comparing the values of `sortKey1` in the contexts. If two values are equal, the values of `sortKey2` are compared. If these values are equal as well, the order of the contexts is determined using `sortKey3`. If the contexts remain unsorted because the values of every sortkey are identical in all contexts then the final order of the contexts is undefined. If `sortkey1` has not been specified, the context list will be sorted by the sortkeys that have been specified for the folder containing the respective files.

Using the `sortModifier` modifier, it is possible to specify how the values of two sortkeys are to be compared with each other. `alpha` sorts by character value, `numeric` numerically, i. e. the character sequence is interpreted as a number. `ascending` sorts in ascending, `descending` in descending order. One of the two keywords of each modifier pair (`alpha/numeric` and `ascending/descending`) can be combined with each other (e.g. `numeric descending`). The default sort mode is `alpha` and the default sort direction is `ascending`. Thus, the default modifier combination is `alpha ascending`.

Selecting elements

If the attribute `start` is specified and if `startIndex` has a value greater than 0, then `startIndex` is the index of the first element of the context list to be considered. The first element has the index 1. If `startIndex` has a value less than 0, then the first element to be considered will be determined from the end of the context list. If `startIndex` equals 0, then the attribute will be ignored.

The attribute `length` allows you to specify the number of elements in the context list. If `length` equals 0 (default) then the elements beginning at `start` up to the end of the list will be returned. A value greater than 0, on the other hand, determines the maximum number of elements to be returned, beginning at `start`. If `length` is less than 0, then `start` specifies the last element in the list whose maximum number of elements is determined by the absolute value of `length`. For example, `start="-1"` and `length="-2"` returns the second last and the last element of the list.

As an alternative to `length`, the `end` attribute can be specified. If `end` is a positive number, then this number specifies the index of the last element in the original list to be considered. The first element has the index 1. If `end` is a negative value, then the absolute value of `end` specifies the number of elements to be removed from the end of the list.

The effect of `start` and `length` or `end`, respectively, is shown in the following overview. The element selected with `start` is colored green.

Selected elements	start	length	end	
1 ■ ■ ■ ■ ■ 6				
1 □ □ ■ □ □ □ 6	3	2		
1 □ □ □ ■ □ 6	5	-2		
1 □ □ □ ■ □ □ 6	-3	2		
1 □ □ □ □ ■ 6	-1	-3		
1 □ □ ■ □ □ □ 6	3		5	
1 □ ■ □ □ □ □ 6	2		-2	
1 □ □ □ ■ □ □ 6	-3		5	
1 □ □ ■ □ □ □ 6	-4		-1	
				Legend
				□ Not selected element
				■ First selected element

■ Other selected element

If the `reverse` attribute has been specified, the order of the elements is reversed. This is done after the elements have been sorted and before they are truncated using `start` and `length` or `end`, respectively.

13.16.3 Example

```
<npsobj list="toclist" start="2" length="3" sortkey1="title" sortmodifier1="ascending">
  <!-- This is processed for each element of the list -->
  <npsobj insertvalue="anchor" name="self">
    <npsobj insertvalue="var" name="title" /><br />
  </npsobj>
</npsobj>
```

13.17 npsobj micronavigation

13.17.1 Syntax

```
npsobj_micronavigation ::=
  <npsobj micronavigation = "fieldname" [ levels = "number" ] >separator</npsobj>
```

13.17.2 Task

This element is replaced by a series of links. If the series consists of more than one link the *separator* will be evaluated and inserted between the links. If *separator* is empty a slash will be inserted.

To create the series of links, a series of files is computed first. This series consists of the current file's parent folder as the last element, this file's parent as the second last element, and so on until the base folder is reached. If `levels="number"` is specified then only *number* elements of this series are used. If *number* is greater than zero, the elements are counted from the beginning, otherwise from the end.

For each of these files an `a` element is created. To the `href` attribute of an `a` tag the path of the respective file is assigned. The content of the `a` element is the result of a request for *fieldname* in the context of the respective file.

Example

```
<npsobj micronavigation="title" levels="-3"> / </npsobj>
```

13.18 npsobj modifyvar append

13.18.1 Syntax

```
npsobj_modifyvar_append ::=
<npsobj modifyvar = "append"
  varname = "name"
  name = "varValueKey"
  [ local = "( true | false )" ] />
```

13.18.2 Task

First, this element checks whether the [export variable](#) *name* exists and whether it is a context list. An error is generated if the variable exists and is no context list. If the variable does not exist, it is created as an empty context list. To the list, another list is appended, namely the list resulting from the evaluation of *varValueKey*.

If the attribute *local*, which is available from version 6.7.1, is specified and its value is *true*, and if the export variable does not yet exist, it will be created as a local variable. This will restrict its scope to the current layout file, meaning that it is neither visible in the calling layout nor in sublayouts. Use unique names for variables or a [modifyvar-set](#) instruction with local scope prior to *modifyvar append* to ensure that the variable is local. Please note, that from version 6.7.1, variables can be passed to layouts included by means of [insertvalue-template](#) instructions.

To the export variable a link context list as well as a file context list can be appended, independently of the contexts stored in the export variable. If the resulting list potentially contains link contexts as well as file contexts, your NPSOBJ code should differentiate between these two cases by querying a field that only exists for files and is never empty, e.g. *objClass*. If the check results in an empty value, then the context must be a link context.

13.18.3 Example

```
<npsobj modifyvar="append" varname="mixedList" name="freelinks2" />
<npsobj list="mixedList">
  <npsobj condition="isEmpty" name="objClass">
    <!-- Must be link context -->
  </npsobj>
  <npsobj condition="isNotEmpty" name="objClass">
    <!-- Must be file context -->
  </npsobj>
</npsobj>
```

13.19 npsobj modifyvar clear

13.19.1 Syntax

```
npsobj_modifyvar_clear ::=
<npsobj modifyvar = "clear" varname = "name" />
```

13.19.2 Task

This element deletes the *name* [export variable](#). Not only the variable value but the variable itself is removed, meaning that it is undefined afterwards.

13.19.3 Example

```
<npsobj modifyvar="clear" varname="string25" />
```

13.20 npsobj modifyvar range

13.20.1 Syntax

```
npsobj_modifyvar_range ::=
  <npsobj modifyvar = "range" varname = "name"
    [ start = "startIndex" ]
    [ ( length = "length" ) | ( end = "end" ) ] />
```

13.20.2 Task

This element truncates the context list stored in the export variable named *name*. The start, length, and end attributes have the same function as in the [NPSOBJ-list instruction](#).

13.20.3 Example

```
<npsobj modifyvar="range" varname="mylist" start="2" end="4" />
```

13.21 npsobj modifyvar set

13.21.1 Syntax

```
npsobj_modifyvar_set ::=
  ( <npsobj modifyvar = "set"
    varname = "name"
    name = "varValueKey"
    [ local = "( true | false )" ] /> ) |
  ( <npsobj modifyvar = "set"
    varname = "name"
    [ local = "( true | false )" ]>varValue</npsobj> )
```

13.21.2 Task

During the export of a file, [export variables](#) can be used to store character strings or context lists under a name. The values can be accessed by referencing the names in the same manner as other names that are valid during the export process.

If the attribute `local`, which is available from version 6.7.1, is specified with `true` as its value, the new export variables are only visible in exactly the layout file in which they were created. In this case, their scope is restricted to the current layout, meaning that the variables are neither available in the calling layout nor in sublayouts. Please note that from version 6.7.1, variables can also be passed as arguments to layouts that are included by means of [insertvalue-template](#) instructions.

Up to version 6.7.0, or if `local="true"` has not been specified, the export variables are globally available from the moment they are created until the export of the respective file finishes (unless the variable is deleted in the meantime using a [modifyvar-clear](#) instruction). This means that you can access a variable even after context switches, e.g. in an `NPSOBJ-list` or `NPSOBJ-context` instruction.

The names of export variables have precedence over other names such as field names. If, for example, an export variable named `author` is created and subsequently assigned the string `John Smith` via the instruction

```
<npsobj modifyvar="set" varname="author">John Smith</npsobj>
```

then `<npsobj insertvalue="var" name="author" />` will always return `John Smith`, even if the currently exported file also has a field named `author` to which a different value has been assigned. It is possible, however, to use the `self` prefix in order to explicitly refer to the value of a field. Analogously, the `export` prefix can be used to explicitly include the value of an export variable.

The `modifyvar-set` instruction is available in the two variants shown above.

Variant 1: Value assignment by means of a tag attribute

In the first version, the variable is assigned the value of the field or export variable named `varValueKey`. `varValueKey` can (like `parent.title`, for example) yield a string or a list of link or file contexts. The names yielding file or link contexts are listed in the sections [Available Names in a Context of Type file](#) und [Available Names in a Context of Type link](#). The following instruction assigns the list of the current file's sibling files (including itself) to the variable named `siblings`:

```
<npsobj modifyvar="set" varname="siblings" name="parent.children" />
```

Using the `modifyvar-append` instruction, you can append further context lists to export variables containing link or file context lists (see [npsobj modifyvar append](#)). Furthermore, the contexts in export variables can be sorted (see [npsobj modifyvar sort](#)).

Variant 2: Value assignment by means of the element body

The second variant of the `modifyvar-set` instruction can be used to assign a character string to an export variable. NPSOBJ elements in this string are evaluated. Thus, layouts in which the same code is to be used several times can be written more efficiently.

Please note that using the `modifyvar-set` instruction in this way creates a new context. This context corresponds to its predecessor context, meaning that no further action is required, normally. If,

however, you need to address a context above the current context, you require an additional context prefix as shown in the following example:

```
<npsobj list="relatedLinks">
  <npsobj modifyvar="set" varname="theLinkListCode"><p>
    <npsobj list="context.context.linkImage">
      <npsobj insertvalue="image" name="destination" />
    </npsobj>
    <npsobj insertvalue="anchor" name="self">
      <npsobj insertvalue="var" name="destination.title"/>
    </npsobj></p>
  </npsobj>
</npsobj>
```

You can also query the source link property to determine the CMS file that contains the link.

13.21.3 Example

The following code stores a link pointing to the parent file in a variable and then exports it.

```
<npsobj modifyvar="set" varname="uplink">
  <npsobj insertvalue="anchor" name="parent">
    <npsobj insertvalue="var" name="parent.title" />
  </npsobj>
</npsobj>
<npsobj insertvalue="var" name="uplink" />
```

13.22 npsobj modifyvar sort

13.22.1 Syntax

```
npsobj_modifyvar_sort ::=
  <npsobj list = "name"
    sortkey1 = "sortKey1"
    [ sortmodifier1 = sortModifier ]
    [ sortkey2 = "sortKey2"
      [ sortmodifier2 = sortModifier ]
      [ sortkey3 = "sortKey3"
        [ sortmodifier3 = sortModifier ]]] />

sortModifier ::=
  "alpha" |
  "numeric" |
  "ascending" |
  "descending" |
  "alpha ascending" |
  "alpha descending" |
  "numeric ascending" |
  "numeric descending"
```

13.22.2 Task

This element sorts the value of the *name* export variable containing a context list. See the [NPSOBJ-list instruction](#) for a description of the meaning of the *sortkey1*, *sortkey2*, *sortkey3*, *sortmodifier1*, *sortmodifier2*, *sortmodifier3* attributes as well as the value of *sortModifier*.

13.22.3 Example

```
<npsobj modifyvar="set" varname="siblings" name="parent.children" />
<npsobj modifyvar="sort" varname="siblings" sortkey1="name" sortkey2="id"
sortmodifier2="numeric" />
```

13.23 npsobj newslst all

13.23.1 Syntax

```
npsobj_newslst_all ::=
  <npsobj newslst = "all" [ ( length = "length" ) ] >
    content
  </npsobj>
```

13.23.2 Task

This instruction is used for generating news lists. *all* indicates that all news are to be taken into account independently of the channels to which they are assigned. The instruction generates a list containing at most *length* file contexts and replaces the element with a sequence of texts resulting from evaluating *content* for each element in the context list.

13.23.3 Example

```
<npsobj newslst="all" length="20">
  This is evaluated for each news item
</npsobj>
```

13.24 npsobj newslst selected

13.24.1 Syntax

```
npsobj_newslst_selected ::=
  <npsobj newslst = "selected"
  ( name = "channel_field" | value = string )
  [ ( length = "length" ) ] >
    content
  </npsobj>

string = "string"
```

13.24.2 Task

This instruction is used for generating newlists. *selected* indicates that only news from the specified channels are taken into account. The channels are specified either indirectly via the *channel_field* field or directly as a string. *channel_field* can be of the type *string*, *text*, *selection*, or *multiple selection*. Channels specified directly or indirectly as a string must be separated by a comma from each other. The instruction generates a list containing at most *length* file contexts and replaces the element with a sequence of texts resulting from evaluating *content* for each element in the context list.

13.24.3 Example

```
<npsobj newlist="selected" name="channels" length="20">
  This is evaluated for each news item
</npsobj>
```

13.25 npsobj switch

13.25.1 Syntax

```
npsobj_switch ::=

<npsobj switch = "switchName">
  { <npsobj casecond = "caseCondition"
    [ negate = "" ]
    [ proceed = ( "yes" | "no" ) ]
    ( name = "caseName" | value = "caseValue" ) >
    content
  }
  [ <npsobj casecond = "default" [ proceed = ( "yes" | "no" ) ] >
    content
  ]
</npsobj>
```

13.25.2 Task

This element lets you compare a value (*switchName*) available in the current context to several other values (*caseName*) or constants (*caseValue*). For every comparison, one of the comparison operators (*caseCondition*) also available for [npsobj_condition](#) can be used. If a unary comparison operator (such as *isEmpty*) is applied to *switchName*, *caseName* and *caseValue* are ignored. Otherwise *switchName* is compared to either *caseName* or *caseValue*, depending on whether *name* or *value* has been specified. If a condition applies, the content of the *caseCond* element concerned is evaluated.

For *switchName* and *caseName* any name available in the current context can be used. To compare *switchName* to a string constant, use *caseValue*. By means of the optional *negate* attribute, the result of the comparison operation can be reversed. If *caseCondition* is no comparison operator but *default* instead, the comparison always applies. In a *switch* instruction, *default* is typically the last comparison. This makes it possible to handle the case that none of the preceding comparisons applied.

By means of the optional `proceed` attribute you can control whether the next `casecond` instruction in the `switch` block is to be processed even if the condition applies (default: `no`).

Inside a `switch` element only `casecond` elements are evaluated. Text as well as other elements on the same level are ignored.

13.25.3 Example

```
<npsobj switch="name">
  <npsobj casecond="isEqual" value="news1">
    News folder 1
  </npsobj>
  <npsobj casecond="isEqual" value="news2">
    News folder 2
  </npsobj>
  <npsobj casecond="default">
    No news folder
  </npsobj>
</npsobj>
```

13.26 npsobj table

13.26.1 Syntax

```
npsobj_table ::=

<npsobj table = "name"
[ sortedby = "sortkey" ]
[ sortkey1 = "sortKey1"
  [ sortmodifier1 = sortModifier ]
  [ sortkey2 = "sortKey2"
    [ sortmodifier2 = sortModifier ]
    [ sortkey3 = "sortKey3"
      [ sortmodifier3 = sortModifier ]]]]
[ start = "startIndex" ]
[ ( length = "length" ) | ( end = "end" ) ]
[ reverse = "" ]
[ columns = "columns" ]
[ direction = ( "horizontal" | "vertical" ) ]
{ other_attribute } >
content
</npsobj>

sortModifier ::=
"alpha" |
"numeric" |
"ascending" |
"descending" |
"alpha ascending" |
"alpha descending" |
"numeric ascending" |
"numeric descending"

other_attribute ::= string = "string"
```

13.26.2 Aufgabe

The instruction creates a table and fills the individual cells with texts. The individual texts result from the evaluation of the values *name*, *sortkey*, *startIndex*, *length*, and *end* as well as *sortKey1*, *sortKey2*, *sortKey3*, *sortModifier1*, *sortModifier2*, *sortModifier3* und *reverse* as described in the section [npsobj_list](#). If a text only contains space characters (thus spaces, tabs or line breaks), then the cell will be filled with the text ` `.

The table contains *columns* columns. If *columns* is not specified, then a single column is created. The created `table` tag contains as attributes all attributes of the NPSOBJ tag specified as *other_attribute*. For all rows a `tr` element with an opening tag and a closing tag are generated. For every cell a `td` element with an opening and a closing tag is generated. The generated `tr` and `td` start tags have no attributes.

The cells of the tables are filled with the calculated texts row by row from left to right, beginning in the uppermost line if *direction*="horizontal" is specified. Otherwise the cells are filled column by column from top to bottom, beginning with the first column. A number of rows or columns are generated that is exactly enough to create a dedicated cell for each text. Cells at the end of the last line or column for which no more text is available will be filled with the text ` `.

13.26.3 Example

```
<npsobj table="children" columns="2" direction="vertical" border="1">
  <npsobj insertvalue="var" name="title"/>
</npsobj>
```

14

14 Providing Additional Means for Editing Content in the Preview

The [preview](#) is a means for the editorial staff to edit field values and links directly, i. e. without having to leave the preview. For this purpose, small editing symbols (so-called markers) are displayed next to links and fields in the preview. The editorial system automatically embeds these markers into the preview, i. e. without editors having to activate an option. However, by means of an `npsgui` element (available from version 6.5.0), which you can embed into layouts, the markers can be explicitly switched on and off.

From version 6.5.0, in addition to automatic markers, further working aids can be given to editors, using `npsgui` elements.

The `npsgui` element provides the following features:

- You can have parts of the layout displayed in the preview in order to [supply editors with additional information for editing](#), for example. These sections in the layout may contain any kind of HTML text. They are not exported.
- You can [offer editors to perform actions](#) by clicking a marker or a linked text. As actions, menu commands, wizards, or editing a field value can be defined. In the preview of a news article, for example, you might offer the editors a possibility to edit the news article using a wizard or to create a new article.
- Furthermore, you can, as indicated above, [enable or disable the automatic generation of markers](#) so that the preview looks like the corresponding live page (unless it contains markers or hints you placed into the preview yourself).

`npsgui` elements are used analogously to `NPSOBJ` elements.

14.1 Displaying Hints in the Preview

You can have hints for editing displayed in the preview by defining a section using an `npsgui` element and placing HTML text into this section. During the export, this section is ignored. Example:

```
<npsgui>
  This text is only visible in the editorial preview.
</npsgui>
```

Using Comments

The `npsgui` element can also be used for including comments in layouts. The comments are not displayed in the Content Navigator and are not present in the published pages. Example:

```
<npsgui><!-- This layout calculates the secondary (left) navigation --></npsgui>
```

14.2 Offering Actions

In the inline preview, editors can be offered [menu commands or wizards](#) for execution or single fields for editing.

An action is displayed in the preview as a link whose appearance can be modified by specifying an icon or HTML text. If neither an icon nor HTML text is specified, the configured icon or the localized title of the action, respectively, is used. The following screenshot illustrates how this functionality can be used by the editorial staff to re-sort the items in a menu:



The actions that can be triggered effect the file in whose [context](#) they are defined. This makes it possible, for example, to edit the files displayed in a generated table of contents.

Editing elements are also offered for [mirror files](#) (which are available from version 6.5.0). The elements affect the corresponding original files.

14.2.1 Syntax

```
<npsgui insertMarker="type" name="name" icon="iconFile">HTMLcode</npsgui>
```

Parameters

- *type* defines the type of the link to be inserted:
 - *item*: The link inserted allows a menu command or a wizard to be executed.
 - *attribute*: The link inserted allows a field to be edited.
- *name*: If *type* is *item*, *name* is an [ID of a registered action](#). If *type* is *attribute*, *name* specifies the name of the field to be edited.
- *iconFile* is the name of an icon in the theme directory of the GUI web application. The size of the icon should be 31×31 pixels. This parameter is optional.

- *HTMLcode* is the HTML code to be linked so that the action can be triggered. If both an icon and HTML text are given, the text has precedence over the icon. *HTMLcode* is optional as well.

Example

The editing element with which a menu item can be moved upwards (as shown in the screenshot above) is generated by means of the following code (please note the syntax of @ references was changed in version 6.7.1):

```
<npsgui insertMarker="item" name="changeSortOrder" objId1="@{_previous}"
  objId2="@{self}">
  
</npsgui>
```

14.2.2 Parameterizing Wizards

If a link with which a wizard is executed is embedded into the inline preview, you can pass parameters to the [wizard](#). For example, in order to include a wizard for editing a particular field(../04_RTC/02_Fields/02_Edit/index.html), specify the additional `attributeName` tag attribute:

```
<npsgui insertMarker="item" name="attributeEditWizard" attributeName="name">Edit name</npsgui>
```

All tag attributes except `insertMarker`, `name`, `icon`, and `context` are added to the parameters passed to the wizard. The parameter names are prefixed with `wizard.` so that the wizard can access `attributeName`, for example, under `wizard.attributeName`.

14.2.3 Dynamically Positioned Editing Elements

From version 6.7.1, markers of the `attribute` type can be positioned dynamically to minimize the corrupting effects they may have on the layout. If this is used, the GUI will not insert the markers directly where the values are output. Instead, only `span` elements that serve as anchors will be generated. The editing elements themselves are placed into invisible containers that are positioned and made visible using JavaScript as soon as the markers need to be displayed. The editing elements are then placed on top of the respective content to be edited:

This positioning method does not work with content that is hidden or displayed using JavaScript (such as JavaScript menus).

To position markers dynamically, specify the `placeWithJS` attribute and assign `true` to it:

```
<npsgui insertMarker="attribute" placeWithJS="true" ... />
```

Please note that dynamically positioned markers may not consist of custom HTML code. Also, the icon displayed can only be changed via the [display theme](#), not by means of the `icon` attribute.

To improve dynamic positioning, or to apply effects such as highlighting to the content to be edited, the `npsgui`-Element may enclose this content. The GUI will then place the content inside the `span` element used as an anchor:

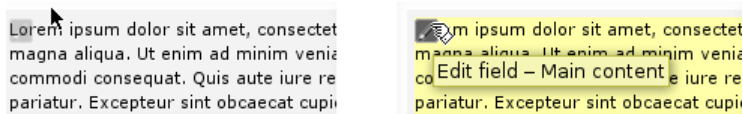
```
<npsgui ... placeWithJS="true"><npsobj insertvalue="var" name="title" /></npsgui>
```

By means of the `containerTag` attribute, a different anchor element can be specified in case `span` is disallowed (because, for example, the HTML code enclosed contains a block element):

```
<npsgui ... placeWithJS="true" containerTag="div"><p>...</p></npsgui>
```

Changing Marker Display

The way how dynamically positioned markers are displayed can be influenced by means of JavaScript. The following example illustrates how markers can be made transparent, and how the content to be edited can be highlighted when the mouse hovers over the marker.



For these effects either the JavaScript libraries [Prototype](#) and [Scriptaculous](#), or [jQuery](#) can be used. The respective libraries need to be present in the CMS as content and must be included in the `head` section of the HTML pages. The following extract from a layout file provides the required part of this section.

```
<!-- Version using Prototype and Scriptaculous -->

<npsgui>
  <script type="text/javascript" language="JavaScript"
    src="/javascripts/prototype.js"></script>
  <script type="text/javascript" language="JavaScript"
    src="/javascripts/scriptaculous.js?load=effects"></script>

  <script type="text/javascript">

    /*****
    /* Helper functions for highlighting */

    function rememberHighlightingStyle(myElement)
    {
      myElement.rememberBackgroundColor = myElement.getStyle('backgroundColor');
      myElement.rememberZoom = myElement.getStyle('zoom');
      myElement.rememberColor = myElement.getStyle('color');
    }

    function highlight(myElement)
    {
      // setting the zoom works around incomplete redraws on ie6
      myElement.setStyle({
        backgroundColor: 'ffffaa',
        zoom: 1,
        color: '#000000'
      });
    }

    function startHighlighting(myElement)
    {
      rememberHighlightingStyle(myElement);
      highlight(myElement);
    }

    function stopHighlighting(myElement)
    {
      myElement.setStyle({
        backgroundColor: myElement.rememberBackgroundColor,
        zoom: myElement.rememberZoom,
        color: myElement.rememberColor
      });
    }
  </script>
</npsgui>
```

```

/*****
/* Function for positioning and customizing the markers */

function placeEditMarker(marker, anchor, offsetLeft, offsetTop)
{
    marker.onmouseover = function() {
        new Effect.Opacity(this, { from: 0.3, to: 1.0, duration: 0.3 });
        anchor.descendants().each(function(e) { rememberHighlightingStyle(e) });
        anchor.descendants().each(function(e) { highlight(e) });
        return startHighlighting(anchor);
    };
    marker.onmouseout = function() {
        new Effect.Opacity(this, { from: 1.0, to: 0.3, duration: 0.3 });
        anchor.descendants().each(function(e) { stopHighlighting(e) });
        return stopHighlighting(anchor);
    };
    new Effect.Opacity(marker, { from: 1.0, to: 0.3, duration: 0 });
    nps.editMarker.placeEditMarker(marker, anchor, offsetLeft, offsetTop);
}
</script>
</npsgui>

<!-- Version using jQuery -->

<npsgui>
<script type="text/javascript" language="JavaScript"
    src="/javascripts/jquery.js"></script>

<script type="text/javascript">

/*****
/* Helper functions for highlighting */

function rememberHighlightingStyle(myElement)
{
    myElement.rememberBackgroundColor = $(myElement).css('backgroundColor');
    myElement.rememberZoom = $(myElement).css('zoom');
    myElement.rememberColor = $(myElement).css('color');
}

function highlight(myElement)
{
    // setting the zoom works around incomplete redraws on ie6
    $(myElement).css({backgroundColor: '#ffffaa', zoom: 1, color: '#000000'});
}

function startHighlighting(myElement)
{
    rememberHighlightingStyle(myElement);
    highlight(myElement);
}

function stopHighlighting(myElement){
    $(myElement).css({
        backgroundColor: myElement.rememberBackgroundColor,
        zoom: myElement.rememberZoom,
        color: myElement.rememberColor
    });
}

/*****
/* Function for positioning and customizing the markers */

function placeEditMarker(marker, anchor, offsetLeft, offsetTop)
{
    marker.onmouseover = function() {
        $(this).fadeTo("slow", 1);
        $(anchor).find(":visible").each(function(i) {
            rememberHighlightingStyle(this)
        });
    };
}

```

```

        $(anchor).find(":visible").each(function(i) {
            highlight(this)
        });
        return startHighlighting(anchor);
    };
    marker.onmouseout = function() {
        $(this).fadeOut("slow", 0.3);
        $(anchor).find(":visible").each(function(i) {
            stopHighlighting(this);
        });
        return stopHighlighting(anchor);
    };
    $(marker).fadeOut(1, 0.3);
    nps.editMarker.placeEditMarker(marker, anchor, offsetLeft, offsetTop);
}
</script>
</npsgui>

```

If the JavaScript function `placeEditMarker` is defined the GUI calls it to position the markers. Otherwise, the GUI will call the internal JavaScript function `nps.editMarker.placeEditMarker` for this purpose.

The `marker` and `anchor` parameters become the HTML elements of the marker and, respectively, its anchor. `offsetLeft` and `offsetTop` are used to correct the position that was calculated. They can be specified as attributes in the `npsgui` element:

```

<!-- Move marker 10px down and 20px to the left -->
<npsgui ... placeWithJS=" true" offsetTop="10" offsetLeft="-20">...</npsgui>

```

Positive values for `offsetTop` shift the marker position down, negative ones shift it up. Analogously, positive values for `offsetLeft` shift the marker position to the right while negative ones shift it to the left.

14.3 Controlling Automatic Display of Editing Elements

You can explicitly disable or enable the automatic display of markers in the preview in order to preserve the look of the pages or to have only your own markers displayed.

The automatic display of markers can be controlled individually for subsections of your website. It is also possible to activate the markers in a subsection where they are deactivated or vice versa. You can define the sections in the following way:

```

<npsgui generateAutomaticEditMarkers="Mode">...</npsgui>

```

Mode can be one of:

- `all`: All automatically generated markers are displayed.
- `brokenLinks`: Only markers for editing unresolved links are displayed.
- `none`: No automatically generated markers are displayed.