Infopark CMS Fiona

Tcl Interface Reference

Infopark CMS Fiona

Tcl Interface Reference

Contents

1 Preface

This document describes the commands of the Tcl command interfaces of the Content Management Server and the Template Engine. This document is intended for CMS and system administrators who are familiar with the Tcl script language and the functions of Infopark CMS Fiona.

The syntax of Tcl and the functions of the CMS Content Manager and the Template Engine are therefore not explained here. For an in-depth description of Tcl, see the book "Tcl and the Tk Toolkit" (John K. Ousterhout, Addison-Wesley Publishing Company, 1994) as well as a number of websites including:

- Tcl Resources
- Tcl Developer Xchange

2

2 Using the Tcl Command Interface

Both the Content Management Server and the Template Engine feature a Tcl command interface that can be used to perform administrative tasks in a quick, efficient way. This is also true for editorial work in the Content Manager's Tcl interface.

2.1 Opening the Tcl Command Interface

The Tcl command interface of a CMS Fiona application (CM, TE, or SES) can be accessed using a Tcl shell. A Tcl shell can be installed and opened on any computer, not only on the CMS server machine.

The standard CMS Fiona installation provides you with a script that opens the supplied server-side Tcl shell and executes the commands necessary to connect to the Tcl server.

In order to be able to successfully connect to the Tcl server, the respective application must be running. By default, you can log-in to the Content Manager using the credentials given on the welcome page that was displayed after the installation of CMS Fiona.

Proceed as follows to start the shell:

1. Log in to the host computer if you have not already done so.
2. Change to the directory into which the CMS application has been installed.
3. Run the *client* script file.
4. Enter the command
   `connect host port`
   where *host* is the name of the computer and *port* is the port number. Both specifications are installation-specific.
5. If you are connecting to the Content Manager enter your login name and your password when you are prompted to do so.

After you have correctly entered all data, you will be connected with the server, and you can enter Tcl commands or run Tcl scripts.

The complete calling scheme is:

```
client [-f scriptPath ] [host [port [login [password ]]]]
```

The arguments have the following meaning:

- *scriptPath*: The path to a Tcl script. This script is read and executed with the `source` command after the connection to the server has been established.

- *host*: The name of the Tcl server.
- *port*: The port at which the Tcl server can be reached.
- *login*: The user's login.
- *password*: The user's password.

2.2 Keyboard Commands

The `readline` and `history` GNU libraries are integrated into the Tcl command interface. This allows you to use the following shortcuts (EMACS notation), for example:

Cursor position

`<CTRL>-a`	Jumps to the beginning of the line
`<CTRL>-e`	Jumps to the end of the line
`<META>-b`	Jumps to the beginning of the previous word
`<META>-f`	Jumps to the beginning of the next word

Line editing

`<CTRL>-k`	Deletes the current line
`<CTRL>-h` `<backspace>`	Deletes the previous character
`<META>-<CTRL>-h` `<META>-<backspace>`	Deletes the previous word
`<CTRL>-t`	Reverses the previous two characters
`<META>-t`	Reverses the current and the previous word
`<META>-r`	Restores line from the history
`<CTRL>-_`	Undoes last change

Command history

`<CTRL>-p` `<CURSOR_UP>`	Previous command
`<CTRL>-n` `<CURSOR_DOWN>`	Next Command
`<CTRL>-r`	Incremental search backward
`<CTRL>-s`	Incremental search forward (does not work on most terminals because <CTRL>-s will be interpreted as X-OFF)
`<META>-<`	First command in the history
`<META>->`	Last command in the history

Additional commands

`<CTRL>-l`	Delete contents of window
`<META><number><command>`	Repeats <command> <number> times

2.3 Using Your Own Tcl Procedures

In any file, you can define your own Tcl procedures that you personally want available every time you start the Tcl shell. You can read them in using the Tcl command `source`.

During the initialization procedure, Tcl scripts are read from the following directories.

```
share/script/common/clientCmds
share/script/app/clientCmds
share/script/common/serverCmds
share/script/app/serverCmds

instance/default/script/common/clientCmds
instance/default/script/app/clientCmds
instance/default/script/common/serverCmds
instance/default/script/app/serverCmds
```

In the paths listed above, *app* stands for the application-specific script directory *cm* or *ses*. Please note that the Template Engine reads the scripts of the Content Management Server at startup. Scripts in the `common/clientCmds` and `common/serverCmds` directories are read by all applications at startup.

Scripts contained in the directories listed in the first group are common to all instances and are read by the corresponding CMS applications, independently of the instance.

You can place your instance-specific scripts in the directories listed in the second group. These scripts are read after the scripts of the first group so that procedures stemming from the first group can be redefined by instance-specific procedures if required.

Tcl scripts make it much easier to carry out repetitive and difficult tasks. You should check the functionality of your scripts with test data first, before working with the production data. In most cases, it is not possible to undo changes made with a script without difficulty.

2.4 Opening the Online Help

To get help on the command groups and other functions to be carried out from the Tcl shell, enter

```
help command
```

at the prompt of the Tcl shell. *command* stands for one of the keywords that are output if only `help` is entered.

2.5 Changing the Encoding

Under Windows, the Tcl client is not able to reliably determine the character encoding used by the terminal. The Tcl client therefore assumes that the encoding typically used under Windows, cp1252, is

active. If this is not the case or if data in a different or more extensive encoding (such as UTF-8) needs to be transferred, the terminal's encoding must be set accordingly. This can be done by means of the following commands (UTF-8 serves as an example here):

```
CM>fconfigure stdout -encoding utf-8
```

```
CM>fconfigure stdin -encoding utf-8
```

```
CM>fconfigure stderr -encoding utf-8
```

Additionally it can be necessary to adapt the encoding of the data read from or written to files. This is required, for example, for sourced scripts to be interpreted correctly. Please use the following command for this:

```
CM>encoding system utf-8
```

2.6 The Safe Tcl Interpreter

Tcl scripts are executed in an interpreter. The script language has a safe interpreter that blocks all access to the system and therefore completely protects the system from being compromised.

Scripts that can be maintained via the GUI or the XML interface are always executed in the safe interpreter. This is true for the following checks and functions:

- Value assignment function (`callback`) and value display function (`displayValueCallback`) for fields.
- Version assignment function (`recordSetCallback`). Here the `open` command, restricted to the blob files passed to the function, is additionally available.
- Workflow assignment function (`workflowModification`)
- Completion check (`completionCheck`)

Conversely, all routines that require write access to files are not executed in the safe interpreter but in the standard interpreter:

- The link function (`linkCallback`)
- The post-action function (`notificationCmd`)
- SystemExecute procedures
- Formatter procedures for field values and links (`dynamicLinkFormatter`)
- `generateThumbnail`
- User manager functions (*usermanAPI*)
- All Tcl files read in during system startup

Most of the procedures associated with these system calls are also available in the safe interpreter. This is a requirement for procedures executed by custom commands. It is desirable for user manager functions.

Tcl procedures can be registered with the safe interpreter using `safeInterp alias` *`serverProc clientProc`*.

3

3 On the Command Descriptions

3.1 Structure of the command descriptions

The Tcl command descriptions in this manual are ordered by command groups (e.g. `obj` or `attribute`). The command descriptions of a group are complemented by a table listing the fields or parameters for that group. For each field or parameter its name, the type of its value, and a short explanation are given. The table also specifies how the field or parameter can be accessed (read or write) and whether the value of the field or parameter can be set when creating a new instance (e.g. of a CMS file).

The command descriptions themselves contain the function declarations with an explanation of the task, the parameters, and the return value. The permissions a user must have to execute the command are listed as well.

3.2 Data types and notation

The data types of the field values and parameters and the return values of the function calls can be:

- string: character string
- stringlist: lists and arrays of strings
- number: integer number
- datetime: date and time
- bool: boolean value (`no | yes` or `false | true` or `0 | 1`)
- void: nothing (no return value, no parameter)

In Tcl funktion declarations the following symbols are used:

Symbol	Usage and example
( )	Links elements logically: `obj (withId` *`id`* `)	(withPath` *`path`* `) get` *`parameter`*
[ ]	The element in parenthesis is optional: `logout [`*`login`*`]`
{ }	The element in braces must be present at least once: `obj withId` *`id`* `mget {`*`parameter`*`}`
|	The vertical bar means that either the element to the left or to the right of it must be specified: `obj (withId` *`id`*`)	(withPath` *`path`*`) get` *`parameter`*

3.3 Date format

All date and time stamp specifications are stored internally in GMT in canonical form as a 14-place string (starting from the left: Year (4 digits), Month (2 digits), Day (2 digits), Hour (2 digits), Minute (2 digits), Second (2 digits)).

For information on converting date and time stamp specifications from the canonical form to a conventional form, see System Configuration. The user defaults for the time zone and the output format are used during conversions of a date or time specification if you use the `userConfig` command instead of the `systemConfig` command (see User Preferences).

3.4 Identifier requirements

No identifier used in the Content Manager (for field names, logins, group names, CMS file names, etc.) may contain spaces or special characters. All characters except a-z, A-Z, 0-9, and the underscore are considered special characters. CMS file names may also contain the Dollar sign and the hyphen. On export, disallowed characters in file names are converted to underscores.

4

4 The Tcl Commands of the Content Manager and the Template Engine

The Tcl commands of the Content Manager and the Template Engine are divided into several groups according to their application areas:

4.1 attribute (Version Fields)

4.1.1 Field Parameters

The field parameters can be accessed using field commands (for examples, please refer to the individual command descriptions). The access types available for a field parameter are indicated in the columns to the right of the explanation column. The `create` column indicates whether a parameter can be specified when creating a field. In the `descr` column, the parameters included in field descriptions (see the `description` field command) are tagged.

Parameters	Type	Explanation	get	set	create	descr
callback	string	Tcl script that is called after a value has been assigned to a field	•	•	•	•
displayTitle	string	The string that is displayed as the field's title in the HTML user interface	•			
displayValueCallback	string	Tcl script that is called to calculate a custom field's display value (see also the `displayValue` version field in the section Version fields)	•	•	•	•
editField	stringlist	Definition of the input field to be used for value assignment	•			•
editFieldSpec	stringlist	Complete specification of the input field - also contains field names and enumeration values	•			
getKeys	stringlist	List of parameters that can be queried with `get`	•			
helpText	string	Help text for the field	•	•	•	•
helpText. *language*	string	Help text for the field in the respective language (from the localization section of the system configuration)	•	•	•	•

isSearchableInCM	bool	Specifies whether the field values can be searched in the Content Manager if the search engine is running.	•	•	•	•
isSearchableInTE	bool	Specifies whether the field values can be searched on the live server if the search engine is running.	•	•	•	•
maxSize	integer	Maximum number of links in the list (from version 6.7.0)	•	•	•	•
minSize	integer	Minimum number of links in the list (from version 6.7.0)	•	•	•	•
name	string	Name of the field	•		•	•
setKeys	stringlist	List of parameters that can be set with `set`	•			
title	string	Title of the field in the user-specific language	•	•	•	•
title. *language*	string	Title of the field in the respective language (from the localization section of the system configuration)	•	•	•	•
type	string	Type of the field (permitted: `string` (default), `text, date, enum, multienum, html, signature, linklist`, from version 6.6 additionally `markdown`)	•		•	•
validEditFieldKeys	stringlist	List of the possible parameter names in the input field definition (depends on input field type)	•			
validEditFieldTypes	stringlist	List of the possible input field types (depends on the field type)	•			
values	stringlist	Enumeration values (only with *enum* and *multienum* fields)	•	•	•	•
wantedTags	stringlist	List of the HTML tags allowed in the field value (only for fields of *html* and `markdown` types) (the empty value means: all).	•	•	•	•

4.1.2 Input Field Parameters

Input field parameter	Type	Explanation	get	set
editorUrl	string	Prior to NPS 6.0: With attributes of type HTML the URL with which the custom HTML editor can be invoked (available as a parameter only for the `custom` input field type).	•	•
length	number	The display width most suited for a field (available as a parameter only for `textfield`, `passwordfield`, `textarea`)	•	•

maxlength	number	The maximum length of a text that can be entered in a field (only available as a parameter for `textfield`, `passwordfield`)	•	•
nilAllowed	bool	Specifies with `multiselect` and `popup` input fields whether the empty value can be selected in addition to the field's enumeration values. The empty value is displayed as a slash in the input field.		
objClasses	list	A list of file formats to which the link destination selection is restricted (as a parameter available for `linklistfield` from version 6.6.2).	•	•
rows	number	The number of lines of the field to be displayed (available as a parameter only for `textarea`, `multiselect`)	•	•
startPub	string	The path of the starting folder for the link destination selection (as a parameter available for `linklistfield` from version 6.6.2).	•	•
type	string	The type of input field. The following types are available: `textfield`, `passwordfield`, `textarea`, `multiselect`, `popup`, `radio`, `checkboxes`, `custom`, `html`, `hidden`, `external`, `linklist`, `wizard`. Not all types are available for all field types.	•	•
wizard	string	The wizard to be used (as a parameter only available if the `type` is `wizard`). The wizard name is equal to its Tcl name space.	•	•

The input field type `linklist` is available from version 6.5.0. From this version, link lists can also be edited by means of a wizard.

4.1.3 Field Commands

attribute create

Available for: Content Management Server

Task: Creates a new user defined field with the specified values.

Syntax:

`attribute create {`*`parameter value`*`}`

Function parameters:

- *parameter* specifies the name of the field parameter whose value is to be set during creation of the field. The field name must be specified.
- *`value`* is the value to be set for the parameter.

Return value if successful: name of the field (string)

Necessary permissions: The user must have the `permissionGlobalRTCEdit` permission.

Example:

```
CM>attribute create name texttype type multienum
texttype
```

attribute list

Available for: Content Management Server

Task: Lists the name of all fields.

Syntax:

`attribute list`

Function parameters: none

Return value if successful: list of field names (stringlist)

Necessary permissions: no restrictions

Example:

```
CM>attribute list
author revision toc_list_type
```

attribute types

Available for: Content Management Server

Task: Lists the existing field types.

Syntax:

`attribute types`

Function parameters: none

Return value if successful: list of field types (stringlist)

Necessary permissions: no restrictions

Example:

```
CM>attribute types
date enum html multienum signature string text markdown
```

attribute where

Available for: Content Management Server

Task: Searches for fields with particular parameter values, e.g. fields whose `type` is `string`, or whose `name` contains a given sequence of characters. If more than one parameter/value pair is specified, a field must meet all search criteria for the field to become part of the search result.

Syntax:

`attribute where {parameter value}`

Function parameters: (If no parameter is specified, the command returns the names of all fields.)

- *parameter* specifies the name of the parameter whose value is to be searched for or specifies the number of hits. The following parameters can be used:
  - `maxResults` specifies that *value* is a number that limits the number of hits. If number is less than or equals zero, the number of hits is unlimited.
  - `name` specifies that the names of the fields are to be checked for occurrences of *value*.
  - `type` specifies that *value* is a single attribute type or a list of attribute types. A field's type must be equal to the single value or part of the list, respectively, to be included in the search result.
- *value* specifies the search term for the respective parameter.

Return value if successful: the names of the fields that match the search criteria (stringlist)

Necessary permissions: no restrictions

Example:

```
CM>attribute where name dat
datasheet datasheet1 datasheet2
```

attribute attrRef addEnumValues

Available for: Content Management Server

Task: Adds the specified enumeration values to an enum or multienum field if they do not already exist.

Syntax:

`attribute withName attrName addEnumValues {enumValue}`

Function parameters:

- *enumValue* (string) specifies a new enumeration value to be added to the existing enumeration values of the field.

Return value if successful: none

Necessary permissions: The user must have the `permissionGlobalRTCEdit` permission.

Example:

```
CM>attribute withName texttype addEnumValues Report Advertisment
Report Advertisment
```

attribute attrRef delete

Available for: Content Management Server

Task: Deletes the specified field.

Syntax:

`attribute withName attrName delete`

Function parameters: none

Return value if successful: none

Necessary permissions: The user must have the `permissionGlobalRTCEdit` permission.

Example:

```
CM>attribute withName texttype delete
```

attribute attrRef description

Available for: Content Management Server

Task: Returns a string representation of the data from the field with the specified name.

Additional information: The format of the string representation is the property list format of a dictionary.

Syntax:

`attribute attrRef description`

Function parameters: none

Return value if successful: string representation of the field (string)

Necessary permissions: no restrictions

Example:

```
CM>attribute withName datasheet description
{
  editField = {
    length = 65;
    type = textfield;
  };
  name = datenblatt;
  title = datenblatt;
  type = string;
}
```

attribute attrRef editField get

Available for: Content Management Server

Task: Returns the value of the specified parameter of the input field specification of the field *attrName*.

Syntax:

`attribute withName attrName editField get parameter`

Function parameters:

- *parameter* specifies the name of the parameter of the input field specification whose value is being searched for.

Return value if successful: value of the specified input field parameter (string)

Necessary permissions: no restrictions

Example:

```
CM>attribute withname datasheet editfield get type
textfield
```

attribute attrRef editField mget

Available for: Content Management Server

Task: Returns the values of the specified parameters of the input field specification of the field *attrName*.

Syntax:

`attribute withName` *attrName* `editField mget {`*parameter*`}`

Function parameters:

- *parameter* specifies the name of the parameter of the input field specification whose value is being searched for.

Return value if successful: the values of the specified input field parameter (stringlist)

Necessary permissions: no restrictions

Example:

```
CM>attribute withName datasheet editField mget type length
textfield 65
```

attribute attrRef editField set

Available for: Content Management Server

Task: Sets the specified input field parameter of the *attrName* field to the specified values.

Syntax:

`attribute withName` *attrName* `editField set {`*parameter value*`}`

Function parameters:

- *parameter* specifies the parameter from the field whose value is to be set.
- *value* is the field parameter value to be set.

Return value if successful: 1

Necessary permissions: The user must have the `permissionGlobalRTCEdit` permission.

Example:

```
CM>attribute withName datasheet editField set type textfield
```

attribute attrRef get

Available for: Content Management Server

Task: Returns the value of the specified field parameter of the *attrName* field.

Syntax:

`attribute withName` *attrName* `get` *parameter*

Function parameters:

- *parameter* specifies the name of the parameter whose value is being searched for.

Return value if successful: the value of the entered parameter (string)

Necessary permissions: no restrictions

Example:

```
CM>attribute withName datasheet get editField
type textfield length 65
```

attribute attrRef mget

Available for: Content Management Server

Task: Returns the values of the specified parameters of the *attrName* field.

Syntax:

`attribute withName` *attrName* `mget {`*parameter*`}`

Function parameters:

- *parameter* specifies the name of the parameter whose value is being searched for.

Return value if successful: the list of values for the parameter concerned (stringlist)

Necessary permissions: no restrictions

Example:

```
CM>attribute withName datasheet mget type editField
string {type textfield length 65}
```

attribute attrRef removeEnumValues

Available for: Content Management Server

Task: Deletes the specified enumeration values of a selection or multi-selection (`enum`/`multienum`) field.

Syntax:

`attribute withName` *`attrName`* `removeEnumValues {`*`enumValue`*`}`

Function parameters:

- *`enumValue`* indicates the enumeration value to be deleted from the defined enumeration values of the field.

Return value if successful: none

Necessary permissions: The user must have the `permissionGlobalRTCEdit` permission.

Example:

```
CM>attribute withName texttype removeEnumValues Report
```

attribute attrRef set

Available for: Content Management Server

Task: Sets parameters of a field to the specified values.

Syntax:

`attribute withName` *`attrName`* `set {`*`parameter value`*`}`

Function parameters:

- *`parameter`* specifies the parameter of the field whose value is to be set.
- *`value`* is the value to be set for the parameter.

Return value if successful: none

Necessary permissions: The user must have the `permissionGlobalRTCEdit` permission.

Example:

```
CM>attribute withName texttype set title {Text type}
```

4.2 attributeGroup (Field Groups)

4.2.1 AttributeGroup Parameters

Parameter	Type	Explanation	get	set	create	descr
attributes	stringlist	The names of the fields assigned to the group. The names are returned in the same order as they are assigned to the group.	•			•
displayTitle	string	Title of the field group displayed in the HTML user interface	•			
getKeys	stringlist	List of parameters which can be queried with `get` .	•			
identifier	string	Name of the file format and name of the field group to be created, separated by a period.			•	
index	number	Index of the group in the file format's list of groups. The index of the first field group is 0.	•		•	•
isDefaultGroup	bool	Returns *1* if the group is the base group, otherwise *0*.	•			
isEmpty	bool	Returns *1* if the group contains no fields, otherwise *0*.	•			
localizedTitle	string	Localized title in the language the user has selected. If this title is empty the name of the group will be returned.	•			
name	string	Name of the field group. The name of the base group (`baseGroup`) cannot be changed.	•	•		•
objClass	string	Name of the file format the group belongs to.	•			
setKeys	stringlist	List of parameters which can be set with `set`.	•			
title	string	Title of the field group in the user-specific language.	•	•	•	•
title. *language*	string	Title of the field group in the respective language.	•	•	•	•

4.2.2 AttributeGroup Commands

In `attributeGroup` commands the field groups are referenced by identifiers made up of two parts. Please specify as the first part the name of the format the group belongs to. The second part is the group's name. It must be separated from the first part by a period. Example:

attributeGroup create

Available for: Content Management Server

Task: Creates a field group.

Syntax:

`attributeGroup create {parameter value}`

Function parameters:

- *parameter* specifies the name of an attributeGroup parameter whose value is to be set when the group is created. The parameter *identifier* must be specified.
- *value* is the value to be set for the attributeGroup parameter *parameter*. As value of the *identifier* parameter the identifier of the new group must be specified. It is made up of the format name, a period and the name of the new group.

Return value if successful: The name of the new group.

Necessary permissions: The user must have the permissionGlobalRTCEdit permission.

Example: In the format newsPub this creates a field group named headlines. Its title is set to Headlines:

```
CM>attributeGroup create identifier newsPub.headlines \
title Headlines
headlines
```

attributeGroup attrGroupRef addAttribute

Available for: Content Management Server

Task: This command adds a field to a field group.

Additional information: The field must have been assigned to the file format the group belongs to. You cannot assign a field to the base group. Only fields assigned to the base group can be assigned to a different group.

Syntax:

`attributeGroup withIdentifier groupId addAttribute {parameter value}`

Function parameters:

- *parameter* specifies the name of a parameter to be taken into account when the field is assigned to the field group. The following parameters can be specified:
  - attribute denotes that *value* is the name of the field. This parameter is obligatory.
  - index denotes that *value* specifies the position at which the field is to be inserted into the group's field list. If this parameter is not specified the field will be appended to this list. The index of the first field is 0.
- *value* specifies the value of the respective parameter.

Return value if successful: none.

Necessary permissions: The user must have the `permissionGlobalRTCEdit` permission.

Example: Add the field `main` to the field group `headlines` in the format `newsPublication` and make it the first field of the group.

```
CM>attributeGroup withIdentifier newsPub.headlines \
addAttribute attribute main index 0
```

attributeGroup attrGroupRef addAttributes

Available for: Content Management Server

Task: With this command fields can be added to a field group.

Additional information: Each specified field must have been assigned to the format the group belongs to. You cannot assign a field to the base group. Only fields assigned to the base group can be assigned to a different group.

Syntax:

`attributeGroup withIdentifier groupId addAttributes {`*`attribute`*`}`

Function parameters:

- *`attribute`* specifies the name of a field that is to be assigned to the group.

Return value if successful: none.

Necessary permissions: The user must have the `permissionGlobalRTCEdit` permission.

Example: Add the fields `head1` and `head2` to the `headlines` group belonging to the format `newsPub`:

```
CM>attributeGroup withIdentifier newsPub.headlines \
addAttributes head1 head2
```

attributeGroup attrGroupRef delete

Available for: Content Management Server

Task: This command deletes a field group.

Additional information: The fields contained in the group will automatically be added to the base group. The base group cannot be deleted.

Syntax:

`attributeGroup withIdentifier` *`groupId`* `delete`

Function parameters: none.

Return value if successful: none.

Necessary permissions: The user must have the `permissionGlobalRTCEdit` permission.

Example: Delete the `related` field group belonging to the `newsPub` file format:

```
CM>attributeGroup withIdentifier newsPub.related delete
```

attributeGroup attrGroupRef description

Available for: Content Management Server

Task: Returns a string representation of a field group's data. The representation is formatted in the property list format of a dictionary.

Syntax:

`attributeGroup withIdentifier` *`groupId`* `description`

Function parameters: none.

Return value if successful: the string representation of the field group (string).

Necessary permissions: none.

Example:

```
CM>attributeGroup withIdentifier newsPub.headlines description
{
  name = headlines;
  title = Headlines;
  title.de = "";
  title.en = "";
  index = 1;
  attributes = (head1, head2);
}
```

attributeGroup attrGroupRef get

Available for: Content Management Server

Task: This command returns the value of the specified `attributeGroup` parameter.

Syntax:

`attributeGroup withIdentifier` *`groupId`* `get` *`parameter`*

Function parameters:

- *`parameter`* specifies the name of the `attributeGroup` parameter whose value is to be returned.

Return value if successful: the parameter's value.

Necessary permissions: none.

Example: List the fields assigned to the `newsPub.headlines` group:

```
CM>attributeGroup withIdentifier newsPub.headlines get attributes
head1 head2
```

attributeGroup withIdentifier groupId mget {parameter }

Available for: Content Management Server

Task: This command returns the values of any number of attributeGroup parameters.

Function parameters:

- *parameter* specifies the name of a field group parameter whose value is to be read.

Return value if successful: the values of the parameters specified.

Necessary permissions: none.

Example: Get the index of the field group `newsPub.headlines` and the names of the fields assigned to it:

```
CM>attributeGroup withIdentifier newsPub.headlines mget index attributes
```

```
1 {head1 head2}
```

attributeGroup attrGroupRef moveAttribute

Available for: Content Management Server

Task: This command moves a field group's field to a different location.

Syntax:

`attributeGroup withIdentifier` *groupId* `moveAttribute {`*parameter value*`}`

Function parameters:

- *parameter* specifies the name of a parameter to be taken into account when the field is moved. The following parameters must be specified:
  - `attribute` denotes that *value* is the name of the field to be moved.
  - `index` denotes that *value* specifies the position at which the field is to be inserted into the group's field list. The index of the first field is 0.
- *value* specifies the value of the respective parameter.

Return value if successful: none.

Necessary permissions: The user must have the `permissionGlobalRTCEdit` permission.

Example: Make the field `main` the second field on the field group `newsPub.headlines`:

```
CM>attributeGroup withIdentifier newsPub.headlines moveAttribute attribute main index 1
```

attributeGroup attrGroupRef moveToIndex

Available for: Content Management Server

Task: This command moves a field group to a different location in the format's list of field groups.

Syntax:

`attributeGroup withIdentifier` *`groupId`* `moveToIndex` *`index`*

Function parameters:

- *`index`* specifies the position at which the field group is to be inserted into the file format's list of groups. The index of the first group is 0.

Return value if successful: none.

Necessary permissions: The user must have the `permissionGlobalRTCEdit` permission.

Example: Make the field group `newsPub.headlines` the third group in the file format's list of groups:

```
CM>attributeGroup withIdentifier newsPub.headlines moveToIndex 2
```

attributeGroup attrGroupRef removeAttribute

Available for: Content Management Server

Task: This command removes a field from a field group.

Additional information: You cannot remove fields from the base group. Fields removed from groups will automatically be assigned to the base group.

Syntax:

`attributeGroup withIdentifier` *`groupId`* `removeAttribute` *`attribute`*

Function parameters:

- *`attribute`* specifies the name of the field to be removed.

Return value if successful: none.

Necessary permissions: The user must have the `permissionGlobalRTCEdit` permission.

Example: Remove the field `head2` from the field group `newsPub.headlines`:

```
CM>attributeGroup withIdentifier newsPub.headlines removeAttribute head2
```

attributeGroup attrGroupRef set

Available for: Content Management Server

Task: This command sets parameter values for the field group specified.

Syntax:

`attributeGroup withIdentifier` *`groupId`* `set {`*`parameter value`*`}`

Function parameters:

- *parameter* specifies the name of a field group parameter whose value is to be set.
- *value* specifies the value to which the respective field group parameter is to be set.

Return value if successful: none.

Necessary permissions: The user must have the `permissionGlobalRTCEdit` permission.

Example: Set the English title of the field group `newsPub.headlines` to `Headlines`:

```
CM>attributeGroup withIdentifier newsPub.headlines \
set title.en Headlines
```

4.3 channel (News Channels)

4.3.1 Channel Parameters

Parameter	Type	Explanation	get	set	create	descr
name	string	Name of the channel.	•	•	•	•
title. *language*	string	Title of the channel in the respective language. The language identifiers are defined in the *localizers* system configuration entry.	•	•	•	•

4.3.2 Channel Commands

Channels allow you to categorize versions according to their topic. If the Portal Manager is used, website visitors can subscribe to channels in order to access their favorite documents.

The channels are maintained in the Content Management Server. By means of the `channels` version field, versions can be assigned to any number of the defined channels.

channel create

Available for: Content Management Server

Task: This command creates a channel.

Syntax:

`channel create {`*parameter value*`}`

Function parameters:

- *parameter* specifies the name of a channel parameter whose value is to be set when the channel is created. The parameter `name` must be specified.
- *value* is the value to be set for the corresponding parameter.

Return value if successful: The name of the new channel.

Necessary permissions: The user must have the `permissionGlobalRTCEdit` permission.

Example:

```
CM>channel create name weather.europe title.en "European Weather"
weather.europe
```

channel list

Available for: Content Management Server

Task: The command returns the list of the names of all channels.

Syntax:

`channel list`

Function parameters: none.

Return value if successful: none.

Necessary permissions: no restrictions.

Example:

```
CM>channel list
weather.europe weather.europe.france weather.europe.germany
```

channel where

Available for: Content Management Server

Task: Allows you to search for all channels for which the value of the specified parameter contains the specified string.

Syntax:

`channel where {`*`parameter value`*`}`

Function parameters: (If no parameter is specified, the command outputs all channels.)

- *`parameter`* specifies the name of the parameter whose value is to be searched for or specifies the number of hits. The following parameters are allowed:
  - `maxResults` specifies that *`value`* is a number that limits the number of hits. If number is less than or equals zero, the number of hits is unlimited.
  - `name` specifies that the names of the channels are to be checked for correspondence with *`value`*.
  - `namePrefix` specifies that the names are to be compared from the beginning with *`value`*.
- *`value`* contains the value of the corresponding parameter.

Return value if successful: the list of names of the matching channels (stringlist)

Necessary permissions: no restrictions

Example:

```
CM>channel where namePrefix weather
weather weather.europe weather.europe.france weather.europe.germany
```

channel channelRef delete

Available for: Content Management Server

Task: This command deletes the specified channel.

Syntax:

`channel withName` *`channelName`* `delete`

Function parameters: none.

Return value if successful: none.

Necessary permissions: The user must have the `permissionGlobalRTCEdit` permission.

Example:

```
CM>channel withName weather.europe.france delete
```

channel channelRef description

Available for: Content Management Server

Task: Returns a string representation of a channel's data. The representation is formatted in the property list format of a dictionary.

Syntax:

`channel withName` *`channelName`* `description`

Function parameters: none.

Return value if successful: the string representation of the channel (string).

Necessary permissions: none.

Example:

```
CM>channel withName weather.europe description
{
  name = weather.europe;
  title.de = "Europawetter";
  title.en = "European Weather";
}
```

channel channelRef get

Available for: Content Management Server

Task: This command returns the value of the specified channel parameter.

Syntax:

`channel withName channelName get parameter`

Function parameters:

- *parameter* specifies the name of the channel parameter whose value is to be returned.

Return value if successful: the parameter's value.

Necessary permissions: none.

Example:

```
CM>channel withName weather.europe get title.en
European Weather
```

channel channelRef mget

Available for: Content Management Server

Task: This command returns the values of any number of channel parameters.

Syntax:

`channel withName channelName mget {parameter}`

Function parameters:

- *parameter* specifies the name of a channel parameter whose value is to be read.

Return value if successful: the values of the parameters specified.

Necessary permissions: none.

Example:

```
CM>channel withName weather.europe get title.en title.de
European Weather Europawetter
```

channel channelRef set

Available for: Content Management Server

Task: This command sets parameter values for the channel specified.

Syntax:

`channel withName channelName set {parameter value}`

Function parameters:

- *parameter* specifies the name of a channel parameter whose value is to be set.

- *value* specifies the value to which the respective channel parameter is to be set.

Return value if successful: none.

Necessary permissions: The user must have the `permissionGlobalRTCEdit` permission.

Example:

```
CM>channel withName weather.europe set title.en "European Weather"
```

4.4 content (Versions)

4.4.1 Version Fields

The following overview of the version fields applies to the Content Management Server. In the Template Engine, only fields are available whose names do not include further components (such as `.plain`). Furthermore, in the Template Engine all field values are read-only.

Field	Type	Explanation	get	set	descr
anchors	stringlist	List of the anchor names in the version (only for files of type `folder` or `document`)	•		
blob blob.plain	string	For layouts, folders and documents only: String representation of the version's main content.	•	•	
blob.base64	string	The version's main content., base64-encoded.	•	•	
blob.stream	string	For read access the streaming ticket is returned with which the blob can be downloaded via the streaming interface. For write access the streaming ticket is specified that was returnd when the blob was uploaded.	•	•	
blobLength	number	For files of the `image` and `generic` type the length of the binary data is returned, for other file types the size of the main content in bytes.	•		
body	string	For layouts, folders and documents only: Pre-processed version of the HTML code of the version that is used for internal purposes	•		
channels	stringlist	The list of channels to which the version is assigned.	•	•	•
contentType	string	The file name extension of the blob assigned to the version. To versions of `document` and `folder` files the following applies: If the MIME type belonging to the file name extension is *text/html*, then the	•	•	•

		main content is processed by the Content Manager with the effect that, for example, links contained in it are managed by the link management. Otherwise the main content is treated as pure text. See also the `mimeTypes` system configuration entry.			

displayTitle	string	The title of the version as it is displayed in the HTML user interface (corresponds to the value of `title`).	•		
editor	string	The login name of the user who edits the version	•		•
exportBlob	string	For layouts, folders and documents only: String representation of the blob. Using the appropriate layouts and depending on the file the version belongs to, the blob is put in the final export form (for folders and documents). Which layouts are appropriate also depends on the `preferEditedTemplates` user preference.	•		
exportBlob.plain	string	like *exportBlob*	•		
exportBlob.base64	string	like `exportBlob`, base64-encoded	•		
exportBlob.stream	string	like `exportBlob`, however transferred via the streaming interface (see description of `blob.stream`)	•		
exportFiles	stringlist	A list containing a pair of values for each file that will be generated when the version is exported. The first value in such a pair is the file name (without path), the second value is the version of that file.	•		
externalAttrNames	stringlist	The list of all custom fields assigned to the version. The list is assembled using the file format of the file the version is assigned to.	•		
frameNames	stringlist	The list of all the names of the frames generated when the version is exported.	•		
freeLinks	stringlist	The list of the IDs of the free links occurring in the version	•		
getKeys	stringlist	List of version fields that can be queried with `get`	•		
hasNewsItem	bool	Specifies whether the version has news items, i. e. whether it is released, channels are assigned to it, and `canCreateNewsItems` is 1 in its file format.	•		
hasThumbnail	bool	Indicates whether the version has a thumbnail.	•		
height	number	The height of the image contained in versions belonging to files of the `image` type, if the image has one of the supported formats (GIF, JPG, PNG). 0 otherwise.	•		
isActive (from version 6.5.0)	bool	Indicates whether the version is temporally valid.	•		•
isCommitted	bool	Indicates whether the version is a committed version	•		
isComplete	bool	Indicates whether the version is complete	•		

isEdited	bool	Indicates whether the version is a draft version	•		
isReleased	bool	Indicates whether the version is a released version.	•		
lastChanged	string	The date the version was last changed	•		
linkListAttributes	stringlist	The list of the linklist fields assigned to the version.	•		
mimeType	string	The MIME type of the version determined by means of the assignments of content types to MIME types in the system configuration (`mimeTypes` entry).	•		
nextEditGroup	string	The name of the next group in the processing workflow	•		
nextSignGroup	string	The name of the next group in the signature workflow	•		
objectId	string	ID of the file belonging to the version	•		
reasonsForIncompleteState	stringlist	The list of the reasons why a draft version is incomplete	•		
setKeys	stringlist	List of version fields that can be set with `set`	•		
signatureAttrNames	stringlist	The list of all signature fields assigned to the version. The list is assembled from the workflow of the file belonging to the version.	•		
sortKey1	string	First sort key component (can only be set for folders)	•	•	
sortKey2	string	Second sort key component (can only be set for folders)	•	•	
sortKey3	string	Third sort key component (can only be set for folders)	•	•	
sortKeyLength1	number	Number of significant characters of the first sort key component (can only be set for folders)	•	•	
sortKeyLength2	number	Number of significant characters of the second sort key component (can only be set for folders)	•	•	
sortKeyLength3	number	Number of significant characters of the third sort key component (can only be set for folders)	•	•	
sortOrder	string	Sorting order (only for folders)	•	•	
sortType1	string	Sorting mode of the first sort key component (only for folders)	•	•	
sortType2	string	Sorting mode of the second sort key component (only for folders)	•	•	
sortType3	string	Sorting mode of the third sort key component (only for folders)	•	•	

subLinks	stringlist	The list of the IDs of the links occurring in the version	•		
textLinks	stringlist	The list of the IDs of the text links and the include links occurring in the version	•		
thumbnail	string	A base-64-encoded preview image in the JPEG format. Only available for *image* or `generic` files. This field may only be used in the GUI (Content Navigator, Wizards), not during export.	•		
title	string	The title of the version	•	•	•
validFrom	string	The date the validity of the version begins in the 14-place canonical format (ignored for layouts). If `validUntil` is set, its value must be greater than `validFrom`.	•	•	
validSortKeys	stringlist	The list of all valid values for the sortKey*n* field	•		
validSortOrders	stringlist	The list of all valid values for the `sortOrder` field	•		
validSortTypes	stringlist	The list of all valid values for the `sortTypen` field	•		
validUntil	string	The date the validity of the version ends in the 14-place canonical format (ignored for layouts).	•	•	
width	number	The width of the image contained in versions belonging to files of the `image` type, if the image has one of the supported formats (GIF, JPG, PNG). 0 otherwise.	•		
workFlowComment	string	The comments on the last workflow action applied to this version	•		
xmlBlob	string	String representation of the blob of the version as an XML document (not available for layouts)	•	•	
Custom field		The value of the custom field	•	•	•

The built-in version fields listed above are supplemented by the custom version fields. The values of these fields can also be read and set (setting them is only possible in the Content Management Server). Furthermore, with each version field a display value can be associated. This value is calculated by the field's display value function whenever the field is assigned a value. To read display values, you can use code as shown in the following example:

```
content withId 65429 get customattribute.displayValue
```

For further information about display value functions, see also Field parameters.

4.4.2 Version Commands

The version commands make it possible to access CMS file versions directly via their ID instead of using one of the following file subcommands:

- `obj (withId objId) | (withPath path) | root editedContent versionSubCommand`
- `obj (withId objId) | (withPath path) | root releasedContent versionSubCommand`

content contentRef addLinkTo

Available for: Content Management Server

Task: Creates a free link and adds it to the specified field in the draft version.

Syntax:

`content withId` *`contentId`* `addLinkTo attribute` *`attrName`* `destinationUrl` *`urlString`*

Function parameters:

- *`attrName`* is the name of the field to which the free link is to be added. The concerning field's type must be `linklist`.
- *`target`* is the target frame.
- *`title`* is the link title.
- *`destinationUrl`* is the link destination.

Return value if successful: the ID of the new link (string).

Necessary permissions: The user must be the editor of the version and have the `permissionWrite` permission for the file the version belongs to.

Example: Add a free link to the `imageLinks` field in version with id 25687; the link points to the `location` file in the same folder::

```
CM>content withId 25687 addLinkTo \
   attribute imageLinks destinationUrl location
9983433
```

Example: Add a free link to the `imageLinks` field in the draft version of the `newslist` file; the link points to the `location` file in the same folder:

```
CM>obj withPath /newslist editedContent addLinkTo \
   attribute imageLinks destinationUrl location
9983434
```

content contentRef debugExport

Available for: Content Management Server (up to version 6.0.x, from version 6.5.0 see obj *objRef* debugExport)

Task: For the purpose of finding errors in the code of layout files, this command simulates the export of the version with the specified ID. It becomes clear from the output which NPSOBJ tags from which source layouts are evaluated in which order and which errors occurred during the evaluation process.

Additional information:

- The command provides information about the following errors: unclosed tags, unbalanced number of opening and closing tags, disallowed attributes in NPSOBJ tags, disallowed values of NPSOBJ tag attributes, read-access to undefined names.
- The contents of `systemExecute` instructions will neither be searched for errors nor formatted. However, the output of such instructions is formatted.

Syntax:

```
content withId contentId debugExport
  [templateName templateName]
  [detailed (yes | no)]
  [quoteHtml (yes | no)]
  [errorPrefix errorPrefix]
  [errorSuffix errorSuffix]
  [htmlPrefix htmlPrefix]
  [htmlSuffix htmlSuffix]
  [infoPrefix infoPrefix]
  [infoSuffix infoSuffix]
  [preferEditedTemplates (yes | no)]
  [allowEditedContents (yes | no)]
```

Function parameters:

- *templateName*: Specifies the base layout (initial layout file) with which the export test is to be performed. If *templateName* has not been specified, the user-specific standard layout is used (*mastertemplate* by default).
- *detailed*: Specifies whether the output is to be restricted to error messages (`no`) or should rather be detailed (`yes`). The default value is `no`.
- *quoteHtml*: Specifies whether the characters <, >, and & are to be output as HTML entities (`yes`) or not (`no`). The prefix und suffix parameters are not affected by this. The default value is `no`.
- *errorPrefix*: Specifies a character string which is output prior to each error message. The default value is `"***"`.
- *errorSuffix*: Specifies a character string which is output after each error message (empty by default).
- *htmlPrefix*: Specifies a character string which is output prior to HTML text. The default value is "<pre>".
- *htmlSuffix*: Specifies a character string which is output after HTML text. The default value is "</pre>".
- *infoPrefix*: Specifies a character string which is output prior to each NPSOBJ information message (empty by default).
- *infoSuffix*: Specifies a character string which is output after each NPSOBJ information message (empty by default).
- *preferEditedTemplates*: Specifies whether the draft versions of layout files are to be preferred to the released versions. The default value corresponds to the user configuration setting of the same name, which is also used for the preview.
- *allowEditedContents*: Specifies whether draft versions are to be used if released versions are not available. The default value corresponds to the state of the version for which this command is executed. The command does not refer to this version but to the versions it references.

Return value if successful: the formatted export report (string).

Necessary permissions: The user must have the `permissionGlobalExport` permission or the `permissionRead` permission for the file the version belongs to.

Example: (see below for a sample output)

```
CM>content withId 25687 debugExport detailed yes
```

Example for the usage as a subcommand of `obj`: (output strongly abridged)

```
CM>obj withId 2792 editedContent debugExport detailed yes
<HTML>
    <HEAD>
  ...
NPSOBJ insertvalue=var: title
The title
END NPSOBJ insertvalue=var
</TITLE>
    </HEAD>
NPSOBJ insertvalue=var: body
<a ...>Linked text</a>

*** ERROR [140008] The instruction attribute was missing from an NPSOBJ tag.
...
```

content contentRef delete

Available for: Content Management Server

Task: Deletes an archived version. For deleting a draft version, use the `obj revert` command.

Syntax:

`content withId` *`contentId`* `delete`

Function parameters: none

Return value if successful: none

Necessary permissions: The user must have the `permissionRoot` permission for the file the version belongs to.

Example: Delete all archived versions of a file:

```
CM>foreach i [obj withPath /news get archivedContentIds] \
   {content withId $i delete}
```

content contentRef description

Available for: Content Management Server

Task: Returns a string representation of the data of the version with the specified ID.

Additional information: The representation is formatted in the property list format of a dictionary.

Syntax:

`content contentRef description`

Function parameters: none

Return value if successful: the string representation of the version (string)

Necessary permissions: The user must have the `permissionRead` permission for the file the version belongs to.

Example: (see below for a sample output)

```
CM>content withId 25687 description
```

Example using the corresponding `obj` command:

```
CM>obj withPath /news editedContent description
{
  contentType = html;
  editor = katrin;
  sortKeyLength1 = 50;
  sortKeyLength2 = 50;
  sortKeyLength3 = 50;
  sortOrder = ascending;
  sortType1 = alphaNumeric;
  sortType2 = alphaNumeric;
  sortType3 = alphaNumeric;
  title = untitled;
  validFrom = 20101216230000;
}
```

content contentRef generateThumbnail

Available for: Content Management Server

Task: Creates the thumbnail image belonging to the draft version with the specified ID and stores it in the `thumbnail` version field. When images are imported into the CMS (manually or by restoring a dump), the system automatically generates thumbnail images by calling this function. Therefore, this function only needs to be called if thumbnail generation failed because of incompatible image data, for example.

Syntax:

`content withId contentId generateThumbnail`

Function parameters: none.

Return value if successful: none.

Necessary permissions: The user must have at least one of the `permissionRead` and `permissionRoot` permissions for the file to which the version belongs.

Example:

```
CM>content withId 43219 generateThumbnail
```

content contentRef get

Available for: Content Management Server

Task: Returns the value of the specified version field.

Syntax:

`content withId *contentId* get *contentAttr*`

Function parameters:

- *`contentAttr`*: The name of the version field whose value is being queried.

Return value if successful: the value of the version field (string)

Necessary permissions:

- The user must have the `permissionRead` permission for the file the version belongs to.
- The `permissionGlobalExport` permission is also sufficient for querying the `exportBlob` field.

Example:

```
CM>content withId 25687 get title
Newslist
```

The value of a version field can also be retrieved using an `obj` command:

```
CM>obj withPath /news editedContent get title
Newslist
```

content contentRef getExportBlobForFrame

Available for: Content Management Server

Task: The function returns the version of the file that will be created when the Content Manager exports the frame belonging to the version with the specified ID.

Syntax:

`content withId *contentId* getExportBlobForFrame *frameName*`

Function parameters:

- *`frameName`*: the name of the frame whose export blob is to be queried.

Return value if successful: the export blob.

Necessary permissions: The user must have the `permissionGlobalExport` permission or the `permissionRead` permission for the file the version belongs to.

Example:

```
CM>content withId 25687 getExportBlobForFrame leftFrame
(The frame's export blob)
```

Example for a corresponding `obj` command:

```
CM>obj withId 24235 editedContent getExportBlobForFrame leftFrame
(The frame's export blob)
```

content contentRef load

Available for: Content Management Server

Task: Loads the blob specified in the parameters into the version.

Additional information:

- One of the parameters `blob`, `blob.plain`, `blob.base64`, or `blob.stream` must be specified.

Syntax:

`content withId` *`contentId`* `load {`*`contentAttr value`*`}`

Function parameters:

- *contentAttr* refers to a parameter necessary for loading the blob. The following names are allowed here:
  - `blob`, `blob.plain`, `blob.base64`, or `blob.stream` specifies that *`value`* contains the properly encoded version to import or, respectively, that *`value`* contains a streaming ticket under which the version to import was uploaded. If several of these parameters are specified, it is undefined which one is evaluated.
  - `charset`: the character set of the blob (for files that are no images). If not specified, the character set is taken from the `charset` user preference. The default of this value is `iso8859-1`. The Content Manager converts the blob to UTF-8. The list of the available character sets can be determined using the Tcl command `encoding names`.
  - `contentType` specifies that *`value`* refers to the file name extension.
- *`value`* specifies the value of the respective parameter.

Return value if successful: none

Necessary permissions: The user must have the `permissionWrite` permission for the file to which the version belongs. She must be the editor of the file.

Example: Load a file as the main content of a version:

```
CM>content withId 25687 load blob [loadFile \
/Users/NPS/Upload/news.html] contentType html
```

Example for a corresponding `obj` command:

```
CM>obj withId 242235 editedContent load blob [loadFile \
/Users/NPS/Upload/news.html] contentType html
```

content contentRef mget

Available for: Content Management Server

Task: Returns the values of the specified version fields.

Syntax:

`content withId` *`contentId`* `mget {`*`contentAttr`*`}`

Function parameters:

- *contentAttr* : The name of a queried version fields (see Version fields).

Return value if successful: the list of values of the queried version fields (stringlist)

Necessary permissions:

- The user must have the *permissionRead* permission for the file the version belongs to.
- The *permissionGlobalExport* permission is sufficient to query the value of the *exportBlob* field.

Example:

```
CM>content withId 25687 mget lastChanged validFrom
20101221174909 20101216230000
```

Example for a corresponding `obj` command:

```
CM>obj withId 242235 editedContent mget lastChanged validFrom
20101221174909 20101216230000
```

content contentRef resolveRefs

Available for: Content Management Server

Task: Resolves the URLs contained in the blob and in HTML fields of the respective version, i.e. assigns links to them.

Syntax:

`content withId` *`contentId`* `resolveRefs`

Function parameters: none

Return value if successful: none

Necessary permissions: The user must be editor of the version and have the `permissionWrite` permission for the file to which the version belongs.

Example:

```
CM>content withId 25687 resolveRefs
```

Example for a corresponding `obj` command:

```
CM>obj withId 242235 editedContent resolveRefs
```

content contentRef set

Available for: Content Management Server

Task: Sets the specified version field to the corresponding value.

Syntax:

`content withId` *`contentId`* `set {`*`contentAttr value`*`}`

Function parameters:

- *`parameter`* specifies the version field whose value is to be set.
- *`value`* is the value of the field to be set.

Return value if successful: none

Necessary permissions: The user must be the editor of the version and have the `permissionWrite` permission for the file the version belongs to.

Example:

```
CM>content withId 25687 set title {News list}
```

Example for a corresponding `obj` command:

```
CM>obj withId 242235 editedContent set title {News list}
```

4.5 group (User Groups)

4.5.1 User Group Parameters

Parameters	Type	Explanation	get	set	create	descr
displayTitle	string	The group's title as it is displayed in the HTML user interface. This title is a combination of the name and the full name.	•			
getKeys	stringlist	List of parameters which can be queried with `get`	•			
globalPermissions	stringlist	List of global permissions	•	•	•	•
name	string	Name of the group	•		•	•
owner	string	Owner of the group	•	•	•	•
realName	string	Full name of the group	•	•	•	•

setKeys	stringlist	List of parameters which can be set using `set`	•			
users	stringlist	List of the users who are members of the group	•	•	•	•

Owner of a user group

The name of the user or the group permitted to change the parameters and permissions of a group is stored in the *owner* parameter of a group. The *owner* group parameter works in a way similar to the *owner* user parameter (see Owner of a user).

4.5.2 User Group Commands

group create

Available for: Content Management Server

Task: Creates a new user group with the specified parameter values.

Additional information:

- When a group is created, the `owner` is set to the logged-in user if `owner` is not specified as a parameter when `create` is called.
- The name of a group cannot be changed after creation.

Syntax:

`group create {`*`parameter value`*`}`

Function parameters:

- *`parameter`* specifies the name of the group parameter whose value is to be set during creation of the group. The `name` parameter must be specified.
- *`value`* is the value of the parameter to be set.

Return value if successful: the name of the new group (string)

Necessary permissions: The user must have the `permissionGlobalUserEdit` permission.

Example:

```
CM>group create name editors realName Editors
Editors
```

group list

Available for: Content Management Server

Task: Lists the names of all managed user groups.

Syntax:

`group list`

Function parameters: none

Return value if successful: list of the names of the groups (stringlist)

Necessary permissions: none

Example:

```
CM>group list
admins editors news_editors online_editors
```

group where

Available for: Content Management Server

Task: Lists the names of all managed groups in whose parameter *name* or *realName* the specified character string appears.

Syntax:

`group where {parameter value}`

Function parameters:

- *parameter* specifies the name of the parameter whose value is to be searched for or specifies the number of hits. The following parameters are allowed:
  - `maxResults` specifies that *value* is a number that limits the number of hits. If number is less than or equals zero, the number of hits is unlimited.
  - `groupText` specifies that *value* is the string for which the group parameters `name` and `RealName` are searched.
- *value*: The value of the corresponding parameter.

Return value if successful: list of the group names (stringlist)

Necessary permissions: none

Example:

```
CM>group where groupText editors maxResults 20
editors news_editors online_editors
```

group groupRef addUsers

Available for: Content Management Server

Task: Puts each specified user into the group with the name `name` if the user is not already a member.

Syntax:

`group withName name addUsers {login}`

Function parameters:

- *login*: Login of a user.

Return value if successful: none

Necessary permissions:

- The logged-in user must be a super user or
- The logged-in user must have the `permissionGlobalUserEdit` permission and be direct or indirect `owner` of the `name` group.

Example:

```
CM>group withName editors addUsers mike jane
```

group groupRef delete

Available for: Content Management Server

Task: Deletes the group with the name `name` .

Syntax:

`group withName` *`name`* `delete`

Function parameters: none

Return value if successful: none

Necessary permissions:

- The logged-in user must be a super user, or
- the logged-in user must have the `permissionGlobalUserEdit` permission and be the direct or indirect `owner` of the `name` group.

Example:

```
CM>group withName unwanted delete
```

group groupRef description

Available for: Content Management Server

Task: provides a string representation of the data of the group with the name `name`.

Additional information: The representation is formatted in the property list format of a dictionary.

Syntax:

`group withName` *`name`* `description`

Function parameters: none

Return value if successful: the string representation of the group data (string)

Necessary permissions: none

Example:

```
CM>group withName editors description
{
  globalPermissions = (
  );
  name = editors;
  owner = root;
  realName = Editors;
  users = (
    jane,
    michael
  );
}
```

group groupRef get

Available for: Content Management Server

Task: Returns the value of the specified group parameter of the group with the name `name`.

Syntax:

`group withName` *`name`* `get` *`paramName`*

Function parameters:

- *`paramName`* specifies the name of the parameter whose value is being searched for.

Return value if successful: the value of the specified parameter (string)

Necessary permissions: none

Example:

```
CM>group withName editors get realName
Editors
```

group groupRef grantGlobalPermissions

Available for: Content Management Server

Task: Grants the specified permissions to the group with the name `name` .

Syntax:

`group withName` *`name`* `grantGlobalPermissions {`*`permission`*`}`

Function parameters:

- `permission` designates a global permission.

Return value if successful: none

Necessary permissions:

- The logged-in user must be a super user or
- The logged-in user must have the `permissionGlobalUserEdit` permission and be direct or indirect `owner` of the `name` group.

Example:

```
CM>group withName admins grantGlobalPermissions permissionGlobalRTCEdit
```

group groupRef hasGlobalPermission

Available for: Content Management Server

Task: Checks whether the user group with the name `name` has the specified permission.

Additional information: A group has the *`permission`* permission if and only if it was granted this permission. It is not sufficient if a member of the group has the permission.

Syntax:

`group withName` *`name`* `hasGlobalPermission` *`permission`*

Function parameters:

- *`permission`* designates a global permission.

Return value if successful: 1, if the user has the permission, otherwise 0.

Necessary permissions: none

Example:

```
CM>group withName admins hasGlobalPermission permissionGlobalRTCEdit
1
```

group groupRef mget

Available for: Content Management Server

Task: Returns the values of the specified parameters of the group with the name `name`.

Syntax:

`group withName` *`name`* `mget {`*`paramName`*`}`

Function parameters:

- *`paramName`* specifies the name of a group parameter whose value is being searched for.

Return value if successful: list of the parameter values (stringlist)

Necessary permissions: none

Example:

```
CM>group withName editors mget realName users
Editors {mike jane}
```

group groupRef removeUsers

Available for: Content Management Server

Task: Removes the users with the specified logins from the user group with the name `name`.

Additional information: A user cannot be removed from his default group.

Syntax:

`group withName` *`name`* `removeUsers {`*`login`*`}`

Function parameters:

- *`login`*: Login of the user to be removed from the group

Return value if successful: none

Necessary permissions:

- The logged-in user must be a super user or
- The logged-user must have the `permissionGlobalUserEdit` permission and be the direct or indirect `owner` of the `name` group.

Example:

```
CM>group withName admins removeUsers steve
```

group groupRef revokeGlobalPermissions

Available for: Content Management Server

Task: Revokes the specified permissions of the group with the name `name`.

Syntax:

`group withName` *`name`* `revokeGlobalPermissions {`*`permission`*`}`

Function parameters:

- *`permission`* designates a global permission.

Return value if successful: none

Necessary permissions:

- The logged-in user must be a super user or
- The logged-user must have the `permissionGlobalUserEdit` permission and be the direct or indirect `owner` of the `name` group.

Example:

```
CM>group withName editors revokeGlobalPermissions permissionGlobalRTCEdit
```

group groupRef set

Available for: Content Management Server

Task: Sets the specified parameters to the specified values in the group with the name `name`.

Syntax:

`group withName` *`name`* `set {`*`paramName value`*`}`

Function parameters:

- *`paramName`* specifies the name of the parameter whose value is to be set.
- *`value`* is the parameter value to be set.

Return value if successful: none

Necessary permissions:

- The logged-in user must be a super user or
- The logged-user must have the `permissionGlobalUserEdit` permission and be direct or indirect `owner` of the `name` group.

Example:

```
CM>group withName editors set realName Editors
```

4.6 incrExport (Incremental Export)

The incremental export is a method used to automatically synchronize a web server, i.e. to update the content available for the website visitors. You need to run the Template Engine on your live server system if you want to make use of this method, unless you are using the Rails Connector for Infopark CMS Fiona to deliver your content.

The Content Management Server continually transfers updated versions to the Template Engine via the XML interface and instructs it in regular, configurable intervals to export and activate them. Updated data as well as instructions are transferred from the Content Management Server to the Template Engine in the form of so called update records.

The configuration values responsible for the incremental export can be defined by means of the `export` system configuration entry. In particular, the `incrementalUpdate.isActive` value lets you control the action the Content Manager takes when versions or other data like channels are updated. If `isActive` is `true`, all changes are transferred to the Template Engine. If `isActive` has the value `false`, changes to files will not be recorded.

In order to use the `incrExport` commands in the Tcl interface, you need to be a superuser or have the `permissionGlobalExport` permission.

4.6.1 IncrExport Parameters

Parameter	Type	Explanation	get
updateRecordCount	string	Returns the number of update records not yet transferred to the Template Engine.	•
updateRecords	stringlist	returns in `active` mode a list containing update records. Each update record is itself a list made up of two elements, `updateRecordId` and `updateType` . In `suspended` mode `updateRecords` returns the empty list, and if mode is `off` the error message *incExport is turned off* will be returned. The update records returned will not be deleted from the list of update records. This must explicitly be triggered with the `incrExport removeUpdateRecords` command.	•
mode	string	This returns the state the incremental export is in. The mode can be `on` or `off` . It can be set with the `export.incrementalUpdate.isActive` system configuration entry.	•
getKeys	stringlist	The list of `incrExport` parameters that can be queried with the `get` subcommand.	•

The following update record types exist:

Update-Record-Type	Meaning	Transfer
1	File was created	CM -> TE
2	File was modified	CM -> TE
3	File was deleted	CM -> TE
4	Version was modified	CM -> TE
5	Version was deleted	CM -> TE
6	Reset was started	CM -> TE
7	Reset was finished	CM -> TE
8	Start publishing process	CM -> TE
9	Channel was deleted	CM -> TE -> PM
10	File was moved or modified	CM -> TE

4.6.2 IncrExport Commands

incrExport get

Available for: Content Management Server

Task: Returns the value of the specified `incrExport` parameter.

Syntax:

`incrExport get` *`paramName`*

Function parameters:

- *`paramName`* specifies the name of the parameter whose value is to be returned.

Return value if successful: the value of the specified parameter (string).

Necessary permissions: The user must be a super user or have the `permissionGlobalExport` permission.

Example:

```
CM>incrExport get mode
suspended
```

incrExport getUpdateData

Available for: Content Management Server

Task: Returns all file data required for exporting the file referenced in the update record identified with `id`. The data will be formatted as an XML document suitable to be interpreted by the Template Engine.

Syntax:

`incrExport getUpdateData updateRecordId` *`id`*

Function parameters:

- *`id`* specifies the ID of the update record referring to the file whose data are to be returned.

Return value if successful: the corresponding XML document containing the data (string). The document's root element is `updateData`.

Necessary permissions: The user must be a super user or have the `permissionGlobalExport` permission.

Example:

```
CM>incrExport getUpdateData updateRecordId 4711
(the update data)
```

incrExport mget

Available for: Content Management Server

Task: Returns the values of the specified `incrExport` parameters.

Syntax:

`incrExport mget {`*`paramName`*`}`

Function parameters:

- *paramName* specifies the name of a parameter whose value is to be returned.

Return value if successful: the values of the specified parameters (string).

Necessary permissions: The user must be a super user or have the `permissionGlobalExport` permission.

Example:

```
CM>incrExport mget mode updateRecordCount
active 481
```

incrExport removeUpdateRecords

Available for: Content Management Server

Task: Deletes the update records whose IDs have been specified. All additional data like XML files associated with the update records are deleted as well.

Syntax:

`incrExport removeUpdateRecords ({`*id*`}) | all`

Function parameters:

- *id* specifies the ID of an update record. If instead of one or more IDs the string `all` is specified then all update records will be deleted.

Return value if successful: The number of update records deleted (number).

Necessary permissions: The user must be a super user or have the `permissionGlobalExport` permission.

Example:

```
CM>incrExport removeUpdateRecords 3461 3462 5471
3
```

incrExport reset

Available for: Content Management Server

Task: Deletes all update records and rebuilds the database table. The amount of time the Content Manager needs to process this command is in proportion to the number of files managed.

Additional information: **While this command is executed, update records must not be generated. The `incrExport reset` command must only be executed in single mode and only if it is guaranteed that no write access to the data takes place. Otherwise the exported data will be incomplete or destroyed. Therefore, please make sure that the CM is not running as a server and no other CMs are executed in single mode while this command is being executed.**

Syntax:

`incrExport reset`

Function parameters: none.

Return value if successful: none.

Necessary permissions: The user must be a super user or have the `permissionGlobalExport` permission.

Example:

```
CM>incrExport reset
```

4.7 job (Jobs)

The `job` command makes it possible to execute custom Tcl scripts once or repeatedly.

The Content Management Server and the Template Engine differentiate between 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). Jobs in different categories, however, are processed in parallel.

Please note that jobs are only executed if the CMS application concerned (CM or TE) runs in server mode. If the application was started with the *-single* command line argument and is not running as server at the same time, jobs will be queued but only executed after the server has been started again.

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

4.7.1 Job Parameters

Parameter	Type	Explanation	get	set create	descr

category	string	The job category (`user` or `system`)	•		•
comment	string	Job description	•	•	
displayTitle	string	The job's title as it is displayed in the HTML user interface	•		
execLogin	string	Login under which the script is executed	•	•	•
execPerm	string	The global permission a user must have, in order to be able to execute the job, i.e. put it in the queue. If no permission is specified all users may execute the job.	•	•	
getKeys	stringlist	List of the *job* parameters that can be queried with `get`.	•		
id	number	The job's ID.	•		•
isActive	bool	Specifies whether a job is active (1) or not (0). Only active jobs can be executed.	•	•	•
lastExecEnd	datetime	Point in time, at which the most recent execution of a job ended.	•		
lastExecResult	string	Result of the last script execution.	•		
lastExecStart	datetime	Point in time, at which the job was most recently executed.	•		
lastLogEntry	stringlist	The most recent log entry of the job. See the `getLogEntry` subcommand for a description of the format.	•		
lastOutput	string	The job output that was generated most recently.	•		
log	stringlist	List of the IDs of the most recent log entries of this job. The maximum number of entries is defined in the *jobMaxLogLength* system configuration entry.	•	•	•
logEntries	stringlist	The most recent log entries of this job. Each entry is itself a list. See the `getLogEntry` subcommand for a description of the format.	•	•	•
name	string	Name of the job. Names of jobs in the `user` category must not begin with *system* or an underscore.	•	•	•

nextExecStart	datetime	Point in time, at which the job will be executed the next time. Empty if the job is running or there is no such point in time.	•		•
queuePos	number	If the value is greater than 0 it specifies the job's position in the execution queue; if it equals 0 the job is currently being executed. Otherwise the job has not been queued.	•		•
schedule	stringlist	The execution schedule (see below for a description).	•	•	
script	string	The Tcl script to be run when the job is executed (only for jobs not in the `system` category).	•	•	
setKeys	stringlist	The list of parameters that can be set with `set`.	•		
title	string	The title of the job.	•	•	•

The Execution Schedule

Every job has an execution schedule assigned to it. In this schedule the (possibly recurring) job execution times are defined. Each execution time is made up of information about the years (`years`), months (`months`), the days (`days`) or weekdays (`weekdays`), the hours (`hours`), and the minutes (`minutes`). The specifications need not be complete, i. e. components can be left out. The following job will be executed daily in July, 2010, at 6.30 and 22.30:

```
job withName twiceADayInJuly2010 set schedule {
  {minutes 30 hours {6 22} months 7 years 2010}
}
```

The execution schedule is a list of entries, each of which is also a list. The parts of an entry are specified using the identifiers mentioned above and their respective values.

It is essential for the logic of repetitions that a component missing from an entry is interpreted as if all values possible for this component had been specified. If, for example, the days or weekdays are not given for the execution of the job, it will be executed daily. As a consequence, an empty entry for an execution time has the effect that the job will be executed every minute, until the entry is deleted or modified. A single execution time (without repetitions) can be defined by specifying exactly one value for each component except for `weekdays`. `days` and `weekdays` are combined using *or*, and the result is combined with the other categories using *and*.

Hours are specified as numbers from 0 to 23, weekdays as numbers from 1 to 7, 1 standing for Monday, 2 for Tuesday and so on.

All entries in execution schedules refer to the server's time with its specific timezone. If execution times in entries overlap, the entry that comes first in the list has priority over the other ones.

4.7.2 Job Commands

job create

Available for: Content Management Server, Template Engine

Task: Creates a new job with the specified parameters.

Syntax:

`job create {parameter value}`

Function parameters:

- *parameter* specifies the name of a parameter that can be used with the create command. If the parameter `execLogin` is not specified, its value will be assigned the current user's login. The `name` parameter must be specified.
- *value* is the value of the respective parameter.

Return value if successful: The new job's name.

Necessary permissions (CM only): The user must be a super user.

Additional information: It is recommended to define at least one global permission (`permissionGlobalJobExec`, for example) and assign it to new jobs as necessary execution permission. Otherwise, new jobs can be executed by all users.

Example:

```
CM>job create name myJob title "Export the root publication" \
execPerm permissionGlobalJobExec \
script "obj root exportSubtree filePrefix /tmp"
myJob
```

job list

Available for: Content Management Server, Template Engine

Task: Returns the names of all jobs.

Syntax:

`job list`

Function parameters: none.

Return value if successful: The list of the names of all jobs (stringlist).

Necessary permissions (CM only): none.

Example:

```
CM>job list
testJob adminExportFailureNotification systemSwitch systemTransferUpdates
```

job listQueue

Available for: Content Management Server, Template Engine

Task: Returns the list of all jobs in the specified category that have been queued for execution, sorted in ascending order by the `queuePos` parameter.

Syntax:

`job listQueue [category]`

Function parameters:

- *category* specifies the name of a job category (`user` or `system`). Default: `user`.

Return value if successful: The list of the names of the jobs in the queue (stringlist).

Necessary permissions (CM only): none.

Example:

```
CM>job listQueue system
systemSwitch systemTransferUpdates
```

job where

Available for: Content Management Server, Template Engine

Task: Searches for jobs that meet all the conditions that have been specified as parameter-value-pairs.

Syntax:

`job where {parameter value}`

Function parameters:

- *parameter* is one of the following identifiers:
  - `maxResults` specifies that *value* is a number that limits the number of hits. If number is less than or equals zero, the number of hits is unlimited.
  - `category`: The jobs whose `category` parameter equals the value *value* will be returned.
  - `comment`: Returns the jobs whose `comment` parameter contains the value *value*.
  - `execLogin`: Returns the jobs whose script will be executed under the login specified as *value*.
  - `isActive`: Returns the jobs whose `isActive` parameter equals *value*.
  - `isQueued`: If *value* is not 0 then all jobs will be returned whose `queuePos` value is greater than or equal to zero. Otherwise, all jobs with a `queuePos` value less than zero are returned.
  - `name`: The jobs whose `name` parameter equals the value *value* will be returned.
  - `title`: The jobs whose `title` parameter contains the value *value* will be returned.
- *value* contains the value for the respective parameter.

Return value if successful: the list of the names of the matching job entries (stringlist).

Necessary permissions (CM only): no restrictions

Example:

```
CM>job where isActive 1 execLogin root
rootJob1 rootJob2
```

job jobRef cancel

Available for: Content Management Server, Template Engine

Task: Removes the specified job from the execution queue.

Additional information: The job's `queuePos` parameter will be set to -1. The job currently beeing executed (`queuePos` = 0) can not be cancelled.

Syntax:

`job (withName jobName) | (withId jobId) cancel`

Function parameters: none.

Return value if successful: none.

Necessary permissions (CM only): The user must be a super user or the user with the `execLogin` login or have the permission specified as the value of the `execPerm` parameter.

Example:

```
CM>job withId 3099 cancel
```

job jobRef delete

Available for: Content Management Server, Template Engine

Task: Deletes the specified job from the job list.

Syntax:

`job (withName jobName) | (withId jobId) delete`

Function parameters: none.

Return value if successful: none.

Necessary permissions (CM only): The user must be a super user.

Example:

```
CM>job withId 2023 delete
```

job jobRef description

Available for: Content Management Server, Template Engine

Task: Returns a string representation of the job with the ID *jobID* .

Additional information: The format of the string representation is the property list format of a dictionary.

Syntax:

`job (withName jobName) | (withId jobId) description`

Function parameters: none.

Return value if successful: the job's string representation (string).

Necessary permissions (CM only): no restrictions.

Example:

```
CM>job withId 3245 description
{
  id = 3245;
  name = someJob;
  category = user;
  queuePos = -1;
  title = "Export Rootpub";
  execLogin = root;
  isActive = 1;
}
```

job jobRef exec

Available for: Content Management Server, Template Engine

Task: Places a job in the job execution queue.

Additional information: If the job has already been queued for execution it will not be queued a second time.

Syntax:

`job (withName jobName) | (withId jobId) exec`

Function parameters: none.

Return value if successful: The new value for `queuePos`. If the job has already been queued the current `queuePos` value will be returned.

Necessary permissions (CM only): The user must be a super user or the user with the `execLogin` login or have the permission specified as the value of the `execPerm` parameter.

Example:

```
CM>job withId 2023 exec
5
```

job jobRef get

Available for: Content Management Server, Template Engine

Task: Returns the *paramName* job parameter.

Syntax:

`job (withName jobName) | (withId jobId) get paramName`

Function parameters:

- *paramName* specifies the name of the job parameter whose value is being searched for.

Return value if successful: The value of the specified job parameter.

Necessary permissions (CM only): no restrictions.

Example:

```
CM>job withId 2245 get title
Export the root publication
```

job jobRef getLogEntry

Available for: Content Management Server, Template Engine

Task: The command returns a log entry of the specified job.

Syntax:

`job (withName` *jobName*`) | (withId` *jobId*`) getLogEntry` *entryId*

Function parameters:

- *entryId* specifies the ID of a log entry belonging to the job concerned. The list of log IDs is returned by the `log` job parameter.

Return value if successful: a list of name-value pairs (stringlist).

Necessary permissions (CM only): no restrictions.

Example:

```
CM>job withName myJob getLogEntry 3461
entryId 3461 execStart 20021219111651 execEnd 20021219111735 execResult {The Job Result}
```

job jobRef getOutput entryId

Available for: Content Management Server, Template Engine

Task: The command returns the logged output of the specified job.

Syntax:

`job (withName` *jobName*`) | (withId` *jobId*`) getOutput` *entryId*

Function parameters:

- *entryId* specifies the ID of a log entry belonging to the job concerned. The list of log IDs is returned by the `log` job parameter.

Return value if successful: the output generated by the job (string).

Necessary permissions (CM only): no restrictions.

Example:

```
CM>job withName myJob getOutput 3461
Output generated by the job.
```

job jobRef mget

Available for: Content Management Server, Template Engine

Task: Returns one or more job parameters given as *paramName* parameters.

Syntax:

`job (withName jobName) | (withId jobId) mget {paramName}`

Function parameters:

- *paramName* specifies the name of a job parameter whose value is being searched for.

Return value if successful: The values of the specified job parameters.

Necessary permissions (CM only): no restrictions.

Example:

```
CM>job withId 2245 mget execLogin execPerm
root permissionGlobalJobExec
```

job jobRef set

Available for: Content Management Server, Template Engine

Task: Sets the specified job parameters to the specified values.

Syntax:

`job (withName jobName) | (withId jobId) set {paramName value}`

Function parameters:

- *paramName* specifies the name of a job parameter whose value is to be set.
- *value* specifies the value of the job parameter specified as *paramName*.

Return value if successful: none.

Necessary permissions (CM only): The user must be a super user.

Example:

```
CM>job withId 2245 set execLogin smith isActive 1
```

4.8 licenseManager (License Information)

4.8.1 LicenseManager parameters

Parameters	Type	Explanation	get
clients	stringlist	(Up to version 6.6.1.) The instances to which users are logged in. The result is a list containg in pairs the instance names and the list of users (see the example below).	•
extensions	stringlist	(Up to version 6.6.1.) The list of licensed extensions such as the *Content Service*.	•
getKeys	stringlist	List of the licenseManager parameters which can be queried with `get`	•
idLimit	number	For licenses with a limited number of IDs, the largest possible ID is returned.	•
license	string	Returns the license data as an XML element.	•
licenseExpirationDate	string	Returns the date the license expires.	•
licenseType	string	Returns *demo* or *full* , depending on the type of license installed.	•
loginCount	string	Returns the number of users currently logged into the system.	•
logins	stringlist	Returns a list of dictionaries. Each dictionary is related to a logged-in user and contains two values, *login* and *timeStamp*. *login* specifies the interface used, the session, and the login (for example `T(1069544507) root` where T=tcl, X=XML, J=Jobs). The *timeStamp* value specifies the date and time the user has last accessed the system.	•
maxConcurrentClients	string	(Up to version 6.6.1.) The maximum number of instances to which users can be logged-in at the same time. This number does not refer to a single user but to all users.	•
maxConcurrentUsers	string	The maximum number of users allowed to be logged-in at the same time.	•
remainingIds	number	For licenses with a limited number of IDs, the number of remaining IDs is returned.	•

4.8.2 LicenseManager Commands

licenseManager checkLicense

Available for: Content Management Server, Template Engine (up to version 6.6.1)

Task: The command checks whether a license is present for the specified application in the specified version.

Syntax:

`licenseManager checkLicense` *`appName version`*

Function parameters:

- `appName` specifies the name of an application whose presence in the `license.xml` license file is to be checked. The name must be specified exactly the way it has been specified in the license file.
- `version` is the version character string which is compared with the version string in the license file.

Return value if successful: If the application `appName` is listed in the license file and if the version number in the license file begins with `version`, the return value is the empty value. Otherwise a corresponding error message is returned.

Necessary permissions: no restrictions

Example:

```
CM>licenseManager checkLicense CM 5
```

licenseManager get

Available for: Content Management Server, Template Engine

Task: Returns the value of the *`paramName`* `licenseManager` parameter.

Syntax:

`licenseManager get` *`paramName`*

Function parameters:

- *`paramName`* specifies the name of the `licenseManager` parameter whose value is being searched for.

Return value if successful: the value of the specified `licenseManager` parameter.

Necessary permissions: no restrictions

Example:

```
CM>licenseManager get loginCount
15
CM>licenseManager get clients
default {root designer} myinstance {editor designer chiefeditor}
```

4.9 link (Links)

4.9.1 Link Parameters

Parameters	Type	Explanation	Applic.	get	set	descr

attributeName	string	Name of the version field containing the link (only with free links).	CM, TE	•	•	•
canHaveAnchor	bool	Specifies whether the link can have an anchor.	CM, TE	•		
canHaveTarget	bool	Specifies whether the link can have a frame target in which the target file is displayed	CM, TE	•		
destination	string	File ID of the target file (only for internal links)	CM, TE	•		•
destinationUrl	string	The URL that represents the link	CM, TE	•	•	•

displayTitle	string	The title of the link used in the HTML user interface. If the link is internal and has no title, then the title of the destination file is returned.	CM, TE	•		•
expectedPath	string	Path to the referenced file in the folder hierarchy (only for internal links)	CM, TE	•		•
getKeys	stringlist	List of the link parameters which can be queried with `get`	CM, TE	•		
id	string	The ID of the link	CM, TE	•		•
isComplete	bool	Specifies whether the link is resolved	CM, TE	•		•
isContextLink	bool	Specifies whether the link is a context link, i. e. can be found in an NPSOBJ-context instruction.	CM, TE	•		•
isDynamicLink	bool	Specifies whether the link is a dynamic link, i. e. can be found in an NPSOBJ-insertvalue-dynamiclink instruction.	CM, TE	•		•
isExternalLink	bool	Specifies whether the link is an external link	CM, TE	•		•
isFreeLink	bool	Specifies whether the link is a free link (i.e. assigned to a linklist field).	CM, TE	•		•
isIncludeLink	bool	Specifies whether the main content of the link destination is to be inserted (`<NPSOBJ includetext="...">`)	CM, TE	•		•
isInlineReferenceLink	bool	Specifies whether it is an inline reference link	CM, TE	•		•
isLinkFromCommittedContent	bool	Specifies whether the link is contained in the committed version	CM	•		
isLinkFromEditedContent	bool	Specifies whether the link is contained in the draft version	CM	•		
isLinkFromReleasedContent	bool	Specifies whether the link is contained in the released version	CM	•		
isWritable	bool	Specifies whether the logged-in user can edit the link. This is the case if the link is in the draft version and the user has file write permission	CM	•		•

position (from version 6.6.1)	integer	If the link is contained in a link list, `position` functions as a sort value that determines the position of the link in the linklist. The value can be assigned freely. Therefore, when sorting the linklist, the positions of all its links need to be analyzed and changed as desired.	CM, TE	•	•	
setKeys	stringlist	List of the link parameters that can be set with `set`	CM	•		
source	string	The ID of the file that contains the link	CM, TE	•		•
sourceContent	string	The ID of the file version that contains the link	CM, TE	•		•
sourcetagAttribute	string	The name of the HTML tag attribute the link is defined in	CM, TE	•		•
sourcetagName	string	The name of the HTML tag the link is defined in	CM, TE	•		•
target	string	The destination frame or window.	CM, TE	•	•	•
title	string	The title of the link	CM, TE	•	•	•

4.9.2 Link Commands

link list

Available for: Content Management Server, Template Engine

Task: Outputs a list of the IDs of all links.

Syntax:

```
link list
```

Function parameters: none

Return value if successful: list of the IDs of the links (stringlist)

Necessary permissions (CM only): The user must be a super user.

Example:

```
CM>link list
23585 32914 42739 81203
```

link linkRef delete

Available for: Content Management Server, Template Engine

Task: Deletes the *linkId* link if it is a free link contained in the draft version.

Syntax:

`link withId linkId delete`

Function parameters: none

Return value if successful: none

Necessary permissions (nur CM): The user must be the editor of the file in whose draft version the link is defined in. A user must have the `permissionWrite` permission for a file of which he wants to become the editor.

Example:

```
CM>link withId 4273.9 delete
```

link linkRef description

Available for: Content Management Server, Template Engine

Task: Returns a string representation of the data of the link *linkId*.

Additional information: The format of the string representation is the property list format of a dictionary.

Syntax:

`link withId linkId description`

Function parameters: none

Return value if successful: representation of the link (string)

Necessary permissions (CM only): The user must have the `permissionRead` permission for the file in which the link is defined.

Example:

```
CM>link withId 4273.9.1 description
{
  destination = 6518;
  destinationUrl = /index.html;
  displayTitle = untitled;
  id = 4273.9.1;
  isComplete = 1;
  isContextLink = 0;
  isExternalLink = 0;
  isIncludeLink = 0;
  isInlineReferenceLink = 0;
  isFreeLink = 1;
  isWritable = 1;
  source = 2012;
  sourceContent = 2012.7;
```

```
  target = "_new";
}
```

link withId linkId get paramName

Task: Returns the value of the *paramName* link parameter for *linkId*.

Syntax:

`link linkRef get`

Function parameters:

- *paramName* specifies the name of the link parameter whose value is being searched for.

Return value if successful: the value of the specified link parameter (string)

Necessary permissions: The user must have the `permissionRead` permission for the file in which the link is defined.

Example:

```
CM>link withId 4273.4.9 get destinationUrl
/index.html
```

link linkRef mget

Available for: Content Management Server, Template Engine

Task: Returns the values of the specified link parameters for the link *linkId*.

Syntax:

`link withId linkId mget {paramName}`

Function parameters:

- *paramName* specifies the names of the link parameters whose values are being searched for.

Return value if successful: list of the parameter values (stringlist)

Necessary permissions (CM only): The user must have the `permissionRead` permission for the file the link is defined in.

Example:

```
CM>link withId 4273.4.9 mget destinationUrl source
/index.html 2012
```

link linkRef set

Available for: Content Management Server, Template Engine

Task: Sets the specified link parameters of the link *linkid* to the specified values. The link must be contained in a draft version.

Syntax:

`link withId linkid set {paramName value}`

Function parameters:

- *paramName* specifies the name of the link parameter whose value is to be set.
- *value* is the value of the link parameter to be set.

Return value if successful: none

Necessary permissions (CM only): The user must be the editor of the file in whose draft version the link is defined in. A user must have the `permissionWrite` permission for a file of which he wants to become the editor.

Example:

```
CM>link withId 2084.4.13 set title {Main Page}
```

4.10 linkChecker (link checking)

The `linkChecker` command checks external links in the CMS for availability.

4.10.1 Link Checker Parameters

The link checker command and its parameters are available from version 6.5.0.

Parameter	Type	Description	get
status	string	The current state of the Link Checker: `running` or `idle`	•
count	number	The number of external link targets to check.	•
checkedCount	number	The number of external link targets already checked since the last start.	•
startedAt	string	The point in time the Link Checker was started last.	•
unreachableUrls	stringlist	The list of the link targets that cannot be reached. The list may contain URLs as well as paths of deactivated files.	•

4.10.2 Link Checker Commands

The `linkChecker` command and its parameters are available from version 6.5.0

linkChecker get

Available for: Content Management Server (from version 6.5.0)

Task: Returns the value of a linkChecker parameter.

Syntax:

`linkChecker get parameter`

Function parameters:

- *parameter*: the name of the parameter whose value is queried.

Return value if successful: the value of the parameter.

Required permissions: none

Example:

```
CM>linkChecker get status
idle
```

linkChecker mget

Available for: Content Management Server (from version 6.5.0)

Task: Returns the values of one or more linkChecker parameters.

Syntax:

`linkChecker mget {parameter}`

Function parameters:

- *parameter*: the name of a parameter whose value is to be queried.

Return value if successful: the values of the specified parameters (stringlist).

Required permissions: none

Example:

```
CM>linkChecker mget status startedAt
running 20110601181500
```

linkChecker start

Available for: Content Management Server (from version 6.5.0)

Task: The command starts checking all external links for availability. The command is executed in the background and therefore returns immediately.

Syntax:

`linkChecker start`

Function parameters: none.

Return value if successful: none.

Required permissions: none.

Example:

```
CM>linkChecker start
```

linkChecker stop

Available for: Content Management Server (from version 6.5.0)

Task: The command stops checking the external links for availability.

Syntax:

```
linkChecker stop
```

Function parameters: none.

Return value if successful: none.

Required permissions: none.

Example:

```
CM>linkChecker stop
```

4.11 logEntry (Log Entries)

4.11.1 LogEntry Parameters

Parameters	Type	Explanation	where	deleteWhere
fromDate	string	The start date of the period in which the log entry is being searched for	•	•
logText	string	The text contained in the comment of a log entry	•	•
logTypes	stringlist	The log types being searched for	•	•
objectId	string	The ID of the file for which the log entry is being searched for	•	•
receiver	string	The name of the group having received a file by means of a workflow action	•	•
untilDate	string	The end date of the period in which the log entry is being searched for	•	•
userLogin	string	The user who caused the log entry to be created	•	•

4.11.2 Log Types

Log type	Triggering action
action_comment	A comment was added to a file.
action_create_subobj	A file was created. The file ID of the folder the file was created in is recorded in the entry as objectId.
action_deactivate	Deactivation of a file.
action_delete_subobj	Deleting a file.
action_admin_obj	Moving a file or modifying file fields.
action_edit_obj	Workflow action *Edit*
action_give_obj	Workflow action *Give*
action_take_obj	Workflow action *Take*
action_forward_obj	Workflow action *Forward*
action_commit_obj	Workflow action *Commit*
action_reject_obj	Workflow action *Reject*
action_revert_obj	Workflow action *Revert*
action_sign_obj	Workflow action *Sign*
action_release_obj	Workflow action *Release*
action_unrelease_obj	Workflow action *Unrelease*

4.11.3 LogEntry Commands

logEntry deleteWhere

Available for: Content Management Server

Task: Deletes the log entries that match the search criteria.

Syntax:

`logEntry deleteWhere {`*`parameter value`*`}`

Function parameters:

- *`parameter`* is the name of a parameter whose values are to be checked for occurrences of *`value`*.
- *`value`* (string) contains the search string for the respective parameter.

Return value if successful: number of deleted entries (number)

Necessary permissions: The user must be a super user.

Example:

```
CM>logEntry deleteWhere objectId 2003
```

logEntry where

Available for: Content Management Server

Task: Searches for log entries for which the value of the specified parameters has the respective specified value.

Syntax:

`logEntry where {parameter value}`

Function parameters:

- *parameter* is the name of a parameter whose values are to be checked for occurrences of *value*.
- *value* contains the search string for the respective parameter.

Return value if successful: the list of names from the matching log entries (stringlist)

Necessary permissions: no restrictions

Example:

```
CM>logEntry where logTypes action_take_obj
{objectId 2012 logType action_take_obj receiver root logTime
20000103143526 logText { } userLogin root}
```

4.12 obj (Files)

4.12.1 File Parameters

For being able to read the values of file fields, the read permission is required for the file concerned. This is not the case for the following fields whose values can be queried without having been granted any permissions: `id`, `name`, `visibleName`, `path`, `visiblePath`, `objType`.

If the *set* column is marked, the value can be set, however only in the Content Management Server. In the Template Engine, only read access is permitted for the values. Version-specific values always refer to the draft version in the Content Manager, if present, otherwise to the released version. In the Template Engine, version-specific values refer to the released version because no draft versions exist in the Template Engine.

Field	Type	Explanation	Applic.	get	set	descr

archivedContentIds	stringlist	List of the IDs of all archived versions.	CM	•		
children	stringlist	List of the IDs of the files in the folder	CM, TE	•		
committedContentId	number	ID of the file's committed version.	CM	•		
contentIds	stringlist	List of the IDs of all versions of the file	CM	•		
dependencies	string	Returns for debugging purposes the file's dependencies in the property list format. For each entry the following is listed: the dependency type (`depType`), the dependent file (`expObjId`), and the referenced file (`refObjId`). The dependency types have the following meaning: 1 - `object`, 2 - `content`, 3 - `reference`, 4 - `children`, 5 - `usesAll`.	TE	•		
dependingObjectIds	stringlist	The list of files depending on this file.	TE	•		
dependOnObjectIds	stringlist	The list of files on which the file depends.	TE	•		
editedContentId	number	ID of the file's draft version.	CM	•		
exportMimeType	string	The mime type for the file name extension of the released version (or, if it does not exist, of the draft version)	CM, TE	•		
getKeys	stringlist	List of the file fields that can be queried with `get`	CM, TE	•		
hasChildren	bool	Specifies whether the file contains other files.	CM, TE	•		•
hasMirrors	bool	(From version 6.5) Specifies whether mirror files of this file exist.	CM, TE	•		
hasSuperLinks	bool	Specifies whether there is a reference to the file	CM, TE	•		•
id	string	The file ID	CM, TE	•		•
isCommitted	bool	Specifies whether the file has a committed version	CM	•		•
isEdited	bool	Specifies whether the file has a draft version	CM	•		•
isExplicitMirror	bool	(From version 6.5) Specifies whether the file is an explicit mirror file (meaning that it was not created by the system but by a user or a script).	CM, TE	•		
isExportable	bool	Specifies whether the file has a released version which is valid with regard to time (checks `validFrom` and `validUntil`).	CM, TE	•		
isMirror	bool	(From version 6.5) Specifies whether the file is a mirror file.	CM, TE	•		•

isReleased	bool	Specifies whether the file has a released version	CM	•		•
isRoot	bool	Specifies whether the file is the base folder	CM, TE	•		
mirrors	stringlist	(From version 6.5) Returns the list of the IDs of all mirror files of the file concerned.	CM, TE	•		
name	string	Name of the file	CM, TE	•	•	•
next	string	The ID of the file that is the successor of the specified file in the folder containing it.	CM, TE	•		
objClass	string	The name of the file's format	CM, TE	•	•	•
objCount	number	The number of files in the file hierarchy excluding implicit mirror files	CM, TE	•	•	•
objType	string	The file type	CM, TE	•		•
objectsToRoot	stringlist	List of the IDs of the files that are on the path from the file to the base folder (including the file and the basefolder themselves)	CM, TE	•		
original	string	(From version 6.5) If the file concerned is a mirror file, the ID of the original file	CM, TE	•		
parent	string	The ID of the folder containing the specified file	CM, TE	•	•	•

path	string	The path of the file (formed from the names of the files)	CM, TE	•		•
permalink	string	(From version 6.6) A permanent URI for this file (permalink) that persists even if the file is renamed or moved.	CM	•	•	
prefixPath	string	The path of the file, with a "/" at the end for folders	CM, TE	•		
previous	string	The ID of the file that is the predecessor of the specified file in the folder in which it is contained	CM, TE	•		
releasedContentId	number	ID of the file's released version.	CM	•		
releasedVersions	stringlist	The list of the IDs of all already released versions of the file (current released version and all archived versions)	CM	•		
reminderComment	string	(From version 6.5) The comment associated with the reminder of the file	CM	•		
reminderFrom	string	(From version 6.5) The date the reminder becomes active (in the 14-digit format)	CM	•		
reminderGroups	stringlist	(From version 6.5) The list of user groups whose members will be reminded of the file	CM	•		
reminderUsers	stringlist	(From version 6.5) The list of users who will be reminded of the file, not including the members of the `reminderGroups`	CM	•		
setKeys	stringlist	The list of all file fields which can be set with `set`	CM	•		
sortValue	string	The value according to which the files are sorted within the folder that contains them. Determined by the sort keys of the folder.	CM, TE	•		
superLinks	stringlist	The list of all IDs of the links that refer to the file	CM, TE	•		
superObjects	stringlist	The list of all IDs of the files that contain links to the file	CM, TE	•		
suppressExport	bool	Specifies whether the file is exported	CM, TE	•	•	•

toclist (publications only)	stringlist	The list of the IDs of a folder's files that appear in a toclist (`<NPSOBJ list="toclist"></NPSOBJ>`) (all folders and documents that are valid with respect to time and have a released version for which `suppressExport` is not set)	CM, TE	•		
validControlActionKeys	stringlist	The names of the actions that can be executed for the file and the respective user – subset of {`edit, commit, release, unrelease, revert, reject`}	CM	•		
validCreateObjClasses	stringlist	The list of the names of the file formats the user may use when creating files in the current folder. (This corresponds to the list of `validSubObjClasses` that are specified in the format of the file, for which the user has the appropriate file creation permissions.)	CM	•		
validObjClasses	stringlist	The list of the names of the file formats the user may assign to the file (all `validSubObjClasses` of the parent folder that are valid for the file's type).	CM	•		
validPermissions	stringlist	The list of all available permissions for the file	CM	•		
version	number	The version of the file	CM, TE	•		•
visibleExportTemplates	stringlist	The list of the IDs of the layouts that are visible from the file (i. e. are located in the `objectsToRoot` path) and which can be applied during export or preview. The value of this parameter also depends on the `preferEditedTemplates` user preference.	CM	•		
visibleName	string	The name of the file plus the file name extension (except for folders)	CM, TE	•		
visiblePath	string	The path of the file with the name extension added ("`/index.`*`contentType`*" is added for folders).	CM, TE	•		
workflowName	string	The name of the workflow assigned to the file	CM, TE	•	•	•

parent.*objAttr*		The file field value of the folder containing the file	CM, TE	•		
previous.*objAttr*		The file field value of the predecessor in the folder	CM, TE	•		
next.*objAttr*		The file field value of the successor in the folder	CM, TE	•		
up.*objAttr*		The file field value of the file itself is output if it is not empty, otherwise the folder containing the file is queried for the entire key	CM, TE	•		
permissions.*permissionName*	stringlist	The list of groups that have the specified access permission for the file	CM, TE	•		

4.12.2 File Commands

obj contentTypesForObjType

Available for: Content Management Server

Task: Returns the list of the file name extensions that are in principle supported by the specified file type. However, the valid file name extensions of a particular CMS file may be different ones since they depend on the file format of the file in question. In a file format, the valid file name extensions can be set using `validContentTypes`.

Syntax:

`obj contentTypesForObjType` *`objType`*

Function parameters:

- *`objType`* specifies the file type whose supported file name extensions are to be output.

Return value if successful: the list of the file name extensions (stringlist)

Necessary permissions (CM only): no restrictions

Example:

```
CM>obj contentTypesForObjType image
jpg jpeg gif
```

obj get

Available for: Content Management Server, Template Engine

Task: Returns the value of the specified parameter, which refers to all CMS files.

Syntax:

`obj get` *`parameter`*

Function parameters:

- *parameter* specifies the name of the parameter whose value is being searched for. The following names are valid:
  - `dependenciesCount` (TE only): The number of entries in the dependency table.
  - `getKeys`: The list of parameters available for this command.
  - `objCount`: The number of files currently managed.
  - `objTypes`: The list of the file types
  - `rootPubId`: The ID of the base folder

Return value if successful: the value of the corresponding parameter.

Necessary permissions (CM only): no restrictions

Example:

```
CM>obj get objTypes
generic image publication document template
```

obj list

Available for: Content Management Server, Template Engine

Task: This command returns the list of the IDs of all CMS files.

Additional information: The command also returns the IDs of explicitly created mirror files. However, it does not return the IDs of implicit mirror files (created automatically by the system).

Syntax:

`obj list`

Function parameters: none

Return value if successful: the list of the files (stringlist)

Necessary permissions (CM only): no restrictions

Example:

```
CM>obj list
2001 2003 2012 2033
```

obj search

Available for: Content Management Server

Task: The command searches by means of the Search Engine Server for files that match the specified search query.

Syntax:

`obj search {`*parameter value*`}`

Function parameters:

- *parameter* specifies the name of a parameter whose value is passed as *value*. *value* can be one of the following:
  - query: specifies that *value* is a search query in the syntax of the search engine installed. The search query is checked for syntactical correctness neither by the Content Manager nor by the Search Engine Server. This parameter is obligatory.
  - minRelevance (optional): *value* specifies the minimal relevance that files matching the search query must have in order to be included in the search result. The value of *value* must be an integer number in the range from 0 to 100 (including both values), with 0 being the lowest and 100 the greatest possible relevance.
  - maxDocs (optional): Specifies that *value* is the maximum number (positive integer number) of hits to be returned.
  - offsetStart (optional): Specifies that *value* is the index of the first version ID to be returned by the Search Engine Server. *value* must be an integer number. The index of the first version in the search result is 1.
  - offsetLength (optional): Specifies that *value* contains the number of version IDs the Search Engine Server is to return. *value* must contain a positive integer number.
  - parser (optional): Specifies that *value* identifies the parser to be used by the search engine to process search queries. Valid values are explicit , freetext, and simple (default). Information about the parsers can be taken from the *Verity Search Cartridge / Search Engine Server* manual.
  - collections (optional): Specifies that *value* is a list of collections to which the search query is to be applied. If the parameter is not specified, all collections are searched.
  - returnOption (optional): Specifies that *value* identifies the type of the return value. *value* can be one of the following:

    objects (default): A list of file IDs is returned. Each ID is the aggregation of the version IDs returned by the search engine.

    includeContents: A list of lists is returned. Each sublist contains as its first element the file ID and as further elements the IDs of this file's versions that match the search query.

    statistics: A list of four elements is returned, containing the following elements: hits *number_of_hits* searched *number_of_versions_searched*.
- *value* is the value of the corresponding parameter.

Return value if successful: A list whose versions depends on the value of returnOption (see above).

Necessary permissions (CM only): no restrictions.

Additional information:

- The Search Engine Server always returns version IDs which are by default put down to file IDs by this command.
- The values of maxDocs, offsetStart, and offsetLength refer to the versions returned by the Search Engine Server in the search result and not to the results returned by this command.
- This command returns only files to which the user has read access.

Example:

```
CM>obj search query {stock <#AND> options} \
returnOption includeContents
```

```
{2073 3044.3 2142.9} {4982 6683.1}
```

obj touchAll

Available for: Template Engine

Task: Deletes the cache and the internal dependency table. This causes all your content to be regenerated the next time the content is exported. Therefore, this command should be used cautiously with large amounts of content.

Syntax:

```
obj touchAll
```

Additional information: This command is not executable during the export process.

Function parameters: none.

Return value if successful: none.

Example:

```
TE>obj touchAll
```

obj where

Available for: Content Management Server, Template Engine

Task: Searches for all IDs of files that meet the specified search criteria. Only files for which the user has read permission are output.

Additional information: The command does not return implicit mirror files (created automatically by the system).

Syntax:

```
obj where {parameter value}
```

Function parameters:

- *parameter* specifies the search criteria or restricts the search result list in another way. The following parameters are available.
  - `ids` is a list of file IDs. If this parameter is specified, only the files with the IDs in the list are searched, otherwise all files.
  - `maxResults` specifies the maximum number of hits to be returned. The value 0 (default) deactivates this limitation. See also the `maxSearchResultSize` system configuration entry. The value of this entry has priority over the value of `maxResults`.
  - `condition`: *value* is a condition a file must meet to be included in the search result list. *value* is a list composed of three elements, a property, a comparison operator, and a value with which the property is to be compared. The following table contains all possible combinations of these three items:

Property	Operator	Comparison value

`name`
`title`
`nameOrTitle`
`objClass`
	`is`
	`contains`
	`startsWith`
		String

`objType`	`is`
	`isOneOf`
		`document`
		`publication`
		`generic`
		`image`
		`template`
		List of file types (see `is`)

`state`	`is`
	`isNot`
	`isOneOf`
		`edited`
		`committed`
		`released`
		List of workflow states (see `is`)

`object`	`is`
	`has`
	`hasNo`
		`inactive (from version 6.5.0)`
		`mirror (from version 6.5.0)`
		`superLinks`
		`superLinks`

Several conditions can be specified, which are then combined with *and*. Example:

```
obj where condition {name contains foo} \
condition {objType isOneOf {document publication}}
```

If an empty list is specified for `objType isOneOf` or `state isOneOf`, the results list is empty.

`state isNot` is the only negation available. It is required because workflow states are not exclusive (a file can have an edited or a committed version plus, optionally, a released version.

`object is inactive` returns all objects without any or without a temporally valid version.

The `obj where` conditions can also be used in the Template Engine. However, since only released files exist in the Template Engine, `state` returns the following:

`state is` / `isOneOf` ...

- `released`: the condition is ignored
- `committed` / `edited`: the results list is always empty

`state isNot` ...

- `released`: the results list is always empty
- `committed` / `edited`: the condition is ignored

The `object` property is not available in the Template Engine.

- *`value`* contains the value for the corresponding parameter.

Return value if successful: the list of IDs of the matching files (stringlist)

Necessary permissions (CM only): no restrictions

Examples:

```
CM>obj where condition {state isOneOf {edited committed}}
CM>obj where ids {2001 3002} condition {name contains foo}
CM>obj where condition {name is bar} condition {title contains foo}
CM>obj where condition {name is foo} maxResults 20
```

obj objRef addComment

Available for: Content Management Server

Task: Adds a comment to a file. The comment is written to the log.

Syntax:

`obj (withId` *`objId`*`) | (withPath` *`path`*`) | root addComment` *`comment`*

Function parameters:

- *`comment`* specifies the comment to add to the log.

Return value if successful: none

Necessary permissions (CM only): The user must have the `permissionWrite` or `permissionRoot` permission for the specified file.

Examples:

```
CM>obj withId 2003 addComment "External links corrected."
CM>logEntry where objectId 2003
{objectId 2003 logType action_comment receiver {} logTime
20100622163745 logText {External links corrected.} userLogin root}
...
```

obj objRef commit

Available for: Content Management Server

Task: The workflow action `commit` is executed, thereby converting the draft version to a committed version. The version will be released instead if its workflow contains no signing groups.

Syntax:

`obj (withId` *`objId`*`) | (withPath` *`path`*`) | root commit {`*`parameter value`*`}`

Function parameters:

- *`parameter`* specifies the name of a parameter of this workflow command. The following values are allowed:
  - `comment` specifies that *`value`* is the comment that is entered in the next task.
- *`value`* contains the value of the respective parameter.

Return value if successful: none

Necessary permissions (CM only): The user must have the `permissionWrite` permission for the specified file and be its editor, or the user must be the administrator of the file.

Example:

```
CM>obj withId 2003 commit
```

obj objRef copy

Available for: Content Management Server

Task: The specified file is copied and the copy receives a draft version.

Syntax:

`obj (withId objId) | (withPath path) | root copy {parameter value}`

Additional information: If the source file has a released version, it is copied to the new draft version. If there is no released version, an attempt is made to use the committed version. If this is not available either, the draft version is used. If this does not exist, the file receives an empty draft version.

Function parameters:

- *parameter* specifies the name of a parameter that specifies important data for copying (the target folder for example). The following values are allowed:
  - `parent`: Specifies that *value* specifies the path or the ID of the target folder. If this parameter is missing, the folder the file is located in is assumed to be the target folder.
  - `name`: Specifies that *value* contains the name that the file copy is to receive. If this name is missing, the name of the file to be copied is used. If it already exists in the target folder, a new name is calculated according to the *Namex* pattern, with *x* being a number.
  - `recursive`: If the file concerned is a folder, *value* specifies whether files contained in it are copied as well (recursively) (`YES` , `TRUE` or `1`) or not (`NO`, `FALSE` or `0`). Links inside the tree to be copied point to the copies whereas links to files outside the tree still point to the same link targets. Default: `FALSE`.
- *value* contains the value of the respective parameter.

Return value if successful: the ID of the new file (string)

Necessary permissions (CM only):

- The user must have the `permissionRead` permission for the source file.
- The user must have the `permissionCreateChildren` permission in the target folder.
- The user must have the global file creation permission defined in the format of the target file.

Example:

```
CM>obj withId 20099 copy parent /news/latestnews name incoming
65811
```

obj objRef create

Available for: Content Management Server

Task: Creates a new file with a draft version. The specified file in which the new file is to be created must be a folder.

Syntax:

`obj (withId objId) | (withPath path) | root create {parameter value}`

Function parameters:

- *parameter* is the name of a parameter important for file creation. The value of the parameter is specified as the corresponding value *value*. *parameter* can be:
  - objClass: the name of the format to be assigned to the new file
  - name: The name the new file is to receive. If it is not specified, the name is generated according to the *newx* pattern, with *x* being a counter that counts from 0 to 999.
  - suppressExport: Specifies whether this file is to be exported.
- *value* contains the value for the respective parameter.

Return value if successful: the ID of the newly created file (string)

Necessary permissions (CM only):

- The user must have the permissionCreateChildren permission in the target folder.
- The user must have the global file creation permission defined in the format of the target file.

Example: Create a folder named *newslist* below the base folder:

```
CM>obj root create name newslist objClass publication
4812
```

obj objRef createAndLoad

Available for: Content Management Server

Task: Creates a new file with a draft version and loads the specified blob. The specified file in which the new file is to be created must be a folder.

Syntax:

`obj (withId objId) | (withPath path) | root createAndLoad {parameter value}`

Additional information:

- One of the parameters blob, blob.plain, blob.base64, blob.stream or file must be specified. If more than one of these parameters is specified, it is undefined which one is evaluated.
- The *contentType* parameter (file name extension) must be set.
- Specifying the file format is not necessary if a file name extension was specified which is a validContentType in only one format, so that the format can be determined uniquely.

Function parameters:

- *parameter* is the name of a parameter important for file creation. The value of this parameter is specified in the corresponding *value* parameter. *parameter* can be:
  - blob, blob.plain, blob.base64 or blob.stream specifies that value contains the properly encoded version to import or, respectively, that *value* contains a streaming ticket under which the version to import was uploaded. If several of these parameters are specified, it is undefined which one is evaluated.
  - charset: the character set of the blob (for file types other than image and resource). If not specified, the character set is taken from the *charset* user preference. The Version Manager converts the blob to UTF-8. The list of the available character sets can be determined with the Tcl command *encoding names*.
  - contentType: The blob's file name extension.
  - file: The path of the correctly encoded file to import. The path can be composed of up to two components. It is relative to the user's temporary directory and must not begin with the parent directory. This parameter is intended to be used by developers only.
  - objClass: The name of the format to be assigned to the new file.
  - name: The name the new file is to receive. A name is generated if none is specified.
  - suppressExport: Specifies whether this file is to be exported.
- *value* contains the value of the respective parameter.

Return value if successful: the ID of the new file (string)

Necessary permissions (CM only):

- The user must have the permissionCreateChildren permission in the target folder.
- The user must have the global file creation permission set in the format of the target file.

Example:

```
CM>obj root createAndLoad name news objClass newspub \
blob.base64 [encodeFile /opt/NPS/upload/newslist.html]
news
```

obj objRef createAndLoadXml

Available for: Content Management Server

Task: Creates a new file with a draft version and loads the specified XML blob. The specified file in which the new file is to be created must be a folder.

Syntax:

obj (withId *objId*) | (withPath *path*) | root createAndLoadXml {*parameter value*}

Function parameters:

- *parameter* is the name of a parameter important for file creation. The value of this parameter is specified in the corresponding *value* parameter. parameter can be:
  - xmlBlob: the blob in XML format to be loaded into the version of the new file
  - name: the name the new file is to receive. A name is generated if none is specified.
- *value* contains the value of the respective parameter.

Return value if successful: the ID of the new file (string)

Necessary permissions (CM only):

- The user must have the `permissionCreateChildren` permission in the target folder.
- The user must have the global file creation permission set in the format of the target file.

Example:

```
CM>obj root createAndLoadXml name news xmlBlob [obj withPath /olds releasedContent get
 xmlBlob]
news
```

obj objRef deactivate

Available for: Content Management Server (from version 6.5.0)

Task: This command deactivates the specified file. This is done by making it temporally invalid.

Syntax:

`obj (withId` *`objId`*`) | (withPath` *`path`*`) | root deactivate`

Additional information: The status of the file becomes `invalid` which can be used as a search criterion in the `obj where` command

Function parameters: none

Return value if successful: none

Necessary permissions (CM only): The user must be a super user or have the `permissionWrite` permission for the specified file.

Example:

```
CM>obj withId 2003 deactivate
```

obj objRef debugExport

Available for: Content Management Server (from version 6.5.0)

Task: For the purpose of finding errors in the code of layout files, this command simulates the export of the file with the specified ID. It becomes clear from the output which NPSOBJ tags from which source layouts are evaluated in which order and which errors occurred during the evaluation process.

Additional information:

- The command provides information about the following errors: unclosed tags, unbalanced number of opening and closing tags, disallowed attributes in NPSOBJ tags, disallowed values of NPSOBJ tag attributes, read-access to undefined names.
- The contents of `systemExecute` instructions will neither be searched for errors nor formatted. However, the output of such instructions is formatted.

Syntax:

```
obj withId objectId debugExport
  [templateName templateName]
  [detailed (yes | no)]
  [quoteHtml (yes | no)]
  [errorPrefix errorPrefix]
  [errorSuffix errorSuffix]
  [htmlPrefix htmlPrefix]
  [htmlSuffix htmlSuffix]
  [infoPrefix infoPrefix]
  [infoSuffix infoSuffix]
  [preferEditedTemplates (yes | no)]
  [allowEditedContents (yes | no)]
```

Function parameters:

- *templateName*: Specifies the base layout (initial layout file) with which the export test is to be performed. If *templateName* has not been specified, the user-specific standard layout is used (*mastertemplate* by default).
- *detailed*: Specifies whether the output is to be restricted to error messages (no) or should rather be detailed (yes). The default value is no.
- *quoteHtml*: Specifies whether the characters <, >, and & are to be output as HTML entities (yes) or not (no). The prefix und suffix parameters are not affected by this. The default value is no.
- *errorPrefix*: Specifies a character string which is output prior to each error message. The default value is "***".
- *errorSuffix*: Specifies a character string which is output after each error message (empty by default).
- *htmlPrefix*: Specifies a character string which is output prior to HTML text. The default value is "<pre>".
- *htmlSuffix*: Specifies a character string which is output after HTML text. The default value is "</pre>".
- *infoPrefix*: Specifies a character string which is output prior to each NPSOBJ information message (empty by default).
- *infoSuffix*: Specifies a character string which is output after each NPSOBJ information message (empty by default).
- *preferEditedTemplates*: Specifies whether the draft versions of layout files are to be preferred to the released versions. The default value corresponds to the user configuration setting of the same name, which is also used for the preview.
- *allowEditedContents*: Specifies whether draft versions are to be used if released versions are not available. The default value is no.

Return value if successful: the formatted export report (string).

Necessary permissions: The user must have the permissionGlobalExport permission or the permissionRead permission for the specified file.

Example: (output strongly abridged)

```
CM>obj withId 25687 debugExport detailed yes
<HTML>
    <HEAD>
  ...
NPSOBJ insertvalue=var: title
```

```
The title
END NPSOBJ insertvalue=var
</TITLE>
    </HEAD>
NPSOBJ insertvalue=var: body
<a ...>Linked text</a>

*** ERROR [140008] The instruction attribute was missing from an NPSOBJ tag.
...
```

obj objRef delete

Available for: Content Management Server

Task: Deletes the specified file.

Syntax:

`obj (withId` *`objId`*`) | (withPath` *`path`*`) delete`

Additional information: The file can only be deleted if it is not the base folder, contains no files, and is not referenced by links. Links in archived versions do no prevent the file from being deleted, however. In such links, all information about the target file including `expectedPath` is removed, and `expectedPath` is set to the path the file had at the time of deletion.

Function parameters: none

Return value if successful: none

Necessary permissions (CM only): The user must be a super user or have the `permissionRoot` permission for the specified file.

Example:

```
CM>obj withId 87193 delete
```

obj objRef deleteFromIndex

Available for: Content Management Server

Task: When Infopark Search Server is in use, all data concerning the specified file is deleted from the search engine's index.

Syntax:

`obj (withId` *`objId`*`) | (withPath` *`path`*`) | root deleteFromIndex`

Function parameters: none.

Return value if successful: none.

Necessary permissions (CM only): The user must have the `permissionRoot` permission for the specified file.

Example:

```
CM>obj withId 85392 deleteFromIndex
```

obj objRef description

Available for: Content Management Server, Template Engine

Task: Returns a string representation of the data of the file with the specified ID.

Syntax:

`obj (withId objId) | (withPath path) | root description`

Additional information: The representation is formatted in the property list format of a dictionary (see The property list format).

Function parameters: none

Return value if successful: The string representation of the file (string). With the CM, the data returned includes information about the file's workflow state, which the TE does not have because it only knows about released versions.

Necessary permissions (CM only): The user must have the `permissionRead` permission for the specified file.

Example:

```
CM>obj root description
{
  hasChildren = 1;
  hasSuperLinks = 1;
  id = 2001;
  isCommitted = 0;
  isEdited = 1;
  isReleased = 1;
  name = ROOTPUB;
  objClass = publication;
  objType = publication;
  path = "/";
  suppressExport = 0;
  version = 1;
}
```

obj objRef edit

Available for: Content Management Server

Task: A draft version is created for the specified file. If a draft version already exists, an error is generated.

Syntax:

`obj (withId objId) | (withPath path) | root edit {parameter value}`

Function parameters:

- *parameter* is the name of a parameter whose value is specified in the corresponding *value* parameter. *parameter* can be:

- `comment`: is the comment to be entered in the next task. The parameter is optional.
- `contentId`: the ID of a version belonging to the file (see the `releasedVersions` file parameter). The parameter is optional. If it has not been specified, a copy of the released version is made. If no released version exists, an empty draft version will be created.

- *`value`* contains the value of the respective parameter.

Return value if successful: none

Necessary permissions (CM only): The user must have the `permissionWrite` permission for the specified file.

Example:

```
CM>obj withId 2003 edit
```

obj objRef editedContent

Available for: Content Management Server

Task: Functions from this function group allow access to the draft version or the committed version of a file.

Syntax:

`obj (withId` *`objId`*`) | (withPath` *`path`*`) | root editedContent` *`contentSubCommand`*

Function parameters:

- *`contentSubCommand`* specifies the subcommand. See Versions.

Return value if successful (CM only): Depends on subcommand.

Necessary permissions: Depends on subcommand.

Example:

```
CM>obj withId 25687 editedContent get title
The title of the draft version
```

For further examples, please refer to the section about the content command.

obj objRef exportSubtree

Available for: Content Management Server

Task: Exports the folder hierarchy starting at the specified folder.

Syntax:

`obj (withId` *`objId`*`) | (withPath` *`path`*`) | root exportSubtree {`*`parameter value`*`}`

Function parameters:

- *parameter* specifies the name of a parameter whose value modifies the export results. The following values are valid for *parameter*:
  - absoluteFsPathPrefix: Specifies that links that are exported as file paths are to be exported as absolute paths and that *value* is to be used as their prefix. In the context of this command, this parameter overrules the corresponding entry in the export system configuration entry. If this entry does not exist, / is used.
  - absoluteUrlPrefix: same as absoluteFsPathPrefix but for the paths in URLs.
  - exportCharset: Specifies that *value* names the character set to be used for the documents generated during the export. The supported values are listed in the export.charsetMap system configuration entry. If this parameter is not specified, the system configuration value export.charset is used. If this entry is missing, utf-8 is used.
  - filePrefix: Specifies that *value* contains a prefix to be attached to the export file name.
  - maxDepth: specifies that *value* contains an integer number specifying the number of folder levels to export. If this parameter is missing the numer is unlimited.
  - purgeCollections: specifies that *value* determines whether the collections are to be deleted prior to the export (1, default) or not (0).
  - templateName: Specifies that *value* contains the name of a layout that should be used during export instead of the value of the the user-specific configuration entry defaultTemplate. If this parameter is not specified and the defaultTemplate entry does not exist, mastertemplate is used.
- *value* contains the value for the respective parameter.

Additional information:

- This command can only be used for folders.
- During export, a directory structure corresponding to the folder hierarchy is created. A dedicated directory is created for each folder in the folder hierarchy. The version of the folder are stored as the index.html file in this directory. Documents are exported to this directory as *filename*.html. Images and resources receive the file name extension of their respective version.
- With the export.abortOnError system configuration entry (YES or NO) you can control whether the export will be terminated or continued if an error occurs.
- If the Search Engine Server is running and the indexing.staticExport.isActive entry in the system configuration of the Content Manager has been set to true, then the UTF-8 encoded web documents are indexed during the export, using the configured collections.

Return value if successful: none

Necessary permissions (CM only): The user must have the permissionGlobalExport permission for the file.

Example:

```
CM>obj withId 4567 exportSubtree filePrefix /tmp
```

obj objRef forward

Available for: Content Management Server

Task: Forwards the file to the next group in the processing workflow.

Syntax:

`obj (withId objId) | (withPath path) | root forward [comment taskComment]`

Function parameters:

- *taskComment* is the comment that is to be entered in the new task.

Return value if successful: none

Necessary permissions (CM only):

- The user must have the `permissionWrite` permission for the specified file.
- The user must be editor of the specified file.

Example:

```
CM>obj withId 27814 forward
```

obj objRef get

Available for: Content Management Server, Template Engine

Task: Returns the value of the specified file field.

Syntax:

`obj (withId objId) | (withPath path) | root get objAttr`

Additional information: Instead of file fields you can specify version fields, provided that the file has at least one version. Regarding the values returned, the released versions have precedence over edited or committed versions. To the Template Engine only released versions are known.

Function parameters:

- `objAttr` specifies the name of the file field whose value is being searched for (see File attributes).

Return value if successful: the value of the queried file field (string)

Necessary permissions (CM only): The user must have the `permissionRead` permission for the specified file. The file fields `id`, `name`, `visibleName`, `path`, `visiblePath`, and `objType` can be queried without having the `permissionRead` permission.

Example:

```
CM>obj withId 2003 get children
2033 4127 98314
```

obj objRef getHierarchy

Available for: Content Management Server, Template Engine

Task: Returns the file hierarchy located below the given folder.

Syntax:

`obj (withId objId) | (withPath path) | root getHierarchy {parameter value}`

Additional information: The hierarchy contains only the IDs of the files to which the user has read access (TE users have root access and therefore can read all files). The hierarchy returned is a Tcl list with two elements. The first element of this list can be 0 or 1 and specifies whether the hierarchy had to be cut off (1) or not (0). The second element is a list that contains the actual hierarchy. In this list, every CMS file except folders is represented by its ID. Folders are represented by a list with two elements of which the first is the ID of the folder, and the second is a list containing the contents of the folder.

Function parameters:

- *parameter* specifies the name of a parameter whose value is expected in the corresponding *value*. The following parameter names exist:
  - `maxDepth`: Specifies the maximum depth the computed hierarchy may have. The default value is 4 unless it is overridden by the value of the `maxHierarchyDepth` key (either in the user's personal preferences or in the general `userManagement.preferences` section).

    The value `-1` specifies that no depth limit is to be applied (from Version 6.7.2).
  - `maxLines`: Specifies the maximum number of files to be added to the hierarchy returned. The default value is 200 unless it is overridden by the value of the `maxHierarchyLines` key (either in the user's personal preferences or in the general `userManagement.preferences` section).

    The value `-1` specifies that no limit regarding the number of files is to be applied (from Version 6.7.2).
  - `objTypes`: Specifies the list of file types to which the hierarchy is to be restricted. As a default, only `publication` files are contained in the hierarchy. If this parameter is specified and the `publication` type is missing from the list, no hierarchy will be returned.
- *value* contains the value for the respective parameter.

Return value if successful: the hierarchy as an encoded Tcl list (stringlist)

Necessary permissions (CM only): The user must have the `permissionRead` permission for the specified file.

Example:

```
CM>obj root getHierarchy
0 {2001 {2012 {2086 2099}}}
```

Use the following command to modify your personal `maxHierarchyLines` preference.

```
CM>userConfig setTexts maxHierarchyLines 250
```

obj objRef giveTo

Available for: Content Management Server

Task: Forwards the file to a different group or a different user for editing.

Syntax:

`obj (withId objId) | (withPath path) | root giveTo {parameter value}`

Function parameters:

- *parameter* can be one of the following:
  - `userLogin` indicates that *value* specifies the user to which the task is assigned. Either this parameter or `groupName` must be specified.
  - `groupName` indicates that *value* specifies the group to which the generated task is assigned. This value must be specified if `userLogin` is not specified.
  - `taskComment` is optional and indicates, if present, that *value* is the comment that is to be assigned to the new task.
- *value* is the value of the corresponding parameter.

Return value if successful: none

Necessary permissions (CM only): The executing user must have the `permissionWrite` permission for the specified file. The receiver must have the `permissionWrite` permission.

Additional information: The following changes apply from version 6.5: The user executing this command needs to have the `permissionWrite` permission for the specified file or be its editor. There are no restrictions for the recipient of the file.

Example:

```
CM>obj withId 65123 giveTo groupName admins
```

obj objRef mget

Available for: Content Management Server, Template Engine

Task: Returns the values of the specified file fields for the specified file.

Syntax:

`obj (withId objId) | (withPath path) | root mget {objAttr}`

Additional information: Instead of file fields you can specify version fields, provided that the file has at least one version. Regarding the values returned, the released versions have precedence over edited or committed versions. In the Template Engine, only released versions exist.

Function parameters:

- `attrName` specifies the name of a file field whose value is being searched for (see File Fields).

Return value if successful: the values of the queried file fields (stringlist)

Necessary permissions (CM only): The user must have the `permissionRead` permission for the specified file. The file fields `id`, `name`, `visibleName`, `path`, `visiblePath`, and `objType` can be queried without having the `permissionRead` permission.

Example:

```
CM>obj withId 2003 mget objClass isEdited
publication 1
```

obj objRef mirror

Available for: Content Management Server (from Version 6.5.0)

Task: Creates a mirror file of the specified file. The original file itself must not be a mirror file.

Syntax:

`obj (withId objId) | (withPath path) | root mirror {parameter value}`

Function parameters:

- `parent` specifies that *`value`* specifies the path or the ID of the destination folder. If this parameter is missing, it is assumed that the destination folder is the folder containing the original file.
- `name` specifies that *`value`* specifies the name of the mirror file. If this parameter is missing, the name of the original file will be used. If this name already exists in the destination folder, a new name will be calculated by appending the next available number to the name.

Return value if successful: the ID of the new file (string)

Necessary permissions:

- The user must have the `permissionRoot` permission for the original file.
- The user must have the `permissionCreateChildren` permission for the destination folder.
- The user must have the `permissionGlobalMirrorHandling` global permission.

Additional information: The `permissionGlobalMirrorHandling` permission does not include the permission to delete mirror files. However, when a mirror file is created, the `permissionRoot` permission of this mirror file is assigned to the same groups who have been granted the `permissionCreateChildren` permission for the `parent` folder of the mirror file. As a consequence, a user who has created a mirror file is permitted to delete it (as long as he has the `permissionRoot` permission via his group membership or has the `permissionGlobalRoot` permission).

Example:

```
CM>obj withId 20099 mirror parent /news/latestnews name incoming
65811
```

obj objRef permission get

Available for: Content Management Server

Task: Returns the list of user groups whose members have the specified permission for the specified file (analogously to `obj objRef get permissions.permission`).

Syntax:

`obj (withId objId) | (withPath path) | root permission permission get`

Function parameters:

- *`permission`* specifies the permission for which the user groups are to be retrieved. Use `obj withId id get validPermissions` to output the list of the existing file permissions.

Return value if successful: The list of the names of the groups to which the given permission is assigned (stringlist)

Necessary permissions (CM only): The user must have the `permissionRead` permission for the specified file.

Example:

```
CM>obj withId 2001 permission permissionCreateChildren get
admins
```

obj objRef permission grantTo

Available for: Content Management Server

Task: The specified permission for the respective file is granted to the specified group. Other permissions of the group remain unaffected.

Syntax:

`obj (withId` *`objId`*`) | (withPath` *`path`*`) | root permission` *`permission`* `grantTo group {`*`group`*`}`

Function parameters:

- *`permission`* specifies the permission to be granted. The `validPermissions` file parameter returns the names of the existing permissions.
- *`group`* specifies the name of the group the permission is to be granted to.

Return value if successful: none

Necessary permissions (CM only): The user must have the `permissionRoot` permission for the specified file.

Example:

```
CM>obj withId 41735 permission permissionCreateChildren grantTo editors
```

obj objRef permission isGrantedToGroup

Available for: Content Management Server

Task: Checks whether the specified group has the specified permission for a CMS file.

Additional information: The members of the group always have all file-related permissions if they have the `permissionRoot` permission or are superusers, even if other file-related permissions have not been granted to them.

Syntax:

`obj (withId` *`objId`*`) | (withPath` *`path`*`) | root permission` *`permission`* `isGrantedToGroup` *`group`*

Function parameters:

- *permission* specifies the permission to be checked. The validPermissions file parameter returns the names of the existing permissions.
- *group* specifies the name of the group which is tested for having the specified permission.

Return value if successful: 1, if the group has the permission, otherwise 0 (bool).

Necessary permissions (CM only): The user must have the permissionRead permission for the specified file.

Example:

```
CM>obj withId 41735 permission permissionCreateChildren \
isGrantedToGroup editors
1
```

obj objRef permission isGrantedToUser

Available for: Content Management Server

Task: Checks whether one of the groups of which the user is a member has been granted the specified permission to access the file concerned.

Syntax:

obj (withId *objId*) | (withPath *path*) | root permission *permission* isGrantedToUser *login*

Additional information: The members of the group always have all file-related permissions if they have the file-related permission permissionRoot or are super users, even if other file-related permissions have not been granted to them.

Function parameters:

- *permission* specifies the permission to be determined. The validPermissions file parameter returns the names of the existing permissions.
- *login* specifies the login name of the user whose *permission* is checked.

Return value if successful: 1, if the user has the permission, otherwise 0 (bool).

Necessary permissions (CM only): The user must have the permissionRead permission for the specified file.

Example:

```
CM>obj withId 41735 permission permissionCreateChildren \
isGrantedToUser jane
1
```

obj objRef permission revokeFrom

Available for: Content Management Server

Task: The specified permission for the respective file is revoked for the specified group.

Syntax:

`obj (withId objId) | (withPath path) | root permission permission revokeFrom group {group}`

Function parameters:

- *permission* specifies the permission to be revoked from the group. The `validPermissions` file parameter returns the names of the existing file permissions.
- *group* specifies the name of the group the permission is to be revoked from.

Return value if successful: none

Necessary permissions (CM only): The user must have the `permissionRoot` permission for the specified file.

Example:

```
CM>obj withId 41735 permission permissionCreateChildren revokeFrom editors
```

obj objRef permission set

Available for: Content Management Server

Task: The groups to which the access permission for a file are granted, are redefined.

Syntax:

`obj (withId objId) | (withPath path) | root permission permission set {permSpec}`

Function parameters:

- *permission* specifies the permission to which the assignment refers. The `validPermissions` file parameter returns the names of the existing permissions.
- *permSpec* specifies the name of a group to which the *permission* permission is to be granted.

Return value if successful: none

Necessary permissions (CM only): The user must have the `permissionRoot` permission for the specified file.

Example:

```
CM>obj withId 41735 permission permissionCreateChildren set editors admins
```

obj objRef reject

Available for: Content Management Server

Task: Cancels the current workflow and begins a new one (workflow action *reject*).

Syntax:

`obj (withId objId) | (withPath path) | root reject [comment taskComment]`

Additional information: If the file has a committed version, it is converted to a draft version.

Function parameters:

- *taskComment* is the comment that is to be entered in the new task.

Return value if successful: none

Necessary permissions (CM only):

- If there is a draft version, the user must have the permissionWrite permission for the file.
- If there is a committed version, the user must have the permissionRead permission for the file and be a member of one of the remaining entitled groups or have the permissionRoot permission for the specified file.

Example:

```
CM>obj withId 2003 reject
```

obj objRef release

Available for: Content Management Server

Task: The specified file is released.

Syntax:

obj (withId *objId*) | (withPath *path*) | root release {parameter *value*}

Additional information: If the file has not been committed, it is committed before it is released (this means it is subjected to a commit operation). If signatures are still missing, they will be generated.

Function parameters:

- *parameter* specifies the name of a parameter of this workflow command. The following values are allowed:
  - *comment* specifies that *value* is a comment that appears in the log file (no task is generated for the workflow action *release*).
- *value* contains the value of the respective parameter.

Return value if successful: none

Necessary permissions (CM only):

- If there is a draft version, the user must be the editor and have write permission (permissionWrite). Furthermore, the release action must be the last step in the version's workflow except when the user has the permissionRoot permission. In this case the user can release the version independently of the workflow if he is the editor.
- If there is a committed version and only one signature is missing, the user must be a member of a group authorized for this signature and must have read permission.

Example:

```
CM>obj withId 2003 release
```

obj objRef releasedContent

Available for: Content Management Server

Task: Functions from this function group allow access to the released version of a file.

Syntax:

`obj (withId objId) | (withPath path) | root releasedContent versionSubCommand`

Function parameters:

- *versionSubCommand* specifies the subcommand. See Versions.

Return value if successful: Depends on the subcommand.

Necessary permissions (CM only): Depends on the subcommand.

Example: (Please see Versions for examples.)

obj objRef removeActiveContents

Available for: Content Management Server

Task: This function removes (i.e. deletes) all active versions of the specified file, i.e. the current draft version or the committed version and the released version. Archived versions will not be deleted by this function (see `content contentRef delete`).

Syntax:

`obj (withId objId) | (withPath path) | root removeActiveContents`

Additional information: If a released version exists, it will only be deleted if archiving is switched off. Otherwise the version will be converted to an archived version.

Function parameters: none.

Return value if successful: none.

Necessary permissions (CM only): the user must be the administrator of the file the version belongs to.

Example:

```
CM>obj withId 2003 removeActiveContents
```

obj objRef removeArchivedContents

Available for: Content Management Server

Task: This function removes (i.e. deletes) all archived versions of the specified file, i.e. all versions except the current draft version or the committed version and the released version.

Syntax:

`obj (withId objId) | (withPath path) | root removeArchivedContents`

Function parameters: none.

Return value if successful: none.

Necessary permissions (CM only): the user must be the administrator of the file the version belongs to.

Example:

```
CM>obj withId 2003 removeArchivedContents
```

obj objRef revert

Available for: Content Management Server

Task: The draft version of the file is deleted.

Syntax:

`obj (withId objId) | (withPath path) | root revert [comment taskComment]`

Function parameters:

- *taskComment* is the comment that is to be entered in the new task.

Return value if successful: none

Necessary permissions (CM only): The user must have the `permissionWrite` permission for the specified file.

Example:

```
CM>obj withId 20033 revert
```

obj objRef set

Available for: Content Management Server

Task: Sets the specified field of the file to the specified value.

Syntax:

`obj (withId objId) | (withPath path) | root set {objAttr value}`

Function parameters:

- *objAttr* The field of the file whose value is to be set (see File attributes).
- *value* is the value to be set for the respective file field.

Return value if successful: none

Necessary permissions (CM only): The user must have the `permissionRoot` permission for the specified file.

Example:

```
CM>obj withId 2003 set parent /news/newslist
```

obj objRef setPermissions

Available for: Content Management Server

Task: First, the specified access permission for the given file is revoked from all groups that currently have this permission. Then, the permission is given to the specified groups.

Syntax:

`obj (withId objId) | (withPath path) | root setPermissions {permission permSpec}`

Function parameters:

- *permission* specifies the permission whose group assignment is to be modified.
- *permSpec* specifies the list of groups to which the *permission* permission is to be granted. This permission is revoked from all user groups not included in this list.

Return value if successful: none

Necessary permissions (CM only): The user must have the `permissionRoot` permission for the specified file.

Example:

```
CM>obj withId 20013 setPermissions permissionRoot admins \
permissionCreateChildren {editors subeditors}
```

obj objRef sign

Available for: Content Management Server

Task: Applies the signature of the logged-in user to the committed version and generates a new task.

Syntax:

`obj (withId objId) | (withPath path) | root sign {parameter value}`

Additional information:

- The last signature in a workflow can only be provided by releasing the file.
- A file administrator is always permitted to sign the versions of his files.

Function parameters:

- *parameter* specifies the name of a parameter of this workflow command. The following values are allowed:
  - `comment` specifies that *value* is the comment that is entered in the next task.
- *value* contains the value of the respective parameter.

Return value if successful: none

Necessary permissions (CM only): The user must have the `permissionRead` permission for the specified file and be member of a group that is authorized to make the signature.

Example:

```
CM>obj withPath /news/newslist sign
```

obj objRef take

Available for: Content Management Server

Task: Makes the logged-in user the editor of the file and generates a new task.

Syntax:

`obj (withId` *`objId`*`) | (withPath` *`path`*`) | root take [comment` *`taskComment`*`]`

Function parameters:

- *`taskComment`* is the comment to be entered in the new task.

Return value if successful: none

Necessary permissions (CM only): The user must have the `permissionWrite` permission for the specified file.

Example:

```
CM>obj withId 7916 take
```

obj objRef touch

Available for: Content Management Server

Task: If this command is executed in the Template Engine, it invalidates the released version of the file concerned, simulating changes to it. This causes the file to be updated on the live server during the next export cycle.
Used in the Content Manager, this command generates an update record for the released version of the file. This also causes the file to be updated on the live server during the next export cycle.

Syntax:

`obj (withId` *`objId`*`) | (withPath` *`path`*`) | root touch`

Additional information: During the export, the command cannot be executed in the Template Engine.

Function parameters: none.

Return value if successful: none.

Necessary permissions (CM only): The user must have the `permissionGlobalExport` permission.

Example:

```
CM>obj withId 91663 touch
```

obj objRef unrelease

Available for: Content Management Server

Task: The command revokes the release of the specified file.

Additional information: The currently released version is archived if archiving is enabled. If there is no draft version, the released version is transferred to a new draft version. The file then no longer has a released version.

Syntax:

`obj (withId` *`objId`*`) | (withPath` *`path`*`) | root unrelease [comment` *`taskComment`*`]`

Function parameters:

- *`taskComment`* is a comment that appears in the log file.

Return value if successful: none

Necessary permissions (CM only): The user must have the `permissionRoot` permission for the specified file.

Example:

```
CM>obj withId 24521 unrelease
```

obj objRef updateCache

Available for: Template Engine

Task: This instruction causes the Template Engine to immediately export the CMS file concerned to the offline directory, independently of the incremental export. This can be helpful when debugging export errors.

Syntax:

`obj (withId` *`objId`*`) | (withPath` *`path`*`) | root touch`

Additional information: The command cannot be executed during the export. Prior to using this command for debugging purposes, the the logging levels should be increased. For this, set the `level` values in the `server.log` system configuration entry to `3` for the `info.log` and `error.log` files. Restart the Template Engine for the changes to take effect. When the command is then executed, detailed information about the errors that occurred can be found in the log files mentioned. After the investigation, the `level` values should be decreased again.

Function parameters: none.

Return value if successful: none.

Necessary permissions: none.

Example:

```
CM>obj withId 91663 updateCache
```

obj objRef updateInIndex

Available for: Content Management Server

Task: All versions of the specified file are reindexed. The command can only be used if the Infopark Search Cartridge is running.

Syntax:

`obj (withId` *`objId`*`) | (withPath` *`path`*`) | root updateInIndex`

Function parameters: none.

Return value if successful: none.

Necessary permissions (CM only): The user must have the `permissionRoot` permission for the specified file.

Example:

```
CM>obj withId 24521 updateInIndex
```

obj objRef verifyExport

Available for: Content Management Server, Template Engine

Task: After exporting the specified folder, this command checks whether an export file exists for each file that is to be exported and whether no export files exist for files that must not be exported. Basically, the existence of a released version, its validity period and its `suppressExport` field are checked. The command does not test whether files exist for nonexistent files.

Additional information:

- Using this command, you can also check whether the exported data of the Template Engine matches the internal data of the Content Manager, i. e. whether all files intended to be exported have in fact been exported by the Template Engine. For this purpose, the command is executed using the Content Manager, specifying the Template Enigine's export directory as directory (see the example below).
- The result of this command is only reliable if no files are modified after the export and until the end of the verification process.

Syntax:

`obj (withId` *`objId`*`) | (withPath` *`path`*`) | root verifyExport filePrefix` *`exportDir`*

Function parameters:

- *`exportDir`* is the path to the directory in which the export files to be examined are located (specified as in `exportSubtree`).

Return value if successful: none. In the case of failure, the relative paths of the missing files and of the files unexpectedly present are output line by line (see the example below).

Necessary permissions (CM only): The user must have the `permissionGlobalExport` permission.

Example:

```
CM>obj withPath /pub verifyExport \
filePrefix /opt/Infopark/NPS/instance/default/export/online/docs
missing: /index.html
missing: /pub/p35.html
unexpected: /pub/s44.html
```

4.13 objClass (File Formats)

4.13.1 File Format Parameters

Parameters	Type	Explanation	get	set	create	descr
attributeGroupNames	stringlist	The names of the field groups of the formats. The order in which the names are returned corresponds to the order of the groups (see `attributeGroup`).	•			
attributes	stringlist	The list of fields used in the format	•	•	•	•
availableBlobEditors	stringlist	The list of editors available to users for editing the main content of a draft version. The list may contain any combination of the following elements: `internalEditor`, `externalEditor`, `tinymceEditor`, `htmlEditor`. In Fiona 6.8.0, Microsoft HTML Editor (`msieEditor`) was replaced by TinyMCE (`tinymceEditor`).	•	•	•	
bodyTemplateName	string	Name of the layout used for exporting the bodies of all files based on this format.	•	•	•	
canCreateNewsItem	bool	Specifies whether a news item for the files of this format is to be created on the live server. (Only available if a Portal Manager license is present.)	•	•	•	•

completionCheck	string	Custom Tcl script which can be used for additional completeness checks. The script is always called when a version is committed. A version can only be committed if all links are resolved, the obligatory fields are assigned valid values and the `completionCheck` string is empty or contains a script which returns result 1.	•	•	•	
contentTypes	stringlist	The list of permitted file name extension for the version of a file based on this format. It is calculated from `validContentTypes` and `obj contentTypesForObjType` .	•			
createPermission	string	The permission required to create a file in this format.	•	•	•	•
defaultAttributeGroupName	string	The name of the base group (`baseGroup`).	•			
displayTitle	string	The title of the format as displayed in the HTML user interface	•			
emptyAttributeGroups	stringlist	List of field groups with no fields assigned to them.	•			
customBlobEditorUrl	string	The URL to which a request is sent when the main content of a draft version is to be edited with the custom editor (see also `availableBlobEditors`).	•	•	•	
getKeys	stringlist	List of parameters which can be queried with `get`	•			
goodAttributeGroupAttributes	stringlist	List of fields that can be added to the format's field groups	•			
goodAttributes	stringlist	List of fields that can be added to the format.	•			
goodMandatoryAttributes	stringlist	List of fields that can be added to the format's list of `mandatoryAttributes`	•			
goodPresetAttributes	stringlist	List of fields that can be added to the format's list of `presetAttributes`	•			
goodPresetFromParentAttributes	stringlist	List of fields that can be added to the format's list of `presetFromParentAttributes`	•			
isEnabled	bool	Indicates whether the format can be assigned to files	•	•	•	•

localizedTitle	string	`title.language` in the language the user has selected. If this title is empty then `name` is returned.	•			
mandatoryAttributes	stringlist	List of obligatory fields for a file with this format (only custom or file fields (see `obj get` *`parameter`*)	•	•	•	•
name	string	The name of the file format	•	•	•	•
objType	string	The type of a file based on this format (document, publication, template, image, generic)	•		•	•
presetAttributes	stringlist	List of fields which are assigned predefined values from the format when the file is created. The list contains the attribute name and the values (all those given in the `attributes` list and the predefined fields are permitted).	•	•	•	•
presetFromParentAttributes	stringlist	List of fields whose values are transferred from the parent folder to a new file based on this format (all those given in the `attributes` list and the predefined fields are permitted)	•	•	•	•
recordSetCallback	string	Custom Tcl code that is called when field values are assigned to the draft version of a file based on this format.	•	•	•	
setKeys	stringlist	The list of parameters which can be set with `set`	•			
title	string	The title of the format in the user-specific language	•	•	•	•
title.*language*	string	The title of the format in the respective language (from the `localizer` section of the system configuration.)	•	•	•	•
validContentTypes	stringlist	The list of permitted file name extensions for the version of a file based on this format. If this value is empty, all file name extensions are valid that are returned by `obj contentTypesForObjType` for the file type of this format.	•	•	•	•

validSortKeys	stringlist	The field names available as value for `sortKey` (from version 6.7.1)	•			
validSortOrders	stringlist	The available values for `sortOrder` (from version 6.7.1)	•			
validSortTypes	stringlist	The available values for `sortType` (from version 6.7.1)	•			
validSubObjClassCheck	string	Tcl-Code that is called when a file is to be created in a folder based on this format.	•	•	•	
validSubObjClasses	stringlist	The list of format names which are permitted for the files contained in folders with this format. This list may only be assigned in formats which are used to create folder type files.	•	•	•	•
workflowModification	string	Tcl-Code that is called before a workflow is assigned to a draft version of a file based on this format.	•	•	•	
xmldtd	string	The XML-DTD belonging to the format	•			

4.13.2 File Format Commands

objClass create

Available for: Content Management Server

Task: The command creates a new file format.

Syntax:

`objClass create {`*`parameter value`*`}`

Function parameters:

- *`parameter`* specifies the name of the format parameter whose value will be set during the create procedure (see Format parameters).
- *`value`* the value for the format parameter to be set.

Return value if successful: the name of the format (string)

Necessary permissions: The user must have the `permissionGlobalRTCEdit` permission.

Example:

```
CM>objClass create name newslist objType publication title {News list} validSubObjClasses
 news attributes {abstract source}
newslist
```

objClass goodAvailableBlobEditorsForObjType

Available for: Content Management Server

Task: The command returns the list of editors that can in principle be used to edit the main content or field values of the draft version of a file that has the specified type.

Additional information: The list of editors with which the main content or field values of a version can actually be edited can be read from the format parameter `availableBlobEditors` .

Syntax:

`objClass goodAvailableBlobEditorsForObjType` *`objType`*

Function parameters:

- *`objType`* specifies the file type.

Return value if successful: the list of available editors for the specified file type (stringlist).

Necessary permissions: no restrictions.

Example:

```
CM>objClass goodAvailableBlobEditorsForObjType document
internalEditor externalEditor tinymceEditor htmlEditor
```

objClass list

Available for: Content Management Server

Task: Returns a list of the names of all managed formats.

Syntax:

`objClass list`

Function parameters: none

Return value if successful: the list of formats (stringlist)

Necessary permissions: no restrictions

Example:

```
CM>objClass list
document generic image news newslist publication template
```

objClass validAttributes

Available for: Content Management Server

Task: Returns the list of the additional (i.e. custom) fields that have been assigned to file formats as a whole.

Syntax:

`objClass validAttributes`

Function parameters: none

Return value if successful: the list of field names (stringlist)

Necessary permissions: no restrictions

Example:

```
CM>objClass validAttributes
abstract keywords price levels list size source
```

Using the following Tcl code you can determine in which file formats a particular field (here `myfield`) is used:

```
set fieldname myfield
foreach c [objClass list] {
  if { [lsearch  [objClass withName $c get attributes] $fieldname]  != -1} {
    puts $c
  }
}
```

objClass where

Available for: Content Management Server

Task: Allows you to search for file formats where the value of the entered parameter contains the specified string.

Syntax:

`objClass where {`*`parameter value`*`}`

Function parameters: (if no parameters are specified the names of all formats will be returned.)

- *`parameter`* specifies the name of the parameter whose value is to be searched for or specifies the number of hits. The following parameters are allowed:
  - `maxResults` specifies that *`value`* is a number that limits the number of hits. If number is less than or equals zero, the number of hits is unlimited.
  - `name` specifies that the name of the formats will be checked for occurrences of *`value`*.
  - `isEnabled` specifies that *`value`* contains a logical value and only the name of the formats in which the `isEnabled` parameter value equals the logical value should be returned. (Only formats in which `isEnabled` is true can be assigned to files.)
  - `objType` specifies that only the name of the formats whose file type is contained in the list specified as *`value`* is to be issued.
- *`value`* contains the value of the corresponding parameter.

Return value if successful: the list of names from the matching formats (stringlist)

Necessary permissions: no restrictions

Example:

```
CM>objClass where objType publication
newslist publication
```

objClass objClassRef delete

Available for: Content Management Server

Task: deletes the specified file format.

Syntax:

`objClass withName` *`objClassName`* `delete`

Function parameters: none

Return value if successful: none

Necessary permissions: The user must have the `permissionGlobalRTCEdit` permission.

Example:

```
CM>objClass withName newslist delete
```

objClass objClassRef description

Available for: Content Management Server

Task: Returns a string representation of the data from the entered file format.

Additional information: The representation is formatted in the property list format of a dictionary (see The property list format).

Syntax:

`objClass withName` *`objClassName`* `description`

Function parameters: none

Return value if successful: the string representation of the format (string)

Necessary permissions: no restrictions

Example:

```
CM>objClass withName newslist description
{
  attributes = (
    abstract,
    source
  );
  isEnabled = 1;
  name = newslist;
```

```
  objType = publication;
  presetAttributes = {};
  title = "News-Liste";
  validSubObjClasses = (
    news
  );
}
```

objClass objClassRef get

Available for: Content Management Server

Task: Returns the value of the specified file format parameter of the given file format.

Syntax:

`objClass withName` *`objClassName`* `get` *`parameter`*

Function parameters:

- *`parameter`* specifies the name of the parameter whose value is being searched for (see Format parameters).

Return value if successful: the value of the entered parameter (string)

Necessary permissions: no restrictions

Example:

```
CM>objClass withName newslist get title
News list
```

objClass objClassRef mget

Available for: Content Management Server

Task: Returns the values of the specified parameters of the given file format.

Syntax:

`objClass withName` *`objClassName`* `mget {`*`parameter`*`}`

Function parameters:

- *`parameter`* specifies the name of the parameter whose value is being searched for (see Format parameters).

Return value if successful: the list of values for the parameter concerned (stringlist)

Necessary permissions: no restrictions

Example:

```
CM>objClass withName newslist mget objType title
publication {News list}
```

objClass objClassRef set

Available for: Content Management Server

Task: Sets the specified parameters of the given file format to the specified values.

Additional information: From version 6.6.1 of Infopark CMS Fiona the name of a file format can be altered using the `set` command. Doing this causes all CMS-internal references to this format to be automatically adapted. This might have the effect that the Template Engine running on the live server exports all files again. In layouts and scripts the format names need to be adapted manually.

Syntax:

`objClass withName` *`objClassName`* `set {`*`parameter value`*`}`

Function parameters:

- *`parameter`* specifies the parameter from the format whose value is to be set (see Format parameters).
- *`value`*: the value to be set for the respective parameter.

Return value if successful: none

Necessary permissions: The user must have the `permissionGlobalRTCEdit` permission.

Example:

```
CM>objClass withName news set title {News article}
```

4.14 reminder

By means of the `reminder` command available from version 6.5.0 reminders can be created, displayed and deleted.

4.14.1 reminder define

Available for: Content Management Server

Task: Creates a reminder date using the specified parameters.

Syntax:

`reminder define {`*`parameter value`*`}`

Function parameters:

- *`parameter`* can be one of the following values:
  - `objectId`: ID of the file for which the reminder is to be created.
  - `from`: The reminder date in the 14-place date format (see example).
  - `users`: The list of the users for which the reminder is to be created.
  - `groups`: The list of the groups for which the reminder is to be created.
  - `comment`: An optional comment for the reminder (e.g. hints for the editor).

- *value* contains the value of the *parameter*.

Return value if successful: none.

Required permissions: Write permission for the specified file.

Example:

```
CM> reminder define objectId 2016 from 20101221080000 \
users {"smith"} comment "update slogan"
```

4.14.2 reminder delete

Available for: Content Management Server

Task: This command deletes a reminder.

Syntax:

`reminder delete {`*parameter value*`}`

Function parameters:

- *parameter* must be `objectId`. This value specifies that *value* is the ID of the file whose reminder is to be deleted.
- *value* contains the value of the specified *parameter*.

Return value if successful: none.

Required permissions: Write permission for the specified file.

Example:

```
CM> reminder delete objectId 2016
```

4.14.3 reminder where

Available for: Content Management Server

Task: Returns a restrictable list of reminders.

Syntax:

`reminder where {`*parameter value*`}`

Function parameters:

- *parameter* defines a criterion which the reminders searched for need to fulfill (several parameters can be specified):
  - `from`: The reminder date must be after the date specified as *value* (in the 14-place format).
  - `until`: The reminder date must be before the date specified as *value* (in the 14-place format).
  - `user`: The user specified as *value* is a recipient.

- `group`: The group specified as *`value`* (i.e. its members) is a recipient.
- `objectID`: The reminder is associated with a file located in a subhierarchy. The starting folder of this subhierarchy is specified as *`value`*.

- *`value`* contains the value of the corresponding *`parameter`*.

Return value if successful: A list containing the data of the matching reminders.

Required permissions: none

Example:

```
CM> reminder where objectId 2016 until 20101225000000 user smith
{objectId 2016 from 201001221080000 users smith objectType document path /news/article1
 groups {} comment {update slogan}}
```

4.15 statistics (Statistical Information)

4.15.1 Statistics Parameters

Parameter	Type	Explanation	Applic.	get	descr
getKeys	stringlist	List of the statistics parameters that can be queried with `get`	CM, TE	•	
names	stringlist	Returns the keywords accepted by the `compute` subcommand (see below).	CM, TE	•	
formats	stringlist	Returns the formats accepted by the `compute` subcommand (see below).	CM, TE	•	

4.15.2 Statistics Commands

statistics compute

Available for: Content Management Server, Template Engine

Task: This command generates version, link, and file statistics.

Syntax:

`statistics compute {`*`parameter value`*`}`

Function parameters:

- *`parameter`* can be one of:
  - `name` specifies that *`value`* contains one of the following keywords indicating the type of statistical information to compute. `name` is mandatory.
    - `all` (CM, TE): returns the results of all other functions listed in the following.

  - `contents` (CM): generates statistical information on versions.
  - `dependencies` (TE): generates statistical information on dependencies.
  - `export` (TE): generates statistical information on the incremental export.
  - `links` (CM): returns statistical information on links.
  - `objects` (CM, TE): generates statistical information on objects.
  - `topReferences` (TE): generates a list of the files that have the most dependencies.
- `format` optionally specifies that *`value`* contains one of the following output formats:
  - `tcl`: the result is to be returned as a tcl list.
  - `text`: the result is to be returned as normal text. This is the default value.
- *`value`* contains the value for the respective parameter.

Return value if successful: the requested statistical information.

Necessary permissions (CM only): the user is required to have the `permissionGlobalRoot` permission.

Example:

```
TE>statistics compute name dependencies
contents
--------
total contents : 258 (100.0%)
edited contents : 8 ( 3.1%)
released contents : 192 ( 74.4%)
archived contents : 58 ( 22.5%)
avg contents/object : 1.3
```

statistics get

Available for: Content Management Server, Template Engine

Task: Returns the value of the specified statistics parameter.

Syntax:

`statistics get` *`parameter`*

Function parameters:

- *`parameter`* specifies the name of the parameter whose value is being searched for. Please refer to section Statistics parameters for a list of valid names.

Return value if successful: the value of the corresponding parameter.

Necessary permissions (CM only): the user is required to have the `permissionGlobalRoot` permission.

Example:

```
TE>statistics get names
all contents links objects
```

4.16 systemConfig (System Configuration)

The `systemConfig` command allows write and read access to the system configuration which the Content Manager and the Template Engine read from their initialization files when they are executed. Changes to the system configuration only affect the running application. The initialization file concerned will not be updated during this process. Therefore, changes you want to be permanent need to be made to the corresponding initialization file.

Data to be stored in the system configuration with the `systemConfig` command need to exist in the XML format. In versions prior to 5.0, the Infopark CMS used the property list format instead.

4.16.1 The Property List Format

In a property list, values are assigned to names. This is done as in the following example:

```
name = value;
```

Each assignment ends with a semicolon. The values in the property lists can be strings, dictionaries (lists of name-value pairs) or arrays (lists of values).

String

A string is a sequence of characters enclosed in quotation marks or any combination of the characters ‘a’ - ‘z’, ‘A’ - ‘Z’, ‘0’ - ‘9’ and the underline ‘_’ character.

Permitted strings are for example:

```
String
"Sequence of characters"
```

Quotation marks can be used in strings enclosed in quotation marks if they are preceded by a backslash as in the following example:

```
"The word \"and\" consists of three characters"
```

Dictionary

A dictionary is a list of name-value pairs which starts with a left curly bracket and ends with a right curly bracket:

```
server = {
  host = localhost;
  tclPort = 3001;
  secondPort = 3002;
};
```

Each value from a name-value pair in a dictionary can be a string, an array or another dictionary.

Array

An array is a list of strings, dictionaries or arrays which starts with a left parenthesis and ends with a right parenthesis. The elements in a list are separated by commas. The following examples give the permitted arrays:

```
name1 = (An, array, with, "five", elements);
name2 = ((An, array), with, (three, elements));
name3 = (
  {
  name1 = An;
  name2 = array;
  },
  {
  name3 = from;
  name4 = (two, dictionaries);
  }
);
```

Comments

Property lists can contain comments. Two backslashes start a comment which covers the rest of the line:

```
// This is the first comment
```

Comments which are enclosed with data must start with the string "/*" and end with the string "*/" as in the following example:

```
/* This is the second comment */
```

4.16.2 System Configuration Commands

systemConfig formatDate

Available for: Content Management Server, Template Engine

Task: The command is obsolete. formatDateTime offers the same functionality.

Syntax:

`systemConfig formatDate` *`date`*

systemConfig formatDateTime

Available for: Content Management Server, Template Engine

Task: Formats the entered date. The date and time are converted to the time zone of the server.

Additional information: The value from the `validDateTimeOutputFormats` dictionary to which the value of the key `userManagement.preferences.dateTimeOutputFormatName` points is used as the output format. If the subcommand is executed via `userConfig`, the user-specific value of `dateTimeOutputFormatName` is used.

Syntax:

`systemConfig formatDateTime` *`dateTime`*

Function parameters:

- *`dateTime`* is the date to be formatted. It should be entered in the form *YYYYMMDDhhmmss* . If the date has too few digits, the year figure will be filled in with zeros, and the month and day figure with *01* and hours, minutes and seconds with zeros. Extra figures will be ignored.

Return value if successful: the formatted time stamp (string)

Necessary permissions (CM only): none

Example:

```
CM>systemConfig formatDateTime 20000115180001
15.01.2000 19:00 MET
```

systemConfig getAttributes

Available for: Content Management Server, Template Engine

Task: Returns the specified tag attributes and their values for the system configuration element stored at the *`elemPath`* location.

Syntax:

`systemConfig getAttributes` *`elemPath`* {*`attrName`*}

Function parameters:

- *`elemPath`* is the path of the configuration element whose tag attributes are to be returned. The parts of a path must be separated by a period.
- *`attrName`* is the name of a a tag attribute whose value is to be returned. If no tag attributes are specified all tag attributes and their values will be returned.

Return value if successful: The list of tag attributes and their values. In this list each attribute and its respective value forms a name-value pair.

Necessary permissions (CM only): none

Example:

```
CM>systemConfig getAttributes userManagement.globalPermissions
type list
```

systemConfig getCounts

Available for: Content Management Server, Template Engine

Task: Returns the number of elements stored under the system configuration element at the specified location.

Syntax:

`systemConfig getCounts {elemPath}`

Function parameters:

- *elemPath* is the path of a configuration element whose number of elements is to be returned. The parts of a path must be separated by a period.

Return value if successful: A list containing as many numbers as paths have been specified. Each number signifies the number of subelements of the corresponding configuration element.

Necessary permissions (CM only): none

Example:

```
CM>systemConfig getCounts content.globalPermissions
5
```

systemConfig getElements

Available for: Content Management Server, Template Engine

Task: Returns system configuration elements at the location *elemPath* .

Syntax:

`systemConfig getElements {elemPath}`

Function parameters:

- *elemPath* is the path of a configuration element to be returned. The parts of a path must be separated by a period.

Return value if successful: A list containing as many configuration elements as paths have been specified. The elements are not formatted.

Necessary permissions (CM only): none

Example:

```
CM>systemConfig getElements tuning.master tuning.slave
```

systemConfig getKeys

Available for: Content Management Server, Template Engine

Task: Returns the names of the subentries of the specified system configuration elements.

Syntax:

`systemConfig getKeys {elemPath}`

Function parameters:

- *elemPath* is the path of a configuration element whose subentries are to be returned. The parts of a path must be separated by a period.

Return value if successful: A list containing the list of subentry names for each *elemPath* specified.

Necessary permissions: none

Example:

```
CM>systemConfig getKeys tuning.master tuning.slave
{maxSlaves slaveIdleTimeout slaveExecMaxFailures slaveExecArguments
slaveShutdownTimeout slaveStartupTimeout minIdleSlaves}
{maxNumberOfRequests requestTimeout}
```

systemConfig getTexts

Available for: Content Management Server, Template Engine

Task: The function returns the contents of the specified system configuration elements.

Additional information: A configuration element can have as content either subelements or a character string value, but not both. Therefore the function returns the empty character string if the queried configuration element has subelements.

Syntax:

`systemConfig getTexts {`*elemPath*`}`

Function parameters:

- *elemPath* is the path of a configuration element whose contents is to be returned. The parts of a path must be separated by a period.

Return value if successful: A list containing the contents of the specified system configuration elements. The contents are not formatted.

Necessary permissions (CM only): none

Example:

```
CM>systemConfig getTexts tuning.master.cm.minIdleSlaves content.sortKey
3 name
```

systemConfig installedLanguages

Task: Returns the list of the languages (shortcuts) for which localizers are available in the system configuration.

Syntax:

`systemConfig installedLanguages`

Function parameters: none.

Return value if successful: the list of the installed languages.

Necessary permissions: none.

Example:

```
CM>systemConfig installedLanguages
en de fr es it
```

systemConfig parseInputDate

Available for: Content Management Server, Template Engine

Task: Interprets the input value as a date and returns the corresponding 14-digit date string. Date and time are converted to GMT from the server time zone.

Additional information: The `userManagement.preferences.timeZone` configuration value is used as the server time zone. If date entries are ambiguous, the `preferences.dateTimeInputFormatName` configuration value will be used to interpret them.

Syntax:

`systemConfig parseInputDate` *`inputDate`*

Function parameters:

- *`date`* is the input date to be interpreted.

Return value if successful: The date as a 14-digit string.

Necessary permissions (CM only): none

Example:

```
CM>systemConfig parseInputDate {15.1.2000 19:00 MET}
20000115180000
```

systemConfig removeAttributes

Available for: Content Management Server, Template Engine

Task: The function deletes the specified tag attributes from the system configuration element with the given path.

Additional information: This command can be used only on application start and in single mode.

Syntax:

`systemConfig removeAttributes` *`elemPath`* {*`attrName`*}

Function parameters:

- *`elemPath`* is the path of a configuration element whose tag attributes are to be deleted. The parts of a path must be separated by a period.
- *`attrName`* is the name of a tag attribute to be deleted. If no tag attribute has been specified all attributes will be deleted.

Return value if successful: none.

Necessary permissions (CM only): The logged-in user must be a super user.

Example:

```
CM>systemConfig removeAttributes tuning.master
```

systemConfig removeKeys

Available for: Content Management Server, Template Engine

Task: The function deletes the specified elements from the system configuration.

Additional information: This command can be used only on application start and in single mode.

Syntax:

`systemConfig removeKeys {elemPath}`

Function parameters:

- *elemPath* is the path of a configuration element that is to be deleted. The parts of a path must be separated by a period.

Return value if successful: none.

Necessary permissions: The logged-in user must be a super user.

Example:

```
CM>systemConfig removeKeys content.globalPermissions tuning
```

systemConfig setAttributes

Available for: Content Management Server, Template Engine

Task: The function modifies the values of tag attributes or adds the tag attributes specified as name-value pairs to the given system configuration element.

Additional information: This command can be used only on application start and in single mode.

Syntax:

`systemConfig setAttributes elemPath {name value}`

Function parameters:

- *elemPath* is the path of the configuration element to which tag attributes are to be added. The parts of a path must be separated by a period.
- *name* is the name of a tag attribute to be added to the system configuration. If the element already has such an attribute, its value will be set to the new value.
- *value* is the new value of the respective tag attribute referred to with *name*.

Return value if successful: none.

Necessary permissions (CM only): The logged-in user must be a super user.

Example:

```
CM>systemConfig setAttributes server.cm fileName cm.xml
```

systemConfig setElements

Available for: Content Management Server, Template Engine

Task: The function replaces system configuration elements or adds elements to the system configuration.

Additional information: This command can be used only on application start and in single mode.

Syntax:

`systemConfig setElements {`*`elemPath element`*`}`

Function parameters:

- *`elemPath`* is the path of a configuration element to be replaced or to be added to the system configuration. The element itself is specified as the *`element`* string immediately following *`elemPath`*. The parts of a path must be separated by a period.
- *`element`* is the new configuration element with the path *`elemPath`*.

Return value if successful: none.

Necessary permissions (CM only): The logged-in user must be a super user.

Example:

```
CM>systemConfig setElements tuning.master \
{<master fileName="master.xml"><maxSlaves>2</maxSlaves><slaveIdleTimeout>60
</slaveIdleTimeout><slaveExecMaxFailures>4</slaveExecMaxFailures>
<slaveExecArguments/><slaveShutdownTimeout>5</slaveShutdownTimeout>
<slaveStartupTimeout>100</slaveStartupTimeout><minIdleSlaves>1
</minIdleSlaves></master>}
```

systemConfig setTexts

Available for: Content Management Server, Template Engine

Task: The function replaces the contents of system configuration elements or adds elements to the system configuration.

Additional information:

- This command can be used only on application start and in single mode.
- A configuration element can have as contents either subelements or a character string value, but not both. Therefore all subelements of a configuration element will be deleted when its contents is set with `setTexts`.

Syntax:

`systemConfig setTexts {elemPath elemText}`

Function parameters:

- *elemPath* is the path of a configuration element to be replaced or to be added to the system configuration. The content of an element (without the surrounding tags) is specified as the *elemText* string immediately following *elemPath*. The parts of a path must be separated by a period.
- *elemText* is the contents of the new configuration element with the path *elemPath*.

Return value if successful: none.

Necessary permissions (CM only): The logged-in user must be a superuser.

Examples:

```
CM>systemConfig setTexts tuning.master.cm.minIdleSlaves 3 content.sortKey title
CM>systemConfig setTexts userManagement.preferences.maxHierarchyLines 250
```

systemConfig validInputCharsets

Available for: Content Management Server, Template Engine

Task: The function returns the value of the `inputCharsets` system configuration entry as a list. This list contains the names of the character sets that are valid for imported versions.

Syntax:

`systemConfig validInputCharsets`

Function parameters: none.

Return value if successful: none.

Necessary permissions (CM only): no restrictions.

Example:

```
CM>systemConfig validInputCharsets
utf-8 iso8859-1 iso8859-2
```

systemConfig validTimeZoneNames

Available for: Content Management Server, Template Engine

Task: The function returns the list of valid timezone names.

Syntax:

`systemConfig validTimeZoneNames`

Function parameters: none.

Return value if successful: none.

Necessary permissions (CM only): no restrictions.

Example:

```
CM>systemConfig validTimeZoneNames
GMT-12 GMT-11 HST HST US/Hawaii GMT-10 US/Hawaii Canada/Yukon US/Yukon GMT-9
Canada/Pacific US/Pacific-New PST8PDT US/Pacific GMT-8 US/Arizona MST GMT-7
US/Mountain MST7MDT Canada/Mountain Canada/Central Canada/East-Saskatchewan
US/Central GMT-6 CST6CDT EST EST5EDT US/East-Indiana Canada/Eastern US/Eastern
GMT-5 Canada/Atlantic GMT-4 Canada/Newfoundland GMT-3 GMT-2 GMT-1 GMT GB-Eire WET
Iceland UTC Greenwich Universal CET GMT+1 Poland MET GMT+2 EET Turkey GMT+3 W-SU
GMT+4 GMT+5 GMT+6 GMT+7 Australia/West Singapore GMT+8 Japan GMT+9 Australia/North
GMT+10 Australia/Queensland Australia/South Australia/Victoria Australia/Tasmania
GMT+11 Australia/NSW NZ GMT+12 GMT+13
```

4.17 task (Tasks)

4.17.1 Task Parameters

Parameters	Type	Explanation	get	descr
comment	string	The comment to the workflow action that caused the task to be generated.	•	•
displayTitle	string	The task's title as it is displayed in the HTML user interface	•	
getKeys	stringlist	List of parameters which can be queried with `get`	•	
groupName	string	Name of the group to whose members the entry applies	•	•
objId	string	ID of the file to which the entry refers	•	•
taskType	string	The type of task to be carried out (`edit`, `sign`)	•	•
timeStamp	string	Date when the task was created	•	•
title	stringlist	Title of the task	•	•
userLogin	string	Name of the user to whom the entry applies	•	•

4.17.2 Task Commands

task list

Available for: Content Management Server

Task: Creates a list of the IDs of all tasks.

Syntax:

`task list`

Function parameters: none

Return value if successful: list of task IDs (stringlist)

Necessary permissions: no restrictions

Example:

```
CM>task list
2140.3 5351.12 2035.4 2893.5 2252.10 5358.7 2042.3 2259.4 5148.1
```

task where

Available for: Content Management Server

Task: Creates a list of the IDs of all tasks in which the specified parameters have the specified values.

Syntax:

`task where {`*`parameter value`*`}`

Function parameters:

- *`parameter`* The name of the parameter whose value is specified in *`value`* and which is being searched for. If several parameters are entered, they will be combined with *and*. The parameters can be:
  - `maxResults` specifies that *`value`* is a number that limits the number of hits. If number is less than or equal to zero, the number of hits is unlimited.
  - `userLogin`: only the tasks which are assigned to the user specified in *`value`* will be returned.
  - `groupNames`: only the tasks which are assigned to the groups specified in the *`value`* string list will be returned.
  - `taskText`: the text given as *`value`* needs to be contained in the comment or title of a task.
  - `objId`: only tasks which refer to the file with the ID given as *`value`* will be listed.
  - `taskType`: only tasks of the type given as *`value`* will be listed. The type can be one of the following values:
    - `edit`: only tasks in an editing workflow are taken into account.
    - `sign`: only tasks in a signature workflow are taken into account.
- *`value`* is the value of the respective parameter.

Return value if successful: list of task file IDs (stringlist)

Necessary permissions: no restrictions

Example:

```
CM>task where taskType edit
2014.5 2035.4 2042.21 2049.16 2056.231 2063.19 2070.34 2084.6
```

task taskRef description

Available for: Content Management Server

Task: Returns a string representation of the data from the task with the specified ID.

Additional information: The representation is formatted in the property list format of a dictionary.

Syntax:

`task withId taskId description`

Function parameters: none

Return value if successful: the string representation of the task (string)

Necessary permissions: none

Example:

```
CM>task withId 2014.7 description
{
  objId = 2001;
  taskType = edit;
  timeStamp = 20011129221803;
  title = "new\ content\ created";
  userLogin = root;
}
```

task get

Verfügbar für: Content Management Server

Task: Returns the value of the specified parameter of the *taskId* task.

Syntax:

`task withId taskId get attrName`

Function parameters:

- *attrName* is the name of the task parameter whose value is being searched for (see Task parameters).

Return value if successful: the value of the specified task parameter (string)

Necessary permissions: none

Example:

```
CM>task withId 2014.7 get title
new content created
```

task taskRef mget

Available for: Content Management Server

Task: Returns the values of the specified parameters of a task.

Syntax:

`task withId` *`taskId`* `mget {`*`attrName`*`}`

Function parameters:

- *`attrName`* specifies the name of the task parameter whose value is being searched for (see Task parameters).

Return value if successful: list of task parameters (stringlist)

Necessary permissions (CM only): none

Example:

```
CM>task withId 2014.7 mget taskType title
edit {new content created}
```

4.18 user (Users)

4.18.1 User Parameters

Parameter	Type	Explanation	get	set	create	descr
defaultGroup	string	User's default group	•	•	•	•
displayTitle	string	The user's login and/or name as it is displayed in the HTML user interface	•			
encryptedPassword	string	User's encrypted password	•	•	•	•
email	string	User's e-mail address	•	•	•	•
externalAttrNames	stringlist	The list of user fields	•			
getKeys	stringlist	List of parameters which can be queried with `get`	•			
globalPermissions	stringlist	List of global permissions	•	•	•	•
groups	stringlist	List of groups in which the user is a member	•	•	•	•
login	string	User's login	•		•	•
owner	string	Owner of user	•	•	•	•
password	string	User's password in plain text		•	•	
realName	string	User's full name	•	•	•	•
setKeys	stringlist	List of parameters which can be set with `set`	•			
userLocked	bool	Indicates whether the user is locked	•	•	•	•

externalAttribute	the field's type	Value of a user field	•	•	•	•

Owner of a user

When a user is created, the logged-in user will automatically become the direct owner (`owner`) of the new user. The owner is allowed to modify the fields and permissions of the new user. The login of the direct owner is defined in the user's `owner` parameter.

The owner of a user can also modify the value in the user's `owner` parameter. This value can be either a login or a group name. When a new owner is entered, the previous owner will pass his permission onto another user or to a group.

If a group is entered in the `owner` parameter of a user, all members of the group will be indirect owners.

In order to change a user's parameters you must be the direct or indirect owner of the user and have the `permissionGlobalUserEdit` permission. Users can therefore not modify their data themselves unless they are their own direct or indirect owner.

The `owner` parameter works in exactly the same way for groups.

4.18.2 User Commands

user create

Available for: Content Management Server

Task: This command creates a new user with the specified parameter values.

Additional information:

- The `login` parameter must be set when a user is created. `Login` cannot be changed subsequently.
- The `owner` of the user will be set to the logged-in user if `owner` is not specified as a parameter when `create` is called.

Syntax:

`user create {`*`parameter value`*`}`

Function parameters:

- *`parameter`* specifies the name of the user parameter whose value is to be set during the create procedure (see The user parameters). The `login` and `defaultGroup` user parameter must be specified.
- *`value`* is the value to be set for the parameter.

Return value if successful: the login of the new user (string)

Necessary permissions: The user must have the `permissionGlobalUserEdit` permission. Furthermore, the user must be the owner of the new user's default group.

Example:

```
CM>user create login jane realName {Jane Mitchum} \
defaultGroup admins
jane
```

user list

Task: Lists the logins of all managed users.

Syntax:

`user list`

Function parameters: none

Return value if successful: list of user logins (stringlist)

Necessary permissions: none

Example:

```
CM>user list
admin jane long michael root walter
```

user where

Available for: Content Management Server

Task: Lists the logins of all managed users in whose name or login the specified string occurs.

Syntax:

`user where {`*`parameter value`*`}`

Function parameters:

- *`parameter`* specifies the name of the parameter whose value is to be searched for or specifies the number of hits. The following parameters are allowed:
  - `maxResults` specifies that *`value`* is a number that limits the number of hits. If number is less than or equals zero, the number of hits is unlimited.
  - `userText` specifies that *`value`* is a string for which the user parameters `name` and `login` are searched.
- *`value`* is the string which is searched for in the values of the `name` and `login` user parameters.

Return value if successful: list of user logins (stringlist)

Necessary permissions: none

Example:

```
CM>user where userText michael
long michael
```

user userRef addToGroups

Available for: Content Management Server

Task: Adds the user with the *login* login to each of the specified groups if he is not already a member.

Additional information: This action does not change the user's *login* but the specified group. The permissions for modifying the groups are therefore required to execute the command.

Syntax:

`user withLogin` *login* `addToGroups {`*groupName*`}`

Function parameters:

- *groupName* is the name of an existing user group. Any number of groups can be specified.

Return value if successful: none

Necessary permissions:

- The logged-in user must be a super user or
- must have the `permissionGlobalUserEdit` permission and be direct or indirect owner (`owner`) of each group to which the user *login* should be added.

Example:

```
CM>user withLogin jane addToGroups editors
```

user userRef checkPassword

Available for: Content Management Server

Task: Checks whether the user with the login *login* has the specified password.

Syntax:

`user withLogin` *login* `checkPassword` *password*

Function parameters:

- *password*: A non-encrypted password.

Return value if successful: 1 if the encrypted password obtained from *password* matches the user's `encryptedPassword` , otherwise 0.

Necessary permissions: none

Example:

```
CM>user withLogin jane checkPassword za1WpM
0
```

user userRef delete

Available for: Content Management Server

Task: Deletes the user with the login *login*.

user was the owner of other users or groups, the root user is set as their new owner.</p>

Syntax:

`user withLogin login delete`

Function parameters: none

Return value if successful: none

Necessary permissions:

- The logged-in user must be a super user or
- have the permissionGlobalUserEdit permissions and be the direct or indirect owner of the *login* user.

Example:

```
CM>user withLogin jane delete
```

user userRef description

Available for: Content Management Server

Task: Returns a string representation of the data for the user with the *login* login.

Additional information: The representation is formatted in the property list format of a dictionary. The displayed fields are listed in the section The user parameters.

Syntax:

`user withLogin login description`

Function parameters: none

Return value if successful: the string representation of the user (string)

Necessary permissions: none

Example:

```
CM>user withLogin jane description
{
  defaultGroup = admins;
  globalPermissions = (
  );
  groups = (
    admins
  );
  login = jane;
  owner = root;
  realName = "Jane\ Mitchum";
```

```
  userLocked = 0;
}
```

user userRef get

Available for: Content Management Server

Task: Returns the value for the specified user parameter of the user with the *login* login.

Syntax:

`user withLogin` *login* `get` *paramName*

Function parameters:

- *paramName* specifies the name of the parameter whose value is being searched for (see The user parameters).

Return value if successful: the value of the entered user parameter (string)

Necessary permissions: none

Example:

```
CM>user withLogin jane get realName
Jane Mitchum
```

user userRef grantGlobalPermissions

Available for: Content Management Server

Task: Grants the specified permissions to the user with the *login* login.

Syntax:

`user withLogin` *login* `grantGlobalPermissions {`*permission*`}`

Function parameters:

- *permission*: The designation for a global permission. The global permissions which are permitted are given in the system configuration under the `content.globalPermissions` key.

Return value if successful: none

Necessary permissions:

- The logged-in user must be a super user or
- have the `permissionGlobalUserEdit` permission and be the direct or indirect `owner` of the *login* user.

Example:

```
CM>user withLogin jane grantGlobalPermissions permissionGlobalRoot
```

user userRef hasGlobalPermission

Available for: Content Management Server

Task: Checks whether the user with the *login* login has the specified global permission.

Additional information: Users have a specific permission if they have the permission permission, if they are super user or they are members of a group with the permission.

Syntax:

user withLogin *login* hasGlobalPermission *permission*

Function parameters:

- *permission* The designation for a global permission.

Return value if successful: 1 if the user has the specified permission, otherwise 0.

Necessary permissions: none

Example:

```
CM>user withLogin jane hasGlobalPermission permissionGlobalRTCEdit
1
```

user userRef isOwnerOf

Available for: Content Management Server

Task: Checks whether the user *login* is direct or indirect owner (owner) of the user with the login or group named *ownedLoginOrGroup* (see Owner of a user).

Syntax:

user withLogin *login* isOwnerOf *ownedLoginOrGroup*

Function parameters:

- *ownedLoginOrGroup* Login which is queried whether its owner is *login*.

Return value if successful: 1 if the user *login* is the direct or indirect owner of the user with the login or group named *ownedLoginOrGroup*, otherwise 0.

Necessary permissions: none

Example:

```
CM>user withLogin michael isOwnerOf jane
1
```

user userRef isSuperUser

Available for: Content Management Server

Task: Checks whether the user with the *login* login is a super user.

Additional information: A user is a super user if his/her login is "root", if he/she has the permissionGlobalRoot permissions or if he/she is member of a group which has this permission. This command has the same result as the user withLogin *login* hasGlobalPermission permissionGlobalRoot command.

Syntax:

```
user withLogin login isSuperUser
```

Function parameters: none

Return value if successful: 1 if the user is a super user, otherwise 0.

Necessary permissions: none

Example:

```
CM>user withLogin jane isSuperUser
1
```

user userRef mget

Available for: Content Management Server

Task: Returns the values for the specified user parameter of the user with the *login* login.

Syntax:

```
user withLogin login mget {paramName}
```

Function parameters:

- *paramName* specifies the name of the parameter whose value is being searched for.

Return value if successful: list of parameter values (stringlist)

Necessary permissions: none

Example:

```
CM>user withLogin jane mget defaultGroup userLocked
admins 0
```

user userRef removeFromGroups

Available for: Content Management Server

Task: Removes the user with the *login* login from the specified groups.

Additional information: A user cannot be removed from his default group.

Syntax:

`user withLogin login removeFromGroups {groupName}`

Function parameters:

- *groupName* Name of a group from which the user is to be removed.

Return value if successful: none

Necessary permissions:

- The logged-in user must be a super user or
- must have the `permissionGlobalUserEdit` permission and be direct or indirect `owner` of each group from which the user *login* should be removed.

Example:

```
CM>user withLogin jane removeFromGroups admins
```

user userRef revokeGlobalPermissions

Available for: Content Management Server

Task: Revokes the specified permissions of the user with the *login* login.

Syntax:

`user withLogin login revokeGlobalPermissions {permission}`

Function parameters:

- *permission* designates a global permission.

Return value if successful: none

Necessary permissions:

- The logged-in user must be a super user or
- have the `permissionGlobalUserEdit` permission and be the direct or indirect owner (`owner`) of the *login* user.

Example:

```
CM>user withLogin jane revokeGlobalPermissions permissionGlobalRTCEdit
```

user userRef set

Available for: Content Management Server

Task: Sets the specified parameters to the corresponding values for the user with the *login* login.

Additional information:

- If the `owner` parameter is explicitly set to the empty string, the logged-in user will be inserted as `owner`.

Syntax:

`user withLogin` *`login`* `set {`*`paramName value`*`}`

Function parameters:

- *`paramName`* specifies the name of the user parameter whose value is to be set (see The user parameters).
- *`value`* is the value to be set for the specified parameter.

Return value if successful: none

Necessary permissions:

- The logged-in user must be a super user or
- the logged-in user must have the `permissionGlobalUserEdit` permission and be the direct or indirect `owner` of the *`login`* user.
- If the logged-in user is not a super user and does not have the `permissionGlobalUserEdit` permission, he can only modify the values of his user fields and his own `defaultGroup`, `email`, `password`, and `realName` parameters. As a new `defaultGroup` he can only specify a group he is a member of.

Example:

```
CM>user withLogin jane set userLocked 0
```

4.19 userAttribute (User Fields)

4.19.1 User Field Parameters

Parameter	Type	Explanation	get	set	create	descr
displayTitle	string	The user field's title as it is displayed in the HTML user interface (corresponds to the name)	•			
getKeys	stringlist	List of parameters which can be queried with `get`	•			
name	string	Name of user field	•		•	•
setKeys	stringlist	List of parameters which can be set with `set`	•			
type	string	Type of user field (permitted: string (default), date, enum, multienum)	•	•	•	•
values	stringlist	Enumeration values (only with *enum* and *multienum* user fields)	•	•		•

4.19.2 User Field Commands

userAttribute create

Available for: Content Management Server

Task: The command creates a new custom user field.

Syntax:

`userAttribute create {`*`parameter value`*`}`

Function parameters:

- *`parameter`* is the name of a parameter whose value is set to *`value`* when the field is created. At least the name of the field must be specified.
- *`value`* is the value to be set for the parameter.

Return value if successful: the name of the user field (string)

Necessary permissions: The user must have the `permissionGlobalUserEdit` permission.

Example:

```
CM>userAttribute create name address
address
```

userAttribute list

Available for: Content Management Server

Task: Returns a list of the names of all custom user fields.

Syntax:

`userAttribute list`

Function parameters: none

Return value if successful: the list of user field names (stringlist)

Necessary permissions: no restrictions

Example:

```
CM>userAttribute list
address phone
```

userAttribute types

Available for: Content Management Server

Task: The command returns the list of the available user field types.

Syntax:

```
userAttribute types
```

Function parameters: none

Return value if successful: the list of user field types (stringlist)

Necessary permissions: no restrictions

Example:

```
CM>userField types
enum string date multienum
```

userAttribute where

Available for: Content Management Server

Task: Allows you to search for all user field names in which the value of the given parameter contains the specified string.

Syntax:

`userAttribute where {parameter value}`

Function parameters:

- *parameter* specifies the name of the parameter whose value is to be searched for or specifies the number of hits. The following parameters are allowed:
  - `maxResults` specifies that *value* is a number that limits the number of hits. If number is less than or equals zero, the number of hits is unlimited.
  - `name`: the names of the user fields are to be checked for occurrences of *value*.
- *value* contains the value of the respective parameter.

Return value if successful: the list of names from the matching user fields (stringlist)

Necessary permissions: no restrictions

Example:

```
CM>userAttribute where name phone
phone phone_car phone_home
```

userAttribute userAttrRef addEnumValues

Available for: Content Management Server

Task: Adds the specified enumeration values to an `enum` or `multienum` user field if they do not exist already.

Syntax:

`userAttribute withName attrName addEnumValues {enumValue}`

Function parameters:

- *enumValue*: indicates the enumeration value to be added to the existing enumeration values of the user field.

Return value if successful: none

Necessary permissions: The user must have the permissionGlobalUserEdit permission.

Example:

```
CM>userAttribute withName department addEnumValues \
administration development marketing services
```

userAttribute userAttrRef delete

Available for: Content Management Server

Task: The command deletes the specified user field.

Syntax:

userAttribute withName *attrName* delete

Function parameters: none

Return value if successful: none

Necessary permissions: The user must have the permissionGlobalUserEdit permission.

Example:

```
CM>userAttribute withName address delete
```

userAttribute userAttrRef description

Available for: Content Management Server

Task: Returns a string representation of the data from the user field with the specified name.

Additional information: the representation is formatted in the property list format of a dictionary (see The property list format).

Syntax:

userAttribute withName *attrName* description

Function parameters: none

Return value if successful: the string representation of the user field (string)

Necessary permissions: no restrictions

Example:

```
CM>userAttribute withName department description
{
  name = department;
  type = multienum;
  values = (
    administration,
    development,
    marketing,
    services
  );
}
```

userAttribute userAttrRef get

Available for: Content Management Server

Task: Returns the value of a user field parameter.

Syntax:

`userAttribute withName` *`attrName`* `get` *`parameter`*

Function parameters:

- *`parameter`* specifies the name of the parameter whose value is being searched for (see User field parameters).

Return value if successful: the value of the entered parameter (string)

Necessary permissions: no restrictions

Example:

```
CM>userAttribute withName department get type
multienum
```

userAttribute userAttrRef mget

Available for: Content Management Server

Task: Returns parameter values of a user field.

Syntax:

`userAttribute withName` *`attrName`* `mget {`*`parameter`*`}`

Function parameters:

- *`parameter`* specifies the name of the parameter whose value is being searched for (see User field parameters).

Return value if successful: the list of values for the parameters concerned (stringlist)

Necessary permissions: no restrictions

Example:

```
CM>userAttribute withName department mget type values
multienum {administration development marketing services}
```

userAttribute userAttrRef removeEnumValues

Available for: Content Management Server

Task: Deletes the specified enumeration values of an enum or multienum user field.

Syntax:

`userAttribute withName` *`attrName`* `removeEnumValues {`*`enumValue`*`}`

Function parameters:

- *`enumValue`* indicates the enumeration value to be deleted from the existing enumeration values of the user field.

Return value if successful: none

Necessary permissions: The user must have the `permissionGlobalUserEdit` permission.

Example:

```
CM>userAttribute withName department removeEnumValues sales services
```

userAttribute userAttrRef set

Available for: Content Management Server

Task: Sets parameter values for the user field called *`attrName`*.

Syntax:

`userAttribute withName` *`attrName`* `set {`*`parameter value`*`}`

Function parameters:

- *`parameter`*: specifies the parameter from the user field whose value is to be set (see User field parameters).
- *`value`* is the value to be set for the parameter.

Return value if successful: none

Necessary permissions: The user must have the `permissionGlobalUserEdit` permission.

Example:

```
CM>userAttribute withName department set type enum
```

4.20 userConfig (User Preferences)

4.20.1 userConfig

Available for: Content Management Server

Task: The `userConfig` command allows write and read access to the logged-in user's personal preferences. The defaults of these preferences are stored in the system configuration under the `userManagement.preferences` and `userManagement.preferences.guiPreferences` configuration elements. The changes a user makes to his preferences are permanently saved.

Additional information:

- The user-specific configuration values are used for the subcommands `parseInputDate` and `formatDateTime`.
- A super user can change any user's preferences using the `sudo` command.

Syntax:

`userConfig` *`command`*

Function parameters:

- `command` is a subcommand. The `userConfig` command has the same subcommands as the `systemConfig` command. Additionally, the subcommand `getAll` is available with the `userConfig` command. With this command the complete user-specific configuration element can be retrieved.

Example:

```
CM>userConfig getTexts timeZone language
MET en
```

4.21 userConfigForUser (Preferences of Other Users)

4.21.1 userConfigForUser

Available for: Content Management Server

Task: The `userConfigForUser` command works analogously to the `userConfig` command. However, while `userConfig` allows access to the personal preferences of the logged-in user, you can use the `userConfigForUser` command to read out and set the preferences of all user logins you own.

Syntax:

`userConfigForUser` *`login command`*

Function parameters:

- *`login`* is the login of the user whose personal preferences you want to read or set.
- *`command`* is a subcommand. The `userConfigForUser` command has the same subcommands as the `systemConfig` command.

Necessary permissions: You can use the command only to access your own preferences and those of the users whose logins you own. As a super user you implicitly own all user logins.

Example:

```
CM>userConfigForUser mustermann setElements guiPreferences.startArea
 <startArea>wizard_page</startArea>
CM>userConfigForUser mustermann getElements guiPreferences.startArea
{<?xml version="1.0" encoding="UTF-8"?>
<startArea>browse_page</startArea>}
```

4.22 workflow (Workflows)

4.22.1 Workflow Parameters

Parameter	Type	Explanation	get	set	create	descr
displayTitle	string	The workflow's title as it is displayed in the HTML user interface	•			
editGroups	stringlist	The groups in the edit workflow	•	•	•	•
name	string	Name of workflow	•		•	•
signatureDefs	stringlist	The signatures and entitled groups in the signing phase of the workflow. Each string is a 2-element list made up of a field name and group.	•	•	•	•
allowsMultiple Signatures	bool	Indicates whether one and the same person can set several signatures under one version	•	•	•	•
getKeys	stringlist	List of parameters which can be queried with `get`	•			
isEnabled	bool	Indicates whether new workflows of this type can be started	•	•	•	•
setKeys	stringlist	The list of parameters which can be set with `set`	•			
title	string	The title of the workflow in the user-specific language	•	•	•	•
title.*language*	string	The title of the workflow in the respective language	•	•	•	•

4.22.2 Workflow Commands

workflow create

Available for: Content Management Server

Task: This command creates a new workflow.

Syntax:

`workflow create {parameter value}`

Function parameters:

- *parameter* specifies the name of the workflow parameter whose value is to be set during the create process (see Workflow parameters).
- *value* is the value to be set for the workflow parameter.

Return value if successful: the name of the workflow (string)

Necessary permissions: The user must have the `permissionGlobalRTCEdit` permission.

Example:

```
CM>workflow create name wf_sig_news editGroups news_editors \
signatureDefs {{sig_news newsadmins}}
wf_sig_news
```

workflow list

Available for: Content Management Server

Task: Lists the names of all managed workflows.

Syntax:

`workflow list`

Function parameters: none

Return value if successful: the list of workflow names (stringlist)

Necessary permissions: no restrictions

Example:

```
CM>workflow list
wf_edit_news wf_sig_news
```

workflow where

Available for: Content Management Server

Task: Allows you to search for all workflow names which contain the specified string.

Syntax:

`workflow where {parameter value}`

Function parameters:

- *parameter* specifies the name of the parameter whose value is to be searched for or specifies the number of hits. The following parameters are allowed:

- `maxResults` specifies that *`value`* is a number that limits the number of hits. If number is less than or equals zero, the number of hits is unlimited.
- `name` specifies that *`value`* is the substring for which the workflow names are to be searched.

- *`value`* is the value of the corresponding parameter.

Return value if successful: the list of names from the matching workflows (stringlist)

Necessary permissions: no restrictions

Example:

```
CM>workflow where name news
wf_edit_news wf_sig_news
```

workflow workflowRef delete

Available for: Content Management Server

Task: The command deletes the specified workflow.

Syntax:

`workflow withName` *`workflowName`* `delete`

Function parameters: none

Return value if successful: none

Necessary permissions: The user must have the `permissionGlobalRTCEdit` permission.

Example:

```
CM>workflow withName wf_edit_news delete
```

workflow workflowRef description

Available for: Content Management Server

Task: Returns a string representation of the workflow with the specified name.

Additional information: The representation is formatted in the property list format of a dictionary (see The property list format).

Syntax:

`workflow withName` *`workflowName`* `description`

Function parameters: none

Return value if successful: the string representation of the workflow (string)

Necessary permissions: no restrictions

Example:

```
CM>workflow withName wf_sig_news description
{
  allowsMultipleSignatures = 1;
  editGroups = (
    editors
  );
  isEnabled = 1;
  signatureDefs = (
    (
      "sig_news",
      newsadmins
    )
  );
}
```

workflow workflowRef get

Available for: Content Management Server

Task: The command returns a parameter value from the workflow with the name *workflowName*.

Syntax:

`workflow withName` *workflowName* `get` *parameter*

Function parameters:

- *parameter* specifies the name of the parameter whose value is being searched for (see Workflow parameters).

Return value if successful: the value of the entered parameter (string)

Necessary permissions: no restrictions

Example:

```
CM>workflow withName wf_sig_news get isEnabled
1
```

workflow workflowRef mget

Available for: Content Management Server

Task: The command returns parameter values from the workflow with the name *workflowName*.

Syntax:

`workflow withName` *workflowName* `mget {`*parameter*`}`

Function parameters:

- *parameter* specifies the name of the parameter whose value is being searched for (see Workflow parameters).

Return value if successful: the list of values for the parameter concerned (stringlist)

Necessary permissions: no restrictions

Example:

```
CM>workflow withName wf_sig_news mget editGroups signatureDefs
editors {{sig_news newsadmins}}
```

workflow workflowRef set

Available for: Content Management Server

Task: Sets the specified parameter values of the workflow *workflowName*.

Syntax:

`workflow withName` *workflowName* `set {`*parameter value*`}`

Function parameters:

- *parameter* specifies the parameter from the workflow whose value is to be set (see Workflow parameters).
- *value* is the value to be set for the specified parameter.

Return value if successful: none

Necessary permissions: The user must have the `permissionGlobalRTCEdit` permission.

Example:

```
CM>workflow withName wf_sig_news set isEnabled 0 title {News Workflow}
```

4.23 Callback Functions

4.23.1 exportFillCallback

Available for: Template Engine

Task: Custom procedure which synchronizes the export directory hierarchy of the Template Engine with the corresponding directory hierarchy on the live server. The callback function can achieve this by executing `rsync`, for example.

Additional information: see Import and Export.

Function parameters:

- *offlineTree*: The path of the export directory hierarchy resulting from the symbolic link `offline` in the instance-specific `export` directory.
- *onlineTree*: The path of the online directory hierarchy resulting from the symbolic link `online` in the instance-specific `export` directory.

Return value if successful: none

4.23.2 exportSwitchCallback

Available for: Template Engine

Task: Custom procedure which swaps the export and live directory hierarchies on the live server via a secure connection, for example. The callback function might take additional actions, if required.

Additional information: see Import and Export.

Syntax:

`exportSwitchCallback` *`offlineTree onlineTree`*

Function parameters:

- *`offlineTree`*: The path of the export directory hierarchy resulting from the symbolic link `offline` in the instance-specific `export` directory.
- *`onlineTree`*: The path of the online directory hierarchy resulting from the symbolic link `online` in the instance-specific `export` directory.

Return value if successful: none

4.23.3 exportSyncCallback

Available for: Template Engine

Task: Custom procedure which updates the remote export directory hierarchy from the live directory hierarchy, i. e. synchronizes the two hierarchies.

Import and Export

Syntax:

`exportSyncCallback` *`offlineTree onlineTree`*

Function parameters:

- *`offlineTree`*: The path of the export directory hierarchy resulting from the symbolic link `offline` in the instance-specific `export` directory.
- *`onlineTree`*: The path of the online directory hierarchy resulting from the symbolic link `online` in the instance-specific `export` directory.

Return value if successful: none

4.23.4 importCallback

Available for: Template Engine from version 6.7.2

Task: Custom procedure for processing the import journal. In the course of processing the journal, other actions, such as tagging CMS files for export, can be taken (see the `obj touch` command).

Additional information: see Import and Export.

Syntax:

`importCallback` *`importJournal`*

Function parameters:

- *`importJournal`*: The path of the import journal file. The function must not modify the file.

Return value if successful: none

The Import Journal

The import journal is created during the import phase. It is a record of all actions relevant to CMS files. Each line in the file represents one journal entry. All lines have the following form:

```
Code FilePath
```

`Code` is a single character that specifies the kind of change made to a CMS file. *`FilePath`* is the CMS path of the file concerned. Lines that start with a hash mark (`#`) are comments.

The following changes are recorded in the import journal and are tagged with the codes specified:

- `c`: The file with the specified path was created.
- `u`: A file field (such as `suppressExport`) has been modified. This type is not recorded if the name of the file or of its parent file is changed. For these actions, the types `m` and `M` are recorded.
- `r`: The file was deleted.
- `m`: The file was moved (its path changed). The specified path is the previous path of the file. This entry type is always followed by an `M` entry.
- `M`: The file was moved (its path changed). The specified path is the new path of the file. This entry type is always preceded by an `m` entry.
- `U`: The contents of the file has changed (the file was released).
- `R`: The contents of the file has been deleted (the file was unreleased).

The import journal is emptied after the `importCB` phase has been finished successfully. If the import callback returns an error all journal entries are preserved. New entries will be appended to the journal.

Example of an import journal:

```
# This file is generated automatically.
# Only the TE is allowed to modify it.
u /
c /internet
c /internet/playland
c /internet/playland/de
c /internet/playland/de/wirueberuns
c /internet/playland/de/wirueberuns/faq
c /internet/playland/de/wirueberuns/faq/probiermodus
c /internet/playland/de/wirueberuns/faq/schadstofffreiheit
c /internet/playland/de/wirueberuns/faq/bestellung
c /internet/playland/de/wirueberuns/faq/verfuegbarkeit
U /
U /internet
U /internet/playland
U /internet/playland/de
U /internet/playland/de/wirueberuns
U /internet/playland/de/wirueberuns/faq
U /internet/playland/de/wirueberuns/faq/probiermodus
U /internet/playland/de/wirueberuns/faq/schadstofffreiheit
U /internet/playland/de/wirueberuns/faq/bestellung
U /internet/playland/de/wirueberuns/faq/verfuegbarkeit
```

4.24 Other Administrative Commands

4.24.1 app closeDbConnection

Available for: Content Management Server, Template Engine

Task: Ends the connection to the database.

Syntax:

`app closeDbConnection`

Function parameters: none

Return value if successful: none

Necessary permissions: This command cannot be related to a CMS user's permissions as it is executed before a user logs himself/herself in.

Additional information: This command can be used only on application start and in single mode.

Example:

```
app closeDbConnection
```

4.24.2 app export

Available for: Template Engine

Task: This command switches the Template Engine from the import phase to the export phase.

Additional information:

- The command is also available in the Tcl client and is executed asynchronously, i. e. it returns immediately.
- If the command must not be executed during the export phase.
- After the export-fill phase the Template Engine returns to the import phase (see also Functions of the Template Engine).

Syntax:

`app export`

Function parameters: none

Return value if successful: none

4.24.3 app get

Available for: Content Management Server, Template Engine, Search Engine Server

Task: Returns the value of the specified parameter. This command is also available in the Tcl-Client.

Syntax:

`app get` *`parameter`*

Function parameters:

- *`parameter`*: the name of the parameter whose value is to be determined. Valid names are:
  - `phase` (TE only): the partial phase the Template Engine is in. For an explanation of the phases see Functions of the Template Engine.
  - `phaseDetails` (TE only): in addition to the current partial phase and for debugging purposes, this returns detailed information about the progress of the phase. Depending on the phase (see the `phase` parameter), the return value is structured as follows:
    - `import {objects` *`objectCount recursiveObjectCount`*`}`:
      `objectCount` specifies the number of files that need to be exported. When the import phase begins, this value reflects the number of files that could not be exported during the most recent export phase. *`recursiveObjectCount`* specifies the number of files whose subhierarchy must also be exported. When the import phase begins, *`recursiveObjectCount`* is 0. During the import phase both values can only increase.
    - `export-init {objects` *`objectCount recursiveObjectCount`*`}`:
      The values have the same meaning as for the `import` phase. At the end of the `export-init` phase, *`recursiveObjectCount`* is 0.
    - `export-fill {`*`sliceType count percentage`*`}` ...:
      `sliceType` is the type of a section from the list of files to be exported. For each of the following types, a sublist is inserted into the resulting list: `available`, `inProgress`, `complete`, `failed`, `total`. The identifiers have the following meaning:
      `count`: The current number of sections of this type. If *`sliceType`* is `total`, the total number of sections is output.
      `percentage`: specifies the percentage of all sections of this type. For the `total` type the percentage is always 100.
    - For the remaining phases, `phaseDetails` only returns the phase identifier.
  - `today`: the current date (14 places), 0.00 h, in GMT.
  - `version`: the version string that can also be queried using the `-version` command line parameter.
  - `appName`: the application's name.
  - `rootConfigPath`: the path of the main configuration file *nps.xml*.
  - `timeZone`: the time zone resulting from the machine's time.
  - `baseDir`: The NPS installation directory.
  - `binDir`: the directory below the instance directory in which the start scripts are located.
  - `commonScriptDir`: the absolute path of the instance-specific *script/common* directory.
  - `configDir`: the directory below the instance directory in which the configuration files are located.
  - `dataDir`: the directory below the instance directory in which data such as blobs and streaming tickets are located.
  - `instanceDir`: the instance directory.
  - `libDir`: the directory below the installation directory in which libraries and executable files are located.

- `logDir`: the directory below the instance directory in which the log files are located.
- `scriptDir`: the directory below the instance directory in which the scripts are located. The *cm* directory is used both by the Content Manager and the Template Engine, the *ses* directory is used by the Search Engine Server.
- `shareDir`: the directory below the installation directory containing files used by all instances, for example the documentation.
- `tmpDir`: the directory below the installation directory in which temporary files such as PID files are located.

Return value if successful:

- `phase` and `rank`: the value of the parameter (string).
- `phaseDetails`: the details for the respective phase (stringlist).
- `timeZone`: the time zone (string).
- `*Dir`: the absolute path of the directory (string).

Necessary permissions (CM only): no restrictions

4.24.4 app makeDbConnection

Available for: Content Management Server, Template Engine

Task: Sets up a connection to the database.

Syntax:

```
app makeDbConnection
```

Function parameters: none

Return value if successful: none

Necessary permissions: This command cannot be related to a CMS user's permissions as it is executed before a user logs himself/herself in.

Additional information: This command can be used only on application start and in single mode.

4.24.5 app publish

Available for: Template Engine

Task: This command initiates a complete export cycle.

Additional information:

- The command is also available in the Tcl client and is executed asynchronously, i. e. it returns immediately.
- The command can be used to resume the export after an error has occurred. See also Functions of the Template Engine.

Syntax:

```
app publish
```

Function parameters: none

Return value if successful: none

4.24.6 clearUsermanCache

Available for: Content Management Server

Task: The command deletes all cached user data.

Additional information: The command should be used if an external user manager is connected to the Content Manager (via the `usermanAPI.tcl` interface file) and user data in this user manager have been modified. The deleted cache is subsequently filled with current user data as soon as such data are requested.

Syntax:

```
clearUsermanCache
```

Function parameters: none

Return value if successful: none

Necessary permissions: no restrictions.

4.24.7 decodeData

Available for: Content Management Server, Template Engine, Search Engine Server

Task: Decodes the specified base64-encoded binary string.

Syntax:

```
decodeData encodedString
```

Function parameters:

- *encodedString* is the base64-encoded string.

Return value if successful: the decoded string.

Necessary permissions (CM only): no restrictions

4.24.8 decodeFile

Available for: Content Management Server, Template Engine, Search Engine Server

Task: Decodes the specified base64-encoded string and writes it in the specified file.

Syntax:

```
decodeFile filename encodedString
```

Function parameters:

- *filename* is the name of the file.

- *encodedString* is the base64-encoded string.

Return value if successful: 1

Necessary permissions (CM only): no restrictions

4.24.9 decodeToString

Available for: Content Management Server, Template Engine, Search Engine Server

Task: Decodes the specified base64-encoded UTF-8 string.

Syntax:

`decodeToString` *encodedString*

Function parameters:

- *encodedString* is the base64-encoded string.

Return value if successful: the decoded string.

Necessary permissions (CM only): no restrictions

4.24.10 encodeData

Available for: Content Management Server, Template Engine, Search Engine Server

Task: base64-encodes a string.

Syntax:

`encodeData` *stringToEncode*

Function parameters:

- *stringToEncode* is the string to encode.

Return value if successful: the base64-encoded string.

Necessary permissions (CM only): no restrictions

4.24.11 encodeFile

Available for: Content Management Server, Template Engine, Search Engine Server

Task: Loads the specified file and returns its contents as a base64-encoded string.

Syntax:

`encodeFile` *filename*

Function parameters:

- *filename* is the name of the file.

Return value if successful: the base64-encoded contents of the file (string)

Necessary permissions (CM only): no restrictions

4.24.12 filterTags

Available for: Content Management Server, Template Engine

Task: The command removes unwanted tags and tag attributes from a main content.

Syntax:

`filterTags blob` *`blob mode taglist`*

Function parameters:

- *`blob`* is the main content to be filtered.
- *`mode`* determines how the main content will be processed. *`mode`* can be one of the following keywords:
  - `allowedTags`: the *`taglist`* list contains the names of the tags allowed to occur in the main content. All other tags will be removed.
  - `disallowedTags`: the *`taglist`* list contains the names of the tags to be removed from the main content. All other tags will remain in the main content.
- *`taglist`* contains the list of tags that are to be removed from or are to remain in the main content (depending on *`mode`*). An element of *`taglist`* can again be a list consisting of two elements. The first element in this list will be interpreted as a tag name, the second as a list of attributes of this tag. If *`mode`* equals `allowedTags` then the attributes specified in this list will be left untouched while all others are removed from the tag. If *`mode`* equals `disallowedTags` then the attributes specified will be removed from the tag.

Return value if successful: the filtered main content.

Necessary permissions (CM only): none.

Example:

```
filterTags blob $oldBlob allowedTags {p br img}
```

4.24.13 getRegisteredCommands

Available for: Content Management Server, Template Engine

Task: Returns the commands registered on the NPS server. In standard installations, this procedure is called by a Tcl client at start-up to retrieve the names of all procedures that the Tcl client needs to forward to the Tcl server for execution.

Syntax:

`getRegisteredCommands`

Function parameters: none

Return value if successful: the list of registered commands (stringlist)

Necessary permissions: no restrictions

4.24.14 indexAllObjects

Available for: Content Management Server

Task: All versions of all files are reindexed. This procedure can only be used if the Search Engine Server has been integrated into the system and is running.

Syntax:

`indexAllObjects`

Function parameters: none.

Return value if successful: none.

Necessary permissions: Up to version 6.7.0, the user must have the `permissionRoot` permission for all files, which is normally only true for super users. From version 6.7.1, the user must be a superuser.

Example:

```
CM>indexAllObjects
 done.
```

4.24.15 listTasks

Available for: Content Management Server

Task: The procedure returns information about tasks. The tasks can be filtered by criteria.

Syntax:

`listTasks {`*`parameter value`*`}`

Function parameters:

- *`parameter`* and *`value`*: see `task where`.

Return value if successful: List of tasks specifying the task type, the task owner, the path of the file concerned as well as the time stamp. The abbreviations have the following meaning: `E` = Edit, `S` = Sign, `F` = FixLink. The abbreviations in the owner column signify the following: `G` = Group and `U` = User.

Necessary permissions (CM only): Only the IDs of files are returned for which the current user has the `permissionRead` permission.

Example:

```
listTasks taskType edit
 T  Owner    Path                              Time
E U root     /internet/index.html              2010090709
E U root     /internet/incoming/index.html     2010090710
2 tasks
```

4.24.16 listTasksOfGroup

Available for: Content Management Server

Task: The procedure returns information about a group's tasks. The tasks can be filtered by criteria.

Syntax:

value}</code></p>

Function parameters:

- *groupName*: the name of the user group whose tasks are to be output.
- *parameter* and *value*: see task where.

Return value if successful: List of tasks specifying the task type, the task owner, the path of the file concerned as well as the time stamp. The abbreviations have the following meaning: E = Edit, S = Sign, F = FixLink. The abbreviations in the owner column signify the following: G = Group and U = User.

Necessary permissions: Only the IDs of files are returned for which the current user has the permissionRead permission.

Example:

```
CM>listTasksOfGroup redakteure taskType edit
 T  Owner          Path                              Time
E G webeditors     /internet/de/index.html           2010090709
1 task
```

4.24.17 listTasksOfUser

Available for: Content Management Server

Task: The procedure returns information about a user's tasks. The tasks can be filtered by criteria.

Syntax:

listTasksOfUser login {*parameter value*}

Function parameters:

- *login*: the login name of the user.
- *parameter* and *value*: see task where.

Return value if successful: List of tasks specifying the task type, the task owner, the path of the file concerned as well as the time stamp. The abbreviations have the following meaning: E = Edit, S = Sign, F = FixLink. The abbreviations in the owner column signify the following: G = Group and U = User.

Necessary permissions: Only the IDs of files are returned for which the current user has the *permissionRead* permission.

Example:

```
CM>listTasksOfUser redakteur taskType edit
```

```
 T  Owner         Path                              Time
E U webeditor     /internet/de/index.html           2010090709
1 task
```

4.24.18 loadFile

Available for: Content Management Server, Template Engine

Task: Returns the contents of the specified file.

Syntax:

`loadFile` *`filename`*

Function parameters:

- *`filename`* is the name of the file.

Return value if successful: the contents of the file (binary)

Necessary permissions (CM only): no restrictions

```
CM>set a [loadFile /tmp/filename]
```

4.24.19 loadTextFile

Available for: Content Management Server, Template Engine

Task: Returns the contents of the specified text file.

Syntax:

`loadTextFile` *`filename`* [*`charset`*]

Function parameters:

- *`filename`* is the name of the file.
- *`charset`* is the character encoding name of the file to read. This parameter is optional. The default value is `utf-8`. Enter `encoding names` in the Tcl shell to retrieve the list of known character encodings.

Return value if successful: the contents of the file

Necessary permissions (CM only): no restrictions

Example:

```
CM>set a [loadTextFile /tmp/textfile iso8859-1]
```

4.24.20 logMessage

Available for: Content Management Server, Template Engine

Task: This command writes a message to the application-specific log file if its level of detail equals or is less than the level that has been set in the system configuration.

Syntax:

`logMessage` *`level message`*

Function parameters:

- *`level`* is an integer number between 0 and 3 (both inclusive) which specifies the detail level of the message (3 = most detailed). You can specify in the `log` system configuration entry how detailed the messages may be that are written to the log file.
- *`message`* is the message to be logged.

Return value if successful: none.

Necessary permissions (CM only): no restrictions

```
TE>>logMessage 1 {The export was started}
```

4.24.21 logWarning

Available for: Content Management Server, Template Engine

Task: This command writes a warning message to the application specific log file.

Syntax:

`logWarning` *`message`*

Function parameters:

- *`message`* is the warning to be logged.

Return value if successful: none.

Necessary permissions (CM only): no restrictions

```
CM>logWarning {This is the message}
```

4.24.22 logout

Available for: Content Management Server

Task: Terminates a connection with the Content Management Server.

Syntax:

`logout [login]`

Function parameters:

- *`login`* is the login name of the user whose connection is to be terminated. You need to be a superuser or the owner of the user to be logged-out in order to be able to log-out a user other than yourself. If the parameter has not been specified, the current connection will be terminated.

Return value if successful: none

Necessary permissions: see above, function parameter `login`.

Example:

```
CM>logout smith
```

4.24.23 news where

Available for: Content Management Server, Template Engine

Task: This command returns a list of news file IDs. The IDs are sorted by date in descending order.

Syntax:

`news where [channels` *`channellist`*`] [length` *`length`*`]`

Function parameters:

- *`channellist`* is an optional list of the channels to take into account.
- *`length`* optionally specifies the maximum length of the list..

Return value if successful: a list of file IDs.

Necessary permissions (only CM): no restrictions

Example:

```
news where channels {politics sports} length 20
```

4.24.24 now

Available for: Content Management Server, Template Engine

Task: Returns the current time in the 14-place canonical format.

Syntax:

`now`

Function parameters: none

Return value if successful: the current time (string)

Necessary permissions (CM only): no restrictions

```
CM>now
20110325163159
```

4.24.25 stream downloadBase64

Available for: Content Management Server, Search Engine Server

under a ticket ID in the Content Management Server as a base64-encoded string.</p>

Additional information: The command can only be executed in `-single` mode or from within wizards.

Syntax:

`stream downloadBase64` *`ticketId`*

Function parameters:

- *`ticketId`* is a ticket ID that was returned by one of the commands `stream uploadFile` or `stream uploadBase64` , for example.

Return value if successful: the base64-encoded data.

Necessary permissions: no restrictions

```
CM>stream downloadBase64 2098
PCFET0NUWVBFIEhUTUwgUFVCTElDICItL...
```

4.24.26 stream downloadFile

Available for: Content Management Server, Search Engine Server

Task: The command saves the data stored under a ticket ID in the Content Management Server in a file.

Additional information: The command can only be executed in `-single` mode or from within wizards.

Syntax:

`stream downloadFile` *`ticketId path`*

Function parameters:

- *`ticketId`* is a ticket ID that was returned by one of the commands `stream uploadFile` or `stream uploadBase64` , for example.
- *`path`* is the path under which the data is to be saved.

Return value if successful: none.

Erforderliche Rechte: no restrictions.

Beispiel:

```
CM>stream downloadFile 2098 /tmp/testfile
```

4.24.27 stream uploadBase64

Available for: Content Management Server, Search Engine Server

Task: The command decodes a base64-encoded string and streams it to the Content Management Server so that other applications or scripts can fetch it from there.

Additional information: The command can only be executed in `-single` mode or from within wizards.

Syntax:

`stream uploadBase64` *`base64EncodedString`*

Function parameters:

- *`base64EncodedString`* is the encoded character string.

Return value if successful: the streaming ticket ID with which the file can be fetched.

Necessary permissions: no restrictions

Example:

```
stream uploadBase64 [encodeFile /tmp/test]
2098
```

4.24.28 stream uploadFile

Available for: Content Management Server, Search Engine Server

Task: This command streams the contents of a file to the Content Management Server so that other applications or scripts can fetch them from there.

Additional information: The command can be called only in `-single` mode or from within wizards.

Syntax:

`stream uploadFile` *`path`*

Function parameters:

- *`path`* is the path of the file to be streamed to the Content Management Server.

Return value if successful: the streaming ticket ID with which the file can be fetched.

Necessary permissions: no restrictions

Example:

```
CM>stream uploadFile /tmp/test
2097
```

4.24.29 sudo

Available for: Content Management Server

Task: The command allows a super user to execute a command under a different login.

Syntax:

`sudo` *`login command`* {*`parameter`*}

Function parameters:

- *`login`* is the login under which the command should be executed.
- *`command`* is the command to be executed. This command must have been registered with safeInterp alias.
- *`parameter`* is a parameter which should be passed to the command.

Return value if successful: none

Necessary permissions: The user must be a super user in order to execute the command.

Example:

```
CM>sudo editor obj withPath /internet/playland forward
```

4.24.30 today

Available for: Content Management Server, Template Engine

Task: Returns the current date in the 14-place canonical format.

Additional information: The value for the hours corresponds to 0 hours on the client.

Syntax:

`today`

Function parameters: none

Return value if successful: the current date (string)

Necessary permissions: no restrictions

Example:

```
CM>today
20110308230000
CM>systemConfig formatDateTime [today]
09.03.2011 00:00
```

4.24.31 valueforLocalizerKey

Available for: Content Management Server, Template Engine

Task: Retrieves a string from the `localizers` system configuration entry, localized in the user's language.

Additional information: The `localizers` key has been removed from the system configuration in Infopark CMS Fiona 6. In wizards, strings can be localized by means of variables. See the wizards supplied with the CMS.

Syntax:

`valueForLocalizerKey` *`key`* [*`category`*]

Function parameters:

- *`key`* is the name of the key whose language-specific value is to be returned.
- *`category`* is the name of a category contained in the `localizers` element.

Return value if successful: the localized string.

Necessary permissions (CM only): no restrictions.

Example:

```
CM>valueForLocalizerKey myKey myCategory
The value of myKey in my language from myCategory
```

4.24.32 whoami

Available for: Content Management Server

Task: Returns the login of the current user.

Syntax:

`whoami`

Function parameters: none

Return value if successful: the login of the current user (string)

Necessary permissions: no restrictions

Example:

```
CM>whoami
editor
```

4.24.33 writeFile

Available for: Content Management Server, Template Engine

Task: Stores data to a file. The data can be binary, i. e. contain null characters, for example.

Syntax:

`writeFile` *`filename data`*

Function parameters:

- *`filename`* is the name of the file.
- *`data`* is the data to be stored.

Return value if successful: none

Necessary permissions: no restrictions

Example:

```
CM>writeFile /tmp/test [obj withId 2001 get exportBlob]
```

4.24.34 writeTextFile

Available for: Content Management Server, Template Engine

Task: Stores text data to a file.

Syntax:

`writeTextFile` *`filename data`* [*`charset`*]

Function parameters:

- *`filename`* is the name of the file.
- *`data`* is the data to be stored.
- *`charset`* specifies the name of the character encoding in which the data is to be stored. This parameter is optional. Its default value is `utf-8`.

Return value if successful: none

Necessary permissions (CM only): no restrictions

Example:

```
writeTextFile /tmp/text {This is an example}
```

4.25 Additional Tcl Procedures

A series of useful Tcl procedures are defined in the `clientCmds.tcl` file which the Content Manager runs when starting. Their functions are described in this section.

4.25.1 contentService

Available for: Content Management Server

Task: The procedure makes it possible to use the ContentService interface of the Content Management Server in Tcl scripts. It transmits the contents of a CMS folder to a different CMS instance.

Syntax:

`contentService` *`sourceUrl sourceLogin sourcePassword sourcePath destUrl destLogin destPassword destPath`*

Function parameters:

- *`sourceUrl`*: The URL of the source instance.
- *`sourceLogin`*, *`sourcePassword`*: The login credentials of the source instance user used to fetch the source CMS files.
- *`sourcePath`*: The path of the source instance folder to be transmitted.
- *`destUrl`*: The URL of the destination instance.
- *`destLogin`*, *`destPassword`*: The login credentials of the user used to modify or create the CMS files in the destination instance.
- *`destPath`*: The path of the destination instance folder that is to receive the CMS files (including *`sourcePath`*).

Return value if successful: none

Necessary permissions: The source instance user requires read permission for all the files to be transmitted. The destination instance user requires write permission for these CMS files in case they already exist. To be able to create CMS files not yet present in the destination instance, this user requires the file creation permission in the respective parent folder.

Example:

```
contentService
    http://localhost:8080/sourceInstance myUser myPassword /news
    http://otherServer:8080/destinationInstance otherUser otherPassword /intranet/content
```

4.25.2 cp

Available for: Content Management Server

Task: The procedure copies a file into a folder. If the file being copied is a folder, its files will not be copied.

Syntax:

`cp` *`sourceObjIdOrPath destObjIdOrPath`*

Function parameters:

- *`sourceObjIdOrPath`* is the ID of the source file or its path. The path must be absolute, i.e. start with a slash.

- *destobjIdOrPath* is the ID of the target folder or its path. The path must be absolute, i.e. start with a slash.

Return value if successful: none

Necessary permissions: The user must have read permission for the source files and have permission to create files in the target folder.

Example:

```
CM>cp /internet/presse /resources
```

4.25.3 findObjectId

Available for: Content Management Server

Task: The procedure determines the id of a file if it exists.

Syntax:

`findObjectId` *idOrPath*

Function parameters:

- *idOrPath* may be the id or path of a file.

Return value if successful: the id of the file (string).

Necessary permissions: none.

Example:

```
CM>findObjectId /myFolder
2145
```

4.25.4 grep

Available for: Content Management Server

Task: The procedure determines from a list of CMS files the ones that contain the given string in a particular field. Additionally, the state of the files to be found can be restricted.

Syntax:

`grep` *expression objects* [*attribute*] [*args*]

Function parameters:

- *expression*: the regular expression to apply
- *objects*: the list of CMS files to be searched
- *attribute*: the field to be searched. This parameter is optional. The default is `blob`
- *args*: optional specification of additional restrictions. *args* can be:

- `-states` *`state`*: This restricts the results set to CMS files of a particular state or a selection of states. *`state`* can be one of:
  - A list containing one or more of the following values: `edited`, `committed`, `released`, `archived`.
  - `all` (the default) stands for any state.
  - `active` stands for any state except `archived`.

Return value if successful: for each file found its path, details about the location where the search term matched, and part of the text containing the search term (string). The location is specified by means of the file ID (`o`), the version ID (`c`), and the line number (`l`). See below for an example.

Necessary permissions: The user must have the read permission for the files that were found.

Example:

```
CM>grep "modifyvar" [obj where condition {objType is template}]
/mastertemplate (o:2011,c:2015,l:4) <npsobj modifyvar="set"
varname="contentTypeAndCharset">text/html;
charset=<npsobj insertvalue="var" name="exportCharset" /></npsobj>
```

4.25.5 incrNpsDate

Available for: Content Management Server

Task: The procedure adds a period of time to a date given in canonical format.

Syntax:

`incrNpsDate` *`date offset`*

Function parameters:

- *`date`* is the date as a sequence of 14 digits.
- *`offset`* is a string containing a relative time specification which consists of an integer number followed by one of the following words: `year`, `fortnight`, `month`, `week`, `day`, `hour`, `minute` (or `min`), `second` (or `sec`). The word can be specified as a singular or plural.

Return value if successful: the calculated absolute date in canonical form (string).

Necessary permissions: none.

Example:

```
CM>incrNpsDate 20110402091300 "-3 months"
20110102091300
```

4.25.6 interactiveLoadSubtree

Available for: Content Management Server

Task: The procedure has the same function as loadSubtree. If the file format can not be determined from the file name extension, the procedure will ask for the missing information. Furthermore, users can specify whether their input should be applied to all files with the file name extension concerned.

Syntax:

`interactiveLoadSubtree` *`parent startDir startName pubObjClass`* [*`mapping`*]

4.25.7 l

Available for: Content Management Server

Task: The procedure returns a table with detailed information on the files in a folder. The file ID, format and type as well as the file name and title are listed.

Syntax:

`l` *`objIdOrPath`*

Function parameters:

- *`objIdOrPath`* is the ID of the folder or its path. The path must be absolute, i.e. start with a slash.

Return value if successful: string

Necessary permissions: Only files for which the user has read permission will be returned.

Beispiel:

```
CM>l /
/ (2001)  total (2)
objId   class               type        name                title
-----------------------------------------------------------------
2006    template            template    mastertemplate      mastertemplate
2050    document            document    test
```

4.25.8 listObjectsWithoutSuperLinks

Available for: Content Management Server

Task: With folders the procedure returns the IDs of all files it contains and which are no link destination files (i. e. to which no links are pointing).

Syntax:

`listObjectsWithoutSuperLinks` *`objIdOrPath`*

Function parameters:

- *`objIdOrPath`* is the ID of the folder or its path. The path must be absolute, i.e. start with a slash.

Return value if successful: List of file IDs

Necessary permissions: The user must have the `permissionRead` permission for the folder. Only the IDs of the files for which the user has the `permissionRead` permission will be returned.

Example:

```
CM>listObjectsWithoutSuperLinks /
objects without superlinks in path /
/test {Test}
/mastertemplate {mastertemplate}
```

4.25.9 listSubtree

Available for: Content Management Server

Task: The procedure returns the IDs of all files which are located in the hierarchy branch of a folder, including the folder itself.

Syntax:

`listSubtree` *`objIdOrPath`*

Function parameters:

- *`objIdOrPath`* is the ID of the folder or its path. The path must be absolute, i.e. start with a slash.

Return value if successful: List of file IDs

Necessary permissions: The user must have the `permissionRead` permission for the folder. Only the IDs of the files for which the user has `permissionRead` permission will be returned.

Example:

```
CM>listSubtree /
2001 2050 2006
```

4.25.10 loadSubtree

Available for: Content Management Server

Task: The procedure imports the files in a directory into the Content Manager. Subdirectories are taken into account recursively.

Additional information: A folder called *`startName`* will be created below the *`parent`* folder. Into this *`startName`* folder, the content is imported. A folder with the same name as the directory will be created for each further directory to be imported. If there is a file named `index.html` in a directory, it will be imported as the main content of the corresponding created folder. The procedure will finally return a list of the files that cannot be imported (because the format could not be clearly determined).

Syntax:

`loadSubtree` *`parent startDir startName pubObjClass`* [*`mapping`*]

Function parameters:

- *`parent`* is the ID of the folder in which the subhierarchy is created.
- *`startDir`* is the path of the directory which is to be imported.

- *startName* is the name of the starting folder.
- *objClass* is the name of the format on which the created folders are to be based.
- *mapping* is a list with assignments. Each setting is a list containing the following parameters (in the order given here):
  - *contentType*: the file name extension for which a format is set.
  - *objClass*: the format on which the files are to be based which are created for *contentType* type files.

Return value if successful: none

Necessary permissions: The user must have the permissionCreateChildren permission for the starting folder. In some cases further permissions are required, for example, permission to use a file format.

Example:

```
CM>loadSubtree /internet /tmp/import press press_folder mapping {{html press_doc} {pdf
 pdf_doc}}
```

4.25.11 ls

Verfügbar für: Content Management Server

Task: The procedure returns a table with information on the files in a folder.

Syntax:

ls *objIdOrPath*

Function parameters:

- *objIdOrPath* is the ID of the folder or its path. The path must be absolute, i.e. start with a slash.

Return value if successful: table string

Necessary permissions: Only files are listed for which the user has read permission.

Example:

```
CM>ls /
/ (2001)  total (2)
objId   class               type        name
---------------------------------------------
2006    template            template    mastertemplate
2050    document            document    test
```

4.25.12 mkpubs

Available for: Content Management Server

Task: The procedure creates a folder from the *objClass* format for each element in the specified path.

Syntax:

`mkpubs` *`path objClass`*

Function parameters:

- *`path`* is the path which is to be created. The path must be absolute.
- *`objClass`* is the format on which the created folders are to be based.

Return value if successful: none

Necessary permissions: The user must have the `permissionCreateChildren` permission for the folders in which files are to be created.

Example:

```
CM>mkpubs /internet/playland/de publication
```

4.25.13 modifyPermissions

Available for: Content Management Server

Task: The procedure modifies access permissions for the specified files and groups.

Additional information: Depending on the *`modifier`*, permissions will be granted to or revoked from the specified groups.

Syntax:

`modifyPermissions` *`objIdsOrPaths modifier groups permissions`*

Function parameters:

- *`objIdsOrPaths`* is the list of files to which the access permissions being modified refer. The list can contain both file IDs and paths. Paths must be absolute, i.e. start with a slash.
- *`modifier`* is one of the following modifiers:
  - `grantTo`: The *permissions* permissions will be granted to the groups.
  - `set`: the system first revokes all file-specific permissions from the group. The *`permissions`* permissions are then granted to the groups.
  - `revokeFrom`: The system revokes the *`permissions`* permissions from the groups.
- *`groups`* is the list of groups for which the access permissions are to be changed.
- *`permissions`* is the list of file access permissions (such as `permissionRead`).

Return value if successful: none

Necessary permissions: The user must have the `permissionRoot` permission for the specified files.

Example:

```
CM>modifyPermissions [obj list] set {admins rootusers} permissionRoot
modifying permissions
done.
```

4.25.14 objWherePath

Available for: Content Management Server

Task: This procedure lists the IDs of all files that can be found in the specified path, i. e. files whose path begins with the path of the specified file. The result can be restricted to particular file types and formats.

Additional information: The command only returns files that have an ID which is not the case with implicit mirror files (created automatically by the system).

Syntax:

`objWherePath` *`idOrPath`* {*`parameter value`*}

Function parameters:

- *`parameter`* is one of the following optional values:
  - `objTypes`: determines that *`value`* contains a list of file types to which the result list is to be restricted.
  - `objClasses`: determines that *`value`* contains a list of file formats to which the result list is to be restricted.
- *`value`* contains the value of the respective parameter.

Additional information: it is possible but makes no sense to specify both parameters since they are combined with AND and the file format already determines the file type. If `objClasses` is specified, implicit mirror files are not taken into account.

Return value if successful: The list of the IDs of the files contained in the specified path and matching the specified parameter values.

Necessary permissions: Only files are listed to which the user has read access.

Example:

```
CM>objWherePath /internet objClasses {publication}
2108 2116 2124
```

4.25.15 performWithEditedContentOfObjects

Available for: Content Management Server

Task: The procedure executes `obj withId` *`id`* `editedContent` *`command`* for each file in the specified *`objIds`* list. If a file does not have a draft version, it is quietly skipped.

Syntax:

`performWithEditedContentOfObjects` *`command objIds`*

Function parameters:

- *command* is one of the available content commands, including the parameters to be passed to this command.
- *objIds* is the list of files for which the command is to be executed.

Return value if successful: none

Necessary permissions: Only the IDs of the files for which the user has the permissionRead permission will be processed. Further permissions may be required to execute the specified command.

Example:

```
CM>performWithEditedContentOfObjects {set validFrom [now]} [obj list]
performing set validFrom [now] with edited Content
done.
```

4.25.16 performWithObjects

Available for: Content Management Server

Task: The procedure executes the obj withId *id command* for each file in the specified *objIds* list.

Syntax:

performWithObjects *command objIds*

Function parameters:

- *command* is one of the available obj commands, including the parameters to be passed to it.
- *objIds* is the list of files for which the *command* command is to be executed.

Return value if successful: none

Necessary permissions: Only the IDs of the files for which the user has permissionRead permission will be processed. Further permissions may be required to execute the specified command.

Example:

```
CM>performWithObjects {set suppressExport 0} {2108 2116 2124}
performing set suppressExport 0
done.
```

4.25.17 removeBlobAndFreeLinks

Available for: Content Management Server

Task: The procedure deletes the draft version and the released version of a file as well as all the links contained in these versions.

Syntax:

removeBlobAndFreeLinks *objId*

Function parameters:

- *objId* is the ID of the file.

Return value if successful: none

Necessary permissions: The user must have the permissionRoot permission for the file.

Example:

```
CM>removeBlobAndFreeLinks [obj withPath /internet get id]
```

4.25.18 removeSubtree

Verfügbar für: Content Management Server

Task: The procedure deletes the specified folder and all the files it contains.

Syntax:

```
removeSubtree [-verbose {0|1}] [-check {0|1}] [-removeExternalMirrors {0|1}] [-
force {0|1}] objectIdOrPath
```

Function parameters:

- *-verbose* determines whether progress information is to be output (1, default) or not (0). If this command is issued via the Tcl client and *verbose* is 0, the command is executed in the server.
- *-check* determines whether access permissions and links are to be checked (1, default) or not (0). The default has the effect that prior to command execution all files are checked for links pointing to them and for sufficient user permissions. Command execution continues only if these conditions are met. If 0 has been specified, this check is not performed, meaning that the subhierarchy might not be completely deleted.
- *-removeExternalMirrors* (from version 6.5.0 PP1) causes mirror files outside the subtree (originating from files inside the subtree) to be deleted as well.
- *-force* forces the command execution to continue (1) even if deleting files causes errors. Errors can occur if files to be deleted are link targets or if the required permission is missing. The default (0) causes the command to terminate on errors.
- *objIdOrPath* is the ID of the folder or its path. The path must be absolute, i.e. start with a slash.

Return value if successful: none

Necessary permissions: The user must have the permissionRoot permission for the folder and all of its files. Otherwise the files concerned cannot be deleted.

Example:

```
CM>removeSubtree -verbose 1 -check 0  -force 1 /internet
```

4.25.19 renameObjClass

Verfügbar für: Content Management Server (from Version 6.6.1)

Aufgabe: This procedure renames a file format and reindexes the files to which this format is assigned. Furthermore, the command outputs the list of files that contain this format name. These files are not modified.

Syntax:

`renameObjClass` *`oldName newName`*

Function parameters:

- *`oldName`* is the current name of the file format.
- *`newName`* is the new name of the file format.

Return value if successful: none.

Necessary permissions: The user must have the `permissionGlobalRTCEdit` permission.

Example:

```
root@localhost:3001 default(CM)> renameObjClass publication Folder

Updating index
-------------------------------------------------------------------------------
performing updateInIndex
done.

Searching for config files that contain references to the original object class name
 'publication'.
-------------------------------------------------------------------------------
/Fiona/instance/default/config/content.xml:28  <defaultPublication>publication</
defaultPublication>
/Fiona/instance/default/config/webDav.xml:4    via WebDAV. Default is "publication". -->
/Fiona/instance/default/config/webDav.xml:5    <defaultPublicationClass>publication</
defaultPublicationClass>
...

Searching for templates that contain references to the original object class name
 'publication'.
-------------------------------------------------------------------------------
/mastertemplate (o:2010,c:2084,l:2) <npsobj caseCond="isEqual" value="publication">


Commands to search again for occurrences
-------------------------------------------------------------------------------
Config files:
  grepInConfig {\ypublication\y}

Templates:
  grepInTemplates {\ypublication\y}
```

4.25.20 rm

Available for: Content Management Server

Task: The procedure deletes a file. If the file is a folder, it can only be deleted if it does not contain any files.

Additional information: The command is based on the `obj delete` command whose restrictions apply here.

Syntax:

`rm objIdOrPath`

Function parameters:

- *objIdOrPath* is the ID of the file or its path. The path must be absolute, i.e. start with a slash.

Return value if successful: none

Necessary permissions: The user must have the `permissionRoot` permission for the file to be deleted.

Example:

```
CM>rm /cats
```

4.25.21 showLinksOfObjects

Available for: Content Management Server

Task: The procedure returns a list of the links contained in each object. Only files which contain links will be taken into consideration.

Syntax:

`showLinksOfObjects objIdsOrPaths [selector]`

Function parameters:

- *objIdsOrPaths* is the list of files which are to be checked. The list can contain both file IDs and paths. The path must be absolute, i.e. it must start with a slash.
- *selector* is one of the following selectors:
  - `all`: all links will be taken into consideration.
  - `dead`: only the links which have not been resolved will be taken into consideration.

Return value if successful: the list of the files containing links. For each link, its ID and its destination is output.

Necessary permissions: Only the IDs of the files for which the user has `permissionRead` permission will be considered.

Example:

```
CM>showLinksOfObjects [obj list] all
showing links
/ (2001)
link 2193 /test.html
done.
```

4.25.22 showPermissions

Available for: Content Management Server

Task: The procedure returns for all files the access permissions and the groups to which these permissions are assigned.

Syntax:

`showPermissions` *`objIdsOrPaths`*

Function parameters:

- *`objIdsOrPaths`* is the list of files which are to be checked. The list can contain both file IDs and paths. Paths must be absolute, i.e. must start with a slash.

Return value if successful: none

Necessary permissions: The user must have the `permissionRead` permission for the specified files.

Example:

```
CM>showPermissions {/ 2050}
/ (2001)
        permissionRoot: rootusers admins
/test (2050)
        permissionRead: admins
        permissionWrite: admins
        permissionRoot: rootusers admins
```

4.25.23 streamFromServer

Available for: Content Management Server

Task: Reads the data stored under a streaming ticket ID from the specified server and stores it in a file.

Additional information:

- This routine uses the streaming interface which allows large pieces of data to be transferred in a memory-saving manner.
- The maximum size of a piece of data transferred does not exceed 64 KB. This does not mean that the amount of data that can be transferred with this routine is limited to this value.

Syntax:

`streamFromServer` *`host port ticketId fileName`*

Function parameters:

- *`host`* is the host name or IP address of the server.
- *`port`* is the port at which the data can be accessed.
- *`ticketId`* is the ID previously returned by the server after the data have been transferred to it.
- *`fileName`* is the name or path of the file to which the data are stored.

Return value if successful: none

Necessary permissions: No restrictions.

Example:

```
CM>streamFromServer localhost 3002 [obj withId 53891 \
editedContent get blob.stream] ~/myDocument.pdf
```

4.25.24 streamToServer

Available for: Content Management Server

Task: Sends the contents of a file to a server and returns a ticket ID for reference.

information:</p>

- This routine uses the streaming interface which allows large pieces of data to be transferred in a memory-saving manner.
- The maximum size of a piece of data transferred does not exceed 64 KB. This does not mean that the amount of data that can be transferred with this routine is limited to this value.
- The validity of the ticket ID is limited to the time specified in the *streamingTicketValidityTimeInterval* system configuration entry.

Syntax:

`streamToServer` *`host port fileName`*

Function parameters:

- *`host`* is the host name or IP address of the server.
- *`port`* is the port to which the data is sent.
- *`fileName`* is the name or path of the file whose contents is to be sent to the server.

Return value if successful: The ticket ID under which the data can be accessed (string).

Necessary permissions: No restrictions.

Example:

```
CM>obj withId 53891 editedContent set blob.stream \
[streamToServer localhost 3002 ~/myDocument.pdf]
```

5

5 Solving Standard Tasks Using Tcl

5.1 Copying File Formats

The following procedure, `copyObjClass`, creates a copy of a file format by reading the `setKeys` of the source and writing them into the new file format. The procedure has two arguments, the names of the source and destination file formats (in this order).

```
# $Id: copyObjClass.tcl, v. 0.9 2010-11-23 13:51:46
#
# (c) 2010 Infopark AG
# Last changed: 2010-11-23
# Use at your own risk!
#
# Purpose:
#   Creates a new file fromat from an existing one
# Parameters:
#   Name of source file format,
#   Name of file format to create

proc copyObjClass {sourceClassName destClassName} {
  set sourceClassType [objClass withName $sourceClassName get objType]

  set setKeys [objClass withName $sourceClassName get setKeys]
  set nameIndex [lsearch $setKeys {name}]
  set setKeys [lreplace $setKeys $nameIndex $nameIndex]
  set values [eval [list objClass withName $sourceClassName mget] $setKeys]

  set setValues [list]
  foreach key $setKeys value $values {
    lappend setValues $key $value
  }

  eval [list objClass create name $destClassName objType $sourceClassType] $setValues
}
```

Please save the script to a single Tcl file. Open a Tcl shell, connect to the Content Manager, and read the Tcl file using `source`. You can now use the procedure. Example:

```
copyObjClass articleDocument newsDocument
```

5.2 Copying Partial Hierarchies

The following two procedures, `copySubtree` and `copySubtreeAndLinks`, copy a folder hierarchy to a different location. After copying, `copySubtreeAndLinks` (in contrast to `copySubtree`) adapts the links pointing to files in the original hierarchy so that they point to the new, copied files.

```
# $Id: copySubtree.tcl,v 1.2 2002/03/27 15:51:06
#
# (c) 2001-2009 Infopark AG
# Last changed: 2009-09-25
# Purpose:
#   copySubtreeAndLinks: Copy a file tree and re-generate the links
#   copySubtree: Simple tree-copying without linkage
# Parameters:
#   File ID or name of the folder to copy,
#   File ID or name of the destination folder
#   [keepStatus]
# keepStatus (optional) tries to assign the same status to the copied files
# as they had before and prints a warning if a file has a released and a
# draft/committed version. In this case only the released version will be copied.

proc copySubtreeAndLinks {{objId ""} {destination ""} {option ""} } {
  if {$destination==""} {
    global copySubtree_usage
    puts $copySubtree_usage
    return
  }
  if {($option!="") && ($option!="keepStatus")} {
    puts "$option is a not allowed option!"
    return
  }
  set to_release 0
  set objId [findObjectId $objId]
  set destination [findObjectId $destination]
  set oldPath [obj withId $objId get path]
  set newPath [obj withId $destination get path]
  set newPath $newPath/[obj withId $objId get name]
  if {[catch {findObjectId $newPath}] == 0} {
    set newPath [obj withId [obj withId $destination create objClass \
      [obj withId $objId get objClass] name [obj withId $objId get name]] get path]
    obj withPath $newPath delete
  }
  puts "Copying files ..."
  copySubtree $objId $destination $option
  puts "\nSubtree successfully copied, now links will be adapted ..."
  set newId [obj withPath $newPath get id]
  foreach i [listSubtree $newId] {
     if !{[obj withId $i get isMirror]} {
        if {[obj withId $i get isReleased] == 1} {
          obj withId $i unrelease
          set to_release 1
        }
        foreach j [obj withId $i get subLinks] {
          set dest [link withId $j get destinationUrl]
          if {[regexp "^$oldPath" $dest]} {
            regsub $oldPath $dest $newPath dest
            link withId $j set destinationUrl $dest
          }
        }
        if {$to_release == 1} {
          obj withId $i release
          set to_release 0
        }
     }
  }
  puts "done"
}


proc copySubtree {{objId ""} {destination ""} {option ""} } {
  if {$destination==""} {
    global copySubtree_usage
    puts $copySubtree_usage
    return
  }
  set objId [findObjectId $objId]
  set destination [findObjectId $destination]
```

```
  if {$objId == $destination} {
    puts "You can't copy a file to itself"
    return
  }
  set newId [obj withId $objId copy parent $destination]
  puts -nonewline "\r$objId"
  flush stdout
  if {$option=="keepStatus"} {
    if {[obj withId $objId get isCommitted]=="1"} {
      obj withId $newId commit
    }
    if {[obj withId $objId get isReleased]=="1"} {
      obj withId $newId release
      if {([obj withId $objId get isCommitted]=="1") ||
          ([obj withId $objId get isEdited]=="1")} {
        puts "\nFile $objId has a released and a draft version.\
          Only the released version was copied"
      }
    }
  }
  if {[obj withId $objId get objType] == "publication" &&
      ![obj withId $objId get isMirror]} {
    foreach i [obj withId $objId get children] {
      copySubtree $i $newId $option
    }
  }
}
```

Please save both scripts to a single Tcl file. Open a Tcl shell, connect to the Content Manager, and read the Tcl file using `source`. You can now use both procedures. The following example completely copies the folder `/en/news` to `/internet/en/news`:

```
copySubtreeAndLinks /en/news /internet/en
```

5.3 Deleting Old Versions and Log Entries

Old archived versions of files often use up a lot of space and might reduce the performance of the CMS. By means of the following script these versions can be deleted. Furthermore, the script removes the log entries associated with the deleted versions.

Please store the script in a file named `cutHistory.tcl`. Afterwards, you can read it into the CMS by means of the Tcl client and execute the `cutHistory` procedure:

```
CM>source /path/to/cutHistory.tcl
CM>cutHistory File_ID Remaining_Entries
```

Here is the script:

```
# (c) 2001 Infopark AG
# Modified: 2010-04-13
# cutHistory.tcl for Fiona 6.x
#
# Purpose:
# Reduce (cut) the number of archived (i.e. previously released) versions
# to the given number. Furthermore, remove the associated log enries.
#
# Parameters:
# - File ID or path
# - Number of released versions to remain in the system
```

```
proc cutHistory {id cutLength} {
    set objectId [findObjectId $id]
    # CMS Fiona 6.5 or later: exclude mirrors
    if {[obj withId $objectId get isMirror] eq "1"} {
        puts "\nMirror files ($objectId) don't have versions..."
    } else {
        set contents [lsort -integer -decreasing \
            [obj withId $objectId get archivedContentIds]]
        set versionLength [llength $contents]
        set logLength [llength [logEntry where logTypes action_release_obj \
            objectId $objectId]]
        puts "Cut by Length:\t$cutLength"
        puts "Log-Size:\t$logLength"
        puts "Versions:\t$versionLength"
        if {[catch {set cutByDate [content withId [lindex $contents $cutLength] \
            get lastChanged]}]} {
            puts "\nFile $objectId has no archived versions..."
        } else {
            foreach i [lrange $contents $cutLength $versionLength] {
                content withId $i delete
                puts "content $i deleted"
            }
            if {[catch {logEntry deleteWhere objectId $objectId \
                    untilDate $cutByDate} msg]} {
                puts "\nCouldn't delete log entries of file $objectId: \n$msg"
            }
        }
    }
}
```

5.4 Finding Files

The Content Navigator, i.e. the web interface of Infopark CMS Fiona features a search form for comfortably finding a particular file. If you only need to locate files via Tcl, and searching their names for the occurrence of a string is sufficient, a small Tcl script will do.

All the script needs to do is determine the names of all files in a partial hierarchy, compare the names with the search string, and output the paths of the files found:

```
proc locate {objId searchString} {
  puts "Getting Ids..."
  set counter 0
  set objects [listSubtree [findObjectId $objId]]
  foreach obj $objects {
    set name [obj withId $obj get name]
    if {[string match *$searchString* $name] == 1} {
      puts "[obj withId $obj get path]"
      incr counter
    }
  }
  puts "\n$counter files found"
}
```

Save this procedure in a script file, for example in `locate.tcl`. Then start the Tcl client of the Content Manager and make the `locate` procedure contained in the script file available using `source locate.tcl`. You can now run the procedure:

```
CM>locate /internet/news business
```

5.5 Moving Files

Using Tcl, individual files or complete subtrees can easily be moved. Just set the `parent` field of the file concerned to the ID of the folder to which the file is to be moved:

```
CM>obj withId 4592 set parent 2001
```

This command moves the file with the ID 4592 to the folder with the ID 2001 (which is the base folder). Of course, you can also specify paths instead of IDs:

```
CM>obj withPath /internet/news set parent [obj root get id]
```

As with creating new files, the restrictions of the target folder apply when files are moved:

- The file format of the file to be moved must be a valid format for files contained in the target folder.
- The user who performs the action needs to have the administration permission for the file to be moved.
- In the target folder, the user needs to have the permission to create files.

5.6 Using Tcl Scripts in More than One Callback Function

Sometimes, the same Tcl procedures need to be used in several callback functions. For not having to maintain identical code at different locations, you can place it in a file which is sourced when the Content Manager is started. For this, proceed as follows:

1. Place a file containing the procedure into a directory whose content is automatically sourced when the Content Manager is started, for example

   `instance/`*`myInstance`*`/script/cm/serverCmds/callbackCommand.tcl`

2. All the arguments passed to the callback function or functions should also be passed to the procedure. Furthermore, the procedure is required to return the values that are relevant to the calling functions. The following code is a sample completion check:

   ```
   proc callbackCommand {contentId} {
     set messages {"Message"}
     set result 0
     return "$messages $result"
   }
   ```

3. Finally, the procedure needs to be registered in the safe interpreter:

   ```
   safeInterp alias callbackCommand callbackCommand
   ```

4. Restart the Content Manager for the script file to be sourced and the procedure to be made available:

   ```
   instance/myInstance/bin/rc.npsd restart CM
   ```

5. You can now call the procedure in the functions concerned and make use of the return value in the desired way:

```
CM> objClass withName document get completionCheck
set returnValue [callbackCommand $contentId]
set messages [lindex $returnValue 0]
set result [lindex $returnValue 1]
```

If a procedure is called from within functions that expect different return values, make the procedure return a list containing all the required return values. In the calling functions extract the required value from the list using `lindex`.
If the functions calling the procedure need to pass different arguments to the procedure, use an additional argument that identifies the function from which the procedure is called. Then make the calculations in the procedure dependent on the value of this additional argument.

5.7 Converting Timestamps

The following Tcl procedure converts a 14-place CMS date to a Unix timestamp:

```
# $Id: nps2unixDate.tcl,v 1.5 2001/12/19 16:09:28 benjamin Exp $
#
# (c) 2001 Infopark AG
# nps2unixDate
#
# Purpose:
#   Converts a CMS timestring (e.g. 20060101123000) to
#   a Unix timestamp (seconds since 01.01.1970).
#
# Parameters:
#   CMS timestring (format: YYYYMMDDHHmmSS)

proc nps2unixDate {timeStamp} {
    # String must contain 14 digits.
    # regular expressions rule!
    if {[regexp "^(1|2)(0|9)\\d\\d(0|1)\\d(0|1|2|3)\\d(0|1|2)\\d\[0-5\]\\d\[0-5\]\\d\$"
 $timeStamp] == "0"} {
        error "nps2unixDate: Parameter is not a valid CMS timestamp"
    }
    set year   [string range $timeStamp 0 3]
    set month  [string range $timeStamp 4 5]
    set day    [string range $timeStamp 6 7]
    set hour   [string range $timeStamp 8 9]
    set minute [string range $timeStamp 10 11]
    set second [string range $timeStamp 12 13]
    set unixTime [clock scan "$month/$day/$year $hour:$minute:$second" -gmt true]
    return $unixTime
}
```

The inverse conversion can be performed using the following procedure:

```
# $Id: unix2npsDate.tcl,v 1.3 2001/12/19 16:08:28 benjamin Exp $
#
# (c) 2001 Infopark AG
# unix2npsDate
#
# Purpose:
#   Converts a Unix timestamp (seconds since 01.01.1970) to
#   a CMS timestring (e.g. 20060519143000).
#
# Parameters:
#   Unix timestring
```

```
proc unix2npsDate {timeStamp} {
    # The string must not contain any non-digits
    if {[regexp "\[^\\d\]" $timeStamp]} {
        error "unix2npsDate: Parameter is not a valid unix timestamp"
    }
    # NOTE
    # The string must not exceed 2.000.000.000 which is
    # Wed, May 18 05:33:20 MET DST 2033
    # Actually, tcl is able to scan timestamps beyond this date
    if {$timeStamp > "2000000000"} {
        error "unix2npsDate: Parameter is too large"
    }
    return [clock format $timeStamp -format "%Y%m%d%H%M%S"]
}
```

Please save both scripts to a Tcl file. Open a Tcl shell, connect to the Content Manager, and read the Tcl file using `source`. You can now use the procedures. Example:

```
nps2unixDate 20110519130500
```

6

6 Functions of the Template Engine

The Template Engine plays a central role on the live system. Its task is to export content it receives from the Content Management Server, i. e. to generate web pages by applying layout files (templates). If the system includes the Verity Search Cartridge, its indexes are updated at the same time. If installed, the Portal Manager is supplied with information about document access restrictions, channels and news.

The Template Engine and the Search Engine Server work with a second directory hierarchy which is switched offline. This hierarchy is switched live by the Template Engine at the command of the Content Manager. Details about the directory hierarchies can be found in section The Directory Hierarchies.

6.1 Import and Export

The Template Engine works in phases. It alternately imports information about CMS file versions from the Content Management Server (import phase) and generates web documents which are then switched online (export phase). The export phase is divided into several subphases. The tasks of the individual phases are described in the following:

- `import`: The Template Engine receives so-called update records from the Content Manager, i. e. information about updated files. If the system includes the NPS Portal Manager, the Template Engine also receives update records relating to channels and news articles.
- `importCB` (from Version 6.7.2): The Template Engine executes a callback function that has access to the journal of import actions and may trigger additional actions.
- `export-init`: The export requests resulting from the update records are processed and grouped. The grouping makes it possible to have several slaves export files.
- `export-fill`: The export requests are executed which means that files are exported to the directory hierarchy switched offline. This phase (and therefore the export procedure as a whole) is aborted, if a configurable number of files could not be exported (system configuration entry `tuning.export.acceptableFailures`). If the system includes the Verity Cartridge, the generated documents are indexed in parallel.
- `export-fillCB`: The Template Engine executes a callback function with which the updated directory hierarchy can be replicated as desired.
- `export-switch`: The updated directory hierarchy is switched live by swapping the targets of two symbolic links pointing to the online and offline hierarchies.
- `export-switchCB`: The Template Engine executes another function with which the targets of the symbolic links can be swapped on other webserver systems as well.
- `export-sync`: The directory hierarchy now switched offline is updated.

- `export-syncCB`: This function allows you to update a remote directory hierarchy which is currently switched offline.
- `export-sync-finished`: The journal of changed files which the Template Engine creates and processes during the `export-fill` and `export-sync` phases, respectively, is rotated. This ends the export phase, and the Template Engine returns to the import phase.

The Template Engine enters the export phase, when it receives a corresponding instruction from the Content Management Server or one of the `app publish` or `app export` Tcl commands is executed. While `app publish` is used to run a complete export and publication cycle or to resume an interrupted cycle, `app export` exports the files and then returns to the import phase. Since the Template Engine exports files incrementally, `app export` can be used to reduce the publication process (`app publish`) to swapping and synchronizing the directories, hence speeding it up. Only errors should make it necessary to use these commands since the server normally receives equivalent instructions as update records from the Content Manager by means of system jobs. Please refer to the Administration/Layout manual for details about these jobs.

If one of the export phases up to and including `export-switch` fails, the Template Engine returns to the import phase. If, on the other hand, one of the next phases fails, the error state persists. In this case it is absolutely necessary to remove the error cause first. Then the export procedure can be resumed with `app publish`, i. e. the directory hierarchies can be swapped and the necessary synchronization processes can be performed. If the cause of the error can not be eliminated (because, for example, a remote machine is unreachable), the functions can be disabled, if necessary, by commenting out their code (it is then required to restart the server).

The Callback Functions

The Template Engine normally delivers content located on the system on which it is installed, meaning that it exports the files into a local, offline directory hierarchy which is then switched online. Finally, the hierarchy previously online is synchronized with the hierarchy which is currently online so that the next export is based on the current data.

With distributed systems, these processes can, by means of callback functions, also be performed on one or more remote machines. The functions call the custom procedures described below. Please note that the functions need to indicate an error using the `error` command. If a callback function failed and is then called again, the result must be the same as if it had been successful upon the first call. Callback functions must not modify the contents of the export directory tree.

The callback functions need to be defined as server commands in a Tcl script file which must be placed in the instance-specific `script/cm/serverCmds` directory.

Phase importCB (from version 6.7.2), callback function *importCallback*

As update records are being imported, the Template Engine creates a journal of the changes made to CMS files. This callback function can process the journal and trigger additional actions depending on the file modifications.

Phase export-fillCB, callback function *exportFillCallback*

Prior to executing the callback function, the Template Engine finishes the export into the offline directory. After execution, an equally complete offline directory hierarchy must exist on the remote machine. It must be possible and safe to execute the callback function several times.

Phase export-switchCB, callback function *exportSwitchCallback*

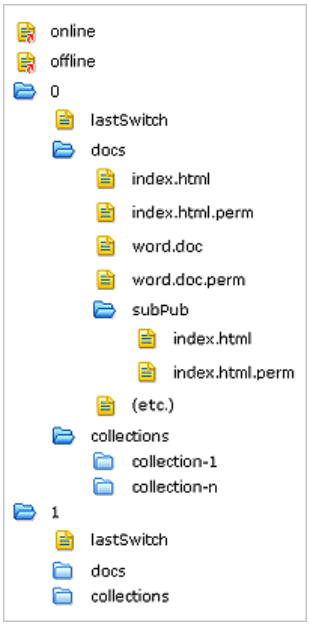
The task of this function is to swap the online and offline directory hierarchies on a remote machine. When the callback function is called, the local online directory hierarchy contains the new data and the offline hierarchy contains the data previously online. Furthermore, the remote machine must be in the state in which it was after the `exportFillCallback` was successfully executed, i. e. the offline hierarchy contains the new data, and the online hierarchy the old data. After successful execution, there must exist an online and an offline directory hierarchy which have been swapped so that their versions correspond to the respective versions on the local system. In the case of an error, the remote directory hierarchies must not have been swapped.

Phase export-syncCB, callback function *exportSyncCallback*

The task of this callback is to update the offline hierarchy on the remote machine so that it corresponds to the online hierarchy. Prior to calling the function, the Template Engine synchronizes the local hierarchies. The callback function is only executed if the `exportSwitchCallback` was executed successfully, i. e. if the current data is online and the outdated data is offline. After the function has been successfully executed, both remote hierarchies must have the same versions.

6.2 The Directory Hierarchies

The Template Engine stores all data relevant to the live system in a directory hierarchy. The starting point of this hierarchy is the instance-specific directory named `export`. The hierarchy has the following structure:



`online` and `offline` are symbolic links, of which one points to the directory `0` and the other one to `1`. The Template Engine swapps the targets of these links in order to activate the hierarchy currently switched offline and vice versa.

The Template Engine places the web documents generated during the export process in the `docs` directory. At the same time it stores the strings which specify the live server read permissions (`permissionLiveServerRead`) of the documents as files of the same name plus `.perm` in the same directory, so that the Portal Manager can access them as soon as the hierarchy has been switched live. If the Verity Cartridge is included in the system, `collections` contains the data collections needed for performing searches on the live server. The Template Engine instructs the Search Engine Server to index the documents stored in the `docs` directory into these collections.

The file named `lastSwitch`, finally, serves to record the most recent point in time at which a hierarchy was activated. By checking this file, applications can determine whether the contents of the corresponding directory hierarchy has changed.

6.3 Dependencies

One of the most important tasks of the Template Engine is to optimize the export of CMS files. On the one hand, using the Template Engine saves the resources of the machine concerned. On the other hand websites can be published faster.

The largest potential for optimization can be found in the export process itself since a file only needs to be exported if the web document that corresponds to it has changed. For being able to predict this, the Template Engine records dependencies between files. These dependencies are explained in this section.

A file not only needs to be exported again after it has been modified. If, for example, a file includes data from another file (by means of links or NPSOBJ instructions) it depends on this other file and needs to be exported even if only parts of the included file have changed.

The Template Engine records such dependencies between the CMS files while it exports them. Since the first export of a folder hierarchy is a complete export, it is guaranteed that all the existing dependencies are recorded. Later on, when the data is updated by means of update records, the Template Engine uses its recordings to check and then delete files that have become obsolete. When the folder hierarchy is finally published, the Template Engine generates the documents that need to be updated.

Infopark CMS Fiona recognizes the following dependency types. The details are listed in the right column. The file on which another file depends is called destination file.

Dependency type	Details
`object (1)`	**Description**: The file depends on the name or the position of the destination file in the folder hierarchy. This is also the case if the name is a component in a link destination path. **Trigger**: Parameter `name, objectsToRoot, object, parent, visibleName/visiblePath` (generates a dependency for all `objectsToRoot` of a link destination file). **Effects**: Files associated with the file are deleted, if `parent` or `name` of the destination file is modified or the destination file is deleted.
`content (2)`	**Description**: The file depends on fields of a different file (other fields than the ones metioned for object, reference, and children dependencies). **Trigger**: Querying parameters, that do not generate one of the dependencies above. **Effects**: When the destination file is released, the file is removed from the cache.
`reference (3)`	**Description**: The file only depends on data of the destination file that rarely change, such as data needed for links and tables of contents. When the destination file is released again and this data remains unchanged and no other

dependencies (especially `content` dependencies) are generated, the exported files are not removed from the cache.

Trigger: Querying the state of the destination file. Reference parameters: `title`, `contentType`, `objClass` for files, `destinationUrl`, `displayTitle` for links plus the fields listed in the `export.referenceAttributes` system configuration entry (do not specify fields of the `linklist` type here).

Effects: The file is only deleted, if the value of a reference parameter has changed.

`children (4)`

Description: The file depends on the list of the (released) files contained in a folder.

Trigger: Querying `children`, `prev`, `next`.

Effects: The files associated with the file are deleted, when files contained in a folder are released (again) or unreleased or lose their released version by some other action.

`usesAll (5)`

Description: The file depends on numerous or an undefined number of files.

Trigger: automatically when `usesAllThreshold` dependencies is exceeded or one of the following parameters is queried: `superObjects`, `superLinks`, `hasSuperLinks`.

Effects: Files in the file system associated with the NPS file are deleted when any data of the Template Engine changes.

During the export, dependencies can also be generated in `systemExecute` procedures. This might be necessary if the procedure reads fields from files for which no dependency has been or will be created.

6.4 Speeding up the Export Process

What Slows down the Export

Especially on large websites, changing a file sometimes has the effect that the Template Engine considers many other files obsolete because they depend on the file that was modified. As a consequence, the export process takes more time because many files have to be exported again.

The following actions considerably slow down the export:

- Changes to layout files. They cause the partial hierarchy, in which the layout is effective, to become obsolete and to be exported again. This is the case even if the layout files are not used during the export.
- Renaming folders. This also has the effect that all files in this partial hierarchy and additionally all files containing links pointing to a file in this hierarchy need to be exported again since the path of the files contained in the folder has changed.

The impact of the following actions is weaker:

- Changes to the file name extension. This causes all files containing links to the file in question to be exported again.
- Changing the size of images. This also causes all files containing links to the file concerned to be exported again because the links potentially contain image dimensions.
- Changes to the `suppressExport` field. References (such as links) to the file concerned are made visible or not by the export process, depending on the value of this field. Changing its value therefore causes the files containing the references to be invalidated.
- If files are created or deleted, all files containing tables of contents of the folders concerned become obsolete.
- Changes to most of the version fields invalidate the file itself as well as all other files from which field values are read while they are exported.

Approaches for Reducing the Export Time

If you include content statically using `includetext`, it is recommended to include the content dynamically instead using `insertvalue = "dynamiclink"`. Furthermore, the keys `hasSuperLinks`, `superLinks`, and `superObjects` should be used with caution in layouts because they have the effect that all other files become dependent on the queried file.

Thus, the time needed for an export does not result from the number of files alone but also from the complexity of the layout structure and the calculation expense produced for each exported file. This calculation expense mainly results from the frequency and the type of the queries contained in the layouts. For example, it is much more time-consuming to generate file or link lists than to query individual field values such as `title`.

Analyzing Dependencies

Whether dependencies are the cause of exports running slowly can be determined in the following way.

- In the Content Manager, execute the command `incrExport reset` and transfer your content to the Template Engine using `job withName systemTransferUpdates exec`.

  Attention: While this command is executed, update records must not be generated. The incrExport reset command must only be executed in single mode and only if it is guaranteed that no write access to the data takes place. Otherwise the exported data will be incomplete or destroyed. Therefore, please make sure that the CM is not running as a server and no other CMs are executed in single mode while this command is being executed.
- After all update records have been transferred, execute the command `app export` in the Template Engine.
- Immediately after the export has completed, check the `usesAll` dependencies that were generated. You can use the command `statistics compute name all` for this and analyze the Template Engine's log file (each time a dependency of this type is generated, a warning is written to the log).
- Try to determine why the files concerned depend on many other files. For this purpose, the value of the system configuration entry `tuning.export.usesAllThreshold` can be increased (the Template Engine must be restarted after that) and afterwards some of the files concerned can be updated (`obj touch` and `obj updateCache`). If, as a result, no `usesAll` dependencies are generated, `obj withId` *`id`* `get dependencies` will show the IDs of the files on which the files concerned depend.

If you would like to determine the layout file and the NPSOBJ instruction that caused the dependency, first increase in the `server.log.logger` system configuration entry named `info` the value for the `level` attribute to 3. Then restart the Template Engine. Update one of the files concerned using `obj withPath` *`path`* `updateCache`. In the `info.log` you can now find the export details mentioned above for this file.

To reduce the number of dependencies, please proceed as described above, i. e. reduce the complexity of your layout structure and the calculation expense in the layout files.

7

7 Error Messages

In this section error messages are explained whose cause is sometimes difficult to determine.

Error code	Explanation
10005	The number of response elements does not match the number of request elements in the request.
20005	The logged-in user needs to have the `permissionGlobalUserEdit` permission to set a user's or a group's owner.
20006	The logged-in user is not allowed to add users to or remove users from groups because he does not have the `permissionGlobalUserEdit` permission.
20007	The logged-in user is not allowed to grant global or file-specific permissions because he does not have the permissions required for this action (`permissionGlobalRoot` or `permissionRoot`, respectively).
20008	The logged-in user is not allowed to revoke global or file-specific permissions because he does not have the permissions required for this action (`permissionGlobalRoot` or `permissionRoot`, respectively).
20009	The logged-in user does not have the required permission (`permissionGlobalUserEdit`) to lock or unlock another user.
20012	The user is not permitted to perform the transaction.
20013	The logged-in user must have the `permissionGlobalExport` permission in order to use the `debugExport` version function or to access one of the version parameters `encodedExportBlob`, `exportBlob`, or `exportFiles`.
20014	The logged-in user is not allowed to read the version parameters because he does not have the `permissionRead` permission for the file to which the version belongs.
20015	The logged-in user is not allowed to read the parameters of the file because he does not have the `permissionRead` permission for the file.
20016	The user does not have the necessary permissions to create a file from the specified file format in this folder. These permissions are the global creation permission of the format and the file creation permission in the folder concerned.
20020	The logged-in user has tried to release a file's draft version without being permitted to do so.
20021	The logged-in user has tried to cut off the workflow of a file and start a new one without being permitted to do so.
33008	The value of the specified identifier is no URL and cannot be converted to an URL.

36001	On execution of a custom command, the configured command name associated with it could not be found.
36024	This message is related to the Portal Manager. In a command expecting exactly one of the named values, none of these values has been specified.
36025	This message is related to the Portal Manager. In a command expecting exactly one of the named values, none of these values has been specified.
40005	The user is not the file administrator and therefore has to be the editor in order to perform the workflow action.
40009	The user has already signed the file. He cannot sign a second time as he is not the file administrator and the workflow does not allow multiple signatures.
40010	The file cannot be committed or signed. It can only be released.
40011	The `commit` operation cannot be carried out as the file has not yet been edited by all groups in the editing workflow. Only the file administrator and a super user have permission to still submit or commit the file.
40012	The file cannot be committed or released as the draft version is incomplete.
40013	The user is neither the administrator of the file nor a super user; therefore, she cannot release the file prematurely.
40014	The user is neither the administrator of the file nor a super user. Therefore, he cannot release the file prematurely.
40015	The user is neither the file administrator nor a super user; therefore he has to be a member of the last entitled group in order to release the file.
40016	There is a committed version so that only members of the remaining entitled groups may reject the file.
40017	An attempt was made to use a signature field more than once. This is not permitted.
50009	The specified name contains invalid characters. The following characters can be used in names: A-Z, a-z, 0-9, and _.
50014	A field which does not exist was specified for a `sortKey` field.
50015	An invalid value was specified for the `sortOrder` field.
50024	The `presetAttributes` specification is not correct. A valid list could not be compiled.
50025	The specified name contains invalid characters. The following characters can be used in names: A-Z, a-z, 0-9, and _.
50034	The specified name contains invalid characters. The following characters can be used in names: A-Z, a-z, 0-9, and _.
50037	The specified name contains invalid characters. The following characters can be used in names: A-Z, a-z, 0-9, and _.
50038	The named value which has been specified in the list of `presetAttributes` is not permitted for this field.
50040	The specified tag contains invalid characters.
50042	The permitted file name extensions can be configured in the `mimeTypes` system configuration entry.
50043	The specified definition contains a nonexistent field or a nonexistent user group.
50044	The specified name contains invalid characters. The following characters can be used in names: A-Z, a-z, 0-9, and _.

50045	An unknown update type was specified in an update record.
50063	Job names must not begin with `system` or the underscore character. Permitted characters in job names are a - z, A - Z, 0 - 9, and the underscore.
55000	The string is made up of two characters in the range from 0 to 9, a to z, or A to Z. The characters are used for encryption.
55001	This error is generated if the server receives a payload in a version not supported by the XML interface. The way the server behaves in such cases is explained in the XML Reference manual.
55003	The link belongs to a tag in which an anchor must not be specified.
55004	The link belongs to a tag which does not support the specification of a frame target.
55005	The target folder specified in the copy or move procedure cannot be accepted as a target. The reason can be one of the following: a file with the name of the file concerned already exists; it is not a folder, you do not have sufficient permissions, it is inside the hierarchy branch to be moved or may not have any subfolders with the format of the source file.
55006	A file with the same name already exists in the directory.
55011	The specified name contains invalid characters. The following characters can be used in names: A-Z, a-z, 0-9, and _.
55015	The specified file name extension or the extension used in the operation is not included in the `validContentTypes` of the format assigned to the file.
55023	The specified format cannot be used for this file. It does not exist or is not contained in the `validSubObjClasses` of the format for the folder containing the file.
55036	The specified field name could not be assigned to a link because the field does not exist in the version to which the link belongs.
55046	This error is generated, if a user tries to generate a thumbnail image for a file whose type is neither *image* nor *resource*.
55047	The specified character set does not exist or has not been defined.
55054	For the `createAndLoad` file command a file with more than two components was specified or the path is absolute or references the parent directory.
55055	At the attempt to access a temporary directory after the Upload Wizard was called, a path consisting of several parts was specified. This is not permitted.
60014	This is an internal error. Please contact Infopark's customer support (support@infopark.de).
60015	This is an internal error. Please contact Infopark's customer support (support@infopark.de).
60016	This message is related to the Portal Manager. The specified path does not exist.
60033	The specified templete is required for the export. However, it does not exist or has not been released.
80001	A folder can only be deleted after the files it contains have been deleted.
80009	The specified field cannot be deleted, as it is still being used in the specified file format.
80010	The specified field cannot be deleted, as it is still being used for signatures in the specified workflows.
80011	The workflow cannot be deleted, as it is referenced in the specified file format in the preset `workflowName` field.

80014	The field value for the specified users must be modified in order to be able to delete the enumeration value.
85013	For the time the exported documents are being switched live, requests in the request queue of the Search Engine Server are temporarily not processed.
85015	The request to the XML interface was not processed because a preceeding request marked as preclusive failed.
85020	You have tried to edit the main content of a version. This is not possible if the file to which the version belongs is based on a format in which a main content layout has been specified.
85028	Fields are automatically added to the base field set when they are deleted from other field sets or added to the file format.
85031	The version is a committed or released version and thus cannot be modified.
85033	The named fields cannot be added to the `mandatoryAttributes` list because they are not contained in the file format's list of fields.
85034	The named fields cannot be added to the `presetAttributes` list because they are not contained in the file format's list of fields.
85035	The named fields cannot be added to the `presetFromParentAttributes` list because they are not contained in the file format's list of fields.
85036	File formats were specified for subfiles even though the file type specified for the format does not permit any subfiles.
85037	You have tried to assign a main content layout to a file format defining layouts, images, or resources. Main content layouts, however, cannot be used with these file types.
85038	You have tried to preset the main content in a file format definition. This is not possible if a main content layout has been specified.
85039	You have tried to preset the main content in a file format definition with the value of the main content of the folder containing the file. This is not possible if a main content layout has been specified.
85041	Context links must point to NPS files.
90007	The specified field occurs in the XML document, however, it is not contained in the file format to which the document belongs.
100000	The specified additional (custom) command could not be executed.
100001	The additional (custom) command could not write to a binary file.
100016	The specified values cannot be used. The value of the `master.maxSlaves` configuration entry must be greater than the `master.minIdleSlaves` value.
100029	This is a database error. The system failed to write to the job execution queue.
100030	This is a database error.
100031	This is a database error.
100032	This is a database error.
100037	This is a database error.
100038	This is a database error.
100039	A NPS application tried to access a nonexistent file for reading. This error can occur if, for example, the streaming interface tries to deliver a file associated with a ticket and this file has been deleted.

100040	This is a database error.
100043	This is a database error.
100044	This is a database or system error. Please contact Infopark's customer support (support@infopark.de).
100052	The file in which the logins and passwords of all users are stored could not be created.
100054	This is a database error.
100055	This is a database error.
100056	This is a database error.
100110	This is a database error.
100111	The draft version could not be created since there is no default file name extension.
100112	This is a database error.
100113	This is a database error.
100114	This is a database error.
100115	This is a database error.
100116	This is a database error.
100119	This is an internal system error.
100121	This is a database error.
100123	The specified type is no internal representation of a task type.
100148	This is a database error.
100149	This is a database error.
100150	This is a database error.
100151	This is a database error.
100152	The application could not access the specified control file. The file might not exist, or the application lacks read permission.
100153	This is a database error.
100154	This is a database error.
100164	During indexing by the Search Engine Server the external processor was not found or the program file was not executable.
100165	During indexing by the Search Engine Server, the attempt to transmit data to the external processor via a pipe failed.
100166	During indexing by the Search Engine Server, the attempt to read the external processor's output via a pipe failed.
100167	This is a database error.
100168	This error can have several causes. Possibly the path contains non-existent directories, or the access permissions are insufficient.
100172	During the initialization of the Verity search engine, no collections were found. They are searched for in the instance-specific *data/collections* directory.
100184	With the SES, actions to be performed are coded into the names of request files. Only two actions, `index` and `delete`, exist. With respect to such a file, an inconsistence occurred. The error is written to the system log, and the file concerned is ignored.

100185	This is a database error.
100188	The attempt to create a streaming ticket failed. Possibly the tickets directory does not exist or the application is not permitted to access it.
100197	The restriction in length is due to the file or operating system.
100209	This error is related to the Template Engine. During the export more errors occurred than the value of the `acceptableExportFailures` system configuration entry tolerates.
100210	The Portal Manager or the application which produced this message is misconfigured. It is also possible that the Portal Manager has not been started.
100219	This is a database error.
109010	Channel check function error.
109012	Version assignment function error.
109016	The stated errors occurred while `sudo` was executed.
109017	The stated errors occurred while the `generateThumbnail` function was executed.
109018	The stated errors occurred while the `usersWhere` procedure in the file *usermanAPI.tcl* was executed.
109021	The stated errors occurred while the `groupsWhere` procedure in the file *usermanAPI.tcl* was executed.
109500	The link function cannot be executed because the list of fields contains the wrong number of elements.
109501	The link function could not be executed properly.
109503	Thumbnails must be stored as JPEG images.
109511	This error is generated and the cause of the error is specified in detail if the Tcl-Code in one of the following functions produces an error: value assignment function, value display function, completion check, version assignment function, creation check for files in a folder, workflow assignment function.
120012	This error can occur if saved data have been deleted or users have worked with the server during the dump procedure.
130007	This is a fatal database error that causes the process of the NPS application concerned to be terminated. The database action concerned has not been performed.
140021	In NPSOBJ instructions only the alias names of Tcl commands can be used. To these names the actual procedure calls must be assigned in the appropriate system configuration entry (`tclSystemExecuteCommands` or `tclFormatterCommands`, for example).